Artificial Intelligence Laboratory (CS-362)
End Semester Report
Group - PSYSTRIKE

Ayush Yadav (201851031) [1], Saksham Jain (201851106)[2], Shah Jainilkumar (201851116)[3]
Utkarsh Tiwari (201851138)[4], Rahin Vadsariya (201851139)[5]
[1]**Indian Institute of Information Technology Vadodara**

CONTENTS

I. Problem 8

Problem Statement: Basics of data structure needed for state-space search tasks and use of random numbers required for MDP and RL.

A. Introduction of MENACE - by Donald Michie

MENACE learns to play noughts and crosses by playing the game repeatedly against another player. In each game it refines its strategy until after having played a certain number of games it becomes almost perfect and its opponent is only able to draw or lose .It is somewhat similar to a neural network which is randomly optimized at the beginning. The learning process involves being "punished" for losing and "rewarded" for drawing or winning and this type of learning is termed as Reinforcement Learning. Here , each matchbox represents a specific board layout of Noughts and Crosses .If each unique layout is considered then the number of boxes required would be very large. So , to reduce the boxes Layouts that are rotated versions of each other or are symmetrical are considered as a single box limiting the number of boxes required to 304.

B. Training MENACE

Initially , all the boxes contain colour-coded beads, where each colour represents a move or position on a board. MENACE makes a move as the human player randomly picks a bead out of the box that represents the game's current state as similar to neural network's weights . The colour of the selected bead from the specific box determines where MENACE will move.

C. Cumulative Reward Equation

Beads in the box are adjusted when there is success or failure. At the end of each game –

- if MENACE loses, each bead MENACE used is removed from each box.
- If MENACE wins, three beads the same as the colour used during each individual turn are added to their respective box.
- If the game results in a draw, one bead is added to each box of the chosen colour.

Considering the above points the equation formed for the number of beads in the first box will be -

3 X Wins + Draws - Losses

After many games are played out, the matchboxes will have more of some beads than the others. This certainly tinkers with the probability of a bead being selected randomly which changes the probability of the next state after every move.

D. Analysis & Results

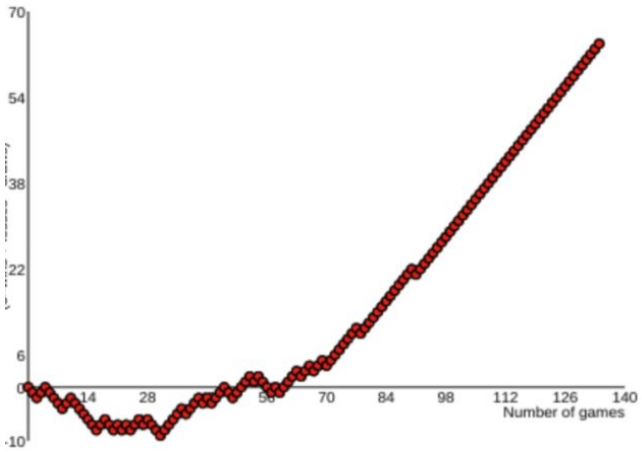


Figure: Variation of Reward Equation (Y axis) with number of games (X axis) for Menace against **Perfect Machine**

As we can see in the graph when MENACE plays a perfect-playing computer initially the value of the equation goes negative indicating the losses but later it increases gradually. Although the number of wins increase rarely but as a draw is considered positive to we observe the given result suggesting that MENACE has learned. The graph suggests that MENACE could never win against the perfect algorithm, but ended up drawing every time after about 90 games, making it equally as perfect as a computer.

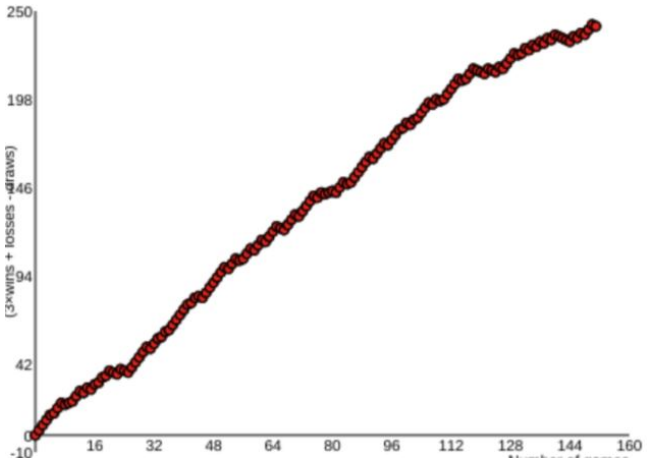


Figure: Variation of Reward Equation (Y axis) with number of games (X axis) for Menace against **Unintelligent Machine**

On the other hand when MENACE plays a with a random picking opponent , the result is a near-perfect positive correlation which indicates that against an unintelligent player the tendency of loss for MENACE is very less although it makes random moves in the beginning still the equation never becomes negative.

II. Problem 9

Problem Statement: Understanding Exploitation-Exploration in simple n-arm bandit reinforcement learning task, epsilon-greedy algorithm. n-armed bandit.

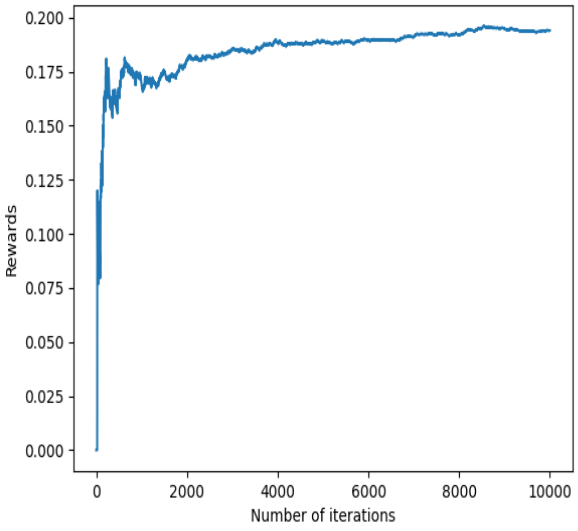
Algorithm 1 Epsilon Greedy Algorithm

Initialize, for $a = 1$ to k:
$Q(a) \leftarrow 0$
$N(a) \leftarrow 0$

Repeat forever:
$A \leftarrow \begin{cases} \arg\max_a Q(a) & \text{with probability } 1-\varepsilon \quad \text{(breaking ties randomly)} \\ \text{a random action} & \text{with probability } \varepsilon \end{cases}$
$R \leftarrow bandit(A)$
$N(A) \leftarrow N(A) + 1$
$Q(A) \leftarrow Q(A) + \frac{1}{N(A)}\left[R - Q(A)\right]$

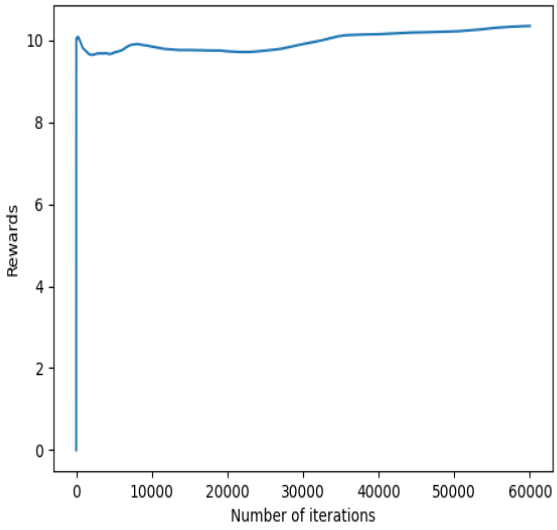
A. Binary bandit with stationary reward (epsilon greedy technique)

- This is a 2 arm-bandit where the rewards are 1 for success and 0 for failure.
- The exploration and exploitation are based on the Epsilon Greedy Algorithm.
- A random number between 0 and 1 is generated. If the generated number is greater than epsilon then exploitation is performed on the basis of the prior knowledge.
- In exploitation, the maximum Q value is selected and the actions which corresponds to maximum reward is performed. In the other case, exploration is done and the reward value corresponding to the action is given.



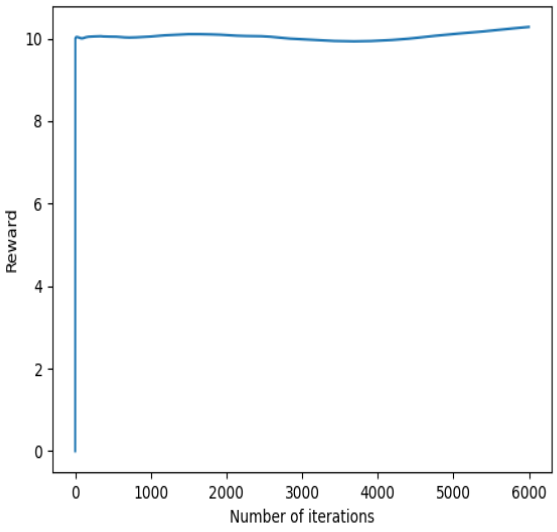
B. Develop a 10-armed bandit in which all ten mean-rewards start out equal and then take independent random walks

- This is a 10 arm-bandit where the mean-rewards is initialized with array of ones.
- The exploration and exploitation is based on the Epsilon Greedy Algorithm.
- A random number between 0 and 1 is generated. If the generated number is greater than epsilon, we perform exploitation on the basis of the prior knowledge otherwise exploration is performed.
- In each itertaion array whith length of 10 and whose values has mean 0 and standard deviation 0.01 is generated.
- –The mean array is updated by adding the values of the generated array to the mean array.The updated mean array is used to give reward to every action.



C. n-arm (10) bandit with non-stationary reward (modified epsilon greedy technique)

- The only difference in this problem from Problem 2 is that in Problem 2 averaging method to update estimation of action reward was used whereas here more weight is given to current earned reward bu using alpha parameter.



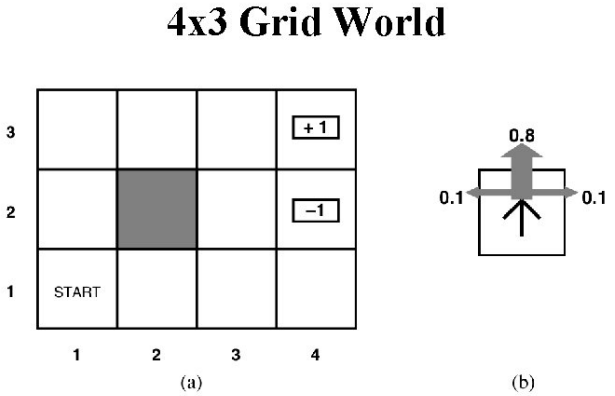
From the above graphs we can conclude that the average

reward in epsilon greedy is more than the simple algorithm.

III. Problem 10

Problem Statement Understand the process of sequential decision making (stochastic environment) and the connection with reinforcement learning.

A. Suppose that an agent is situated in the 4x3 environment as shown in Figure. Beginning in the start state, it must choose an action at each time step. The interaction with the environment terminates when the agent reaches one of the goal states, marked +1 or -1.



Algorithm 2 Value Iteration: Learn function $J : \mathcal{X} \to \mathbb{R}$

Require:
Sates $\mathcal{X} = \{1, \ldots, n_x\}$
Actions $\mathcal{A} = \{1, \ldots, n_a\}$, $A : \mathcal{X} \Rightarrow \mathcal{A}$
Cost function $g : \mathcal{X} \times \mathcal{A} \to \mathbb{R}$
Transition probabilities $f_{xy}(a) = \mathbb{P}(y|x, a)$
Discounting factor $\alpha \in (0, 1)$, typically $\alpha = 0.9$
procedure ValueIteration($\mathcal{X}$, A, g, f, α)
  Initialize $J, J' : \mathcal{X} \to \mathbb{R}_0^+$ arbitrarily
  while J is not converged **do**
    $J' \leftarrow J$
    for $x \in \mathcal{X}$ **do**
      for $a \in A(x)$ **do**
        $Q(x, a) \leftarrow g(x, a) + \alpha \sum_{j=1}^{n_x} f_{xj}(a) \cdot J'(j)$
    for $x \in \mathcal{X}$ **do**
      $J(x) \leftarrow \min_a\{Q(x, a)\}$
  return J

Value iteration computes the optimal state value function by iteratively improving the estimate of V(s). The algorithm initialize V(s) to arbitrary random values. It repeatedly updates the Q(s, a) and V(s) values until they converges. Value iteration is guaranteed to converge to the optimal values.

B. Find the value functions corresponding optimal policy for the following:

1) r(s) = -0.04

Value function Matrix			
-1.22	-0.82	-0.27	0.00
-1.45	0.00	-0.87	0.00
-1.53	-1.45	-1.22	-1.17

2) r(s) = -2

Value function Matrix			
-59.69	-46.00	-24.31	0.00
-65.39	0.00	-21.93	0.00
- 63.08	-52.78	-34.49	-20.74

Giving negative reward reflects Q(1,2) that the value function near the reward state is higher than away from the reward state, and it is expected behaviour of the value iteration policy, because that should only be the way to reach the end state

3) r(s) = 0.1

Value function Matrix			
2.93	2.38	1.44	0.00
3.08	0.00	0.63	0.00
2.83	2.19	1.15	0.22

4) r(s) = 0.02

Value function Matrix			
0.54	0.54	0.45	0.00
0.47	0.00	-0.23	0.00
0.32	0.09	-0.21	-0.58

5) r(s) = 1

Value function Matrix			
29.78	23.13	12.48	0.00
32.44	0.00	10.30	0.00
31.10	25.76	16.42	9.21

while giving positive reward Q.(3,4,5) deflects the agent and force to stay in the state where it is, so in conclusion we can say that it is better to give reward in negative number, that helps agent to go towards the destination state and get maximum rewards, as we can see in (5) when reward = 1, the value at [1,1] is higher than [1,3], it means the agent is more stable at position [1,1] but in ideally it should be at position [1,3]

C. Gbike bicycle rental with policy iteration and improvement

We have used Dynamic Programming to break up problem into smaller subsets and iteratively solve them. This can be done as we have a perfect model of the environment in the form of an Markov Decision Process. The goal here is to setup the environment of the MDP defining the states, actions, probabilities, and rewards and further use Policy Iteration to solve the problems.

Policies are a mapping of the actions that an agent takes from a particular state. Many different actions are possible for the agent, so to solve this problem, it needs to know which action will result in the highest long-term return.

1) Policy Iteration

Policy Iteration (using iterative policy evaluation) for estimating $\pi \approx \pi_*$

1. Initialization
   $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation
   Loop:
   $\Delta \leftarrow 0$
   Loop for each $s \in \mathcal{S}$:
   $v \leftarrow V(s)$
   $V(s) \leftarrow \sum_{s',r} p(s', r|s, \pi(s))[r + \gamma V(s')]$
   $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta < \theta$ (a small positive number determining the accuracy of estimation)

3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
   old-action $\leftarrow \pi(s)$
   $\pi(s) \leftarrow \text{argmax}_a \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')]$
   If *old-action* $\neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*
   If *policy-stable*, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2

Algorithm of Policy Iteration

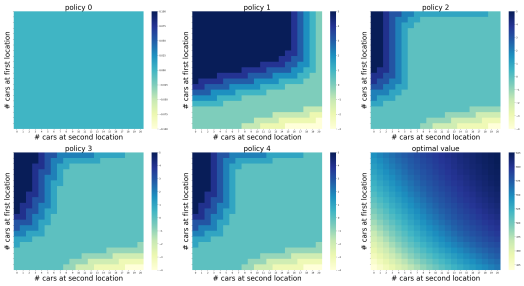
Policy Evaluation is the process of taking a policy, and calculating the return of every single state based on following a particular policy. Once we arrive at a true value function for our policy, we can attempt to improve it.

Policy Improvement is the act of looking at all actions we could take from a given state, acting greedily by choosing the action that gives the highest return with respect to our value function.

Every time we arrive at the true value function for a policy, we determine how much value we should expect to return at each state given a particular policy. This helps us in making a better policy. At the beginning, we randomized our policy but with a true value function, we can exploit the values by changing our policy to choose the best action that will return the highest value from a given position.

Improving the policy involves testing actions at each state and choosing the best action among them. Unlike evaluation, here we iterate through all the actions and then have a list of returns to look over.

The system go through these two processes again and again. It finds the true value function for a policy, and then improve the policy based on the value function, before going back and refining the value function again, whereby we improve the policy and keep doing this until it arrives at the optimal policy for our solution.



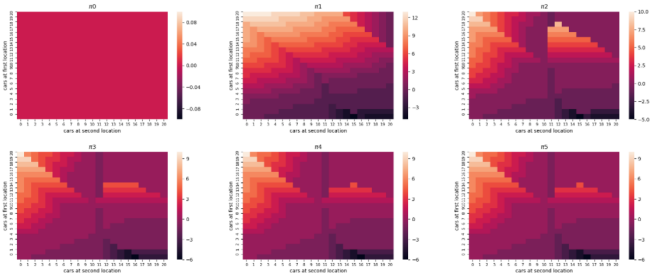
Improvement of policy

As we can see the graph, the left figure of given graph is shown for the initial state of the car which is 0 and 0 both the sides, and then iterating over new policy evaluation and continuous policy improvement, we end up being in the last 6th state, where we can see the optima has been occurred.

D. Synchronized Gbike bicycle rental with policy iteration and improvement

Adding non-linearities to the original rental problem

- One of Jack's employees at the first location rides a bus home each night and lives near the second location. She is happy to shuttle one gbike to the second location for free. Each additional gbike still costs 2, as do all cars moved in the other direction.
- In addition, Jack has limited parking space at each location. If more than 10 gbike are kept overnight at a location (after any moving of gbike), then an additional cost of 4 must be incurred to use a second parking lot (independent of how many gbike are kept there).



Improvement of policy

The above graph also shows the similar property as the above one, we are trying to optimize the policy and after 5 iteration we are getting the optimal policy which can not be improved.

E. Conclusion

We solved the above problems using Dynamic Programming. We stored the intermediate value and policy matrices and used them in our policy evaluation and improvement functions. In order to use Bellman updates though, we need to know the dynamics of the environment, just like we knew the probabilities for rewards and the next states in the rental example. If we can only sample from the underlying distribution and don't know the distribution itself, then Monte

Carlo methods can be used to solve the corresponding learning problem.

IV. Problem 11

Problem Statement : To understand the design of type-1 (Mamdani) Fuzzy expert system

A. Identify the appropriate time needed to wash the load of clothes taking input as two parameters load of clothes and dirtyness of clothes using fuzzy logic

LA Zadah in 1965 , developed a fuzzy membership values ranges from 0 to 1. It is calculated by membership function defined on fuzzy system. There are many types of membership function like Triangular membership function, Trapezoidal membership function, rectangular membership function etc. these function is useful in calculating membership values of linguistic variable of fuzzy system. In fuzzy logic, The crisp data sets are actual input in fuzzy system and it is converted into fuzzy values called Fuzzification. The reverse of fuzzification is called defuzzification. The defuzzification converts the fuzzy values into Crisp values.

To solve the problem of washing machine using fuzzy logic controller. The fuzzy logic is a method of reasoning and fuzzy logic adopts the decision capability like human being. We have taken the following input and output variables in this problem.

The washing machine takes an input –

1) Load of the clothes(mclothes) – We have enter the load of the clothes in range 0-10
   Fuzzy Lingusitic Variables - low_load, medium_load or full_load.
2) Amount of Dirtyness(dclothes) – We have to enter the dirtiness of the clothes in range 0-10
   Fuzzy Lingusitic Variables - not_dirty, lightly_dirty, medim_dirty or very_dirty.

The following rules are involved in the fuzzy controller for producing the result in term of time taken by washing machine to clean the clothes in range 0-20 minutes.

RULES :

rule1 = ctrl.Rule(mClothes['full_load']&dCloths['very_dirty'], wTime['very_long_time']) bullet.

rule2=ctrl.Rule(mClothes['full_load']&dCloths['medium_dirty'], wTime['very_long_time'])

rule3=ctrl.Rule(mClothes['full_load']&dCloths['lightly_dirty'], wTime['long_time'])

rule4=ctrl.Rule(mClothes['full_load']&dCloths['not_dirty'], wTime['little_time'])

rule5=ctrl.Rule(mClothes['medium_load']&dCloths['very_dirty'], wTime['very_long_time'])

rule6=ctrl.Rule(mClothes['medium_load']&dCloths['medium_dirty'], wTime['medium_time'])

rule7=ctrl.Rule(mClothes['medium_load']&dCloths['lightly_dirty'], wTime['medium_time'])

rule8=ctrl.Rule(mClothes['medium_load']&dCloths['not_dirty'], wTime['little_time'])

rule9 = ctrl.Rule(mClothes['low_load']&dCloths['very_dirty'], wTime['long_time'])

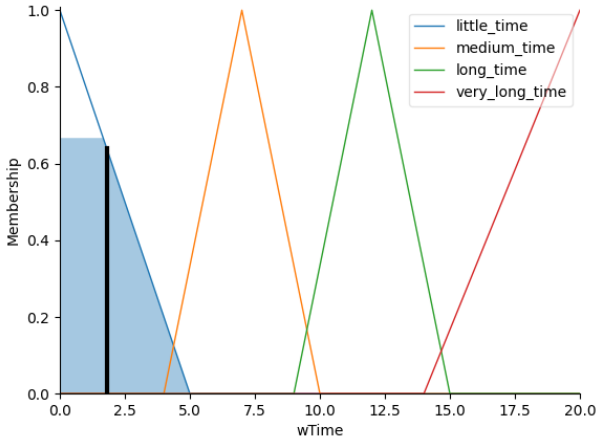
rule10=ctrl.Rule(mClothes['low_load']&dCloths['medium_dirty'], wTime['long_time'])

rule11=ctrl.Rule(mClothes['low_load']&dCloths['lightly_dirty'], wTime['little_time'])

rule12=ctrl.Rule(mClothes['low_load']&dCloths['not_dirty'], wTime['little_time'])

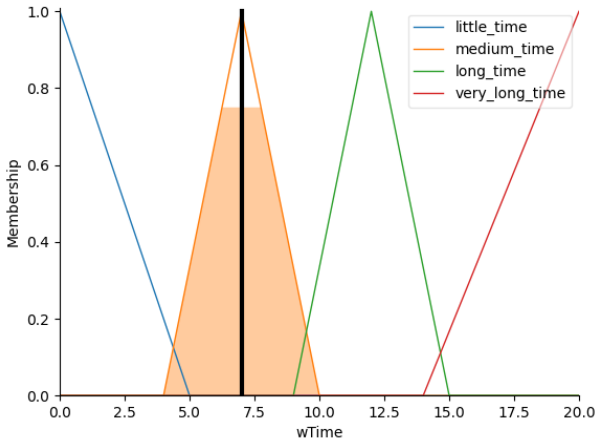
Case 1: Little Time

washing_time.input['mClothes'] = 1
washing_time.input['dCloths'] = 1
washing_time.compute()
wTime.view(sim=washing_time)
Washing time - 18.055555555555558

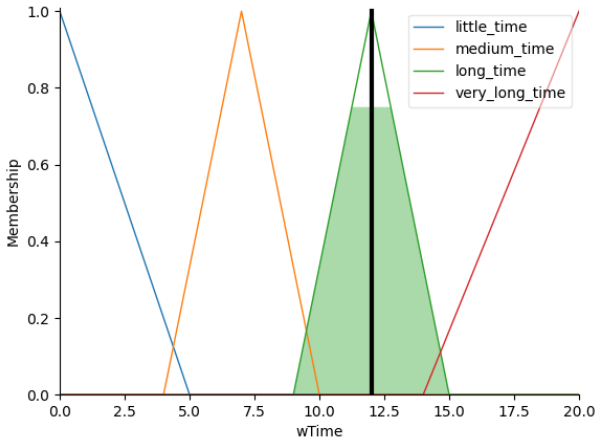


Case 2: Medium Time

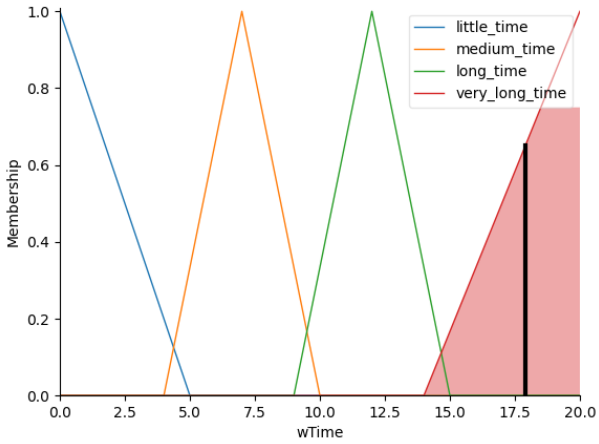
washing_time.input['mClothes'] = 4
washing_time.input['dCloths'] = 5
washing_time.compute()
wTime.view(sim=washing_time)
Washing time – 7.0



Case 3: Long Time
washing_time.input['mClothes'] = 1
washing_time.input['dCloths'] = 9
washing_time.compute()
wTime.view(sim=washing_time)
Washing Time – 12.0



Case 4: Very Long Time
washing_time.input['mClothes'] = 4
washing_time.input['dCloths'] = 9
washing_time.compute()
wTime.view(sim=washing_time)
Washing time – 17.9



By using the proposed fuzzy logic controller to solve this problem, we are able to obtain the different washing time according to the value of two different types of input parameter. i.e the amount of dirt and the load of clothes. So this methodology leads to saving the electricity energy and time taken by washing machine.

V. Acknowledgment

We are highly indebted to Dr Pratik Shah for his guidance and constant supervision as well as for providing necessary information for completion of assignment problems. We would like to express our special gratitude and thanks to Teaching Assistants who helped us in resolving our doubts and putting forward helping hands.

VI. Github Link

Group - PSYSTRIKE Assignment Codes

References

[1] Washing Machine controller using Fuzzy Logic technique
[2] Experiments on the mechanization of game-learning
[3] Menace: the Machine Educable Noughts And Crosses Engine
[4] How 300 Matchboxes Learned to Play Tic-Tac-Toe Using MENACE
[5] Application of fuzzy logic in design of smart washing machine
[6] K Armed Bandit Reinforcement Learning by Richard Sutton and Andrew Barto Chapter 4