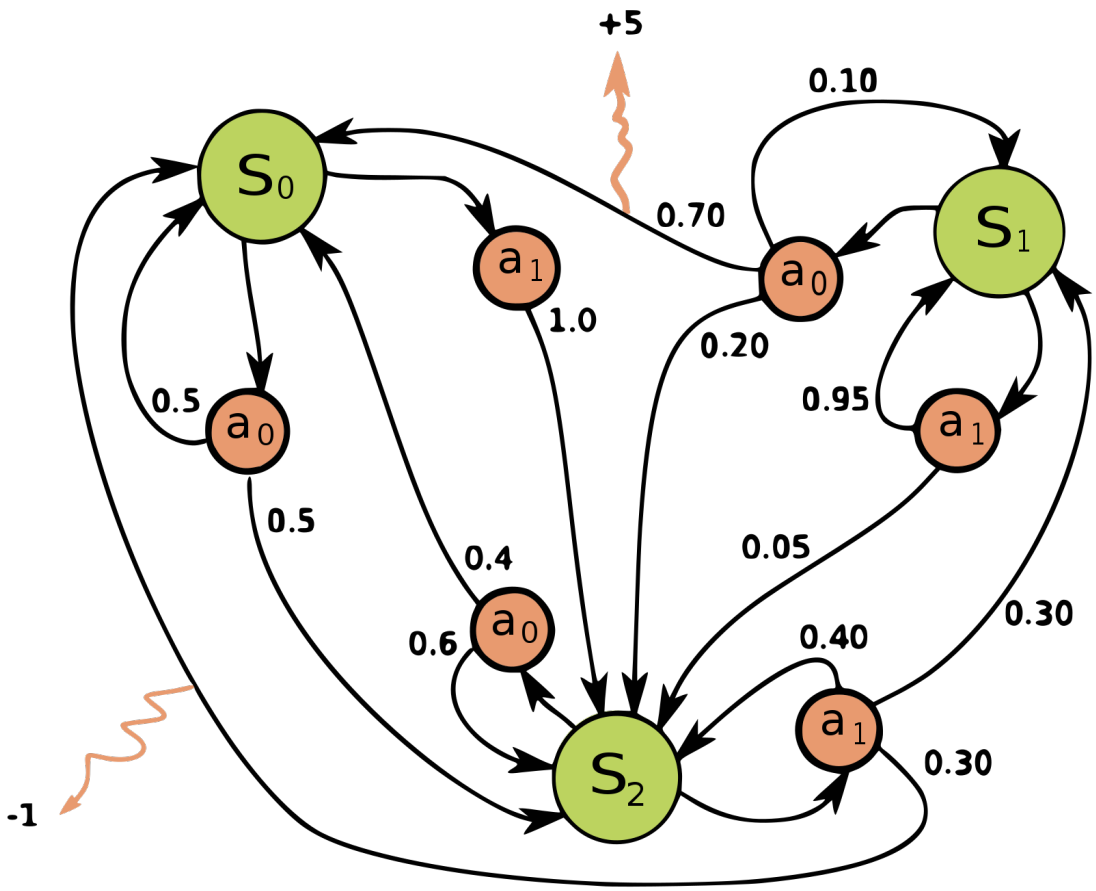

Markov Decision Processes

CS 520 Final Question 2

Spring 2019

Xuenan Wang

NET ID: xw336

MARKOV DECISION PROCESSES

- **Clarify the model**

To begin with, we can regular this question to normal MDP problem. A Markov Decision Process is a 4-tuple (S, A, P_a, R_a), where

S is a finite set of states, which in this case are:
$\{New, Used_1, Used_2, Used_3, Used_4, Used_5, Used_6, Used_7, Used_8, Dead\}$.

A is a finite set of actions, which is this case are:

$\{USE, REPLACE\}$.

$P_a(s, s') = P(s_{t+1} = s' \mid s_t = s, a_t = a)$ is the probability that action a in state s at time t will lead to state s' at time $t + 1$, which in this case are:

$$P(s_{t+1} \in S - \{New\} \mid s_t \in S - \{Dead\}, a_t = \{USE\}) = 0.1 \times t,$$
$$P(s_t \in S - \{Dead\} \mid s_t \in S - \{Dead\}, a_t = \{USE\}) = 1 - 0.1 \times t,$$
$$P(s_{t+1} = \{New\} \mid s_t \in S - \{New\}, a_t = \{REPLACE\}) = 1.$$

$R_a(s, s')$ is the reward received after transitioning from state s to state s', due to action a, which in this case are:

$$R_a(s, s') = \sum_{i=0}^{\infty} \beta^i \times (100 - 10 \times i) \; iff \; a = \{USE\},$$

$$R_a(s, s') = -250 \times \beta^i \; (i \; is \; the \; steps \; number \; when \; this \; been \; called) \; iff \; a = \{REPLACE\}.$$

β is the discounting factor, which is 0.9 here.

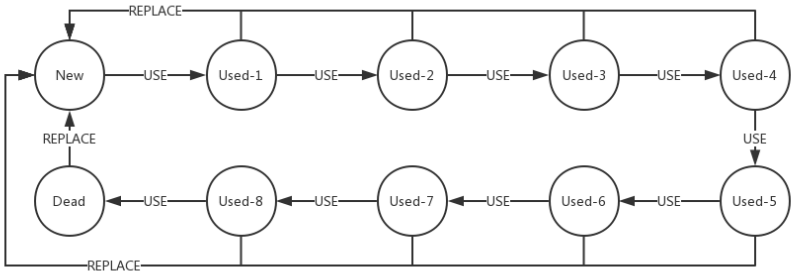

Figure 1. States changing flow chart.

a) For each of the 10 states, what is the optimal utility (long term expected discounted value) available in that state (i.e., U^*(state))?

We define reward_prediction as the reward the machine can get as long as it got replaced, which can be written as: $R_{prediction} = \sum_{i=0}^{\infty} \beta^i \times (100 - 10 \times i)\ when\ a = \{USE\} + (-250 \times \beta^i)\ when\ a = \{REPLACE\}$.

The result of this reward_prediction is shown as Figure 2. We can easily see that when $s = \{Used_5\}$, the reward is still positive, but after that the reward goes negative, which indicates that after this point, the total use reward of current machine is not ideal and current machine should be replaced.

```
[🍀Xuenan👉~/Documents/GitHub/CS520_Final/Question 2🧐: python mdp.py                                    ]
reward_use: [863.2262635856096, 607.685429742165, 605.4328801705567, 331.95053545, 193.447635, 7.924999999999983,
 -73.85000000000002, -24.769500000000022, -205.0]
reward_replace: [-250, -250, -250, -250, -250, -250, -250, -250, 0]
```

Figure 2. Result of predict function.

After a second thought, I realized that single result is not really persuasive because there are probability that the state will or will not change at each action. Therefore, I operate the predict function for 1000 times and averaged the result. This result is shown as Figure 3. Result indicates that our policy is correct because at the point after $s = \{Used_5\}$ the reward goes negative.

```
[🍀Xuenan👉~/Documents/GitHub/CS520_Final/Question 2🧐: python mdp.py                                    ]
average reward_use over 1000: [884.8457369627829, 695.295373831451, 510.2808030917206, 345.3362392792515, 197.
30082576210617, 68.21648130497043, -42.505343687614456, -135.84498589499972, -205.0, 0]
```

Figure 3. Average result of predict function over 1000 times.

b) What is the optimal policy that gives you this optimal utility - i.e., in each state, what is the best action to take in that state?

From Figure 3 we can see that after $s = \{Used_5\}$, the reward_prediction will go negative. Therefore, we should take action USE when $s \in \{New, Used_1, Used_2, Used_3, Used_4, Used_5\}$ and take action REPLACE when $s = \{Used_6\}$.

c) Instead of buying a new machine, a MachineSellingBot offers you the following option: you could buy a used machine, which had an equal chance of being in Used$_1$ and Used$_2$.

Intuitively:
– If the MachineSellingBot were offering you this option for free, you would never buy a new machine.
– If the MachineSellingBot were offering you this option at a cost of 250, you would never take this option over buying a new machine.

What is the highest price for which this used machine option would be the rational choice? i.e., what price should MachineSellingBot be selling this option at?

This question can be thought differently. Instead of trying to compare the rewards between buying used or replace a new one, we can try to calculate which state is the best state of buying a used one. When we trying to evaluate and find a optimal policy for REPLACE, we found that at $s = \{Used_5\}$ the reward is positive and at $s = \{Used_6\}$ the reward turns to negative. Similar here, we do a loop from 0 to 250, which indicate the price for a used machine, and try to find an optimal point that is the last one to be positive for each price. This point is the optimal point for each price to get a used machine instead of continue using the current one. Result is shown as Figure 4. After that, we only keep best point that is bigger than 5 and take its corresponding price. Also shown in Figure 4. As the result shows, the highest price is around 180-186. Of course, we iterate every steps for 100 times and take the average.

```
[🍀Xuenan👉~/Documents/GitHub/CS520_Final/Question 2🧐: python mdp.py                                    ]
best_pos: [8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7
, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7
, 7, 7, 7, 7, 7, 7, 6, 7, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6
, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6
, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 5, 6, 5, 5, 6, 6, 5, 6
, 5, 5, 5, 5, 5, 5, 5, 5, 6, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5
, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5]
accpted price: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26,
27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55
, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83,
84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109,
110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132,
 133, 134, 135, 136, 137, 138, 139, 140, 141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155
, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 17
8, 180, 183, 184, 186, 195]
```

Figure 4. Average result of optimal points and corresponding prices over 1000 times.

d) For different values of β (such that 0 < β < 1), the utility or value of being in certain states will change. However, the optimal policy may not. Compare the optimal policy for β = 0.1, 0.3, 0.5, 0.7, 0.9, 0.99, etc. Is there a policy that is optimal for all sufficiently large β? Does this policy make sense? Explain.

For β = 0.1, 0.3, 0.5, 0.7, 0.9, 0.99, the optimal points are $\{Dead, Dead, Dead, Dead, Used_6, Used_4\}$. Result is shown as Figure 5. There is not a policy that is optimal for all β. This makes sense because β will change the way it keep previous influence. The more it keeps, the easier the machine replace the old part.

```
[🍀Xuenan👉~/Documents/GitHub/CS520_Final/Question 2🧐: python mdp.py
 best_pos: [8, 8, 8, 8, 5, 3]
```

Figure 5. Average result of different β over 10000 times.(Results need to be $+1$)

Bonus: The cost of a new machine is 250. What is the (long term discounted) value of a new machine? Determine the break-even cost of a new machine - the price above which you are operating at a net loss, and below which you are operating at a net gain, i.e., when does a new machine become so expensive this game isn't even worth playing?

The top price it can be is 9606. Result shown as Figure 6.

```
[🍀Xuenan👉~/Documents/GitHub/CS520_Final/Question 2🧐: python mdp.py
price that no longer being meaningful: 9607
```

Figure 6. Price that no longer make sense.

APPENDIX: SOURCE CODE

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
# @Date    : 2019-05-13 04:40:23
# @Author  : Xuenan(Roderick) Wang
# @Email   : roderick_wang@outlook.com
# @Github  : https://github.com/hello-roderickwang

import numpy as np
import math

class Robot:
	def __init__(self):
		# type 0:new 1-8:used1-8 9:dead
		self.type = 0
		self.value = 0.0
		self.beta = 0.9
		self.step = 0

	def draw_lots(self, prob):
		n = np.random.randint(0, 100)
		if n/100 <= prob:
			return True
		else:
			return False

	def use(self):
		if self.type != 9:
			if self.type == 0:
				self.value += 100*math.pow(self.beta, self.step)
				self.type = 1
				self.step += 1
			elif self.type <= 7:
				if self.draw_lots(self.type*0.1) == True:
					self.value += (100-10*self.type)*math.pow(self.beta, self.step)
					self.type += 1
					self.step += 1
				else:
					self.value += (100-10*self.type)*math.pow(self.beta, self.step)
					self.step += 1
			else:
				if self.draw_lots(0.8) == True:
					self.value += 20*math.pow(self.beta, self.step)
					self.type = 9
					self.step += 1
				else:
					self.value += 20*math.pow(self.beta, self.step)
					self.step += 1

	def replace(self):
		if self.type != 0:
			self.type = 0
			self.value += -250*math.pow(self.beta, self.step)
			self.step += 1

	def predict(self, type, utility, price = 250, last_step = 0):
		self.type = type
		step = last_step
		reward = 0
		# print('self.type:', self.type)
		if self.type <= 8:
			if utility == 'use':
				reward += (100-10*self.type)*math.pow(self.beta, step)
				i = self.type+1
				step += 1
				while i <= 8:
					if self.draw_lots(0.1*i) == True:
						reward += (100-10*self.type)*math.pow(self.beta, step)
						i += 1
						step += 1
					else:
						reward += (100-10*self.type)*math.pow(self.beta, step)
						step += 1
					if i == 9:
						reward += -1*price*math.pow(self.beta, step)
			elif utility == 'replace':
				reward += -250
		return reward

	def buy_used(self):
		if self.draw_lots(0.5) == True:
			return 1
		else:
			return 2

	def used_predict(self, type, utility, price = 250):
		self.type = type
		step = 0
		reward = 0
		if self.type <= 8:
			if utility == 'use':
				reward += (100-10*self.type)*math.pow(self.beta, step)
				i = self.type+1
				step += 1
				while i <= 8:
					if self.draw_lots(0.1*i) == True:
						reward += (100-10*self.type)*math.pow(self.beta, step)
						i += 1
						step += 1
					else:
```

```
reward += (100-10*self.type)*math.pow(self.beta, step)
                                                step
+= 1
                                        if i == 9:
                                                reward_used
= self.predict(self.buy_used(), 'use', price = price, last_step =
step)-1*price*math.pow(self.beta, step)
                                                reward_new =
self.predict(0, 'use', last_step = step)-250*math.pow(self.beta,
step)
                                elif utility == 'replace':
                                        reward += -250
                        return reward_used-reward_new

if __name__ == '__main__':
        bot = Robot()
        # stete: New
        # bot.use()
        # state: Used 1
        # reward_use = []
        # for i in range(9):
        #       reward_use.append(bot.predict(i, 'use'))
        # reward_replace = []
        # for i in range(1, 10):
        #       reward_replace.append(bot.predict(i,
'replace'))
        # print('reward_use:', reward_use)
        # print('reward_replace:', reward_replace)
        loop = 1000
        # reward_use = [0,0,0,0,0,0,0,0,0,0]
        # for i in range(loop):
        #       for j in range(9):
        #               reward_use[j] += bot.predict(j,
'use')
        # for i in range(9):
        #       reward_use[i] = reward_use[i]/loop
        # print('average reward_use over 1000:',
reward_use)
        # bot.predict(0, 'use')-bot.used_predict(0, 'use', j)
        # reward_used = []
        # for i in range(loop):
        #       for j in range(0, 100):
        #               if i == 0:
        #
reward_used.append(bot.used_predict(8, 'use', j))
        #                       else:
        #                               reward_used[j] +=
bot.used_predict(8, 'use', j)
        # for i in range(len(reward_used)):
        #       reward_used[i] = reward_used[i]/loop
        #       if reward_used[i] >= 0:
        #               print('i:', i)
        # print('average reward_used over 1000:',
reward_used)
        # best_pos = []
        # for p in range(0, 250):
        #       reward_use = [0,0,0,0,0,0,0,0,0,0]
        #       pos = []
        #       for i in range(loop):
        #               for j in range(9):
        #                       reward_use[j] +=
bot.predict(j, 'use', price = p)
        #       for i in range(9):
        #               reward_use[i] = reward_use[i]/
loop
        #               if reward_use[i] >= 0:
        #                       pos.append(i)
        #       best_pos.append(pos[-1])
        # print('best_pos:', best_pos)
        # array = []
        # for n in range(len(best_pos)):
        #       if best_pos[n] > 5:
        #               array.append(n)
        # print('accpted price:', array)
        # print('average reward_use over 1000:',
reward_use)
        # bot.predict(0, 'use')-bot.used_predict(0, 'use', j)
        # best_pos = []
        # for b in [0.1, 0.3, 0.5, 0.7, 0.9, 0.99]:
        #       reward_use = [0,0,0,0,0,0,0,0,0,0]
        #       bot.beta = b
        #       pos = []
        #       for i in range(loop):
        #               for j in range(9):
        #                       reward_use[j] +=
bot.predict(j, 'use')
        #       for i in range(9):
        #               reward_use[i] = reward_use[i]/
loop
        #               if reward_use[i] >= 0:
        #                       pos.append(i)
        #       best_pos.append(pos[-1])
        #       print('reward_use:', reward_use)
        # print('best_pos:', best_pos)
        cost = 0
        price = 250
        while cost >= 0:
                reward = 0
                for i in range(loop):
                        reward += bot.predict(0, 'use',
price = price)
                cost = reward/loop
                price += 1
        print('price that no longer being meaningful:', price)
```