STOCHASTIC OPTIMIZATION

2022

LUIS ARIAS GONZALES s293581

ALBERTO MARTÍN GARRE s293972

Politecnico di Torino

TABLE OF CONTENTS

INTRODUCTION

The first step in this assignment should be an introduction of the problem we have chosen, explaining the different features of it as well as their relationship with the concepts studied during the course.

We have selected the problem related to dynamic optimization. This problem talks about a specific company with 6 clerks. There is a disease circulating and the company has two options (two actions) for each possible case (state): carry on with the normal way of doing things (action 1, it implies a specific transition matrix as well as a reward matrix) or, instead, apply an 'emergency protocol' (action 2, with its corresponding transition matrix and reward matrix). Action 2 means reducing the probability of infection for the workers but also there is a reduction in the reward.

Therefore, the problem goal is to find the optimal policy (which action should be taken in each state) for maximizing the reward. We are going to deal with the problem through methods based on Q-factors and Reinforcement Learning. Once we have completed the computation of the two methods, we will discuss which method is better.

PROBLEM

DEFINITION OF NECESSARY ELEMENTS

Before getting into explaining the methods used, it is important to define the elements (matrices) that we will need in the computation of the algorithm. This problem represents a Markov Decision Process. This process is composed of different elements: states, actions (with their transition matrices), rewards (with their reward matrices), policies (there is a huge number of possible policies (exactly $2^7 = 128$) and we have to find which is the optimal, the one that provides the maximum reward) and the performance metrics (in our problem we are just going to work with the discounted reward 'λ').

- States. Represent the possible situations of the system, i.e., in our case there could be either 0,1,2,3,4,5 or 6 clerks working at each moment. This composes our set of states, S= {0,1,2,3,4,5,6}.
- Actions & Rewards. We have already explained them in the introduction. There are two possible actions: the normal way of doing things or emergency protocol. From now on we will refer to them as action 1 and action 2 respectively. Each of these two actions has a transition matrix P_a (it characterizes the dynamics on the system under the given actions) and a reward matrix R_a (it describes the reward that each transition would produce). The transition matrix for each action represents the probability of being in one state going to another or the same state. In the context of the problem, it means that if you are in one specific state, for example, 2 working clerks, what is the probability that two of the ill workers recover and the two working clerks don't get infected (going from state 2 to state 4). Each of these transition matrices implies a reward.

- Policies. This is where the assignment is focused. We must find an optimal policy to maximize the profit. For example, a possible policy would be: if there are 2 or fewer clerks currently working (the rest of them are ill) impose the 'emergency protocol', otherwise keep on with the normal way of doing things.
- Performance metrics. As we have anticipated we are going to use the discounted reward as a performance metric. This metric is based on the idea that the value of money reduces with time. A reward might be less worthy if it takes time to earn it.

COMPUTATION OF NECESSARY ELEMENTS

This section is dedicated to the explanation of the implementation of these necessary elements for the problem in MATLAB.

- Actions. In this problem, there is a formula for the transition matrix. Nevertheless, looking closely at the formula specified in the problem it is based on the binomial distribution. Thus, we can use the MATLAB function *'binopdf ()'* when computing the transition matrix. The difference between the two actions is that action1 implies a probability of infection (p_i) of ¼ while action2 supposes a $p_i = 1/20$. Computing the required loop for defining the matrix we obtain:

$$p(i,j) = \sum_{0 \le k \le n} f_B(k, i, p_i) * f_B(j - i + k, n - i, p_r)$$

Formula for computing the transition matrix (f_B refers to the binomial probability)

$$P_{a1} = \begin{pmatrix} 0.2621 & 0.3932 & 0.2458 & 0.0819 & 0.0154 & 0.0015 & 6.4e-05 \\ 0.0819 & 0.3482 & 0.3584 & 0.1664 & 0.04 & 0.0049 & 2.4e-04 \\ 0.0256 & 0.1792 & 0.3936 & 0.2896 & 0.0961 & 0.015 & 9e-04 \\ 0.008 & 0.078 & 0.2715 & 0.3916 & 0.2036 & 0.0439 & 0.0034 \\ 0.0025 & 0.0313 & 0.1502 & 0.3394 & 0.3459 & 0.1181 & 0.0127 \\ 7.8125e-04 & 0.0119 & 0.0732 & 0.2285 & 0.3691 & 0.2689 & 0.0475 \\ 2.4414e-04 & 0.0044 & 0.033 & 0.1318 & 0.2966 & 0.356 & 0.178 \end{pmatrix}$$

Transition Matrix for Action 1

$$P_{a2} = \begin{pmatrix} 0.2621 & 0.3932 & 0.2458 & 0.0819 & 0.0154 & 0.0015 & 6.4e-05 \\ 0.0164 & 0.3318 & 0.3994 & 0.1971 & 0.0490 & 0.0061 & 3.04e-04 \\ 0.001 & 0.0399 & 0.4090 & 0.3843 & 0.1411 & 0.0233 & 0.0014 \\ 6.4e-05 & 0.0037 & 0.0721 & 0.4916 & 0.3423 & 0.0834 & 0.0069 \\ 4e-06 & 3.060e-04 & 0.0088 & 0.1141 & 0.5767 & 0.2675 & 0.0326 \\ 2.5e-07 & 2.3813e-05 & 9.0844e-04 & 0.0174 & 0.1672 & 0.6598 & 0.1548 \\ 1.5625e-08 & 1.7813e-06 & 8.4609e-05 & 0.0021 & 0.0305 & 0.2321 & 0.7351 \end{pmatrix}$$

Transition Matrix for Action 2

- Rewards. Regarding reward matrices, each one has a different formula for its computation.

$$r(i,1,j) = 50 * (e^{\frac{i}{7}} - 1)$$

Formula for computing the reward matrix for action 1

$$R_{a1} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 7.782 & 7.782 & 7.782 & 7.782 & 7.782 & 7.782 & 7.782 \\ 16.5356 & 16.5356 & 16.5356 & 16.5356 & 16.5356 & 16.5356 & 16.5356 \\ 26.7532 & 26.7532 & 26.7532 & 26.7532 & 26.7532 & 26.7532 & 26.7532 \\ 38.5397 & 38.5397 & 38.5397 & 38.5397 & 38.5397 & 38.5397 & 38.5397 \\ 52.1364 & 52.1364 & 52.1364 & 52.1364 & 52.1364 & 52.1364 & 52.1364 \\ 67.8209 & 67.8209 & 67.8209 & 67.8209 & 67.8209 & 67.8209 & 67.8209 \end{pmatrix}$$

Reward Matrix for Action 1

$$r(i,2,j) = \begin{Bmatrix} 0 & if \quad i = 0 \\ 35 * \left(e^{\frac{i-1}{7}} - 1\right) & else \end{Bmatrix}$$

Formula for computing the reward matrix for action 2

$$R_{a2} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 5.3748 & 5.3748 & 5.3748 & 5.3748 & 5.3748 & 5.3748 & 5.3748 \\ 11.5749 & 11.5749 & 11.5749 & 11.5749 & 11.5749 & 11.5749 & 11.5749 \\ 18.7272 & 18.7272 & 18.7272 & 18.7272 & 18.7272 & 18.7272 & 18.7272 \\ 26.9778 & 26.9778 & 26.9778 & 26.9778 & 26.9778 & 26.9778 & 26.9778 \\ 36.4954 & 36.4954 & 36.4954 & 36.4954 & 36.4954 & 36.4954 & 36.4954 \end{pmatrix}$$

Reward Matrix for Action 2

We have decided to build three-dimensional matrices for both actions and rewards to ease the computation of the algorithms. This has been done because when we call either the probability of a transition matrix or a reward it is required to know which is the action we are talking about. Without the three-dimensional matrices, it would be harder to be extracting information from different matrices depending on the action chosen.

ALGORITHMS COMPUTATION

1. Q-FACTOR VALUE ITERATION

This algorithm focuses on discounted case only, and only on the analog of the value iteration algorithm. We associate to every state i ∈ S a value function J(i) that is not related to a specific policy. Considering the Bellman Optimization Equation we define the Q-factor associated with the state-action couple (i,a):

$$Q(i,a) = \sum_{j \in S} p_a(i,j) \left[r(i,a,j) + \lambda * max_{b \in A} Q(j,b) \right]$$

Q-factor formula definition

This algorithm reduces complexity when computing it through MATLAB or any other language. This method is just based on rewriting the value iteration algorithm in terms of Q-factors. At each iteration, we must compute the Q-factors matrix, but it is easier to extract the maximum, in order to obtain the vector J. For both algorithms, we will use the parameter for the discounted reward recommended in the assignment: λ= 0.995. Running the code (for more details go to appendix) we obtain the following optimal policy and the corresponding vector J^*:

$$\mu_{QF}^* = \begin{pmatrix} \mu(0) \\ \mu(1) \\ \mu(2) \\ \mu(3) \\ \mu(4) \\ \mu(5) \\ \mu(6) \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 2 \\ 2 \\ 2 \\ 2 \end{pmatrix} \qquad J_{QF}^* = \begin{pmatrix} 5.043,3 \\ 5.063,2 \\ 5.085,2 \\ 5.110,3 \\ 5.140,0 \\ 5.172,5 \\ 5.207,9 \end{pmatrix}$$

2. REINFORCEMENT LEARNING: Q-LEARNING ALGORITHM

This algorithm is based on the Robbins-Monro algorithm. It introduces a parameter α which is decreasing because we can consider that the more data we have the less relevant the new info is. Now, Q-factors are basically expectations, updated in a Robbins-Monro algorithm. We need to simulate the generation of the next state given a current state and a randomly selected action. This model updates Q-factors according to the following formula:

$$Q^{k+1}(i,a) \leftarrow (1-\alpha_{k+1})Q^k(i,a) + \alpha_{k+1}[\, r(i,a,j) + \lambda * max_{b \in A} Q^k(j,b)\,]$$

Updating Reinforcement Q-factor formula

The main difference between this algorithm and the previous one is that we don't compute the whole matrix at each iteration, we just compute one element per iteration. Nevertheless, this method has much more randomness and thus it will need more iterations. Once we run the algorithm through MATLAB, we obtain the following results:

$$\mu_{RL}^* = \begin{pmatrix} \mu(0) \\ \mu(1) \\ \mu(2) \\ \mu(3) \\ \mu(4) \\ \mu(5) \\ \mu(6) \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 2 \\ 2 \\ 2 \\ 2 \end{pmatrix} \qquad J_{RL}^* = \begin{pmatrix} 5.086,7 \\ 5.101,9 \\ 5.125,9 \\ 5.153,6 \\ 5.181,9 \\ 5.214,2 \\ 5.251,0 \end{pmatrix}$$

CONCLUSIONS

We have correctly computed both algorithms and with the proper specifications for each algorithm, they reach the same result. The best policy, considered as the optimal one, is picking action 1 (the normal way of doing things) in the cases of having either 0, 1, or 2 workers available, and picking action 2 (emergency protocol) in the rest of the cases (3, 4, 5 or 6 workers available).

This assignment may arise the question of which is the best algorithm. Nonetheless, we have seen that they get to the same solution, so the matter is not whether one is better than the other but the advantages that each method has. First, we analyze the number of iterations. For the Q-factor algorithm, it has taken k= 6.256 iterations (lasting 0.575 seconds in the computation) while we have selected k=1e+06 (lasting 1.148 seconds in the computation) for the reinforcement algorithm. At first sight, we would argue that the first method is better than the second one because it takes fewer iterations. However, there is a trade-off: Q-factor method requires a smaller number of iterations, but it takes more inner iteration time due to the computation of the Q-factor matrix at each iteration; Reinforcement Learning algorithm requires more iterations because it is a process affected by randomness, but we only compute the value of one element of the Q-factor matrix at each iteration.

Second, as the policy proposals are the same for both methods, we could compare the J^* vectors so we can see which method reports more profit:

$$J^*_{RL} - J^*_{QF} = \begin{pmatrix} 39{,}7069 \\ 36.4384 \\ 35{,}2039 \\ 34{,}1789 \\ 33{,}0351 \\ 32{,}3747 \\ 35{,}3908 \end{pmatrix}$$

This difference shows that Reinforcement Learning reports a higher value function (that represents the benefit on the reward), nevertheless, even though it is good to know that value functions are different, it is not very relevant because as we have explained, the RL algorithm is affected by randomness (it makes simulations for each state).

APPENDIX

```
%% First of all we initialize all the variables

N=6;
pi=1/4;
pr=1/5;
pi_emergency=1/20;

%% CREATE TRANSITION AND REWARD MATRICES
%ACTION 1.................................

%Transition Matrix Related to Action 1
prob=0;
transition_1=zeros(N);

for i=1:(N+1)
    for j=1:(N+1)
        prob=0;
        for k=0:N
            prob=prob+binopdf(k,(i-1),pi)*binopdf((j-1)-(i-1)+k,N-(i-1),pr);
        end
        transition_1(i,j)=prob;
    end
end

 %Reward Matrix Related to Action 1
 reward_1=zeros(N);

for i=1:(N+1)
    for j=1:(N+1)
        reward_1(i,j)=50*(exp((i-1)/7)-1);
    end
end

 %ACTION 2.............................

 %Transition Matrix for Related to Action 2
 transition_2=zeros(N);

for i=1:(N+1)
    for j=1:(N+1)
        prob=0;
        for k=0:N
            prob=prob+binopdf(k,(i-1),pi_emergency)*binopdf((j-1)-(i-1)+k,N-(i-1),pr);
        end
        transition_2(i,j)=prob;
    end
end
```

```
%Reward Matrix related to Action 2
reward_2=zeros(N);

for i=1:(N+1)
    for j=1:(N+1)
        if i==1
            reward_2(i,j)=0;
        else
            reward_2(i,j)=35*(exp(((i-1)-1)/7)-1);
        end
    end
end
```

```
%% SPECIFY A GENERIC MULTIDIMENSIONAL REWARD MATRIX and TRANSITION MATRIX

%When computing the different dynamic optimization algorithms it is helpful
%to make a tri-dimensional matrix unifying the two reward matrices so, the access is easier

total_reward=reward_1;
total_reward(:,:,2)=reward_2;

total_transition=transition_1;
total_transition(:,:,2)=transition_2;
```

```
%% OPTIMIZATION WITH Q-VALUE VALUE ITERATION

%Initialization
k=0;
Q=zeros([N+1,2]);
Q_new_dyn=Q;
epsilon= 1e-10;
lamda=0.995;
J=zeros([N+1,1]);
J_new_dyn=J;
diff=1000;

tol=epsilon*(1-lamda)/(2*lamda);


%Once we have initialized the algoritm we can get into the loop
```

```
while diff>=tol
    k=k+1;
    Q_old_dyn=Q_new_dyn;
    J_old_dyn=J_new_dyn;
    %We compute at each iteration the Matrix of Q-factors
    for i=1:(N+1)
        for a=1:2
            val=0;
            for j=1:(N+1)
                val=val+total_transition(i,j,a)*(total_reward(i,j,a)+(lamda*(max(Q_old_dyn(j,:)))));
            end
            Q_new_dyn(i,a)=val;
        end
    end
```

```
    %We compute the vector composed by the solutions of the different
    %Bellman optimmality equations
    for i=1:N+1
        J_new_dyn(i)=max(Q_new_dyn(i,:));
    end

    %We compute the infinite norm of the difference
    diff=max(J_new_dyn-J_old_dyn);
end

%We specify the optimal policy
optimal_policy_dyn=zeros([N+1,1]);
for i=1:(N+1)
    j=1;
    max_Q=max(Q_new_dyn(i,:));
    while Q_new_dyn(i,j)~=max_Q
        j=j+1;
    end
    optimal_policy_dyn(i)=j;
end
```

```
%% %% OPTIMIZATION WITH REINFORCEMENT LEARNING: Q-LEARNING ALGORITHM

%Initialization
k=0;
Q_reinf=Q;
lamda=0.995;
k_max=1e+06;

%We select the initial state arbitrarily
in_state=2;
curr_state=in_state;
next_state=curr_state;

%Define alpha as a function handle
A=3000;
B=6000;
alpha=@(k) A/(B+k);
%Once we have initialized the algoritm we can get into the loop
while k<k_max

    curr_state=next_state;

    %We create a uniform random variable in order to pick a random action
    %among the two possible ones
    U=rand;
    if U<0.5
        a=1;
    else
        a=2;
    end
```

```
    %We simulate the next state, given the current one
    Z=rand;
    count=0;
    for j=1:(N+1)
        count=count+total_transition(curr_state,j,a);
        if Z<count
            next_state=j;
            break
        else
        end
    end

    %We update the matrix of Q-Factors according the Reinforcement Learning
    %Rule

    Q_reinf(curr_state,a)=((1-alpha(k+1))*Q_reinf(curr_state,a))+alpha(k+1)*...
        (total_reward(curr_state,next_state,a)+(lamda*(max(Q_reinf(next_state,:)))));

    %We update the number of iterations
    k=k+1;
end

%We have built by simulations the optimal matrix of Q-Factors and now we have to translate that
%information into an optimal policy the optimal policy
 J_RL=zeros(N+1,1);
for i=1:N+1
        J_RL(i)=max(Q_reinf(i,:));
 end

optimal_policy_reinf=zeros([N+1,1]);

for i=1:(N+1)
    j=1;
    max_Q=max(Q_reinf(i,:));
    while Q_reinf(i,j)~=max_Q
        j=j+1;
    end
    optimal_policy_reinf(i)=j;
end
```