Graduate Artificial Intelligence 15-780

Homework 2: *MDPs and Search*

Out on February 15
Due on February 29

Problem 1: MDPs [Felipe, 20pts]

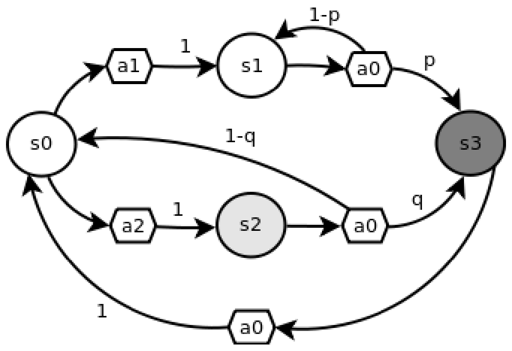

Figure 1: MDP for Problem 1. States are represented by circles and actions by hexagons. The numbers on the arrows from actions to states represent the transition probability, for instance, $P(s_3|s_2, a_0) = q$. Each of the parameters p and q are in the interval $[0, 1]$. The reward is 10 for state s_3, 1 for state s_2 and 0 otherwise.

For this question, consider the infinite-horizon MDP $\mathscr{M}$ represented by Figure 1 with discount factor $\gamma \in [0, 1)$.

a) List all the possible policies for $\mathscr{M}$. [2 pts]

b) Show the equation representing the optimal value function for each state of $\mathscr{M}$, i.e. $V^*(s_0), V^*(s_1), V^*(s_2)$ and $V^*(s_3)$. [4 pts]

c) Is there a value for p such that for all $\gamma \in [0, 1)$ and $q \in [0, 1]$, $\pi^*(s_0) = a_2$? Explain. [5 pts]

d) Is there a value for q such that for all $\gamma \in [0, 1)$ and $p > 0$, $\pi^*(s_0) = a_1$? Explain. [5 pts]

e) Using $p = q = 0.25$ and $\gamma = 0.9$, compute π^* and V^* for all states of $\mathscr{M}$. You can either solve the recursion of item (b) or implement value iteration. In the latter case, the error between V^t and V^* allowed is $\epsilon = 10^{-3}$, therefore the stop criterion should be $\max_s |V^{t-1}(s) - V^t(s)| \leq \epsilon(1-\gamma)/\gamma$. [4 pts]

Problem 2: Another interpretation for the discount factor [Felipe, 40pts]

In this problem you will explore the difference between MDPs and the Stochastic Shortest Path Problems (SSPs) and find another possible interpretation for the discount factor for infinite-horizon MDPs. In order to simplify the math, MDPs are parametrized using a cost function as opposed to a reward function, that is, an MDP is the tuple $\mathscr{M} = \langle S, s_0, A, C, P \rangle$ where:

- S is the finite state space;
- $s_0 \in S$ is the initial state;
- A is the finite action space;
- $P(s'|s, a)$ represents the probability of $s' \in S$ be the resulting state of applying action $a \in A$ in state $s \in S$; and
- $C(s, a) \in (0, \infty)$ is the cost of applying action $a \in A$ in state s.

Using this parametrization, the optimal value function for infinite-horizon MDPs with discount factor $\gamma \in [0, 1)$ is given by equation (1). Notice that the min operator is used instead of max because the parametrization uses cost instead of reward.

$$V^*(s) = \min_{a \in A} \left\{ C(s, a) + \gamma \sum_{s' \in S} P(s'|s, a) V^*(s') \right\} \tag{1}$$

Stochastic Shortest Path Problems

Stochastic Shortest Path Problems (SSPs) are represented as the tuple $\mathscr{S} = \langle S, s_0, G, A, C, P\rangle$, where S, s_0, A, C and P are defined as in the MDP case and $G \subset S$ is the set of goal states. Both infinite-horizon MDPs and SSPs have as solution a policy, however this policy:

- minimizes the expected long-term cost for infinite-horizon MDPs; and
- minimizes the expected cost to reach a goal state for SSPs.

Questions

For all the questions bellow, we assume that for all state $s \in S$, there exists a path from s to $s_G \in G$ for the SSPs.

a) Explain using no more than 2 sentences why infinite-horizon MDPs need the discount factor γ. [2 pts]

b) Explain using no more than 2 sentences why SSPs do **not** need the discount factor γ. [6 pts]

c) Derive the optimal value function for SSPs, i.e. the equivalent of equation (1) for SSPs. [6 pts]

d) SSPs are more expressive than infinite-horizon MDPs, i.e. any infinite-horizon MDP can be transformed to an SSP, however not all SSPs can be represented as an infinite-horizon MDP. Given an infinite-horizon MDP $\mathscr{M} = \langle S, s_0, A, C, P\rangle$ with discount factor $\gamma \in [0,1)$ show how to translate $\mathscr{M}$ into the SSP $\mathscr{S} = \langle S', s_0, G, A, C', P'\rangle$. [20 pts]

e) Give an interpretation for the discount factor γ based in the previous item using no more than 2 sentences. [6 pts]

Problem 3: In search of stolen cheese [John, 40pts]

You are a mouse, and some thief has stolen your cheese! Worse still, this bandit has hidden each of your blocks of cheese in a different, difficult maze. You must retrieve your cheese from these mazes, and you must do it quickly – winter is coming.

Luckily, a sympathetic friend has given you a set of maps, one for each of the mazes. It turns out these mazes consist of $m \times n$ square blocks that are either *passable* or *impassable*. There is exactly one *entrance* to each maze, and exactly one *goal* location in the maze where your cheese sits. You know both these locations, as well as the passable/impassable traits of every block in each maze.

Being a clever, computationally aware mouse, you decide to use the A* search algorithm to plot a path through each maze to its guarded block of cheese. Moving from one block to another takes *energy*; since you'd like to conserve energy for the winter, your goal is to *minimize* the amount of energy you use to retrieve each block of cheese.

a) Assume each movement takes a single unit of energy. If you can only move vertically or horizontally, describe an admissible heuristic for this problem. Now let diagonal movement be allowed; is your heuristic still admissible? Why or why not? [5 pts]

b) Using the admissible heuristic you developed earlier, implement the A* algorithm in the programming language of your choice. Run your implementation of A* on each of the mazes. In a single table, record the *optimal path lengths*, as well as the total number of nodes that were *visited* and total number of nodes that were *explored*, for each maze.

   For this problem and the next, only horizontal and vertical (i.e., no diagonal) movement is allowed. Each movement takes one unit of energy. [15 pts]

c) Is A^* really worth the effort? As you learned as an undergradumouse, DFS and BFS will also find paths to the cheese. To test this, you decide to empirically compare DFS, BFS, and A*.

First, tweak your implementations of A*, DFS, and BFS to include the following randomizations:

A*. When selecting the node with lowest f value, randomly select from all nodes with the same lowest f value.

BFS. Enqueue new nodes (i.e., the unexplored neighbors of the current explored node) in random order.

DFS. Push new nodes (i.e., the unexplored neighbors of the current explored node) onto the stack in random order.

For each maze, run your implementation of A* against implementations of DFS and BFS at least 100 times. In a single table, record the average path lengths, nodes visited, and nodes explored for each of the three algorithms. In general, did A* result in significant gains? [20 pts]

How to load a map of the maze

Your sympathetic friend left a tarball of the maps on the course webpage. They are formatted as follows:

```
<map-name>
<num-rows> <num-columns>
<x-coord-of-entrance> <y-coord-of-entrance>
<x-coord-of-cheese> <y-coord-of-cheese>
0 1 0 ... num-columns ... 0 1
0 0 1 ... num-columns ... 1 1
. . .         .           . .
. . .         .           . .
. . .         .           . .
```

Where a 0 at cell (i, j) means the block at location (i, j) is passable, and a 1 means impassable. These maps are 0-indexed. That is, coordinates run from $(0,0)$ to $(m-1, n-1)$, with m being the number of rows and n the number of columns.

For example, the following describes a map named `easy-map` with 3 rows and 3 columns. The entrance is in the top-left corner of the maze at $(0,0)$, and the cheese is in the bottom-right corner at $(2,2)$.

```
easy-map
3 3
0 0
2 2
0 1 1
0 0 1
1 0 0
```