CS221 Fall 2015 Homework [blackjack]

Problem 1: Value Iteration

1a.

For iteration 0:

V(-2) = 0

V(-1) = 0

V(0) = 0

V(1) = 0

V(2) = 0

For iteration 1:

V(-2) = 0

V(-1) = $max\begin{cases} for_{action} - 1: \ 80\% * (20+0) \ + \ 20\% * (-5+0) = 15.0 \\ for_{action} + 1: \ 70\% * (20+0) \ + \ 30\% * (-5+0) = 12.5 \end{cases} = 15$

V(0) = $max\begin{cases} for_{action} - 1: \ 80\% * (-5+0) \ + \ 20\% * (-5+0) = -5 \\ for_{action} + 1: \ 70\% * (-5+0) \ + \ 30\% * (-5+0) = -5 \end{cases} = -5$

V(1) = $max\begin{cases} for_{action} - 1: \ 80\% * (-5+0) \ + \ 20\% * (100+0) = 16.0 \\ for_{action} + 1: \ 70\% * (-5+0) \ + \ 30\% * (100+0) = 26.5 \end{cases} = 26.5$

V(2) = 0

For iteration 2:

V(-2) = 0

V(-1) = $max\begin{cases} for_{action} - 1: \ 80\% * (20+0) \ + \ 20\% * (-5-5) = 14 \\ for_{action} + 1: \ 70\% * (20+0) \ + \ 30\% * (-5-5) = 11 \end{cases} = 14$

V(0) = $max\begin{cases} for_{action} - 1: \ 80\% * (-5+15) \ + \ 20\% * (-5+26.5) = 12.30 \\ for_{action} + 1: \ 70\% * (-5+15) \ + \ 30\% * (-5+26.5) = 13.45 \end{cases} = 13.45$

V(1) = $max\begin{cases} for_{action} - 1: \ 80\% * (-5-5) \ + \ 20\% * (100+0) = 12 \\ for_{action} + 1: \ 70\% * (-5-5) \ + \ 30\% * (100+0) = 23 \end{cases} = 23$

V(2) = 0

1b.

From the above calculations the optimal policies for all non terminal states are:

$\pi_{opt}(-1) = -1$

$\pi_{opt}(0) = +1$

$\pi_{opt}(1) = +1$

Problem 2: Transforming MDPs

2a.
(see CounterexampleMDP in submission.py)

2b.
For an acyclic MDP an algorithm more efficient than value iteration that would only require one pass over all the *(s, a, s')* triples is as follows.

This algorithm is based on the idea of dynamic programming.

In an acyclic MDP we can always find a set of nodes representing states that have no outgoing transitions, these are the terminal states. Let these terminal states belong to Topological Level 1. After removing these states and their incident edges, we will be left with a new set of states that now have no outgoing transitions. Let these states belong to Topological Level 2. We continue this process to divide all available states of the MDP into a total of Ψ topological levels.

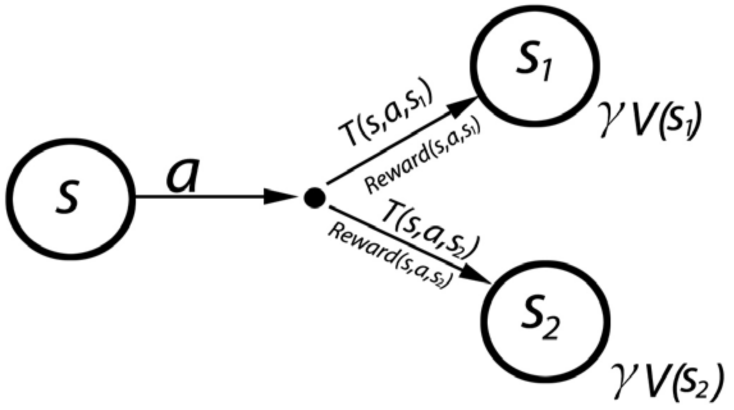
The values of the states in Topological Level 1 don't change. The values of the states in Topological Level 2 can be computed by using $Reward(s,a,s')$ and $T(s,a,s')$ where *s* belongs to Topological Level 2, s' belongs to Topological Level 1 and *a* is an action that creates a transition from *s* to *s'*. Similarly the values of the states in Topological Level 3 can be computed by using $Reward(s,a,s')$ and $T(s,a,s')$ where *s* belongs to Topological Level 3, s' belongs to Topological Level 2 and *a* is an action that creates a transition from *s* to *s'*. This process continues where the values of the states in Topological Level *n* can be computed by using $Reward(s,a,s')$ and $T(s,a,s')$ where *s* belongs to Topological Level *n*, s' belongs to Topological Level *n-1* and *a* is an action that creates a transition from *s* to *s'*. After Ψ levels, the values of all states can therefore be computed by dynamic programming that examines all the *(s,a,s')* triples bottom-up exactly once.

(**2c** is on next page) -->

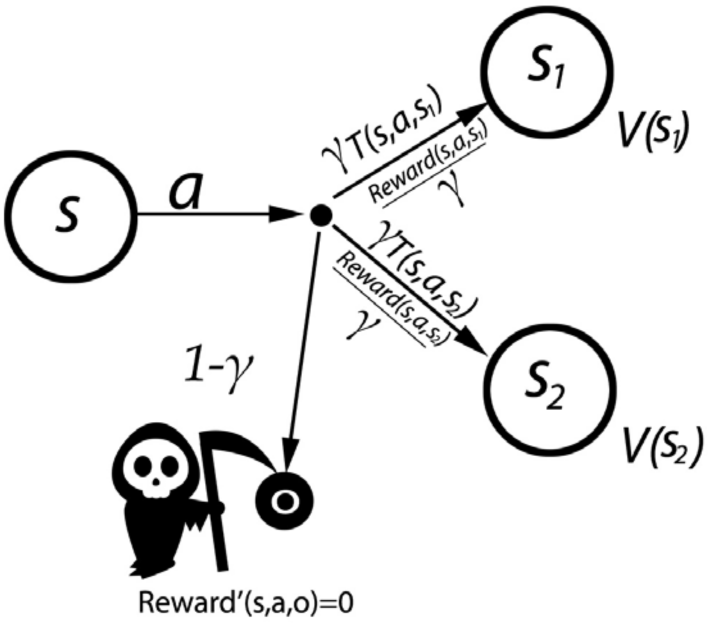
2c. Extra Credit

The transition probabilities $T'(s,a,s')$ in the new MDP will be defined as follows
$T'(s,a,s')=\gamma T(s,a,s')$ Here $T(s,a,s')$ are the transition probabilities from the original MDP. However, $T'(s,a,o)=1-\gamma$ and $T'(o,a,s')=0$

The $Reward'(s,a,s')$ in the new MDP will be defined as follows
$Reward'(s,a,s')=\frac{Reward(s,a,s')}{\gamma}$ Here $Reward(s,a,s')$ are the rewards from the original MDP. However, $Reward'(s,a,o)=0$
Here is an illustration of the original MDP:



Here is an illustration of the new MDP including additional $States'=States \cup \{o\}$



4b. Here's one instance of how policies compare for small and large MDPs (copied output from shell):

```
########################################
######## s m a l l * m d p ###########
########################################
Average rewards of Simulate: 5.39
ValueIteration: 5 iterations
Average rewards of Value Iteration: 4.82098765432
Differences between Simulate (and) Value Iteration:
**Value Iteration Policy:    Take:  (5, 1, (2, 1))
**Simulate Policy:           Quit:  (5, 1, (2, 1))
Small Mdp: number of differences: 1

########################################
######## L A R G E * M D P ###########
########################################
Average rewards of Simulation: 6.3919
ValueIteration: 15 iterations
Average rewards of Value Iteration: 35.5680675568
Differences between Simulate (and) Value Iteration (first 10 only):
**Value Iteration Policy:    Take:  (11, None, (2, 3, 3, 3, 2))
**Simulate Policy:           Quit:  (11, None, (2, 3, 3, 3, 2))
**Value Iteration Policy:    Quit:  (33, 4, (2, 0, 0, 2, 3))
**Simulate Policy:           Take:  (33, 4, (2, 0, 0, 2, 3))
**Value Iteration Policy:    Quit:  (38, 4, (2, 2, 1, 0, 3))
**Simulate Policy:           Take:  (38, 4, (2, 2, 1, 0, 3))
**Value Iteration Policy:    Quit:  (32, 4, (0, 0, 1, 3, 2))
**Simulate Policy:           Take:  (32, 4, (0, 0, 1, 3, 2))
**Value Iteration Policy:    Quit:  (38, 2, (2, 2, 1, 0, 3))
**Simulate Policy:           Take:  (38, 2, (2, 2, 1, 0, 3))
**Value Iteration Policy:    Quit:  (38, None, (0, 3, 2, 3, 0))
**Simulate Policy:           Take:  (38, None, (0, 3, 2, 3, 0))
**Value Iteration Policy:    Quit:  (31, 4, (3, 3, 0, 1, 3))
**Simulate Policy:           Take:  (31, 4, (3, 3, 0, 1, 3))
**Value Iteration Policy:    Quit:  (38, 1, (0, 3, 2, 3, 0))
**Simulate Policy:           Take:  (38, 1, (0, 3, 2, 3, 0))
**Value Iteration Policy:    Quit:  (37, 3, (0, 1, 1, 2, 2))
**Simulate Policy:           Take:  (37, 3, (0, 1, 1, 2, 2))
**Value Iteration Policy:    Peek:  (37, None, (1, 3, 2, 3, 0))
**Simulate Policy:           Take:  (37, None, (1, 3, 2, 3, 0))
Large Mdp: TOTAL NUMBER OF DIFFERENCES: 866
```

What went wrong? Policies from Q-learning w/ large MDP dramatically differ from optimal policies. This is likely due to larger state space and an inadequate, non-general rote-like ϕ. It's also interesting to note the stochastic variation each time examining simulation results due to explorationProb value.

4d.

Here is an example of a reward and the type of policies explored using util.simulate with a FixedRLAgorithm created using the policy learned from running value iteration of different MDP. Notice that the reward never goes over 10:

```
(totalReward = 10): [(0, None, (2, 2)), 'Take', 0, (5, None, (2, 1)), 'Take', 0, (10, None, (2, 0)), 'Quit',
10, (10, None, None)]
```

Here is an example policy of running simulate with the same MDP but using Q Learning (instead of the FixedRLAlgorithm as in the prior example above). Notice that the reward in this example does go above 10:

```
(totalReward = 12): [(0, None, (2, 2)), 'Take', 0, (5, None, (2, 1)), 'Take', 0, (10, None, (2, 0)), 'Take',
0, (11, None, (1, 0)), 'Take', 12, (12, None, None)]
```

The major thing to note are the differences in the results found between the above two example policies despite the fact that they are both simulated using a RL created from the same underlying MDP. Yet in the first example, the policy was actually learned on a MDP with a threshold of only 10, and that learned policy was then copied into the RL used when simulating the first example; this explains why the reward never went above 10 in the first example despite the actual MDP having a threshold of 15 in both examples above.

It's also worth mentioning that the term "simulation" is being used loosely when referring to the first MDP since the FixedRLAlgorithm implements it's own getAction() function which returns pre-computed policy dynamics from it's pre-computed policy rather than learning an entirely new policy, it simulates within the boundaries of the existing one – kind of like a constraint space.