Maggie Jacoby

Homework 3

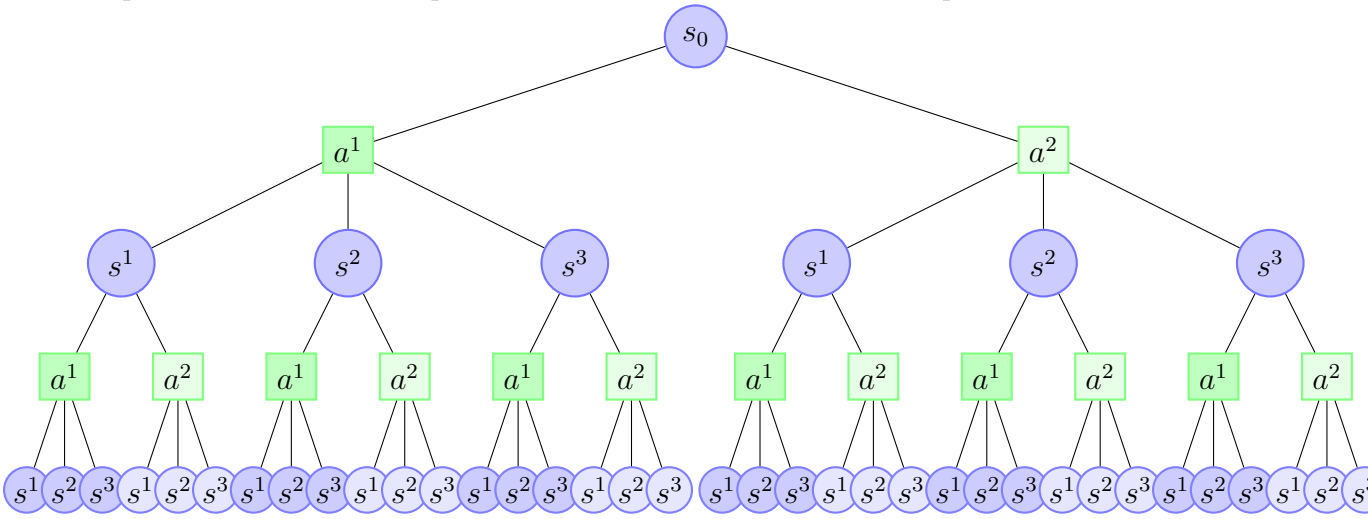
Conceptual Questions

Problem 1: Draw Three Trees

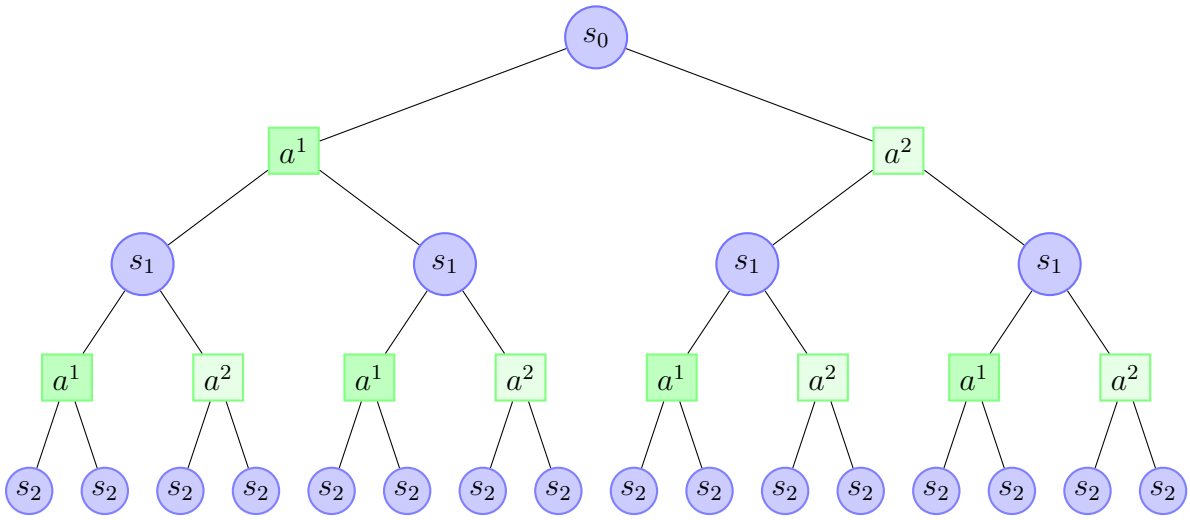
Given
An MDP with three states, $\mathcal{S} = \{s^1, s^2, s^3\}$, and two actions, $\mathcal{A} = \{a^1, a^2\}$.

Draw

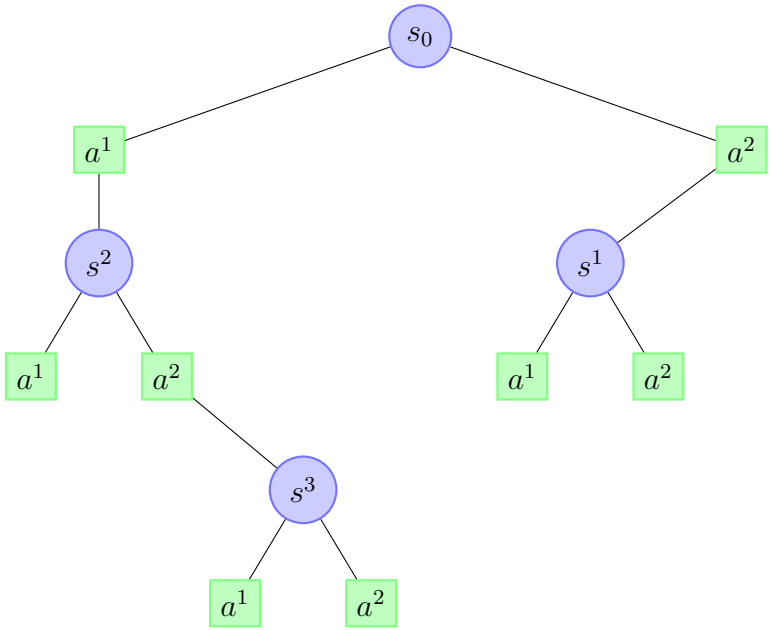
(a) The complete state-action tree produced with forward search to a depth of $d = 2$.



(b) A sparse sampling tree with $n = 2$ to a depth of $d = 2$.



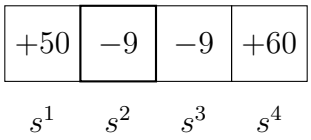
(c) A partial tree after four iterations of Monte Carlo Tree Search to a depth of $d = 2$.



Problem 2: Sparse Sampling Counter Example

Show
An MDP and policy that is a counterexample to the claim:
If a policy π satisfies $|Q^*(s, \pi^*(s)) - Q^*(s, \pi(s))| \leq \beta$ for all $s \in \mathcal{S}$, then it immediately follows that $|R(s, \pi^*(s)) - R(s, \pi(s))| \leq \beta|$.

Solution
Consider an un-discounted 4x1 grid world, where the left and right edges are terminal states that contain rewards of +50 and +60, respectively, and the two center cells both have a cost of -9. Let the actions be completely deterministic, and the rewards be based only on the cell you enter. The problem can be represented as so:



- $\mathcal{S} = \{s^1, s^2, s^3, s^4\}$
- $\mathcal{A} = \{-1, +1\}$
- $\mathcal{T}(s'|s, a) = 1$
- $\mathcal{R}(s, a, s') = s'$
- $\gamma = 1$
- Let $\pi(s)$ be a policy that tells you to move into the closest edge cell.

Assume that you start in cell s^2. The optimal policy would say to move right two spaces, which would then terminate with a total accumulated reward of +51, while the highest immediate reward would be found by following the "move towards the closest edge" policy, which would move left one space, then terminate with a total accumulated reward of +50.

$$|Q^*(s, \pi^*(s)) - Q^*(s, \pi(s))| = |51 - 50| = 1 = \beta$$
$$|R(s, \pi^*(s)) - R(s, \pi(s))| = |-9 - 50| = 59 > \beta$$

Challenge Problem

Problem 3: On-line Planner for Dense Grid Worlds

Solution submitted to leaderboard.

```
using DMUStudent.HW3
using DMUStudent
using POMDPs
using POMDPPolicies
using MCTS

## Function policy for rollouts
# Uses matrix A developed below
my_policy = FunctionPolicy(
    function pi(s)
        return A[s[2],s[1]]
    end
);

## Returns estimate of value
# non-reward value based on averages observed for non-reward states
function Q2_est(m, s, a)
    s in keys(m.rewards) ? 100.0 : -6.0
end


function solver_explore(c=150.0,  d=30, n=100)
    println("n: $n, d: $d, c: $c")
    solver_n = MCTSSolver(
        n_iterations = n,                   # default 100
        max_time = 0.04,                    # default Inf
        depth = d,                          # default 10
        exploration_constant = c,           # default 1.0
        init_Q = Q2_est,
        init_N = 100,
        estimate_value = RolloutEstimator(my_policy),
        reuse_tree = true                   # default false
    );
    return solver_n
end

my_solver = solver_explore()
submit(my_solver, "hw3", "maja8167@colorado.edu", nickname="...")


evaluate(my_solver, "hw3")

### Some of the parameters explored
n_range = (5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70)
d_range = (5, 10, 15, 20, 30)
c_range = (50.0, 60.0, 70.0, 80.0, 90.0, 100.0, 120.0, 150.0)

for c in c_range
    my_solver = solver_explore(c)
    evaluate(my_solver, "hw3")
end
```

```
### Matrix for derermining optimal actions
#   Run once before running solvers

A = Array{Symbol}(undef, (100,100))
for i=1:10
    A[:,i] = fill(:right, (100,1))
end
for i=90:100
    A[:,i] = fill(:left, (100,1))
end
for i=11:89
    if i%20 >= 10
        A[:,i] = fill(:right, (100,1))
    elseif i%20 >= 1
        A[:,i] = fill(:left, (100,1))
    else
        for j=1:10
            A[j,i] = :up
        end
        for j=90:100
            A[j,i] = :down
        end
        for j=11:89
            if j%20 >= 10
                A[j,i] = :up
            elseif j%20 >= 1
                A[j,i] = :down
            else
                A[i,j] = :up
            end
        end
    end
end

### Function used to determine mean non-reward cost
# averages seeemed to be around -6 for most random worlds (-6 used above)
function mean_cost(m)
   r_tot = 0.0
   for i = 1:100, j = 1:100
       r_tot += reward(m, GWPos(i,j))
   end
   return (r_tot-1600)/9984
end
```