CSIT5900 2024 Fall Semester Assignment #2 - The Written Part

Date assigned: Friday, Oct 18

Due time: 23:59 on Sunday, Nov 3

How to submit it: Submit your written answers as a pdf file on canvas.ust.hk.

Penalties on late papers: 20% off each day (anytime after the due time is considered late by one day)

Problem 1 (10%). Given a graph and a node in it as the starting point, the traveling salesman problem (TSP) is about finding a least cost path that starts and ends at the starting node, and goes through each other node in the graph once and exactly once. The edges of the graph are labeled by a number representing the cost of traveling between the connecting two nodes. Formulate this problem as a search problem by specifying the states, possible initial states, goal test, operators, and operator costs.

Problem 2. (10%) (Adapted from Russell and Norvig) Consider the problem of *constructing* crossword puzzles: fitting words into a grid of intersecting rows and columns of squares. Assume that a list of words (i.e. dictionary) is provided, and that the task is to fill in the rows and columns with words from this list so that if a row intersects with a column, their intersecting square has the same letter. Formulate this problem as an assignment (constraint satisfaction) problem by specifying the variables, their domains of possible values, and the constraints.

Problem 3. (10%) Consider the following two investment decisions: buy a certified deposit (CD) which will pay you a 10% annual interest rate, and buy some stocks which will either give you a 30% annual return (with 0.7 probability) or incur a 10% loss for you (with 0.3 probability). In terms of expected utility, you should of course buy stocks. However, suppose you are conservative and cannot tolerate any loss of your principal. In this case, you have no choice but to deposit your money into a CD. Now consider this problem for a 3 years time span. You start with 1 unit of money. Each year you can choose only one way to invest your entire fund (meaning you are not allowed to diversify, like 50% CD, 50% stocks). You definitely don't want to lose any money at the end of the 3 years, but you are okay with a "temporary" loss during the time (for example, it is okay for you to lose some money in the first year as long as you can make it up later). Formalize this problem as a Markov decision process and compute its optimal policy.

Hints A state needs to contain information about how many money you have in this state. In other words, a state needs to encode the history: which actions you have done so far and what their outcomes are if the actions are non-deterministic.

Problem 4. (15%) Recall our boundary following robot. The correct behavior is that the robot should follow a boundary in a consistent way, clockwise if it's inside the wall of the room, and counter-clockwise if it is outside an obstacle in the room.

4.1 Formulate this task as a MDP.

4.2 Formulate this task as a reinforcement learning problem.

Problem 5. (15%) Consider the following search problem:

- state space: $\{S, A, B, G\}$;
- operators: O_1 maps S to A with cost 4, O_2 maps S to B with cost 2, O_3 maps B to A with cost 1, and O_4 maps from A to G with cost 5;
- initial state S;
- goal state G;
- heuristic function h: $h(S) = 7$, $h(A) = 1$, $h(B) = 6$, and $h(G) = 0$.

Find a solution using A^* search by tree. Number the nodes according to their order of expansion, and for each node give its $f(n) = g(n) + h(n)$ value.

Problem 6. (10%) Perform minimax with perfect decision on the tree given in the following figure (the leaves are terminal nodes, and the numbers next to them are their utility values).

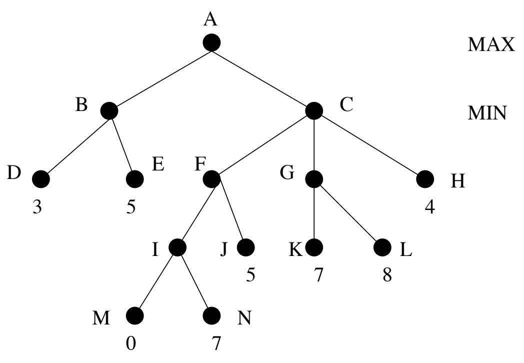

Figure 1: A minimax search tree

Problem 7. (10%) Perform a left-to-right alpha-beta prune on the tree in the above exercise. Perform a right-to-left prune on the same tree. Left-to-right or right-to-left means the order by which the leaf nodes are generated. So for left-to-right, the first leaf node considered is D, followed by E, followed by M and so on.

CSIT5900 2024 Fall Semester Assignment #2 - The Programming Part

Date assigned: Friday, Oct 18

Due time: 23:59 on Sunday, Nov 3

How to submit it: Submit your codes as a zip file named YourStudentID.zip. Do not include absolute path like `‘‘C:/..../...’’` in your code. Use relative path so that our autograder can run your program.

Penalties on late papers: 20% off each day (anytime after the due time is considered late by one day)

Problem 8. (20%) Programming assignment (A*). This programming assignment uses Berkeley AI Project[1] Pacman game for you to practice the A* algorithm. You'll find the relevant files/programs in the archive `assign2-program27.zip` (Python 2.7) or `assign2-program3.zip` (Python 3), including many testing files.

In `searchAgents.py`, you'll find a fully implemented `SearchAgent`, which plans out a path through Pacman's world and then executes that path step-by-step. The search algorithms for formulating a plan are not implemented – that's your job.

First, test that the `SearchAgent` is working correctly by running:

```
python pacman.py -l tinyMaze -p SearchAgent -a fn=tinyMazeSearch
```

The command above tells the `SearchAgent` to use `tinyMazeSearch` as its search algorithm, which is implemented in `search.py`. Pacman should navigate the maze successfully.

Now it's time to write full-fledged generic search functions to help Pacman plan routes! Remember that a search node must contain not only a state but also the information necessary to reconstruct the path (plan) which gets to that state.

Hint: We have implemented the breadth-first search (BFS) algorithm in the `breadthFirst Search` function for you. If you have no idea about where to start, it should be a good reference. Since the algorithms are very similar, only a few changes are needed to implement DFS based on BFS.

Here are some general notes:

1. All of your search functions need to return a list of *actions* that will lead the agent from the start to the goal. These actions all have to be legal moves (valid directions, no moving through walls).

2. Make sure to **use** the `Stack`, `Queue` and `PriorityQueue` data structures provided to you in `util.py`! These data structure implementations have particular properties which are required for compatibility with the autograder.

[1] `http://ai.berkeley.edu`

3. Your code should quickly find a solution for:

   ```
   python pacman.py -l tinyMaze -p SearchAgent
   ```

   ```
   python pacman.py -l mediumMaze -p SearchAgent
   ```

   ```
   python pacman.py -l bigMaze -z .5 -p SearchAgent
   ```

   The Pacman board will show an overlay of the states explored, and the order in which they were explored (brighter red means earlier exploration). Is the exploration order what you would have expected? Does Pacman actually go to all the explored squares on his way to the goal?

4. If you use a `Stack` as your data structure, the solution found by your DFS algorithm for `mediumMaze` should have a length of 130 (provided you push successors onto the fringe in the order provided by getSuccessors; you might get 246 if you push them in the reverse order). Is this a least cost solution? If not, think about what depth-first search is doing wrong.

Tasks

1. **A* Search (10 pts)**

   Implement A* graph search in the empty function `aStarSearch` in `search.py`. A* takes a heuristic function as an argument. Heuristics take two arguments: a state in the search problem (the main argument), and the problem itself (for reference information). The `nullHeuristic` heuristic function in `search.py` is a trivial example.

   You can test your A* implementation on the original problem of finding a path through a maze to a fixed position using the Manhattan distance heuristic (implemented already as `manhattanHeuristic` in `searchAgents.py`).

   ```
   python pacman.py -l bigMaze -z .5 -p SearchAgent\
   -a fn=astar,heuristic=manhattanHeuristic
   ```

2. **Corners Problem: Formulation (5 pts)**

   The real power of A* will only be apparent with a more challenging search problem. This task is for you to formulate a new problem called the corners problem. The nest task is to design a heuristic for it.

   In *corner mazes*, there are four dots, one in each corner. Our new search problem is to find the shortest path through the maze that touches all four corners (whether the maze actually has food there or not). Note that for some mazes like `tinyCorners`, the shortest path does not always go to the closest food first! *Hint*: the shortest path through `tinyCorners` takes 28 steps.

   Implement the `CornersProblem` search problem in `searchAgents.py`. You will need to choose a state representation that encodes all the information necessary to detect whether all four corners have been reached. Now, your search agent should solve:

   ```
   python pacman.py -l tinyCorners -p SearchAgent\
   -a fn=bfs,prob=CornersProblem
   ```

```
python pacman.py -l mediumCorners -p SearchAgent\
-a fn=bfs,prob=CornersProblem
```

Hints You need to define an abstract state representation that *does not* encode irrelevant information (like the position of ghosts, where extra food is, etc.). In particular, do not use a Pacman `GameState` as a search state. Your code will be very, very slow if you do (and also wrong). The only parts of the game state you need to reference in your implementation are the starting Pacman position and the location of the four corners.

3. **Corners Problem: A* Heuristic (5 pts)**

   Implement a non-trivial, admissible heuristic for the `CornersProblem` in `cornersHeuristic`.

   ```
   python pacman.py -l mediumCorners -p AStarCornersAgent -z 0.5
   ```

   Note that `AStarCornersAgent` is a shortcut for

   ```
   -p SearchAgent\
   -a fn=aStarSearch,prob=CornersProblem,heuristic=cornersHeuristic.
   ```

   Notice that you heuristic must be admissible, i.e. the heuristic values must be lower bounds on the actual shortest path cost to the nearest goal (and non-negative). It should not be a trivial one like returning zero everywhere.