3 MDP (12 points)

1. (3 pts) In MDPs, the values of states are related by the Bellman equation:

$$U(s) = R(s) + \gamma \max_a \sum_{s'} P(s'|s,a)U(s')$$

where $R(s)$ is the reward associated with being in state s. Suppose now we wish the reward to depend on actions; i.e. $R(a,s)$ is the reward for doing a in state s. How should the Bellman equation be rewritten to use $R(a,s)$ instead of $R(s)$?

$$U(s) = \max_a (R(a,s) + \gamma \sum_{s'} P(s'|s,a)U(s'))$$

2. (9 pts) Can any search problem with a finite number of states be translated into a Markov decision problem, such that an optimal solution of the latter is also an optimal solution of the former? If so, explain precisely how to translate the problem AND how to translate the solution back; illustrate your answer on a 3 state search problem of your own choosing. If not give a counterexample.

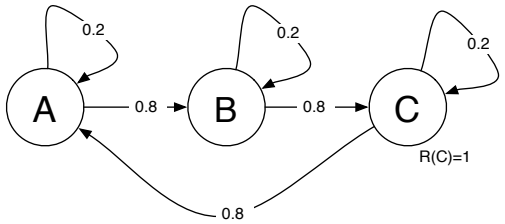
Yes, a finite search problem is a deterministic MDP:

- *states in the search problem are states in MDP*
- *if S_j is descendant of state S_i in the search, call this action $a_{i,j}$, and set $P(S_j|S_i,a_{i,j}) = 1$. All the other transition probabilities are 0.*
- $R(a_{i,j},i) = -cost(S_i,S_j)$
- *goal nodes in the search problem are terminal nodes with no possible actions.*

To translate an optimal policy back to a search path, simply start at the start state and follow the optimal action until the goal is reached.

4 Reinforcement Learning (13 points)

Consider an MDP with three states, called A, B and C, arranged in a loop.



There are two actions available in each state:

- $Move_s$: with probability 0.8, moves to the next state in the loop and with probability 0.2, stays in the same state.
- $Stay_s$: with probability 1.0 stays in the state.

There is a reward of 1 in state C and zero reward elsewhere. The agent starts in state A. Assume that the discount factor is 0.9, that is, $\gamma = 0.9$.

1. (6 pts) Show the values of $Q(a, s)$ for 3 iterations of the TD Q-learning algorithm (equation 21.8 in Russell & Norvig):

$$Q(a, s) \leftarrow Q(a, s) + \alpha(R(s) + \gamma \max_{a'} Q(a', s') - Q(a, s))$$

Let $\alpha = 1$, note the simplification that follows from this. Assume we always pick the Move action and end up moving to the adjacent state. That is, we see a state-action sequence: *A, Move, B, Move, C, Move, A*. The Q values start out as 0.

This wording is a bit ambiguous. If we display the values after each action in one trail, we get:

	iter=0	iter=1	iter=2	iter=3
Q(Move,A)	0	0	0	0
Q(Stay,A)	0	0	0	0
Q(Move,B)	0	0	0	0
Q(Stay,B)	0	0	0	0
Q(Move,C)	0	0	0	1
Q(Stay,C)	0	0	0	0

If we think about the 3 actions as a trial and show what happens after each full trial, we get:

	iter=0	iter=1	iter=2	iter=3
Q(Move,A)	0	0	0	0.81
Q(Stay,A)	0	0	0	0
Q(Move,B)	0	0	0.9	0.9
Q(Stay,B)	0	0	0	0
Q(Move,C)	0	1	1	1
Q(Stay,C)	0	0	0	0

2. (2 pts) Characterize the weakness of Q-learning demonstrated by this example. Hint. Imagine that the chain were 100 states long.

   The Q-learning updates are very local. In a chain of 100 states we would have to go around 100 times before we had non-zero Q values in each state.

3. (3 pts) Why might a solution based on ADP (adaptive dynamic programming) be better than Q-learning?

   ADP updates all the states to make optimal use of whatever information we have discovered. So, for example, all states leading to a good state would have their values increased.

4. (2 pts) On the other hand, what are the disadvantages of ADP based approaches (compared to Q-learning)?

   - *ADP is global. If there are lots of states then the cost of these updates will be very large.*
   - *If we use ADP to learning the utility/value function directly (instead of the Q-function), this will require also learning the transition function so that we can compute the action with the best expected utility.*

14 Markov Decision Processes

a. V(s1) = .9 * .9 * 5.5 = 4.455

b. V(s2) = 5.5

c. V(s3) = 4.5

d. V(s4) = 0

e. V(s5) = 10

Problem 3 Reinforcement Learning

Deterministic world

Part A

You were to fill in a table as follows:

	Initial State: MILD		Action: West New State: HOT		Action: East New State: MILD		Action: East New State: COLD		Action: West New State: MILD	
	East	West	East	West	East	West	East	West	East	West
HOT	0	0	0	0	**5**	0	5	0	5	0
MILD	0	0	0	**10**	0	10	**0**	10	0	10
COLD	0	0	0	0	0	0	0	0	0	**-5**

Part B

You were to determin the number of **policies** there are. There are 8 possible policies, because there are 2 actions out of each of the 3 states.

Part C

You were to explain why is the policy $\pi(s) =$ West, for all states, *better than* the policy $\pi(s) =$ East, for all states.

The policy $\pi(s) =$ West, for all states, is *better than* the policy $\pi(s) =$ East, for all states, because the value of at least one state, in particular the state HOT, is higher for that policy.

Let $\pi_1(s) =$ West, for all states, $\gamma = 0.5$. Then:

- $V^{\pi_1}(\text{HOT}) = 10 + \gamma V^{\pi_1}(\text{HOT}) = 20$.

Let $\pi_2(s) =$ East, for all states, $\gamma = 0.5$. Then:

- $V^{\pi_2}(\text{COLD}) = -10 + \gamma V^{\pi_2}(\text{COLD}) = -20$,
- $V^{\pi_2}(\text{MILD}) = 0 + \gamma V^{\pi_2}(\text{COLD}) = -10$,
- $V^{\pi_2}(\text{HOT}) = 0 + \gamma V^{\pi_2}(\text{MILD}) = -5$.

Nondeterministic world

Part D

D.1

You were to compute the optimal values of each state, namely V^*(S1), V^*(S2), V^*(S3), V^*(S4) according to a given policy.

Remember that $\gamma = 0.9$.

```
V*(S2) = r(S2,D) + 0.9 (1.0 V*(S2))
V*(S2) = 100 + 0.9 V*(S2)
V*(S2) = 100 (1/1 - 0.9)
V*(S2) = 1000.

V*(S1) = r(S1,D) + 0.9 (1.0 V*(S2))
V*(S1) = 0 + 0.9 x 1000
V*(S1) = 900.

V*(S3) = r(S3,D) + 0.9 (0.9 V*(S2) + 0.1 V*(S3))
V*(S3) = 0 + 0.9 (0.9 x 1000 + 0.1 V*(S3))
V*(S3) = 0.9 (900 + 0.1 V*(S3))
V*(S3) = 810 + 0.09 V*(S3)
V*(S3) = 810 (1/1 - 0.09)
V*(S3) = 81000/91.

V*(S4) = r(S4,D) + 0.9 (0.9 V*(S2) + 0.1 V*(S4))
V*(S4) = 10 + 0.9 (0.9 x 1000 + 0.1 V*(S4))
V*(S4) = 82000/91.
```

D.2

You were to determine the Q-value, Q(S2,R).

```
Q(S2,R) = r(S2,R) + 0.9 (0.9 V*(S1) + 0.1 V*(S2))
Q(S2,R) = 100 + 0.9 (0.9 x 900 + 0.1 x 1000)
Q(S2,R) = 100 + 0.9 (810 + 100)
Q(S2,R) = 100 + 0.9 x 910
Q(S2,R) = 919.
```

Part E

E.1

You were to determine the problem with a Q-learning agent that always takes the action whose estimated Q-value is currently the highest.

The agent will probably not learn the optimal policy, as it commits to the first policy that it finds.

E.2

You were to determine the problem with a Q-learning agent that ignores its current estimates of Q in order to explore everywhere.

The agent will take a very long time to converge to the optimal policy and it will not necessarily improve its performance while actively learning.

Part F

You were to consider the Q-learning training rule for a nondetermistic Markov Decision Process:

$$\hat{Q}_n(s,a) \leftarrow (1-\alpha_n)\hat{Q}_{n-1}(s,a) + \alpha_n[r + \gamma \max_{a'} \hat{Q}_{n-1}(s',a')],$$

where $\alpha_n = \frac{1}{1+visits_n(s,a)}$, and $visits_n(s,a)$ is the number of times that the transition s,a was visited at iteration n.

Then you were to answer with True (T) and False (F):

- α_n decreases as the number of times the learner visits the transition s,a increases.

  ANSWER: True.

- The weighted sum through α_n makes the Q-values oscillate as a function of the nondeterministic transition and therefore not converge.

  ANSWER: False.

- If the world is deterministic, the Q-learning training rule given above converges to the same as the specific one for the deterministic worlds.

  ANSWER: True.

4. (10 points)

Consider a house-cleaning robot. It can be either in the living room or at its charging station. The living room can be clean or dirty. So there are four states: **LD** (in the living room, dirty), **LC** (in the living room, clean), **CD** (at the charger, dirty), and **CC** (at the charger, clean). The robot can either choose to **suck** up dirt or **return** to its charger. Reward for being in the charging station when the living room is clean is 0; reward for being in the charging station when the living room is dirty is -10; reward for other states is -1. Assume also that after the robot has gotten a -10 penalty for entering the charging station when the living room is still dirty, it will get rewards of 0 thereafter, no matter what it does.

Assume that if the robot decides to suck up dirt while it is in the living room, then the probability of going from a dirty to a clean floor is 0.5. The return action always takes the robot to the charging station, leaving the dirtiness of the room unchanged. The discount factor is 0.8.

(a) (1 pt) What is $V^*(CC)$ (the value of being in the CC state)?

0

(b) (1 pt) What is $V^*(CD)$?

-10

(c) (2 pts) Write the Bellman equation for $V^*(LC)$.

$V^*(LC) = -1 + 0.8 * max_a \sum_s T(LC, a, s)V(s)$

(d) (2 pts) What is the value of $V^*(LC)$?

-1

(e) (2 pts) Write the Bellman equation for $V^*(LD)$ and simplify it as much as possible.

$V^*(LD) = -1 + 0.8 * max_a \sum_s T(LD, a, s)V(s)$
$V^*(LD) = -1 + 0.8 * max_a\{action_{goBack}, action_{suckUp}\}$
$V^*(LD) = -1 + 0.8 * max_a\{-10, 0.5 * V(LC) + 0.5 * V(LD)\}$
$V^*(LD) = -1 + 0.8 * max_a\{-10, 0.5 * -1 + 0.5 * V(LD)\}$
$V^*(LD) = -1 + 0.8 * max_a\{-10, -0.5 + 0.5 * V(LD)\}$

(f) (2 pts) If $V_0(LD) = 0$ (that is, the initial value assigned to this state is 0), what is $V_1(LD)$, the value of LD with one step to go (computed via one iteration of value iteration)?

$V_0(LD) = 0$
$V_1(LD) = -1 + 0.8 * max_a\{-10, -0.5 + 0.5 * V_0(LD)\}$
$V_1(LD) = -1 + 0.8 * max_a\{-10, -0.5\}$
$V_1(LD) = -1 - 0.8 * 0.5$
$V_1(LD) = -1 - 0.4$
$V_1(LD) = -1.4$

4. (30 points) A robot has to deliver identical packages to locations A, B, and C, in an office environment. Assume it starts off holding all three packages. The environment is represented as a grid of squares, some of which are free (so the robot can move into them) and some of which are occupied (by walls, doors, etc.). The robot can move into neighboring squares, and can pick up and drop packages if they are in the same square as the robot.

   (a) (4 points) Formulate this problem as a search problem, specifying the state space, action space, goal test, and cost function.

   Solution:

   The state space needs to include enough information so that, by looking at the current values of the state features, the robot knows what it needs to do. For this task, the robot needs to be able to determine its position in the grid and which packages it has already delivered.

   - State: { (x,y), deliveredA, deliveredB, deliveredC }
   - Action space: MoveN, MoveE, MoveS, MoveW, DropA, DropB, DropC.
   - Goal test: { (x,y), delA, delB, delC } = { (any x, any y), 1, 1, 1 }
   - Cost function: cost of 1 for each action taken.

Common Mistakes:

- Including only the number of packages in the state space: this is insufficient because, if we know we are holding 2 packages, we don't know if, for example, we've already been to A, or if we should try to go there next.
- Specifying a goal test that couldn't be derived from the state space that was described.

(b) (3 points) Give a non-trivial, polynomial-time, admissible heuristic for this domain, or argue that there isn't one.

Solution:

There were many acceptable solutions. Some examples:

- Straight line or manhattan distance to furthest unvisited drop-off location.
- Straight line or manhattan distance to nearest unvisited drop-off location. (slightly looser underestimate than the previous one)
- Number of packages left to deliver. (Not as "good" as the above two, but still acceptable.)

Common Mistakes:

- Trying to solve the Travelling Salesman Problem: In the general case, not poly in the number of drop-off locations.
- Summing the distances between current robot location and all unvisited drop-off locations: Can overestimate the distance.
- Summing up the perimeter of the convex hull of robot location and drop-off locations: Can overestimate the distance.

(c) (3 points) Package A has to be delivered to the boss, and it's important that it be done quickly. But we should deliver the other packages promptly as well. How could we encode this in the problem?

Solution:

The idea of putting "pressure" on the delivering to the boss is correctly handled in the cost function. We want to emphasize delivery to A, but we don't want to ignore B if we happen to pass by it. Please note that to encode something into the "problem," it must be encoded into one of the problem components: the state space, action space, goal test, or cost function. For example:

while we haven't delivered to A, actions cost 2; after that, actions cost 1.

Common Mistakes:

- Decreasing the cost instead of increasing the cost.
- Constraining the order of the problem; requiring delivery to A before considering the other two locations at all.
- Modifying the search heuristic instead of the cost function.

(d) (3 points) Now, consider the case where the robot doesn't start with the packages, but it has to pick up a package from location 1 to deliver to location A, a package from location 2 to deliver to B, and from 3 to deliver to C. What is an appropriate state space for this problem?

Solution:

Again, multiple solutions were possible. One example:

- State: { (x,y), deliveredA, deliveredB deliveredC, holdingA, holdingB, holdingC }

Common Mistakes:

- Again, just the number of packages and the number of visited locations are insufficient.

(e) (3 points) What is a good admissible heuristic?

Solution:

- The same heuristic as before. (Not so "good," but acceptable.)
- The maximum (distance from my location to unvisited pickup location s_i plus the distance from s_i to unvisited delivery location).

(f) (4 points) This problem can also be treated as a planning problem.

Describe the effects of the action of moving north using situation calculus axioms. Assume that we are representing the location of the robot using integer coordinates for its x, y location in the grid, and that we can name the next integer coordinate greater than x as $x + 1$. Additionally, assume that we have a predicate *occupied* that applies to x, y pairs, indicating whether the robot may pass through the indicated locations.

Solution:

$$\forall x, y, s. atrobot(x, y, s) \wedge \neg occupied(x, y + 1) \rightarrow atrobot(x, y + 1, result(moveNorth, s))$$

A little variability in this answer was acceptable, but the idea of *result* as a function from actions and situations to actions was crucial.

~~(g) (3 points) Describe what frame axioms would be necessary for this domain (but don't write them down).~~

~~**Solution:**~~

- ~~When the robot moves, the positions of the (non-held) packages and the delivery locations don't change; which packages are held doesn't change.~~
- ~~When the robot picks up or drops off a package, the robot's location doesn't change; the locations of packages and delivery locations don't change.~~

Some people mentioned that which squares are occupied doesn't change. Because these facts never change (unless we have moving obstacles), it would probably be best to model them without a situation argument, thereby obviating the need for frame axioms for *occupied*.

(h) (3 points) Either provide a description of the move operation in the STRIPS language, or say why it is difficult to do so.

Solution:

There were a couple of different acceptable answers here. The most common one was some variation on :

move(x,y):

Pre: *at(x)*, ¬ *occupied(y)*, *neighbor(x,y)*
Effect: ¬ *at(x)*, *at(y)*

This requires some mention of the fact that it might be a big pain to specify the *neighbor* relation.

Another answer was

moveNorth(x,y):

Pre: *at(x,y)*, ¬ *occupied(x,y+1)*
Effect: ¬ *at(x,y)*, *at(x,y+1)*

This requires some mention, at least, of the fact that you'd need 4 operators. I gave full credit for this, but, in general, it's not legal to use functions in STRIPS, so you can't write "y+1" in the effects.

One more answer, saying that it was too hard, because you'd need to have a specific operator for each possible location (due to the fact that you can't have functions and/or that the "neighbor" relation would be hard to specify) was also okay.

(i) (2 points) When does it make sense to treat a problem as a planning problem rather than as a basic state-space search problem?

Solution:

There are a lot of decent answers here. I was looking for at least a couple of the following points.

- The initial state is not known exactly (so you have to search over the space of sets of states).
- There is a direction connection between actions and effects that can be heuristically exploited.
- The goal is described conjunctively, so solving the subgoals individually might lead to a plan more efficiently.
- The domain is described using a factored, logical representation (this really contributes to all of the above points).

- There is not a big concern with minimizing cost (because planning methods usually seek any path to the goal).

Saying "when the state space is too big" was not sufficient.

(j) (2 points) Should this problem be treated as a planning problem? Explain why or why not.

Solution:

I accepted a lot of answers here, mostly dependent on how cogently they were argued in the context of the answer to the previous problem.

I actually think that treating this as a planning problem won't be much help because: the initial state is known exactly, the subgoals aren't particularly separable (you can't solve the problem of going to location A without knowing where you're going to be when you start), and we have a strong desire for minimizing path cost.