CS747 Foundation of Learning Agents
Assignment 2

Sanchit Jindal   200020120

1 MDP Planning

To Run:

```
python planner.py --mdp MDPFILE [--algorithm vi/hpi/lp ]/[ --policy POLICYFILE]
```

The Code is available in the file **planner.py** and I have chosen the Value Iteration algorithm to be the default algorithm I have tried to vectorize the code wherever possible to reduce the time required to run the scripts

1.1 Value Iteration

This Algorithm is

```
Initialize Value Vector $V \in \mathbb{R}^n$
do
    $V_{new} \leftarrow \max_{action} \sum_{state} T\{R + \gamma V\}$
    $error \leftarrow ||V - V_{new}||$
    $V \leftarrow V_{new}$
while (error$\leq \Delta$) # A limit parameter
$\pi \leftarrow arg\,max_{action} \sum_{state} T\{R + \gamma V\}$
return $V, \pi$
```

That is we keep iterating the over the Value function by using the Bellman Optimality Operator till the values converge

1.2 Howards Policy Improvements

The Algorithm is

```
Initialize policy $\pi \in A^n$ ($A$ is the set of actions)
do
    Initialize Value Vector $V \in \mathbb{R}^n$
    do
        # Choosing the Values Corresponding to the policies
        $V_{new} \leftarrow \sum_{state} T\{R + \gamma V\}[\pi]$
        $error \leftarrow ||V - V_{new}||$
        $V \leftarrow V_{new}$
    while (error$\leq \Delta$) # A limit parameter

    $\pi \leftarrow arg\,max_{action} \sum_{state} T\{R + \gamma V\}$
while (policy Changed)
return $V, \pi$
```

That is we choose a arbitary policy and calculate the Value Function for each state. Then We check if there exists a policy that can give a better value function by applying the Bellman Operator. If True then we repeat with the new better policy till there is no change. As we are choosing argmax over all the actions if a state is improvable it is being improved.

1.3 Linear Program

We are solving the Problem

```
$MAX_V \quad \sum V$
s.t
    $V(s) \geq \sum T\{R + \gamma V\} \quad \forall s, \forall a$
```

The Problem is solved using the `PulP` library of python As we need to create a PulP Equation The Code could not be vectorized

1.4 Policy Evaluation

The Policy Evaluation is same as the Value Computation code in the Howards Policy Iteration.

```
Given policy $\pi$
    Initialize Value Vector $V \in \mathbb{R}^n$
    do
        # Choosing the Values Corresponding to the policies
        $V_{new} \leftarrow \sum_{state} T\{R + \gamma V\}[\pi]$
        $error \leftarrow ||V - V_{new}||$
        $V \leftarrow V_{new}$
    while (error$\leq \Delta$) # A limit parameter
    return $V$
```

1.5 Utilitities

To Better Organize the Code:-

- I have constructed a MDP class that initializes the Transition and Reward matrix with the given values
- Both the Matrix are of dimension equivalent to state $\times$ action $\times$ state
- The End States are being stored but they are not required as the transition probability value will be zero for them
- Similarly the algorithms are applicable for both Continous and Episodic MDPs and is thus not required

2 MDP for cricket game

The Commands to run the code are:-

```
python encoder.py --states STATEFILE --parameter PARAMETERFILE --q NUM > MDPFILE
python planner.py --mdp MDPFILE > VALUEPOLICYFILE
python decoder.py --value-policy VALUEPOLICYFILE --states STATEFILE
```

2.1 Encoder

I have used the following encoding to produce an MDP

- Using the states file I create 2 sets of states by adding a **O** or a **1** to the start of the state to denote the 2 batsman.
- There are 2 other Terminal states one corresponding to winning the match and the other corresponding to losing.
- I map all the different states to a number from 0 to NumStates
- I also map the possible actions $0, 1, 2, 4, 6$ to the numbers $0, 1, 2, 3, 4$ as the Planner expects then to be in this fashion.
- I make transitions from each states for all possible actions, reading the parameters for the better batsman and using q for the tail ender.
- For each state for each action the destination is state with one ball less and runs required decreased by runs scored.
- To accomodate the possibility that the player may go from one state to another by two possible paths for eg if 2 runs are required and player tries a 6 then on both 4 and 6 runs he reaches the winning terminal state. I had to change the planner.py file to add the transition probability instead of overwriting them
- If runs reach less than or equal to zero then the destination is the winning state and reward is one.
- If the balls reach zero before or the player gets out then the final state is the losing Terminal state.
- Similarly for the tail ender.
- The strike changes when player scores 1 or 3 runs.
- The strike also changed when the number of balls go from 7 to 6 or 13 to 12

The –policy and –algorithm flags for the planner.py are mutually exclusive, so if policy given algorithm is ignored.

2.2 Decoder

The Decoder Assumes that the first lines of the value-policy file correspond to the states in the given file And hence it reverse maps the actions and provide the results in the required format

2.3 Analysis

To collect and plot data I have used two python scripts

- **collectData.py** It has different blocks of code that can be uncommented to run the other files and generate the values.
- **plot.py** It is a python script plots the data on the basis of two files

2.3.1 Value Of q

The value of q is ranging from 0 to 1 and there are 100 data points

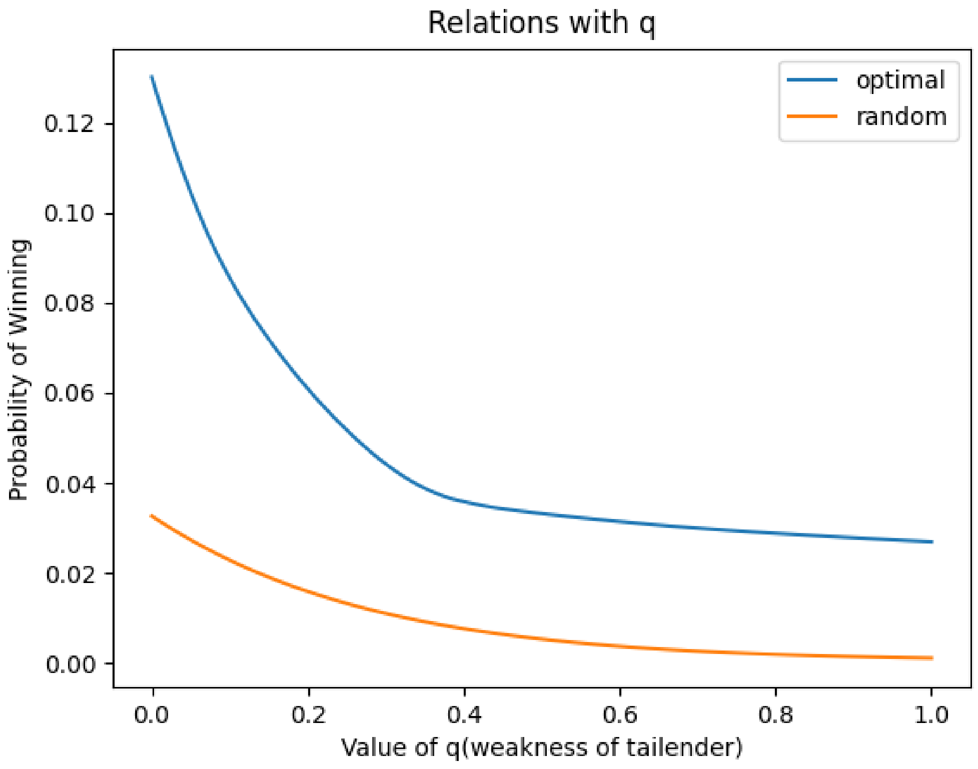

Figure 1: Dependence on q

The Main observations are

- The probability of winning the match is always higher if the optimal policy is used, this is to be expected as we optimized the policy to give the most probability.
- The Other observations is that the probability reduces as the value of **q** increases, This is also expected as q is the weakness coffecient and if it is higher than the other player is weaker and hence the probability of winning will be less.

2.3.2 Number of Runs Required

There are 10 balls left, q is set to 0.25, and the number of runs required is ranging from 20,19 . . . 1

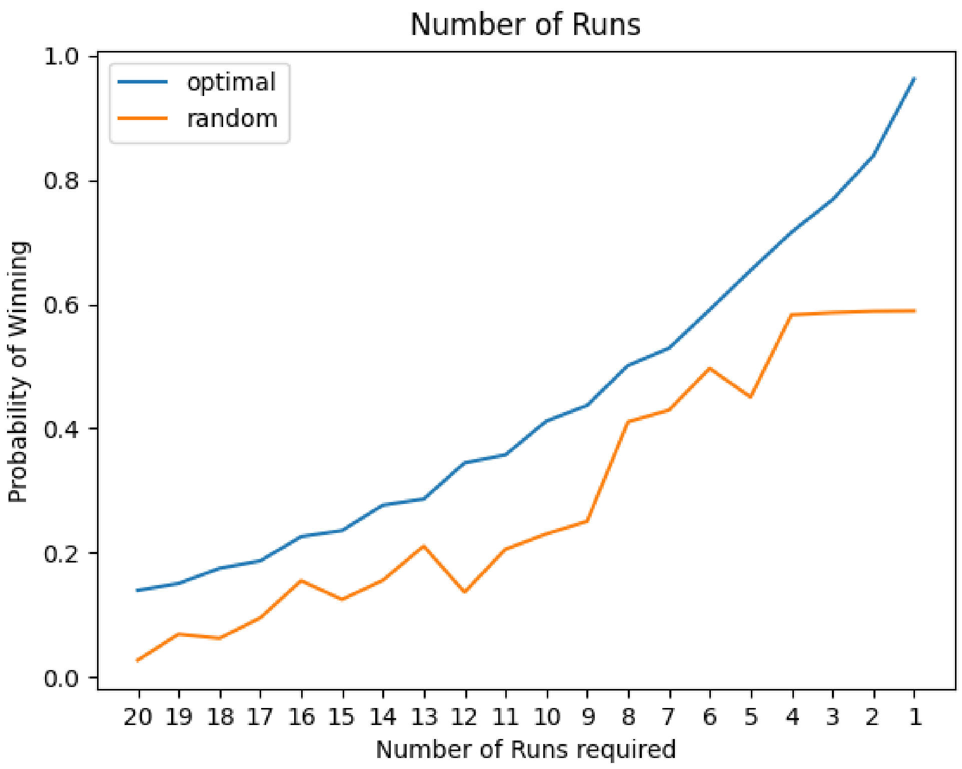

Figure 2: Target to Achive

The observations from the graph are:

- Again the probability of winning is higher if the player followes the optimal policies.
- The Probability of winning increases if the number of runs required Decreases, This is expected it will become easier to win if the number of runs required is less.
- We can see that the probabililty of winning with the random policy fluctuates much more than the optimal as the random policy though benefitting from the less number of runs required is not able to fully utilize that.

2.3.3 Number of Balls Left

10 runs are required, the value of q is again 0.25 and the number of balls left is changed from 15,14 . . . 1

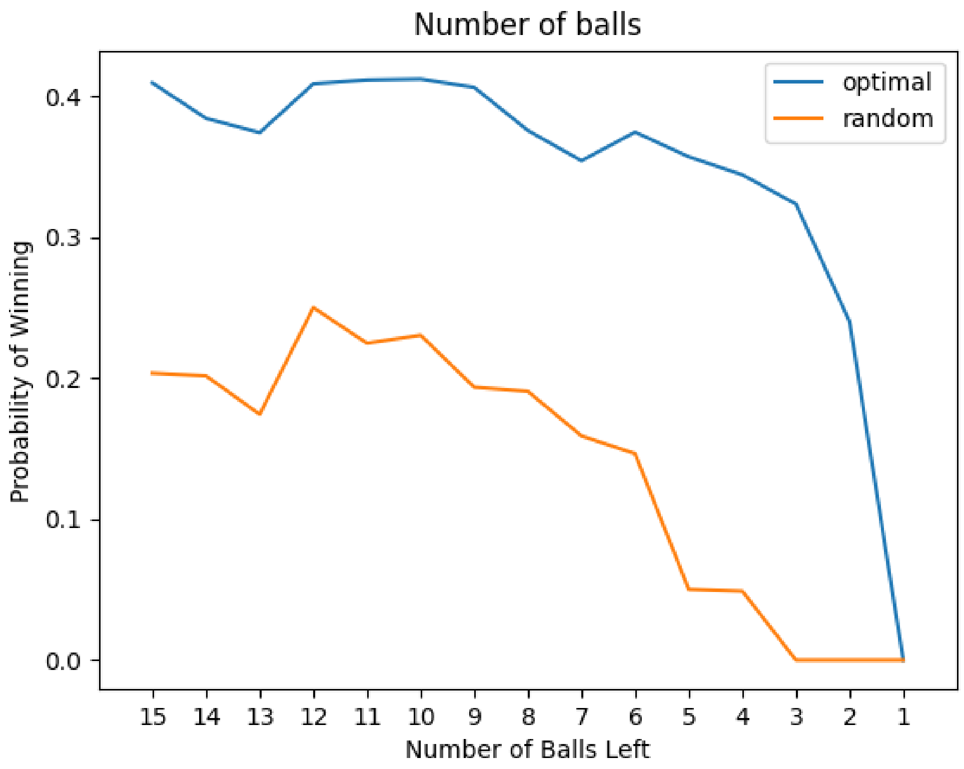

Figure 3: Balls Left

The observations are:-

- Again the Probability of winning is more for the optimal policy, than the random policy,
- The probability decreases as the number of balls left decreases, this is again expected as if there are less number of balls then it is more difficult to win
- We can see that the probability of winning is zero if only one ball is left as it is impossible to make 10 runs in one ball.
- There probabililty increases a little when the balls left are 12 and 6 as the chance of the strike going to the weaker player is reduced.

3 Submitted files

The Folder Structure is:-

```
submission/
    |-- collectData.py
    |-- decoder.py
    |-- encoder.py
    |-- planner.py
    |-- plot.py
    |-- references.txt
    |-- report.pdf
```