Online Methods

Last Time

- Does value iteration always converge?
- Is the value function unique?

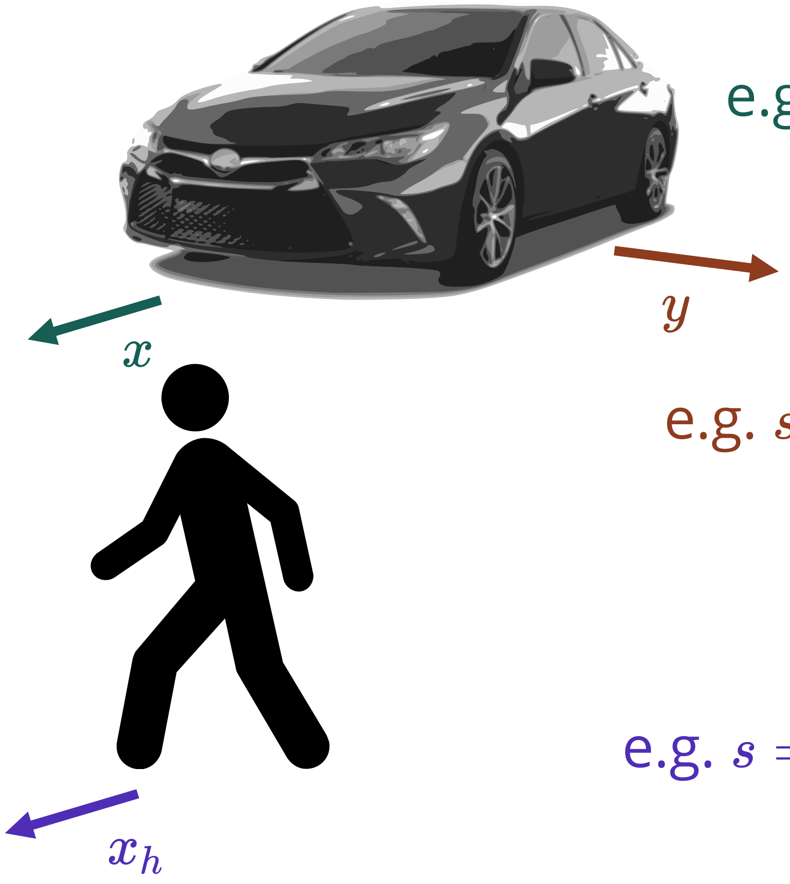
Guiding Questions

- What are the differences between *online* and *offline* solutions?
- Are there solution techniques that require computation time *independent* of the state space size?

Why Do We Need Something Else?

- Problems Policy and Value Iteration may struggle with?
  - Path planning across the country, or interplanetary
  - More realistic car dynamics (continuous states)
- Why are these problems hard?
  - State Space is massive (or infinite)

Curse of Dimensionality



1 dimension

e.g. $s = x \in S = \{1, 2, 3, 4, 5\}$

$|S| = 5$

2 dimensions

e.g. $s = (x, y) \in S = \{1, 2, 3, 4, 5\}^2$

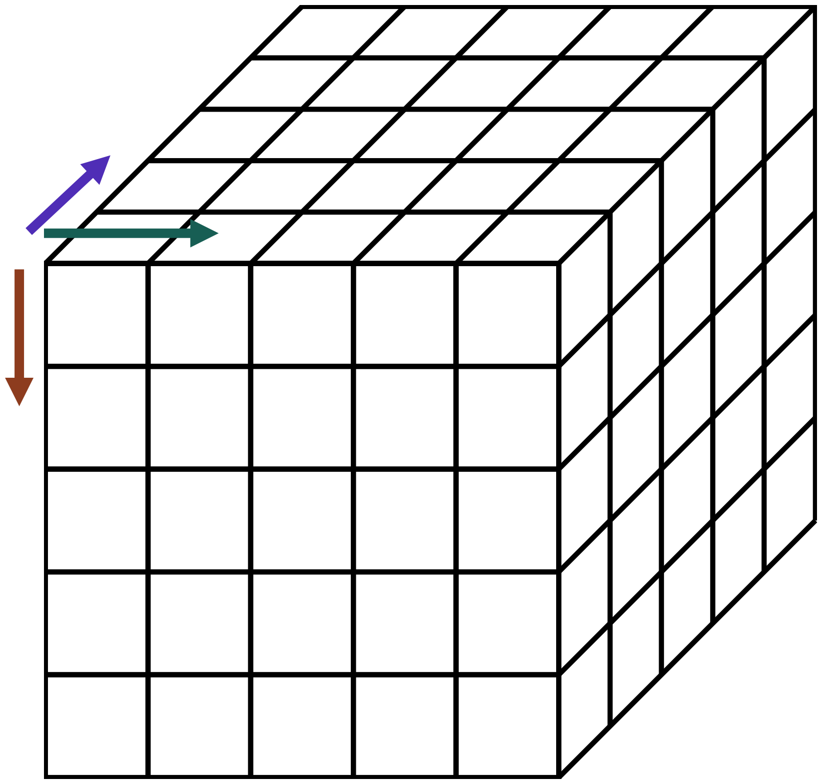
$|S| = 25$

3 dimensions

e.g. $s = (x, y, x_h) \in S = \{1, 2, 3, 4, 5\}^3$

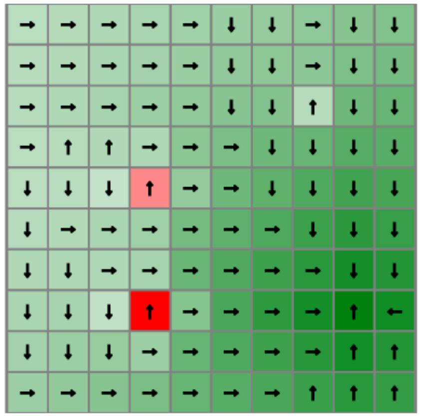
$|S| = 125$

(Discretize each dimension into 5 segments)



d dimensions, k segments $\rightarrow |S| = k^d$

Offline vs Online Solutions

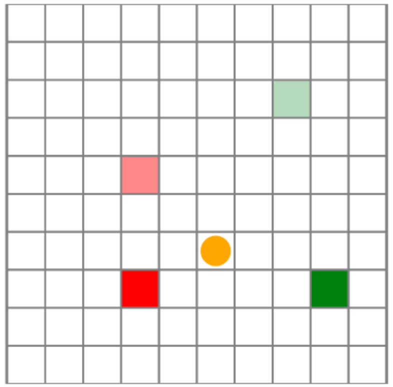


<u>Offline</u>

- Before Execution: find V^*/Q^*
- During Execution: $\pi^*(s) = \operatorname{argmax} Q^*(s, a)$

<u>Online</u>

- Before Execution: <nothing>
- During Execution: Consider actions and their consequences (everything)



- Why?
- Online methods are insensitive to the size of S !

```julia
struct MDP
    γ  # discount factor
    𝒮  # state space
    𝒜  # action space
    T  # transition function
    R  # reward function
    TR # sample transition and reward
end
```

Rollout Lookahead

```julia
struct RolloutLookahead
    𝒫 # problem
    π # rollout policy
    d # depth
end

randstep(𝒫::MDP, s, a) = 𝒫.TR(s, a)

function rollout(𝒫, s, π, d)
    ret = 0.0
    for t in 1:d
        a = π(s)
        s, r = randstep(𝒫, s, a)
        ret += 𝒫.γ^(t-1) * r
    end
    return ret
end

function (π::RolloutLookahead)(s)
    U(s) = rollout(π.𝒫, s, π.π, π.d)
    return greedy(π.𝒫, U, s).a
end
```

```julia
function greedy(𝒫::MDP, U, s)
    u, a = findmax(a->lookahead(𝒫, U, s, a), 𝒫.𝒜)
    return (a=a, u=u)
end
```

```julia
function lookahead(𝒫::MDP, U, s, a)
    𝒮, T, R, γ = 𝒫.𝒮, 𝒫.T, 𝒫.R, 𝒫.γ
    return R(s,a) + γ*sum(T(s,a,s′)*U(s′) for s′ in 𝒮)
```

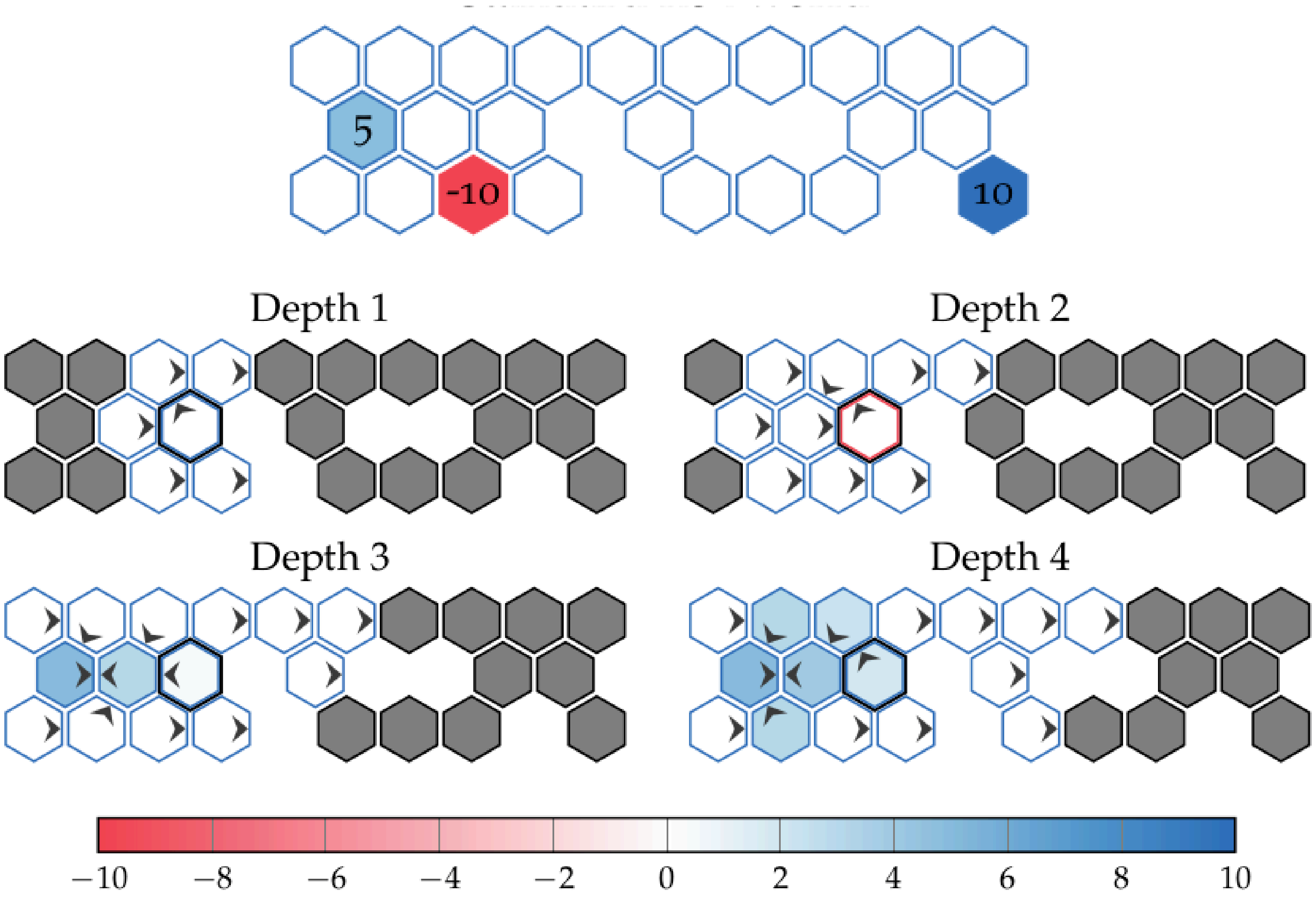
Forward Search

Algorithm 4.6 Forward search

```
function SELECTACTION(s, d)
    if d = 0
        return (NIL, 0)
    (a*, v*) ← (NIL, −∞)
    for a ∈ A(s)
        v ← R(s, a)
        for s′ ∈ S(s, a)
            (a′, v′) ← SELECTACTION(s′, d − 1)
            v ← v + γ T(s′ | s, a) v′
        if v > v*
            (a*, v*) ← (a, v)
    return (a*, v*)
```

$$(|S| \times |A|)^d$$

Forward Search depth



Sparse Sampling

Algorithm 4.8 Sparse sampling

```
function SelectAction(s, d)
    if d = 0
        return (NIL, 0)
    (a*, v*) ← (NIL, −∞)
    for a ∈ A(s)
        v ← 0
        for i ← 1 to n
            (s′, r) ∼ G(s, a)
            (a′, v′) ← SelectAction(s′, d − 1)
            v ← v + (r + γv′)/n
        if v > v*
            (a*, v*) ← (a, v)
    return (a*, v*)
```

$(n|A|)^d$ $|V^{\text{SS}}(s) - V^*(s)| \leq \epsilon$ n, ϵ, and d related, but independent of $|S|$

https://www.cis.upenn.edu/~mkearns/papers/sparsesampling-journal.pdf

Break

Draw the trees produced by the following algorithms for a problem with 2 actions and 3 states:

1. One-step lookahead with rollout
2. Forward search (d=2)
3. Sparse sampling (d=2, n=2)

Monte Carlo Tree Search (MCTS/UCT)

<u>Keep track of:</u>

$Q(s,a)$: Value estimate of that state and action combo

$N(s,a)$: Number of times we visit a state and action combo

$$Q(s,a) + c\sqrt{\frac{\log N(s)}{N(s,a)}} \qquad Q(s,a) + c\frac{N(s)^{\beta}}{\sqrt{N(s,a)}}$$

low $N(s,a)/N(s)$ = high bonus

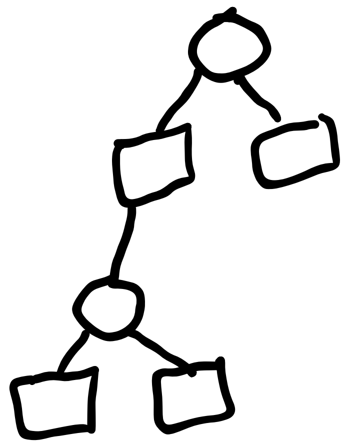
start with $c = 2(\bar{V} - \underline{V})$, $\beta = 1/4$

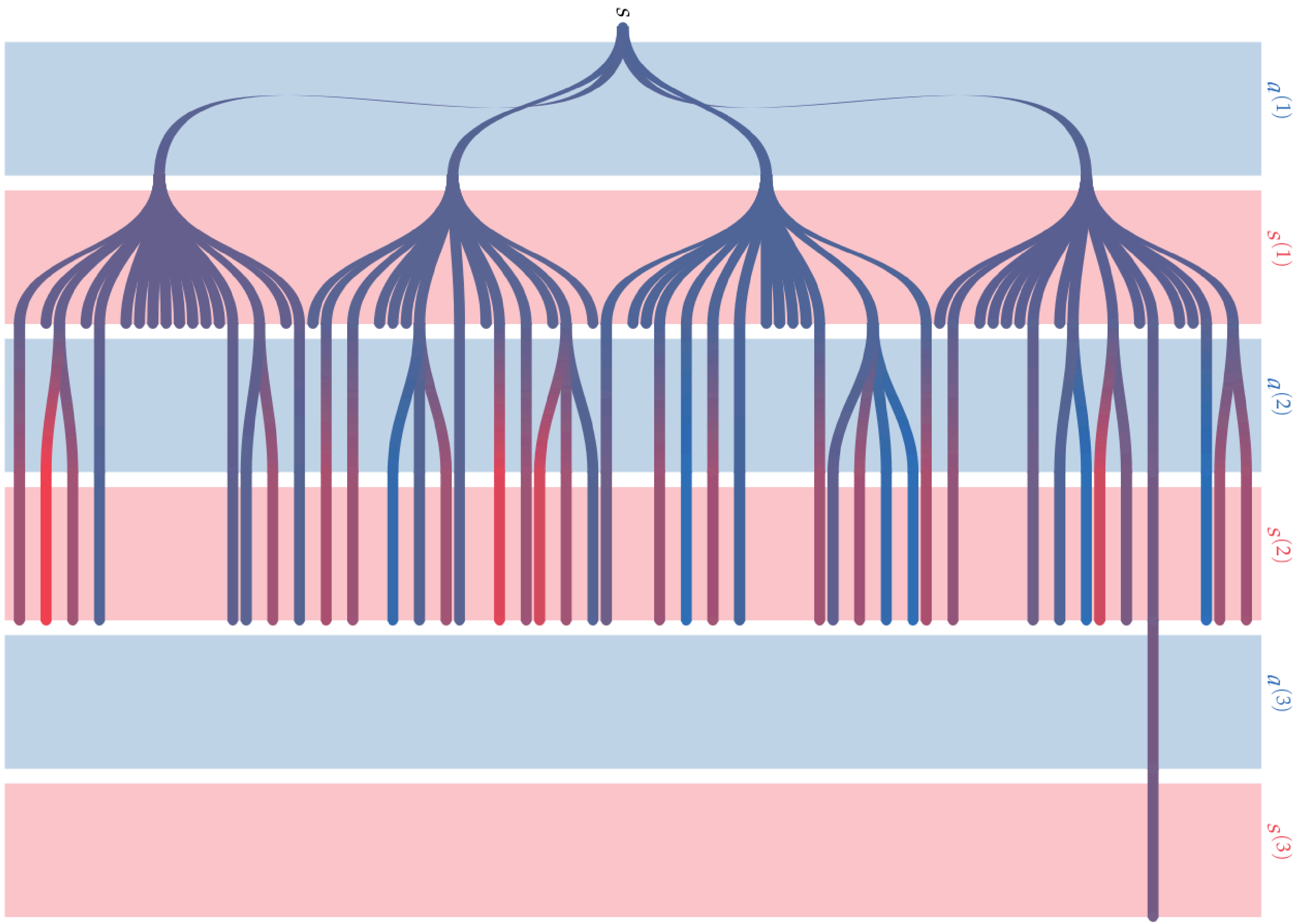
Full story can be found in https://arxiv.org/pdf/1902.05213.pdf

Monte Carlo Tree Search (MCTS/UCT)

```
function (π::MonteCarloTreeSearch)(s)
    for k in 1:π.m
        simulate!(π, s)
    end
    return argmax(a->π.Q[(s,a)], π.𝒫.𝒜)
end
```

```
function simulate!(π::MonteCarloTreeSearch, s, d=π.d)
    if d ≤ 0
        return π.U(s)
    end
    𝒫, N, Q, c = π.𝒫, π.N, π.Q, π.c
    𝒜, TR, γ = 𝒫.𝒜, 𝒫.TR, 𝒫.γ
    if !haskey(N, (s, first(𝒜)))
        for a in 𝒜
            N[(s,a)] = 0
            Q[(s,a)] = 0.0
        end
        return π.U(s)
    end
    a = explore(π, s)
    s′, r = TR(s,a)
    q = r + γ*simulate!(π, s′, d-1)
    N[(s,a)] += 1
    Q[(s,a)] += (q-Q[(s,a)])/N[(s,a)]
    return q
end
```





s
a(1)
s(1)
a(2)
s(2)
a(3)
s(3)

Using Online Methods in a Simulation

Algorithm: Rollout Simulation

Given: MDP (S, A, R, T, γ, b)

$s \leftarrow \text{sample}(b)$

$\hat{u} \leftarrow 0$

for t in $0 \ldots T-1$

    $a \leftarrow \pi(s)$

    $s', r \leftarrow G(s, a)$

    $\hat{u} \leftarrow \hat{u} + \gamma^t r$

    $s \leftarrow s'$

return $\hat{u}$

Guiding Questions

- What are the differences between online and offline solutions?
- Are there solution techniques that are *independent* of the state space size?

Forward Search Sparse Sampling

(FSSS)

Paper: https://cdn.aaai.org/ojs/7689/7689-13-11219-1-2-20201228.pdf

- Sparse Sampling, but only look at potentially valuable states

<u>Things it keeps track of:</u>

$Q(s, a)$: Estimate of the value for the state action pair

$U(s)$: Upper bound for value of state s

$L(s)$: Lower bound for value of state s

$U(s, a)$: Upper bound for value of state-action

$L(s, a)$: Lower bound for value of state-action

Forward Search Sparse Sampling

Algorithm 3 FSSS(s, d)

```
if d = 1 (leaf) then
    L^d(s,a) = U^d(s,a) = R(s,a), ∀a
    L^d(s) = U^d(s) = max_a R(s,a)
else if n_sd = 0 then
    for each a ∈ A do
        L^d(s,a) = V_min
        U^d(s,a) = V_max
        for C times do
            s' ~ T(s,a,·)
            L^{d-1}(s') = V_min
            U^{d-1}(s') = V_max
            K^d(s,a) = K^d(s,a) ∪ {s'}
a* = argmax_a U^d(s,a)
s* = max_{s'∈K^d(s,a*)} (U^{d-1}(s') − L^{d-1}(s'))
FSSS(s*, d − 1)
n_sd = n_sd + 1
L^d(s,a*) = R(s,a*) + γ Σ_{s'∈K^d(s,a*)} L^{d-1}(s')/C
U^d(s,a*) = R(s,a*) + γ Σ_{s'∈K^d(s,a*)} U^{d-1}(s')/C
L^d(s) = max_a L^d(s,a)
U^d(s) = max_a U^d(s,a)
```

If $L(s, a*) \geq \max_{a \neq a^*} U(s,a)$ for best action ($a^* = \arg\max_a U(s,a)$):
then, the node is closed because the best action is found.