CS747: Programming Assignment 2

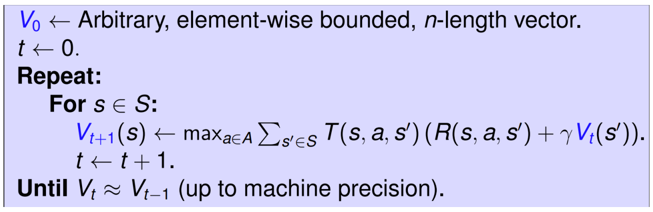
Jash Kabra, 200050054

October 2022

1 Task 1

1.1 Value Iteration

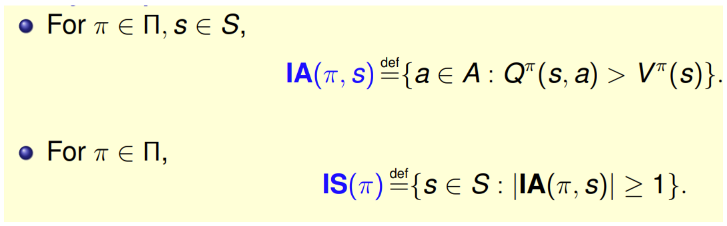
I implemented the algorithm as described in the slides.



```
$V_0 \leftarrow$ Arbitrary, element-wise bounded, $n$-length vector.
$t \leftarrow 0$.
Repeat:
    For $s \in S$:
        $V_{t+1}(s) \leftarrow \max_{a \in A} \sum_{s' \in S} T(s, a, s')\,(R(s, a, s') + \gamma V_t(s'))$.
        $t \leftarrow t + 1$.
Until $V_t \approx V_{t-1}$ (up to machine precision).
```

I initialised V_0 as an array of all zeros. The loop was repeated until $abs(V_t - V_{t-1}) <$ 1e-10 for all elements.
The optimal policy was decided by taking argmax in the formula instead of max at the end.

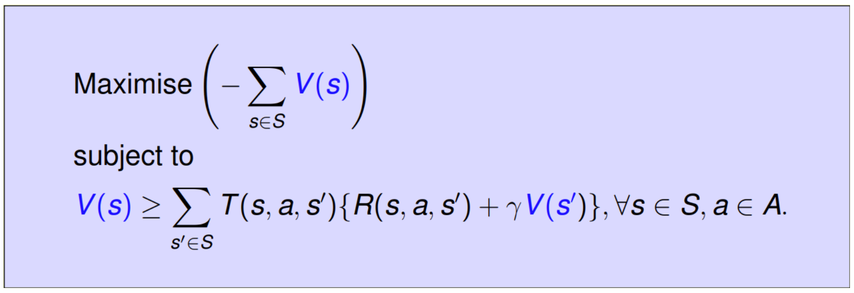
1.2 Howard Policy Iteration



- For $\pi \in \Pi, s \in S$,
$$\mathbf{IA}(\pi, s) \overset{\text{def}}{=} \{a \in A : Q^\pi(s, a) > V^\pi(s)\}.$$
- For $\pi \in \Pi$,
$$\mathbf{IS}(\pi) \overset{\text{def}}{=} \{s \in S : |\mathbf{IA}(\pi, s)| \geq 1\}.$$

For every state s, I chose the policy π for which $Q^\pi(s, a)$ was maximum. This ensured that if $IS(\pi) = \phi$ then we chose the optimal policy π again. Otherwise it switches all improvable states to some improving action as required by Howard's policy iteration algorithm.
I used the policy evaluation function to find our value function for a particular policy π and then improved the policy until possible.

1.3 Linear Programming



$$\text{Maximise} \left(-\sum_{s \in S} V(s) \right)$$

subject to

$$V(s) \geq \sum_{s' \in S} T(s, a, s')\{R(s, a, s') + \gamma V(s')\}, \forall s \in S, a \in A.$$

I used PuLP solver to solve this LP. I added n*k constraints and a maximising objective function. In this way, I found the value function for each state.
To find the policy at each state, I used the same method as Value Iteration.

1.4 Policy Evaluation

I evaluated a policy by solving the set of Bellman Equations. I used PuLP earlier to solve these equations but PuLP was very slow, so I converted the set of n equations to matrix form, and inverted the coefficient matrix and multiplied it with constant matrix to find our value function.

1.5 Some Observations

PuLP solver can be very slow due to time taken to add the constraints into the solver. For large MDPs, even Howard's Policy Iteration is slow due to the time taken to invert the matrix in policy evaluation step. Value iteration works the fastest for large MDPs.
I did not need to use the end states anywhere as their transition probability matrix was just empty (zero filled).

2 Task 2

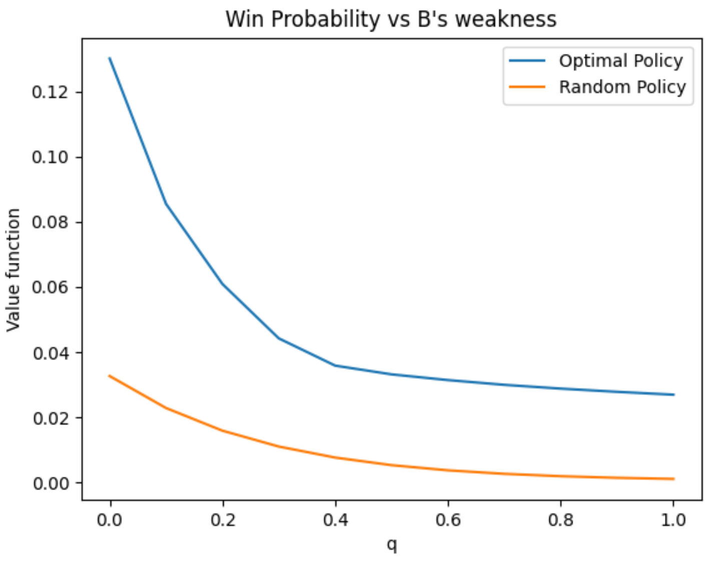
2.1 Design Decisions

I have designed the states as a 3 tuples = {balls,runs,strike} indicating the number of balls left, amount of runs to make, and whether player 1 or player 2 is on strike. Balls go from 0 to total_balls, runs go from 0 to total_runs, strike is either 0 (P1) or 1(P2).
There are 6 actions possible at each state, 5 actions for player 1 (0, 1, 2, 4, 6) and only 1 action for player 2 (as P2 is not in our control).

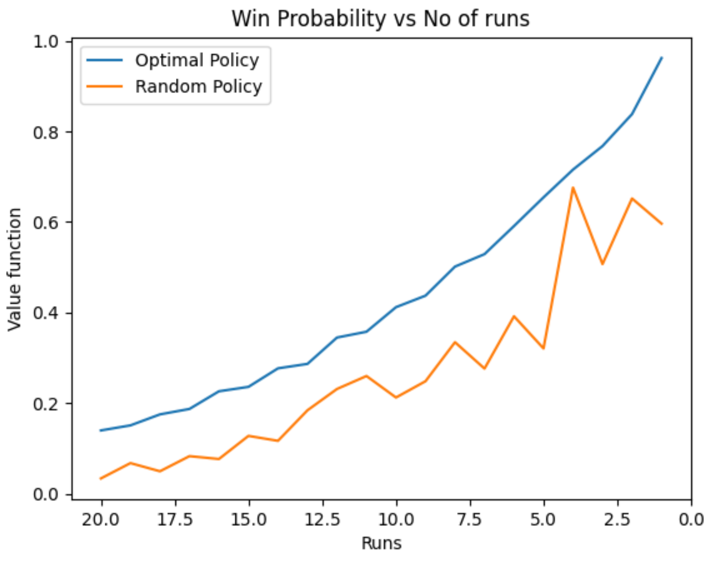
I designed a transition probability matrix and a rewards matrix for each state and each action. Transition probabilities for only 1st 5 actions may be non-zero if P1 is on strike and only last 1 action when P2 is on strike. The rewards are by default zero. There is only a reward 1 when there is a transition to a state where runs=0.
On each ball, we go from a state from {balls,runs,strike} to {balls-1,runs-runs_scored,new_strike}. Strike changes when runs scored are odd or when (balls-1)mod6=0. Strike does not change when both conditions are true.

When P1 is on strike, transition probabilities are taken from parameters file while for P2, they are dependent on q.
When a player gets out, we go to a state where balls=0,runs=1 (Meaning game is lost) and when runs_scored $\geq$ runs_left, we go to a state where runs=0, balls=1 (Meaning game is won).
Note that there are no transitions from states where runs=0 or balls=0 (They are end states).

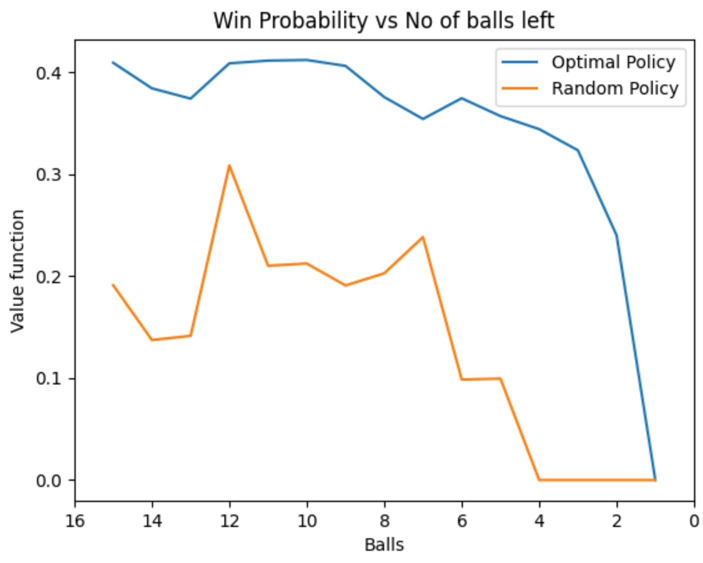
2.2 Analysis



Win probability decreases as q (B's weakness) increases. Our optimal policy performs better than the random policy. Interesting thing to note is even the random policy monotically decreases because starting runs and starting balls are constant and only q changes.



Our optimal policy again performs better than the random policy. Win probability increases as number of runs to score decreases and would've reached 1 for runs=0 (again true with our intuition).



We see that our optimal policy performs better in every situation that the random policy. Win probability decreases as number of balls decreases with runs= constant (again intuitive). We also observe that there is a dip in optimal policy at balls= 7,13 because just 1 ball is left for over to change and strike changes at every new over.