Graduate Artificial Intelligence 15-780

Homework 2: *MDPs and Search*

Out on February 15
Due on February 29

Problem 1: MDPs [Felipe, 20pts]

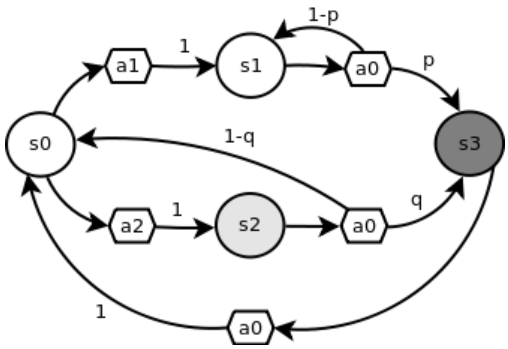

Figure 1: MDP for Problem 1. States are represented by circles and actions by hexagons. The numbers on the arrows from actions to states represent the transition probability, for instance, $P(s_3|s_2,a_0)=q$. Each of the parameters p and q are in the interval $[0,1]$. The reward is 10 for state s_3,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

Answer: There are two possible policies:

	s_0	s_1	s_2	s_3
π_1	a_1	a_0	a_0	a_0
π_2	a_2	a_0	a_0	a_0

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]

Answer:

$$
\begin{aligned}
V^*(s_0) &= \max_{a\in\{a_1,a_2\}} 0+\gamma\sum_{s'}P(s'|s,a)V^*(s') = \gamma\max\{V^*(s_1),V^*(s_2)\}\\
V^*(s_1) &= \max_{a\in\{a_0\}} 0+\gamma\sum_{s'}P(s'|s,a)V^*(s') = \gamma[(1-p)V^*(s_1)+pV^*(s_3)]\\
V^*(s_2) &= \max_{a\in\{a_0\}} 1+\gamma\sum_{s'}P(s'|s,a)V^*(s') = 1+\gamma[(1-q)V^*(s_0)+qV^*(s_3)]\\
V^*(s_3) &= \max_{a\in\{a_0\}} 10+\gamma\sum_{s'}P(s'|s,a)V^*(s') = 10+\gamma V^*(s_0)
\end{aligned}
$$

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]

Answer: Notice that $\pi^*(s_0)=\text{argmax}_{a\in\{a_1,a_2\}}\gamma\sum_{s'}P(s'|s,a)V^*(s')$, therefore if $\gamma=0$ then a_1 and a_2 are tied and π^* is not unique, so we can't guarantee that $\pi^*(s_0)=a_2$. For $\gamma>0$, $\pi^*(s_0)=a_2$ iff $V^*(s_2)>V^*(s_1)$ for all $q\in[0,1]$. If $p=0$, then $V^*(s_1)=\gamma V^*(s_1)=0$ and since $V^*(s_2)=1+\gamma[(1-q)V^*(s_0)+qV^*(s_3)]\geq 1$, we have that $V^*(s_2)>V^*(s_1)$ and $\pi^*(s_0)=a_2$ for all $\gamma\in(0,1)$ and $q\in[0,1]$.

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]

Answer: No. Since $V^*(s_2)=1+\gamma[(1-q)V^*(s_0)+qV^*(s_3)]\geq 1$, then $\pi^*(s_0)=a_1$ iff $V^*(s_1)>V^*(s_2)\geq 1$ for all $\gamma\in[0,1)$ and $p>0$. We have that $V^*(s_1)=\gamma[(1-p)V^*(s_1)+pV^*(s_3)]=\frac{\gamma pV^*(s_3)}{1-\gamma(1-p)}$. Therefore we can always find a value for γ sufficiently small such that $V^*(s_1)<1$. Notice that if γ equals 0, then π^* is not unique (as in the previous item).

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]

Answer:

	s_0	s_1	s_2	s_3
V^*	14.1846	15.7608	15.6969	22.7661
π^*	a_1	a_0	a_0	a_0

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 $\mathcal{M} = \langle S, s_0, A, C, P \rangle$ 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 $\mathcal{S} = \langle S, s_0, G, A, C, P \rangle$, 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]
Answer: **Mathematical explanation:** Since $V^*(s)$ represents the expected accumulated reward, the discount factor $\gamma \in [0,1)$ is necessary in order to guarantee that $|V^*(s)| < \infty$.
Economical interpretation: We can see γ as the inflation per time step, i.e., one unit of reward at time t is worth γ^t units of reward now.

b) Explain using no more than 2 sentences why SSPs do **not** need the discount factor γ. [6 pts]
Answer: SSPs do not need the discount factor γ because they have a termination condition, i.e. when a goal state is reached the problem is finished.

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

$$V^*(s) = \begin{cases} \min_{a \in A} \left\{ C(s,a) + \sum_{s' \in S} P(s'|s,a) V^*(s') \right\} & \text{if } s \in S \setminus G \\ 0 & \text{if } s \in G \end{cases}$$

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]

Answer: Given an MDP $\mathscr{M} = \langle S, s_0, A, C, P\rangle$ with discount factor $\gamma \in [0,1)$, we need to construct an SSP $\mathscr{S} = \langle S', s_0, G, A, C', P'\rangle$ such that, for all $s \in S$, $V^*_{\text{MDP}}(s) = V^*_{\text{SSP}}(s)$. Now, define:

- $S' = S \cup \{s_g\}$
- $G = \{s_g\}$
- $P'(s'|s,a) = \begin{cases} \gamma P(s'|s,a) & \text{if } s \in S \text{ and } s' \in S \\ 1-\gamma & \text{if } s \in S \text{ and } s' = s_g \end{cases}$, for all $a \in A$, and
- $C'(s,a) = \begin{cases} C(s,a) & \text{if } s \in S \\ 0 & \text{if } s = s_g \end{cases}$, for all $a \in A$.

Now, for all $s \in S$, we have that:

$$\begin{aligned} V^*_{\text{SSP}}(s) &= \min_{a\in A}\left\{C'(s,a) + \sum_{s'\in S'} P'(s'|s,a)V^*_{\text{SSP}}(s')\right\} \\ &\overset{(*)}{=} \min_{a\in A}\left\{C(s,a) + (1-\gamma)V^*_{\text{SSP}}(s_g) + \sum_{s'\in S} \gamma P(s'|s,a)V^*_{\text{SSP}}(s')\right\} \\ &\overset{(**)}{=} \min_{a\in A}\left\{C(s,a) + \gamma\sum_{s'\in S} P(s'|s,a)V^*_{\text{SSP}}(s')\right\} \\ &= V^*_{\text{MDP}}(s) \end{aligned}$$

$(*)$ Applying the definition of C', P' and S'. Also, notice that $s \in S$, i.e. $s \neq s_g$.
$(**)$ $V^*_{\text{SSP}}(s_g) = 0$ by definition of SSPs. □

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

Answer: As we can see from the previous item, the discount factor γ for infinite-horizon MDPs can be interpreted as the probability that the agent will keep *playing the game* and accumulating more rewards. In other words, at each time step, the agent *dies* with probability $1-\gamma$.

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]
Answer: Most people got this answer correct. Let the mouse's current position be $m = (x_m, y_m)$ and the goal's position be $g = (x_g, y_g)$. Then a sampling of admissible heuristics include:

Manhattan distance. By far the most popular heuristic in the class. It is named after the grid-like geometry of the New York City streets. Calculated as

$$h^M(m,g) = |x_m - x_g| + |y_m - y_g|$$

Clearly, if we allow diagonal moves this heuristic is no longer admissible. Take as a counterexample $m' = (0,0)$ and $g' = (1,1)$. Then $h^M(m',g') = 2 > 1 = |opt\_path(m',g')|$. The heuristic overestimated the distance to goal, violating the admissibility condition.

Euclidean distance. Second most popular heuristic in the class. Calculated as

$$h^E(m,g) = \sqrt{|x_m - x_g|^2 + |y_m - y_g|^2}$$

We see that $\forall m', g'.h^M(m',g') \geq h^E(m',g')$ and, since h^M is guaranteed never to overestimate the distance to the goal, so to is h^E.

If we allow diagonal moves, this heuristic is no longer admissible. Recall that our cost function was *one energy unit per move*. This means that a move from, e.g., $(0,0)$ to $(1,1)$ has cost 1, not cost $\sqrt{2}$ as it would in Euclidean space. Thus, the heuristic can overestimate the distance to the goal, violating the admissibility condition.

Chebychev distance. The least popular heuristic in the class (two people mentioned it). This heuristic is calculated as

$$h^C(m,g) = \max\left(|x_m - x_g|, |y_m - y_g|\right)$$

We see that $\forall m', g'.h^E(m',g') \geq h^C(m',g')$ and, since h^E is guaranteed never to overestimate the distance to the goal, so to is h^C. This is the "loosest" heuristic reported (there are looser heuristics, e.g., $\forall m', g'.h^{\leq}(m',g') = 0$, which turns A^* search into standard BFS).

Unlike the other heuristics discussed above, if we allow diagonal moves, the Chebychev heuristic is still admissible.

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]
Answer: Again, most people got this answer correct. A few comments:

- In A^*, you are constantly enqueuing nodes that have an associated numeric "score" (i.e., the f value) and then searching for the lowest scoring node in the queue. Many people did a linear search over all nodes in the queue to find a lowest score. This resulted in very long A^* runtimes, even on the very small maps in this homework assignment. A better method would be to use a data structure like a *priority queue* or a *heap*. It's important to look at an algorithm and optimize for the type of operations you'll be doing frequently! For more information, see:
  - `http://en.wikipedia.org/wiki/Priority_queue`
  - `http://en.wikipedia.org/wiki/Heap_%28data_structure%29`

- After the A^* search concludes, we are guaranteed to have an optimal path from start to goal. An easy way to retrieve this path is to, at each node you explore, store a pointer to the node that you visited immediately before (the "parent" node). In a simple grid world like the mazes in this homework, one could also store "parent" coordinates in an $m \times n$ array, although this is *not* recommended due to wasting memory (you're allocating a slot in the array for every possible cell, not just those the algorithm visited). Some solutions I saw did worse things like store entire paths in each node, or search over the final matrix of f values to find a path after A^* converged. This resulted, again, in extremely increased runtime and/or memory requirements for A^*.
- More than half of the class did not read the file format! Specifically, the second line of the maze file is formatted `<num-rows> <num-columns>`, while the third and fourth lines report the start and goal coordinates in `<x> <y>` format. Many people read the start and goal lines in `<y> <x>` format, which made the Scotty maze seem unsolvable. By luck of the draw, all the other mazes still had valid paths from valid start locations to valid goal locations. **I did not take off points for misreading the file format.**

The best runtimes (with correct answers) I saw were as follows:

Map	Path	Length	Explored	Visited Time(s)
cubism 320x320	504	41420	38973	0.093472
diffusion 640x640	1237	165994	164567	0.642666
empty 320x320	508	1521	507	0.012962
maze 320x320	3975	47917	47871	0.067668
polar 640x640	993	108311	107727	0.294654
scotty 214x234	358	4825	4483	0.012298

These were all done by the same student. The algorithm was implemented in OCaml with a priority queue and handily beat your TA's Python implementation. For a very easy-to-read implementation of A^* in a Python-like language (which appears to be, with Matlab, the language of choice in our class), see `http://en.wikipedia.org/wiki/A*#Pseudocode`.

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]

Answer: Once again, most people got this answer entirely or mostly correct. You can change your A^* code to BFS or DFS in at most two lines.

(a) For BFS, set the heuristic function to always return zero. This means that the f-score equals the g-score for all explored nodes.

(b) For DFS, swap the queue out for a stack, and ignore the f values of nodes.

The A^* machinery is overkill, but these changes are extremely programmer-friendly to make, given an A^* implementation from the question above.

The takeaway messages from this problem are as follows:

- BFS still returns an optimal path (albeit potentially a different one than A^*), but may explore many more nodes to do so.
- DFS can return extremely suboptimal paths and, typically, does. However, on some types of maps it can explore fewer nodes than, e.g., the A^* algorithm.
- Randomization can help, but (at least on these maps, with the most common heuristics) not by much. It can help, though!

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
```