Decision Making Under Uncertainty: Homework 3

Julian Lema Bulliard*
University of Colorado Boulder

I. Questions

A. Question 1

For this question we will first start off with the reward structure. Let's say that the reward boxes in orders are 1, 20, 1, 25. If we only assume that we are always starting at state 3, if we went left for the immediate reward we would get a reward of 20. Now if we looked ahead and wanted the best final state reward which is 25 on the right we would us the following equations. Where γ is equal to 0.9.

$$Q_{left} = 20$$

$$Q_{right} = 1 + \gamma * 25 = 23.5$$

Now we can obviously see that this is the optimal policy out of going left or right at state three. The difference between these two states is:

$$|Q* - Q_{left}| = 3.5 = \beta$$

Now for the reward difference, we get that:

$$|R* - R_{left}| = 19$$

This is obviously greater than β which we initialized in the difference of Q. Which would be a counter-example to the Lemma 5. This can also be shown through matrix versions of the problem, but to simplify I kept it and constrained the problem to a one-state and one action pairing.

*Graduate Student, Aerospace Engineering Department

B. Question 2

To create my heuristic policy I assumed that there was only 4 squares with rewards in the problem (sort of ambiguous in the problem statement) which were at [20,20] , [20,40] , [40,20] and [40,40]. Based on this I modded my given position to normalize in 20x20 blocks. From here I chose the closes reward square to me and starting moving my way there. This code can be seen in the appendix. If the starting state was outside this inside box. It would guide it towards the middle and then go to a corner state.

From this policy I achieved the following results:

$$\text{Mean for Random Policy} = -95.0784044987906$$

$$\text{Standard error of the mean Random Policy} = 0.7611037269828965$$

$$\text{Mean for Smart Policy} = -18.425936348094723$$

$$\text{Standard error of the mean Smart Policy} = 0.7611037269828965$$

We can easily see that the difference in the mean is:

$$\text{Difference between Random and Smart Policy} = 76.65246815069588$$

C. Question 3

For my Monte Carlo Tree Search, I outputted the following image. This is easily verifiable as a good MCTS tree since we are starting at state [19,19] and the two actions that we consider taking at the beginning are right or up. From these two actions, we can see that it does end up at the terminal state in the next action. It should be noted that the exploration constant was set to 500. This is due to the maximum reward in the space to 100, and the average reward for taking random movements in the space over time to be around -150. This difference multiplied by two gives the exploration bonus/constant.

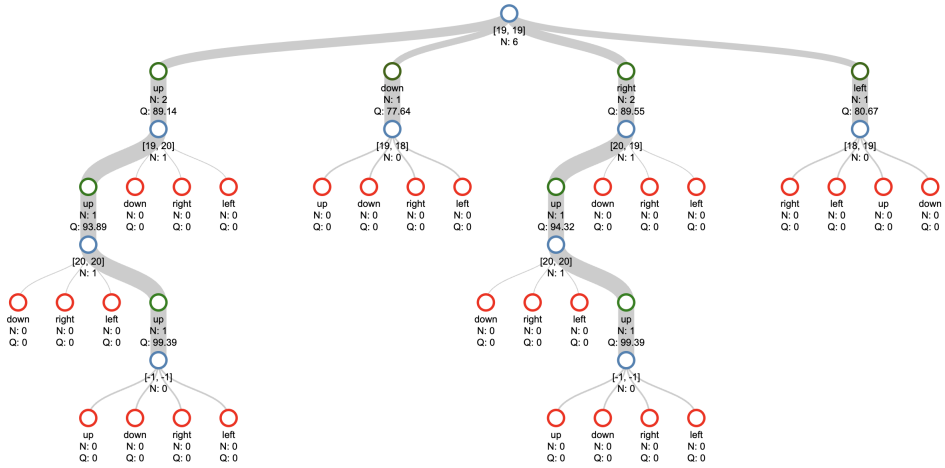

Fig. 1 Monte Carlo Tree Search after 7 Iterations

D. Question 4

From my code, and running 100, 100 step actions I came up with the following results. In general, my code was able to run through 7000-9000 actions every time that it was called.

$$\text{Mean for Random Policy} = -94.0215630004999$$

$$\text{Standard error of the mean Random Policy} = 0.7691807144973255$$

$$\text{Mean for Smart Policy} = 10.812953195560723$$

$$\text{Standard error of the mean Smart Policy} = 2.987935431305634$$

$$\text{Difference between Random and Smart Policy} = 104.83451619606062$$

E. Question 5

For Question 5 I implemented my heuristic policy as I did for problem 2 for the rollout section of the algorithm, but then implemented a recursive Monte Carlo Tree Search to choose the best action after this rollout took place. Through testing, my code only took too long on a maximum 3 or 4 of the data points that it was given. In the end, though, it should be noted that even though the heuristic policy that is given to the program may be lacking, or optimized for the 60 x 60 grid world, it will usually still converge on an adequate solution due to my reasonably high exploration parameter.

I tested several heuristic policies, including changing my own to be able to handle a 100 x 100 grid world, but in the end, it seemed as if the exploration parameter had a higher impact. Overall, I received a score of 78.4 on the problem.

II. Appendix

A. Question 2

```
    using DMUStudent.HW3: HW3, DenseGridWorld, visualize_tree
    using POMDPs: actions, @gen, isterminal, discount, statetype, actiontype, simulate, states, initials
    using D3Trees: inchrome
    using StaticArrays: SA
    using Statistics: mean, std, mean

    ############
# Question 2
############



mdp = HW3.DenseGridWorld(seed = 3)
a = actions(mdp)
s = states(mdp)
policy_function = rand(actions(mdp))
start_state = SA[19, 19]

function rollout(mdp, start_state, max_steps=100)
    r_total = 0.0
    a = actions(mdp)
    s = start_state
    t = 0
    while !isterminal(mdp, s) && t < max_steps
        a = rand(actions(mdp))
        s, r = @gen(:sp,:r)(mdp, s, a)
        r_total += discount(mdp)^t*r
        t += 1
    end
    return r_total # replace this with the reward
end
```

```
function heuristic_policy(s)
    xtrue = s[1]
    ytrue = s[2]

    x = mod(s[1],20)
    y = mod(s[2],20)

    # If between bounds
    if (20 - x > 10) & (20 - x != 20)
        a = :left
    elseif (20 - x <= 10) & (20 - x != 0)
        a = :right
    elseif 20 - y > 10
        a = :down
    else
        a = :up
    end

    # # If outside of internal square
    # if 60 - xtrue > 40
    #     a = :right
    # elseif 60 - xtrue < 20
    #     a = :left
    # elseif 60 - ytrue > 40
    #     a = :up
    # elseif 60 - ytrue < 20
    #     a = :down
    # else
    #     return a
    # end
```

```
    return a
end


function rollout_smart(mdp, start_state, heuristic_policy, max_steps=100)
    r_total = 0.0
    a = actions(mdp)
    s = start_state
    t = 0
    while !isterminal(mdp, s) && t < max_steps
        a = heuristic_policy(s)
        s, r = @gen(:sp,:r)(mdp, s, a)
        r_total += discount(mdp)^t*r
        t += 1
    end
    return r_total # replace this with the reward
end



# This code runs monte carlo simulations: you can calculate the mean and standard error from the results
results_random = [rollout(mdp, rand(initialstate(mdp))) for _ in 1:10000]
mean_random = mean(results_random)
SEM_random = std(results_random)/(length(results_random)^0.5)
@show mean_random
@show SEM_random

results_smart = [rollout_smart(mdp, rand(initialstate(mdp)), heuristic_policy) for _ in 1:10000]
mean_smart = mean(results_smart)
SEM_smart = std(results_smart)/(length(results_smart)^0.5)

@show mean_smart
@show SEM_smart
```

```
@show difference = mean_smart - mean_random
```

B. Question 3

```
    ############
# Question 3
############

########################################
function rollout_smart(mdp, start_state, heuristic_policy, max_steps=100)
    r_total = 0.0
    a = actions(mdp)
    s = start_state
    t = 0
    while !isterminal(mdp, s) && t < max_steps
        a = heuristic_policy(s)
        s, r = @gen(:sp,:r)(mdp, s, a)
        r_total += discount(mdp)^t*r
        t += 1
    end
    return r_total # replace this with the reward
end
########################################
function bonus(N, Ns, s, a)

    if N[(s,a)] == 0 || Ns == 0
        return Inf
    end

    return sqrt(log(Ns)/N[(s,a)])
end
```

```
########################################

function explore(mdp,N,s,Q,c)
    A = actions(mdp)
    Ns = sum(N[(s,a)] for a in A)

    index = argmax(Q[(s,a)] + c * bonus(N,Ns,s,a) for a in A)
    return A[index]
end

########################################
function simulateMCTS(mdp,s,N,Q,d,c,V,T,discount)
    A = actions(mdp)

    if d <= 0
        return rollout_smart(mdp, s, heuristic_policy), N, T, Q
    end



    if !haskey(N, (s,first(A)) )

        for a in A
            N[s,a] = 0
            Q[s,a] = 0
        end
        return rollout_smart(mdp, s, heuristic_policy), N, T, Q
    end

    a = explore(mdp,N,s,Q,c)
    sprime , r = @gen(:sp,:r)(mdp,s,a)

    if !haskey(T, (s,a,sprime) )
        T[s,a,sprime] = 0
```

```
    end

    T[(s,a,sprime)] += 1

    q, N, T, Q = simulateMCTS(mdp,sprime,N,Q,d-1,c,V,T,discount)
    q = r + discount * q

    N[s,a] += 1
    Q[s,a] += (q - Q[(s,a)]) / N[(s,a)]
    return q, N, T, Q

end


########################################
function MCTS(mdp, reps)
    s = SA[19,19]
    dis = discount(mdp)
    d = 15
    c = 500
    V = 0
    S = statetype(mdp)
    A = actiontype(mdp)
    N = Dict{Tuple{S, A}, Int}()
    Q = Dict{Tuple{S, A}, Float64}()
    T = Dict{Tuple{S, A, S}, Int}()
    A = actions(mdp)



    for i in 1:reps
        q, N, T, Q = simulateMCTS(mdp,s,N,Q,d,c,V,T,dis)
    end
```

```
    index = argmax(Q[s,a] for a in actions(mdp))

    return Q, N, T, A[index]

end

mdp = DenseGridWorld(seed=4)

Q,N,T,A = MCTS(mdp,7)



inchrome(visualize_tree(Q, N, T, SA[19,19]))
```

C. Question 4

```
    ############
# Question 4
############

########################################
function rollout_smart(mdp, start_state, heuristic_policy, max_steps=100)
    r_total = 0.0
    a = actions(mdp)
    s = start_state
    t = 0
    while !isterminal(mdp, s) && t < max_steps
        a = heuristic_policy(s)
        s, r = @gen(:sp,:r)(mdp, s, a)
        r_total += discount(mdp)^t*r
        t += 1
    end
```

```
    return r_total # replace this with the reward
end

########################################
function rollout_MCTS(mdp, start_state, max_steps=100)
    r_total = 0.0
    a = actions(mdp)
    s = start_state
    t = 0
    while !isterminal(mdp, s) && t < max_steps
        _,_,_,a = MCTS(mdp,s)
        s, r = @gen(:sp,:r)(mdp, s, a)
        r_total += discount(mdp)^t*r
        t += 1
    end
    return r_total # replace this with the reward
end
########################################
function bonus(N, Ns, s, a)

    if N[(s,a)] == 0 || Ns == 0
        return Inf
    end

    return sqrt(log(Ns)/N[(s,a)])
end



########################################

function explore(mdp,N,s,Q,c)
    A = actions(mdp)
    Ns = sum(N[(s,a)] for a in A)
```

```
    index = argmax(Q[(s,a)] + c * bonus(N,Ns,s,a) for a in A)
    return A[index]
end

########################################
function simulateMCTS(mdp,s,N,Q,d,c,V,T,discount)
    A = actions(mdp)

    if d <= 0
        return rollout_smart(mdp, s, heuristic_policy), N, T, Q
    end



    if !haskey(N, (s,first(A)) )

        for a in A
            N[s,a] = 0
            Q[s,a] = 0
        end
        return rollout_smart(mdp, s, heuristic_policy), N, T, Q
    end

    a = explore(mdp,N,s,Q,c)
    sprime , r = @gen(:sp,:r)(mdp,s,a)

    if !haskey(T, (s,a,sprime) )
        T[s,a,sprime] = 0
    end

    T[(s,a,sprime)] += 1

    q, N, T, Q = simulateMCTS(mdp,sprime,N,Q,d-1,c,V,T,discount)
```

```
    q = r + discount * q

    N[s,a] += 1
    Q[s,a] += (q - Q[(s,a)]) / N[(s,a)]
    return q, N, T, Q

end


########################################
function MCTS(mdp, s)
    start = time_ns()
    dis = discount(mdp)
    d = 15
    c = 120
    V = 0
    S = statetype(mdp)
    A = actiontype(mdp)
    N = Dict{Tuple{S, A}, Int}()
    Q = Dict{Tuple{S, A}, Float64}()
    T = Dict{Tuple{S, A, S}, Int}()
    A = actions(mdp)


    count = 0
    while time_ns() < start + 40_000_000
        q, N, T, Q = simulateMCTS(mdp,s,N,Q,d,c,V,T,dis)
        count += 1
    end
    @show count

    index = argmax(Q[s,a] for a in actions(mdp))
```

```
    return Q, N, T, A[index]

end

mdp = DenseGridWorld(seed=4)

# Q,N,T,A = MCTS(mdp, 7)


# This code runs monte carlo simulations: you can calculate the mean and standard error from the results
results_random = [rollout(mdp, rand(initialstate(mdp))) for _ in 1:10000]
mean_random = mean(results_random)
SEM_random = std(results_random)/(length(results_random)^0.5)
@show mean_random
@show SEM_random

results_smart = [rollout_MCTS(mdp, rand(initialstate(mdp))) for _ in 1:100]
mean_smart = mean(results_smart)
SEM_smart = std(results_smart)/(length(results_smart)^0.5)
@show mean_smart
@show SEM_smart


@show difference = mean_smart - mean_random
```

D. Question 5

```
    ############
# Question 5
############

########################################
function rollout_smart(mdp, start_state, heuristic_policy, max_steps=100)
    r_total = 0.0
```

```
    a = actions(mdp)
    s = start_state
    t = 0
    while !isterminal(mdp, s) && t < max_steps
        a = heuristic_policy(s)
        s, r = @gen(:sp,:r)(mdp, s, a)
        r_total += discount(mdp)^t*r
        t += 1
    end
    return r_total # replace this with the reward
end

########################################
function rollout_MCTS(mdp, start_state, max_steps=100)
    r_total = 0.0
    a = actions(mdp)
    s = start_state
    t = 0
    while !isterminal(mdp, s) && t < max_steps
        _,_,_,a = MCTS5(mdp,s)
        s, r = @gen(:sp,:r)(mdp, s, a)
        r_total += discount(mdp)^t*r
        t += 1
    end
    return r_total
end
########################################
function bonus(N, Ns, s, a)

    if N[(s,a)] == 0 || Ns == 0
        return Inf
    end
```

```
    return sqrt(log(Ns)/N[(s,a)])
end


########################################

function explore(mdp,N,s,Q,c)
    A = actions(mdp)
    Ns = sum(N[(s,a)] for a in A)

    index = argmax(Q[(s,a)] + c * bonus(N,Ns,s,a) for a in A)
    return A[index]
end

########################################
function simulateMCTS(mdp,s,N,Q,d,c,V,T,discount)
    A = actions(mdp)

    if d <= 0
        return rollout_smart(mdp, s, heuristic_policy), N, T, Q
    end



    if !haskey(N, (s,first(A)) )

        for a in A
            N[s,a] = 0
            Q[s,a] = 0
        end
        return rollout_smart(mdp, s, heuristic_policy), N, T, Q
    end

    a = explore(mdp,N,s,Q,c)
```

```
    sprime , r = @gen(:sp,:r)(mdp,s,a)

    if !haskey(T, (s,a,sprime) )
        T[s,a,sprime] = 0
    end

    T[(s,a,sprime)] += 1

    q, N, T, Q = simulateMCTS(mdp,sprime,N,Q,d-1,c,V,T,discount)
    q = r + discount * q

    N[s,a] += 1
    Q[s,a] += (q - Q[(s,a)]) / N[(s,a)]
    return q, N, T, Q

end


##########################################
function MCTS5(mdp, s)
    start = time_ns()
    dis = discount(mdp)
    d = 15
    c = 900
    V = 0
    S = statetype(mdp)
    A = actiontype(mdp)
    N = Dict{Tuple{S, A}, Int}()
    Q = Dict{Tuple{S, A}, Float64}()
    T = Dict{Tuple{S, A, S}, Int}()
    A = actions(mdp)
```

```
    while time_ns() < start + 40_000_000
        q, N, T, Q = simulateMCTS(mdp,s,N,Q,d,c,V,T,dis)
    end


    index = argmax(Q[s,a] for a in actions(mdp))

    return A[index]

end

MCTS5(mdp,SA[5,5])

HW3.evaluate(MCTS5, "julian.lemabulliard@colorado.edu")
```