EECS 598: Reinforcement Learning, Homework 2

Abhishek Venkataraman

October 6, 2017

Problem 1. Stochastic Dyna-Q
In order to account for stochasticity, :

Converged policy for Dyna-Q:

$$Policy = \begin{bmatrix} 0., 0., 0., 1. \\ 0., 0., 0., 1. \\ 0., 0., 0., 1. \\ 0., 0., 1., 0. \\ 0., 1., 0., 0. \\ 1., 0., 0., 0. \\ 1., 0., 0., 0. \\ 1., 0., 0., 0. \\ 1., 0., 0., 0. \\ 0., 0., 1., 0. \\ 0., 0., 0., 1. \\ 0., 0., 0., 1. \\ 0., 1., 0., 0. \\ 0., 1., 0., 0. \\ 0., 1., 0., 0. \\ 1., 0., 0., 0. \\ 1., 0., 0., 0. \\ 1., 0., 0., 0. \\ 0., 1., 0., 0. \\ 0., 0., 1., 0. \\ 0., 1., 0., 0. \\ 0., 1., 0., 0. \\ 0., 1., 0., 0. \\ 1., 0., 0., 0. \end{bmatrix}$$

Parameters: $\#episodes = 400$, $\#planning = 50$, $\epsilon = 0.02$, $\alpha = 0.1$, $\gamma = 0.99$, $\kappa = None$

Reasons:

- I was not able to reach the goal state many times in just 30 episodes
- I changed the number of planning steps but that did not affect the results much

- Large ϵ leads to more random action. Given that the environment has a large number of holes, it is essential that the optimal action is taken more often. Hence ϵ small value lets the cumulative reward increase.
- I used the default learning rate. It did not affect the policy convergence a lot
- since the reward is small and a lot of time, the agent falls into hole, I increased γ so that the goal state helps the agent to be directed towards it

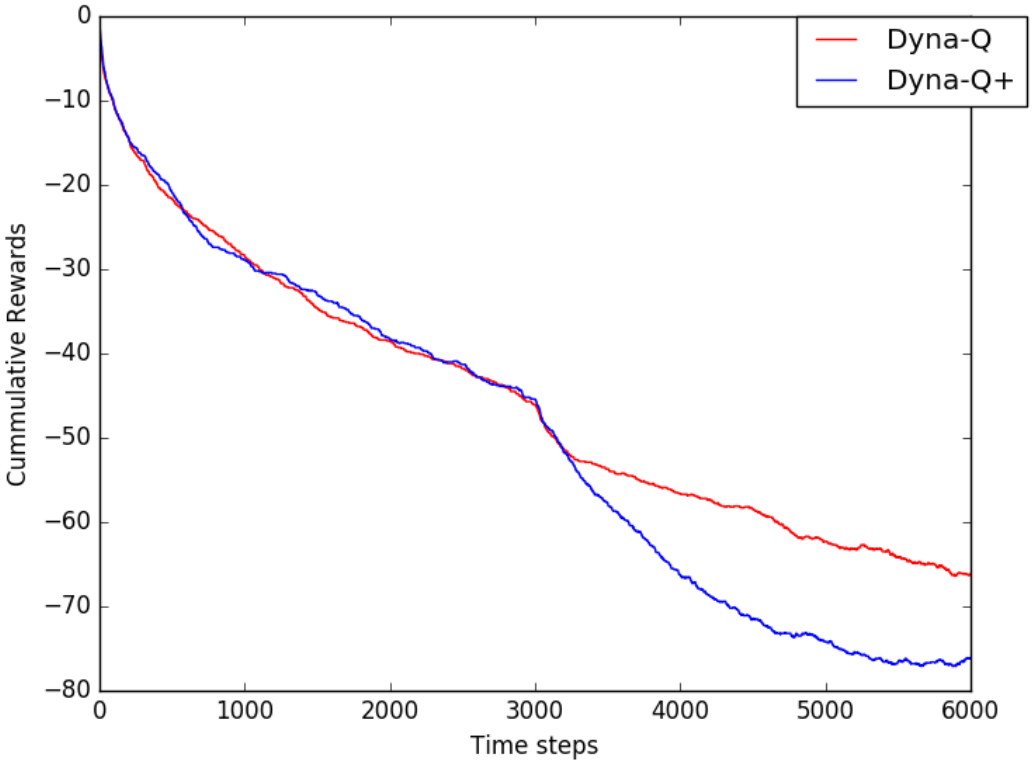
Converged policy for Dyna-Q+:

$$Policy = \begin{bmatrix} 0., 0., 0., 1. \\ 0., 0., 0., 1. \\ 0., 0., 0., 1. \\ 0., 0., 1., 0. \\ 1., 0., 0., 0. \\ 1., 0., 0., 0. \\ 1., 0., 0., 0. \\ 1., 0., 0., 0. \\ 1., 0., 0., 0. \\ 0., 0., 1., 0. \\ 0., 0., 0., 1. \\ 0., 0., 0., 1. \\ 0., 1., 0., 0. \\ 0., 1., 0., 0. \\ 0., 1., 0., 0. \\ 1., 0., 0., 0. \\ 1., 0., 0., 0. \\ 1., 0., 0., 0. \\ 0., 0., 0., 1. \\ 0., 0., 1., 0. \\ 0., 1., 0., 0. \\ 0., 1., 0., 0. \\ 0., 1., 0., 0. \\ 1., 0., 0., 0. \end{bmatrix}$$

Parameters: $\#episodes = 400$, $\#planning = 50$, $\epsilon = 0.02$, $\alpha = 0.1$, $\gamma = 0.99$, $\kappa = 0.0001$

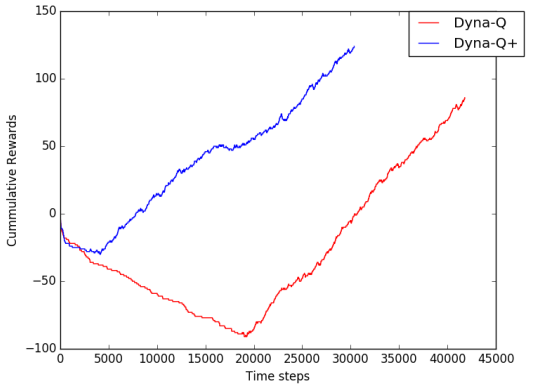
Reason: A large kappa was priority gives preference for visiting non goals states over goal state. This reduces the overall cumulative reward. The other parameters were the same as Dyna-Q method.

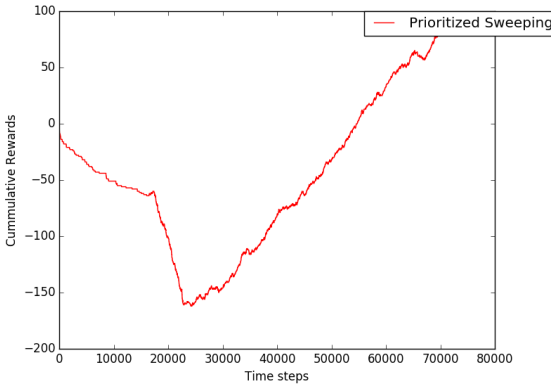
Problem 2. Dyna-Q vs. Dyna-Q+



Problem 3. Prioritized Sweeping

Prioritized Sweeping seems to be converging slower than Dyna-Q and Dyna-Q+. I suspect





that this is highly dependent on the parameters chosen. In case of Dyna-Q/Q+ the num of planning episodes was 50 while in case of this, the number of planning step was only 20. I suspect that this is a main reason for the difference.

Policy obtained by Prioritized Sweeping with the following parameters:
#episodes= 1000, #planning= 20, $\epsilon = 0.02$, $\alpha = 0.1$, $\gamma = 0.99$, $\theta = 0.5$

$$Policy = \begin{bmatrix} 0. & 0. & 0. & 1. \\ 0. & 0. & 0. & 1. \\ 0. & 0. & 0. & 1. \\ 0. & 0. & 1. & 0. \\ 1. & 0. & 0. & 0. \\ 1. & 0. & 0. & 0. \\ 1. & 0. & 0. & 0. \\ 1. & 0. & 0. & 0. \\ 1. & 0. & 0. & 0. \\ 0. & 0. & 1. & 0. \\ 0. & 0. & 0. & 1. \\ 0. & 0. & 0. & 1. \\ 0. & 0. & 1. & 0. \\ 0. & 1. & 0. & 0. \\ 0. & 1. & 0. & 0. \\ 1. & 0. & 0. & 0. \\ 1. & 0. & 0. & 0. \\ 1. & 0. & 0. & 0. \\ 0. & 1. & 0. & 0. \\ 0. & 0. & 1. & 0. \\ 0. & 0. & 1. & 0. \\ 0. & 1. & 0. & 0. \\ 0. & 1. & 0. & 0. \\ 1. & 0. & 0. & 0. \end{bmatrix}$$

Reasons:

- I needed a large number of episodes to reach goal more frequently
- Large number of planning with large number of episodes took a lot of computational time
- Large ϵ leads to more random action. Given that the environment has a large number of holes, it is essential that the optimal action is taken more often. Hence ϵ small value lets the cumulative reward increase.
- I did not change the learning rate
- since the reward is small and a lot of time, the agent falls into hole, I increased γ so that the goal state helps the agent to be directed towards it
- I increase θ so that lesser number of elements were added to the backup list. With smaller θ the convergence rate was slower

Problem 4. Reproducibility & Verification

(i) First 3 lines of requirement.txt :

```
matplotlib==1.5.1
numpy==1.13.3
gym==0.9.3
```

(ii) I created a simple map with deterministic actions

$$\begin{bmatrix} S & F \\ H & G \end{bmatrix}$$

I used numpy to generate random number and test the algorithm I ran for 2 episodes with 1 planning step in each episode.

The following is the sequence and hand calculations for it :

EPISODE 1 :

Actual action :

State : S(0)
Action : Right(2)
Next state : F(1)
Reward : 0

Q(0,2) = Q(0,2) + 0.1 * (0 + 0.95 * max Q(1) - Q(0,2))
Q(0,2) = Q(0,2) + 0.1 * (0 + 0 - 0)

Q values calculated :

$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Planning :

State : S(0)
Action : Right(2)
Next state : F(1)

Reward : 0

Q(0,2) = Q(0,2) + 0.1 * (0 + 0.9 * max Q(1) - Q(0,2))
Q(0,2) = Q(0,2) + 0.1 * (0 + 0.9 * 0.1 - 0)

Q values calculated :

$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Actual action :

State : F(1)
Action : Down(1)
Next state : G(3)
Reward : 1

Q(1,1) = Q(1,1) + 0.1 * (1 + 0.9 * max Q(1) - Q(1,1))
Q(1,1) = Q(1,1) + 0.1 * (1 + 0 - 0)

Q values calculated :

$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0.1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Planning :

State : S(0)
Action : Right(2)
Next state : F(1)
Reward : 0

Q(0,2) = Q(0,2) + 0.1 * (0 + 0.9 * max Q(1) - Q(0,2))
Q(0,2) = Q(0,2) + 0.1 * (0 + 0.9 * 0.1 - 0)

Q values calculated :

$$\begin{bmatrix} 0 & 0 & 0.009 & 0 \\ 0 & 0.1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

EPISODE 2 :

DIRECT RL :

State : S(0)
Action : Left(0)
Next state : S(0)
Reward : 0

Q(0,0) = Q(0,0) + 0.1 * (0 + 0.9 * max Q(0) - Q(0,0))
Q(0,0) = Q(0,0) + 0.1 * (0 + 0.9 * 0.009 - 0)

Q values calculated :

$$\begin{bmatrix} 0.00081 & 0 & 0.009 & 0 \\ 0 & 0.1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Planning :

State : F(1)
Action : Down(1)
Next state : G(3)
Reward : 1

Q(1,1) = Q(1,1) + 0.1 * (0 + 0.9 * max Q(3) - Q(1,1))
Q(1,1) = Q(1,1) + 0.1 * (1 + 0 - 0.1)

Q values calculated :

$$\begin{bmatrix} 0.00081 & 0 & 0.009 & 0 \\ 0 & 0.19 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Actual step :

State : S(0)
Action : Down(1)
Next state : H(2)
Reward : 0

Q(0,1) = Q(0,1) + 0.1 * (0 + 0.9 * max Q(2) - Q(0,1))
Q(0,1) = Q(0,1) + 0.1 * (0 + 0 - 0)

Q values calculated :

$$\begin{bmatrix} 0.00081 & 0 & 0.009 & 0 \\ 0 & 0.19 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Planning :

State : S(0)
Action : Down(1)
Next state : H(2)
Reward : 0

Q(0,1) = Q(0,1) + 0.1 * (0 + 0.9 * max Q(2) - Q(0,1))
Q(0,1) = Q(0,1) + 0.1 * (0 + 0 - 0)

Q values calculated :

$$\begin{bmatrix} 0.00081 & 0 & 0.009 & 0 \\ 0 & 0.19 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

This is consistent with what I got from running the code

(iii) Rendered README.md file `https://github.com/abhven/Reinforced_learning/tree/master/HW2`

Reinforcement Learning - HW2

This is the submission towards HW2 of EECS 598 Section 07. This file contains the instructions to reproduce the results obtained by me.

Setting up the working directory

Folder structure

Dowload and unzip the frozen_lakes.tar.gz file from and unzip it using the command:

```
tar -xzf frozen_lakes.tar.gz
```

Download and unzip the submission file abhven.zip in the same directory as frozen lakes folder. You have the following directory structure:

```
program directory
--hw2.py
--hw2_tests.py
--requirements.txt
--frozen_lakes
  --__init__.py
  --lakes_envs.py
```

Setting up dependencies

For this assignment, the libraries required are

```
numpy
matplotlib
gym
```

To install the dependencies run the command:

```
pip install -r requirements.txt
```

Exection of code

All the commands need to be executed in the program directory

For part 1, use the command

```
python hw2_tests.py 1
```

For part 2, use the command

```
python hw2_tests.py 2
```

For part 3, use the command

```
python hw2_tests.py 3
```

Author

Abhishek Venkataraman, abhven@umich.edu

Problem 5. Theory: Maximum-Likelihood estimate

Proof. Given the MDP of n states, we want to find the transition probability,

$$p(S' = s'|S = s, A = a)$$

Let $n(s, a- > s')$ be the transition counts to state s' from state s by action a. Writing in terms of likelihood,

$$L(p) = Pr(S' = s') \prod_{s\in S, a\in A} \prod_{s'\in S'} p(S' = s'|S = s, A = a)^{n(s,a->s')}$$

Taking log form,

$$\log L(p) = \log Pr(S' = s') + \sum_{s\in S, a\in A, s'\in S'} n(s, a- > s') \log p(s'|s, a)$$

Taking the derivative,

$$\frac{\partial \log L(p)}{\partial p(s'|s, a)} = \frac{n(s, a- > s')}{p(s'|s, a)}$$

This is equal to 0 at $p(s'|s, a)$. Now we can get

$$\frac{\hat{n}(s, a- > all)}{p(all|s, a)} = \frac{\hat{n}(s, a- > s')}{p(s'|s, a)}$$
$$\therefore \frac{p(s'|s, a)}{p(all|s, a)} = \frac{\hat{n}(s, a- > s')}{\hat{n}(s, a- > all)}$$

we know that $p(all|s, a) = 1$

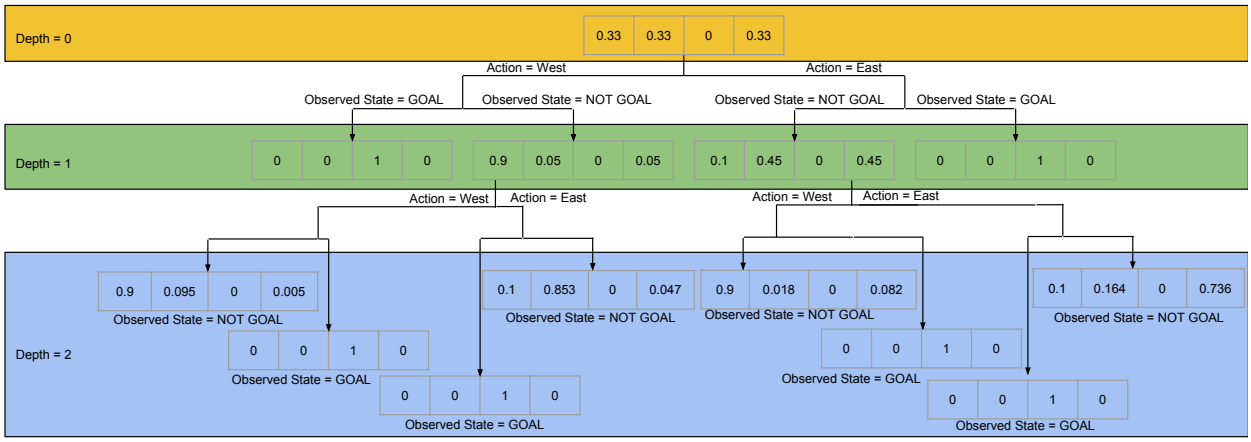
$$\therefore p(s'|s, a) = \frac{\hat{n}(s, a- > s')}{\hat{n}(s, a- > all)}$$

□

Problem 6. Theory: POMPDP belief state

The belief state can be given by:

$$b'(s') = \frac{O(s', a, o)\Sigma_{s\in S} T(s, a, s')b(s)}{Pr(o|a, b)} = \eta.O(s', a, o)\Sigma_{s\in S} T(s, a, s')b(s)$$



where η is the normalization factor. Since the observation is only whether in goal state or not,

$$O(s', a, o) = \begin{cases} 0, \text{ if } s' \in \{1, 2, 4\} \texttt{ and } o = GOAL \\ 1, \text{ if } s' \in \{1, 2, 4\} \texttt{ and } o = NOT\ GOAL \\ 1, \text{ if } s' \in \{3\} \texttt{ and } o = GOAL \\ 0, \text{ if } s' \in \{3\} \texttt{ and } o = NOT\ GOAL \end{cases}$$

Clearly if we observe goal state, then belief state would take the value $[0, 0, 1, 0]$. Hence the below calculations are shown only for non goal state observations.

Depth = 1
Action = West

$$\begin{aligned} b'(1) &= \eta.1.\{T(1, W, 1)b(1) + T(2, W, 1)b(2)\} = \eta\{0.9 \times 0.33 + 0.9 \times 0.33\} &= 0.594\eta \\ b'(2) &= \eta.1.\{T(1, W, 2)b(1)\} = \eta 0.1 \times 0.33 &= 0.033\eta \\ b'(3) & &= 0 \\ b'(4) &= \eta.1.\{T(4, W, 4)b(1)\} = \eta 0.1 \times 0.33 &= 0.033\eta \end{aligned}$$

Solving $\Sigma_{s \in \{1,2,3,4\}} b'(s') = 1$, belief state $= [0.9, 0.05, 0, 0.05]$

Action = East

$$\begin{aligned} b'(1) &= \eta.1.\{T(1, E, 1)b(1) + T(2, E, 1)b(2)\} = \eta\{0.1 \times 0.33 + 0.1 \times 0.33\} &= 0.067\eta \\ b'(2) &= \eta.1.\{T(1, E, 2)b(1)\} = \eta 0.9 \times 0.33 &= 0.297\eta \\ b'(3) & &= 0 \\ b'(4) &= \eta.1.\{T(4, E, 4)b(1)\} = \eta 0.9 \times 0.33 &= 0.297\eta \end{aligned}$$

Solving $\Sigma_{s\in\{1,2,3,4\}} b'(s') = 1$, belief state = $[0.1, 0.45, 0, 0.45]$

Depth = 2

Depth 1 Action = West, Depth 2 Action = West

$$b'(1) = \eta.1.\{T(1,W,1)b(1) + T(2,W,1)b(2)\} = \eta\{0.9 \times 0.9 + 0.9 \times 0.05\} = 0.855\eta$$
$$b'(2) = \eta.1.\{T(1,W,2)b(1)\} = \eta 0.1 \times 0.9 = 0.09\eta$$
$$b'(3) = 0$$
$$b'(4) = \eta.1.\{T(4,W,4)b(1)\} = \eta 0.1 \times 0.05 = 0.005\eta$$

Solving $\Sigma_{s\in\{1,2,3,4\}} b'(s') = 1$, belief state = $[0.9, 0.095, 0, 0.005]$

Depth 1 Action = West, Depth 2 Action = East

$$b'(1) = \eta.1.\{T(1,E,1)b(1) + T(2,E,1)b(2)\} = \eta\{0.1 \times 0.9 + 0.1 \times 0.05\} = 0.095\eta$$
$$b'(2) = \eta.1.\{T(1,E,2)b(1)\} = \eta 0.9 \times 0.9 = 0.81\eta$$
$$b'(3) = 0$$
$$b'(4) = \eta.1.\{T(4,E,4)b(1)\} = \eta 0.9 \times 0.05 = 0.045\eta$$

Solving $\Sigma_{s\in\{1,2,3,4\}} b'(s') = 1$, belief state = $[0.1, 0.853, 0, 0.047]$

Depth 1 Action = East, Depth 2 Action = West

$$b'(1) = \eta.1.\{T(1,W,1)b(1) + T(2,W,1)b(2)\} = \eta\{0.9 \times 0.1 + 0.9 \times 0.45\} = 0.495\eta$$
$$b'(2) = \eta.1.\{T(1,W,2)b(1)\} = \eta 0.1 \times 0.1 = 0.01\eta$$
$$b'(3) = 0$$
$$b'(4) = \eta.1.\{T(4,W,4)b(1)\} = \eta 0.1 \times 0.45 = 0.045\eta$$

Solving $\Sigma_{s\in\{1,2,3,4\}} b'(s') = 1$, belief state = $[0.9, 0.018, 0, 0.082]$

Depth 1 Action = East, Depth 2 Action = East

$$b'(1) = \eta.1.\{T(1,E,1)b(1) + T(2,E,1)b(2)\} = \eta\{0.1 \times 0.1 + 0.1 \times 0.45\} = 0.055\eta$$
$$b'(2) = \eta.1.\{T(1,E,2)b(1)\} = \eta 0.9 \times 0.1 = 0.09\eta$$
$$b'(3) = 0$$
$$b'(4) = \eta.1.\{T(4,E,4)b(1)\} = \eta 0.9 \times 0.45 = 0.405\eta$$

Solving $\Sigma_{s\in\{1,2,3,4\}} b'(s') = 1$, belief state = $[0.1, 0.164, 0, 0.736]$

Problem 7. Theory: Optimum policy with fixed horizon

When the horizon to be 1, clearly, the

$$\pi_1^*(s) = \underset{a}{\operatorname{argmax}}\, R(s,a)$$
$$V_1(s) \coloneqq R(s, \pi_1(s)) \quad \forall s \in S$$

Now for horizon of 2 and 3, using $V_1(s)$,

Initialize $V_2(s) = 0 \; \forall s \in S$
loop for all $t \in \{2, 3\}$
  loop for all $s \in S$
    loop for all $a \in A$
      $Q_t^a(s) \coloneqq R(s,a) + \Sigma_{s' \in S} T(s,a,s') V_{t-1}(s')$
    end loop
    $V_t(s) \coloneqq \max_a Q_t^a(s)$
  end loop
end loop

Hence the optimal policy with 3 steps look ahead can be given by :

$$\pi^*(s) = \underset{a}{\operatorname{argmax}} \left[R(s,a) + \Sigma_{s' \in S} T(s,a,s') V_3(s') \right]$$

Problem 8. Meta
I feel that 1 week was quite a small time for this assignment. I understand the course is being taught for the first time, however, it was quite frustrating that there were so many changes in the HW. Also the additional time constraint and uncertainty made the assignment challenging.

- Prob 1 & 2:It took a lot of time to try out the parameters. I would have spent around 10-12 hours for coding, debugging and fine tuning the parameters.
- Prob 3: The implementation was not too hard. I spent a total of 3 hours on this problem
- Prob 4: I spent around 4 hours on this problem. 1 hour was spend on understanding the use fo requirements.txt and tesing it on a another linux machine. The README.md took around 0.5 hours since I have never made one before. 2.5 hrs was spent towards manually testing the algorithm.
- Prob 5: I spent around 3 hours on this problem. Most of the time was spent in trying to understand the question.
- Prob 6: I spent around 3 hours on this problem
- Prob 7: I spent around 3 hours on this problem