CS747 : Foundations of Intelligent & Learning Agents

Programming Assignment 2

REPORT

Vinit P. Doke
190260018[1]

[1] Email: 190260018@iitb.ac.in

October 11, 2021

Content

1 Task 1

1.1 Storing the Transition Probabilities and Rewards

The data structure used is a python dictionary with format as follows:

Dictionary = { 'Base State' : { 'Valid Action' : { 'End State' : (Probability, Reward)}}}

Note: Here, Valid Actions imply actions that have non-zero probability of transition from a given base state to a possible end state.

Under the main dictionary, there are multiple base states as keys which individually have dictionaries keyed by Valid Actions available to that state, which further have dictionaries keyed by the End State which they result in. For a given Base State - Action - End State combination, we store a tuple with the transition probability and the corresponding reward.

1.2 Value Iteration

The value iteration procedure is implemented in same manner as presented in slides except for given modifications:

- Two initial value vectors are initialized as a complete Zero Vector and a vector with all elements as 1. Further updates are made on one of them and then we check for the euclidean distance between them as exit condition. The tolerance is set to

$$tolerance = 10^{-12}$$

1.3 Howard's Policy Iteration

For Howard's Policy Iteration, the policy is initialized as follows:

- For a given state, we look at all the valid actions available and set the action to be taken as the valid action that was indexed at last. This could also be done stochastically.
- Not having a valid initialisation can create problems like the generated policies having invalid actions for some states.

1.4 Linear Programming

- For this approach, only transitions with non-zero probabilities are appended to the constraints to speed up the calculation. This greatly improved the execution time.

2 Task 2

2.1 Obtaining transitions with non-zero probabilities:

1. For a given player whose policy we want to generate in response to an opponent's policy, we begin with one of the "actionable" states given in the states file of the said player.
2. We look for "0"s as the positions which are valid plays and obtain multiple intermediate states which could be one of the following:
    - If the produced intermediate is not present in the opponent's policy, the game has ended. We then check for win, loss or draw and assign rewards and transition probability as discussed in section 2.2
    - If the produced intermediate is present in the policy of the opponent, we generate multiple end states resulting from the opponent's policy.
3. Once we have the end states, following are the possibilities:
    - If End State in states file of the player, we assign the transition probability according to the opponent's policy and a reward of 0
    - If the End State is not in states file of the player, the game has ended. We then check for win, loss or a draw and assign transition probability and reward as discussed in section 2.2

2.2 Reward Structure and Addition of Terminal States

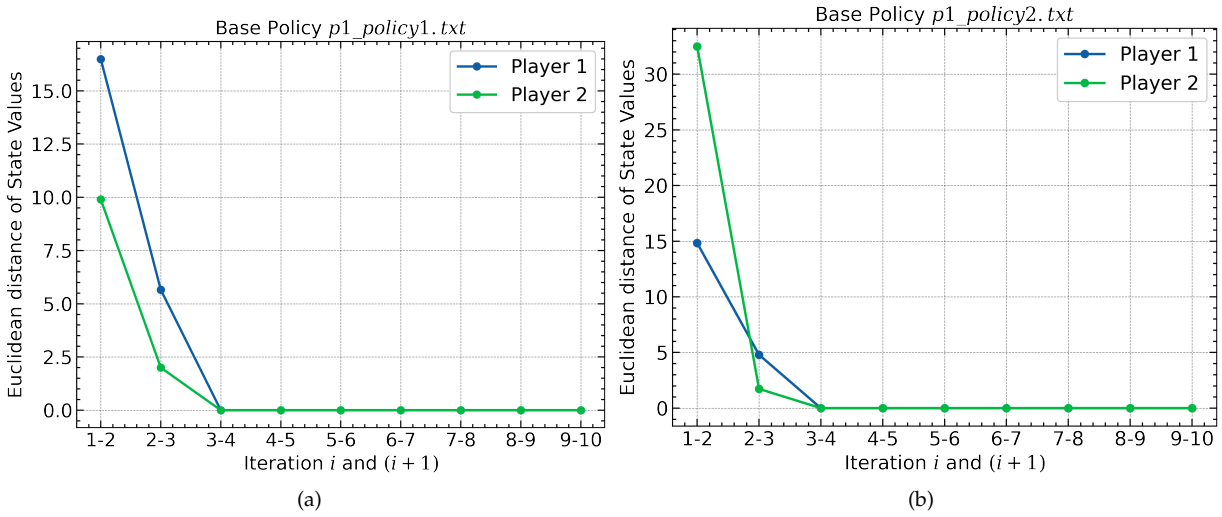
Rewards:
The Rewards are as follows:

- **0** : for draws, losses and game continuing transitions
- **1** : for action leading to games ending in Wins for the Player

Terminal States:
I have added two extra states that are Terminal in nature. The first type is when an action leads to a draw or loss. All such transitions end up in same end state given by this terminal state.
Accordingly, the other type of the terminal state is a "win" terminal state that is mapped to all the actions leading to win for the player under consideration.

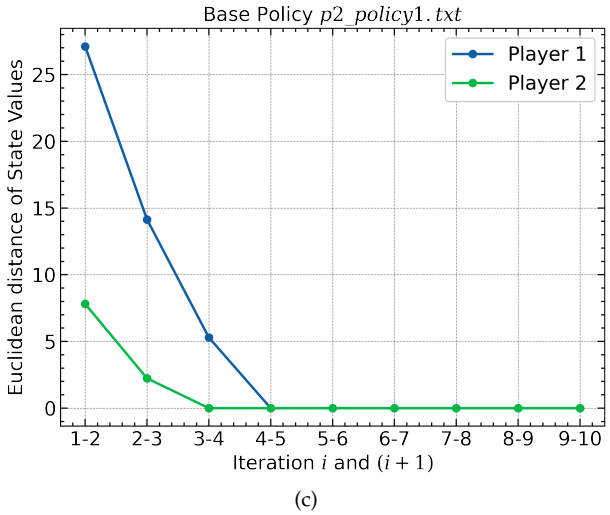
3 Task 3

3.1 Plots of Euclidean Distances between Value Functions of consecutive policies

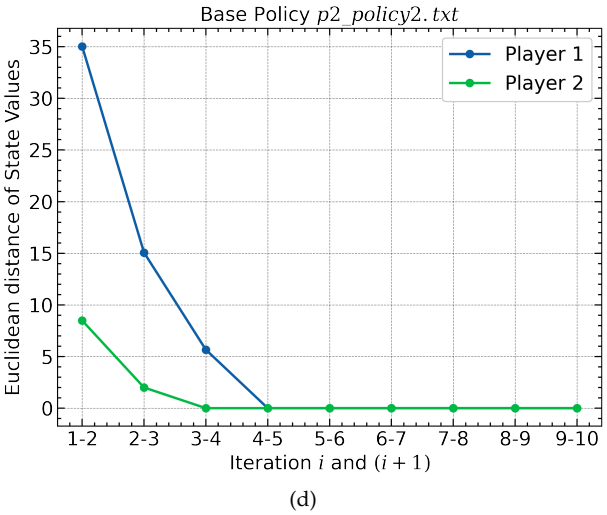


(a)

(b)



(c)



(d)

3.2 Remarks

1. **task3.py** generates a folder named **task3** which has sub-folders **p1** & **p2**. Each such folder contains folders **values** & **policies** with 10 files each (1 more file for the initial player)

2. **Observations :**

   - It is evident by plotting the euclidean distances between the consecutive value functions of the same player that the value functions converge to the same function, implying that the policies converge too.

References

1. subprocess module
   https://docs.python.org/3/library/subprocess.html
2. argparse module
   https://docs.python.org/3.8/library/argparse.html#module-argparse
3. value iteration
   https://towardsdatascience.com/the-value-iteration-algorithm-4714f113f7c5
4. pulp reference
   https://towardsdatascience.com/linear-programming-and-discrete-optimization-with-python-using-pulp-449f3c5f6e99