Homework 1
Search

15-381/781: Artificial Intelligence (Fall 2016)
OUT: Sept 2, 2016
DUE: Sept 19, 2016 at 11:59pm

Instructions

Homework Policy

Homework is due on autolab by the posted deadline. Assignments submitted past the deadline will incur the use of late days.

You have 6 late days, but cannot use more than 2 late days per homework. No credit will be given for homework submitted more than 2 days after the due date. After your 6 late days have been used you will receive 20% off for each additional day late.

You can discuss the exercises with your classmates, but you should write up your own solutions. If you find a solution in any source other than the material provided on the course website or the textbook, you must mention the source. You can work on the programming questions in pairs, but theoretical questions are always submitted individually. Make sure that you include a README file with your andrew id and your collaborator's andrew id.

Submission

Please create a tar archive of your answers and submit to Homework 1 on autolab. You should have three files in your archive: a completed `problems.py` for the programming portion, a PDF for your answers to the written component, and a README file with your Andrew ID and your collaborators' Andrew IDs. Your completed functions will be autograded by running through several test cases and their return values will be compared to the reference implementation. There is a `sample.txt` that contains sample inputs and outputs for reference.

1 Written [40 points]

1.1 Approximate Admissibility [10 points]

Let $h^*(x)$ be the shortest distance between a state x and a goal state. Let $h(\cdot)$ be a heuristic that overestimates $h^*(\cdot)$ by at most ϵ, i.e., for all states x, $h(x) \leq h^*(x) + \epsilon$. Assume that $h(\cdot)$ still assigns 0 to all goal states. Prove that A* *tree search* using h finds a goal state t whose cost is at most ϵ more than the optimal goal. Formally, if s is the start state and t is the goal state returned by A*, then $g(t) \leq h^*(s) + \epsilon$.

1.2 Dominating Heuristic Implies Efficiency [15 points]

Let $h_1(\cdot)$ and $h_2(\cdot)$ be two admissible heuristics, and define $f_1 = g + h_1$, $f_2 = g + h_2$. As before, let $h^*(x)$ be the shortest distance between a state x and a goal state. Let s denote the start node. Prove that if h_1

dominates h_2 then every node expanded by A* tree-search with f_1 is also expanded by A* tree-search with f_2 or is in the set $\{x|f_1(x) = h^*(s)\}$. (Remark: when an algorithm needs to tie-break between nodes with the same f-value on its open-list, assume that it expands the vertex with the smallest value according to an in-order traversal of the tree.)

1.3 Cryptarithmetic Puzzle CSP [15 points]

In this question we will solve the following cryptarithmetic problem.

	O	D	D
$+$	O	D	D
E	V	E	N

Table 1: The cryptarithmetic problem

Each letter stands for a *distinct* digit, and the goal is to find an assignment of digits to letters so that the sum is correct. We also require that there be no leading zeroes.

1.3.1 [5 points]

What are the variables, domains and constraints?

1.3.2 [10 points]

Solve the cryptarithmetic problem using a backtracking search. Use forward checking as you progress as well as the *minimum remaining values* and *least constraining value* heuristics. Additionally, assign a value to a variable in $\{E, V, O, D, N\}$ in the first round. Among the variables in $\{E, V, O, D, N\}$, break ties for variable choice alphabetically, and then for assignment choice by choosing the smallest remaining value. Show all your steps.

1.4 Graduate Problem: Strong Consistency Implies Global Consistency [15 points/5 BONUS points for undergrads]

Let there be a binary CSP with domain $D = \{1, \ldots, d\}$. Prove that if it is strong $(d+1)$-consistent, then it is globally consistent. (Remark: In general, if the CSP has arity r and is strong $(d(r-1)+1)$-consistent, then it is globally consistent). For reference, here are the definitions: A CSP is k-consistent if for every $Y_1, , Y_k$, any legal assignment $for Y_1, , Y_{k-1}$ can be extended to a legal assignment for $Y_1, , Y_k$. Strong k-consistency means k' consistency for every $k' \leq k$. Global consistency means strong n-consistency.

2 Programming [60 points]

Amayon, the online shopping conglomerate, has announced their big initiative to use robots to deliver packages to customers. Being part of their AI team, your job is to program the robots to efficiently plan a good delivery route.

Specifically, the robot will be given a map of a neighborhood, and must figure out a fast route to get to every residence. The map will be a 2D rectangular grid divided into discrete squares. Each square is either free, an obstacle, or a residence. The route must visit every residence at least once. A goal node is any node where the robot has already visited every residence at least once.

As a visual reference, here is an ASCII rendition of a neighborhood map with 2 residences and some walls:

```
S.#..
..#.G
.##.#
....G
```

This will be represented in the code as a list of lists, with blank spaces being 0, obstacles being -1 and residences being 1. The exact structure for the map above is

```
[
[0,0,-1,0,0],
[0,0,-1,0,1],
[0,-1,-1,0,-1],
[0,0,0,0,1]
]
```

The start position is given as a tuple of row-column coordinates; the start position is `(0,0)` for this grid. The cost is uniformly 1 for every attempted move. The robot can only move in the 4 cardinal directions (up, right, down, left), and cannot move off of the grid. Trying to move into walls or off the grid results in staying put.

A node should encompass the current position of the robot, as well as which residences have been visited and which have not. Two nodes with the same position, but different visited residences are different nodes.

Note, when adding successors of a search node onto your priority queue, make sure to add them in the following order: up, right, down, left, so that they will be explored in that order. Tie-breaking for A* should be done by favoring nodes added earlier. Tie-breaking for BFS is the same; however tie-breaking for DFS, if using a stack, should result in adding the successors in the reverse order such that the successors will be visited in the correct order. Also, the autograder will be checking your search algorithms by checking which nodes you expand. An expanded node is either a goal node, or a node for which you have added successors. This means duplicate nodes that you ignore don't count as expanded.

For example inputs and outputs, see `sample.txt`.

2.1 DFS Search [10 points]

Implement depth-first search in the `dfs(grid, start)` function. Note that you should only do cycle detection and not re-expand any nodes along a single path; your algorithm should allow for the same node to be visited by different paths. Cycle detection preserves the low memory requirements of DFS.

2.2 BFS Search [10 points]

Implement breadth-first search in the `bfs(grid, start)` function. Note that you should globally keep track of visited nodes and not expand any node more than once.

2.3 A* Search [15 points]

Implement A* search in the `astar(grid, start, hfunc)` function. The `heapq` module is one possible way to implement your priority queue. Be sure to use A* graph search, so that you don't do redundant work. Reaching the goal should count as expanding the goal, so the goal node should be the last node in your expanded list output. The `hfunc` parameter should be a function with a single argument i.e. `hfunc(data)`, but it is up to your A* function how to use the argument. The autograder will test your A* function by passing an `hfunc` that always returns 0. Note that this trivial heuristic is consistent.

2.4 Simple Heuristic [5 points]

Your simple heuristic should return the number of residences that haven't been visited. Implement this in the `simple(data)` function. The autograder will test this by passing your `simple(data)` as the `hfunc` parameter in your A* function.

2.5 IDA* Search [15 points]

Implement IDA* search in the `idastar(grid, start, hfunc)` function. IDA* is iterative deepening but tweaked to being an informed search. Every iteration uses DFS with cycle detection as the search algorithm, but you also keep track of f-values; furthermore, the stopping criterion is no longer the depth, but an f-value threshold. The first threshold should be 0. During an iteration, you should keep track of every time you see a node whose f-value exceeds the current threshold. The threshold for the next iteration will then be the smallest f-value that you saw exceeding the current threshold. This is the A* equivalent of increasing the depth. The autograder will use your simple heuristic `simple(data)` as the `hfunc` parameter in your IDA* function.

2.6 Custom Heuristic Challenge [5 points]

The simple heuristic is quick to compute, but not very effective in guiding A*. Try to come up with your own heuristic that reduces the number of nodes expanded. Your heuristic should be consistent. In your writeup, you should prove that your custom heuristic is consistent. Implement this in the `custom(data)` function. The autograder will test this by passing your `custom(data)` as the `hfunc` parameter in your A* function. You will get full marks by expanding fewer nodes than the simple heuristic. Depending on your placing on the leaderboard, you will receive up to 5 bonus points.