OPTIMAL POLICY USING LPP

TEAM

- Tanvi Karandikar 2018101059
- Mallika Subramanian 2018101041

OVERVIEW

The optimal policy for a Markov Decision Process can be obtained via many ways. Here the problem is posed as a Linear Programming Problem(LPP) and the optimal policy is found by solving the LP.

Setup and Run

```
cd team_62
python3 solution.py
```

This will generate the output.json file that contains the following parameters :

- The A matrix
- The R array
- The alpha array
- The optimised x array
- The derived
- The Deterministic policy
- The final optimised objective value

MATRIX A :

The X vector (that is to be solved for, using the LPP) contains all (STATE, ACITON) pairs which represents the expected number of times an action is taken in a particular state.

The number of rows for the matrix A is equal to the number of **STATES** in the MDP. The number of coloumns for the matrix A is equal to the number of **(STATE, ACTION) pairs.** possible in the MDP. Eg: x11,x12,x12,x21,x22 ... Where each (x_{ij}) represents the action **j** taken in state **i**

Corresponding to each row (state) the entries in the matrix A represent the effect that, that state will have for every (STATE, ACTION) pair.

The probabilities of the (STATE, ACTION) pairs contribute to the value of the corresponding matrix entries.

All **Inflow** probabilites are subtracted and **Outflow** probabilities are added.

That is, consider a any row of matrix A, say *p*, if the (STATE, ACTION) pair causes an incoming edge into *state p* then this probability is **subtracted** whereas if it is an outgoing edge, the probability is **added**.

Our implementation involves filling the matrix A **Coloumn wise**. Here we have considered the effect that a particular (STATE, ACTION) pair will have on all the states of the MDP and accordingly filled the result of the probabilites (outflow-inflow)

The standard that we have followed in case of the NOOP actions taken at the terminal states is that the entry in A corresponing to the row of the TERMINAL state and coloumn for the pair (TERMINAL, NOOP) the entry will be a 1.

In our given MDP for which we have to obtain the policy, the dimensions of A are 60 x 100 - since there are 60 states in total (5 values of health, 4 values of arros x 3 values of stamina) and there are 100 (STATE, ACTION) pairs.

PROCEDURE TO OBTAIN THE POLICY :

The optimal policy for an MDP will be the one that results in the maximum reward. Hence this is the objective function that we have to maximize.

> Objective function = ΣV_i ... TO MAXIMIZE

```
objective = cp.Maximize(cp.sum(cp.matmul(r,x), axis=0))
```

Using the Bellmann equation we know that :

> $V_i <= [R (I , A) + \gamma * \Sigma P (J | I , A) * V_j]$

Therefore the function is subject to to the following constraints:

> Ax = *alpha* Where alpha represents the probabilities of each state being a start state.

And the non-negativity constraints for the values of x that is :

> $x >= 0$

```
constraints = [cp.matmul(a, x) == alpha, x>=0]
```

These are the pre-requisites to solve the LPP. On solving the LPP we obtain the values for the vector **x**.

Each element of **x** (x_{ij}) represents the expected number of times an action **j** is taken in the state **i**.

Once the LPP is solved and the vector *x* is obtained, the optimal policy is one that selects that action with the **maximum expected value** for each particular state.

MULTIPLE POLICIES :

Yes there can be multiple policies that can be generated for a single MDP.

- In order to select an optimal action to be taken in a state, we select the one with the maximum value for *expected number of times the action will be taken in that state*. It may so happen that this *expected* *value* is the same for 2 or more actions in the same state. Hence if the condition to look for the maximum is changed from a strict inequality to a normal inequality, then only the first action amongst these equal ones will be chosen.

- Changing the **reward/step-cost/penalty** for each state can also result in a change in the policy. Since if the penalty is higher (stepcost is lower) the agent will want to and try to reach the terminal state faster with a greater reward.
- Selecting a different start state can also result in generating a different policy to the MDP. That is any change in the *alpha* vector can change the policy.
- Furthermore, changing values of parameters such as the probabilities of taking various different actions, the rewards/penalties received at each state etc will obviously change the policy since the state diagram of the problem itself would be different in such cases.