

Online Methods

Last Time

Last Time

- Does value iteration always converge?
- Is the ^{optimal} value function unique?

Guiding Questions

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?
- Why are these problems hard?

Why Do We Need Something Else?

- Problems Policy and Value Iteration may struggle with?
 - Path planning across the country, or interplanetary
- Why are these problems hard?

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?

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



Curse of Dimensionality

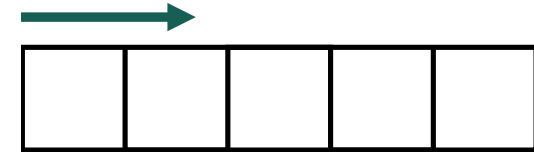


1 dimension

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

$$|S| = 5$$

(Discretize each dimension
into 5 segments)



Curse of Dimensionality



1 dimension

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

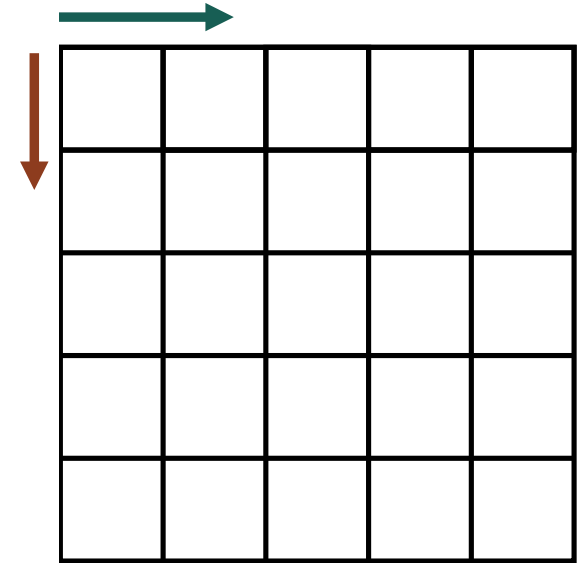
$$|S| = 5$$

(Discretize each dimension
into 5 segments)

2 dimensions

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

$$|S| = 25$$



Curse of Dimensionality

1 dimension

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

$$|S| = 5$$

2 dimensions

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

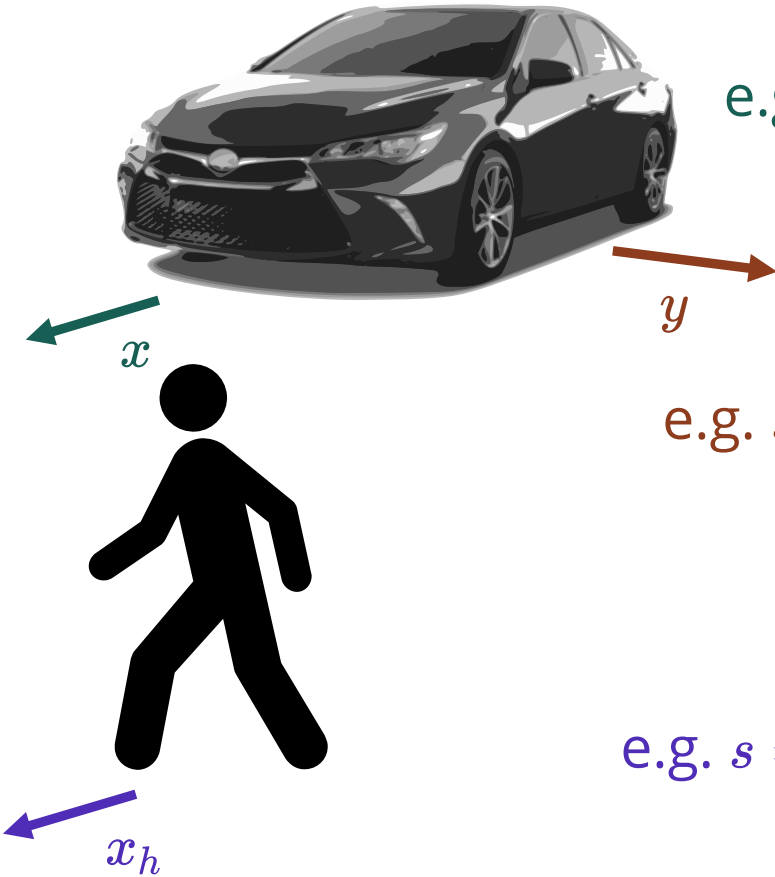
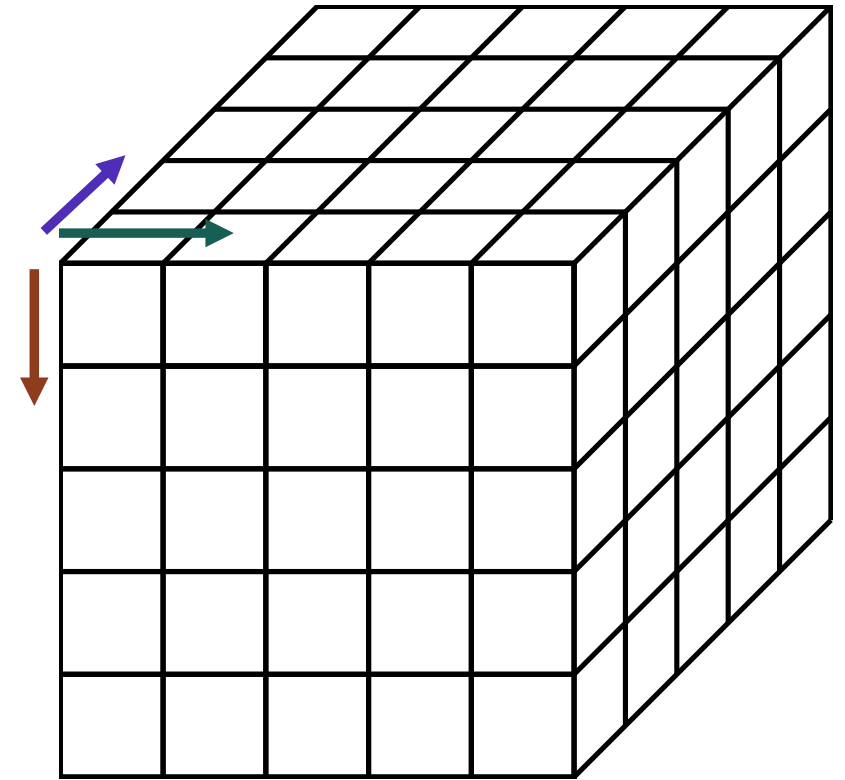
$$|S| = 25$$

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)



Curse of Dimensionality

1 dimension

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

$$|S| = 5$$

2 dimensions

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

$$|S| = 25$$

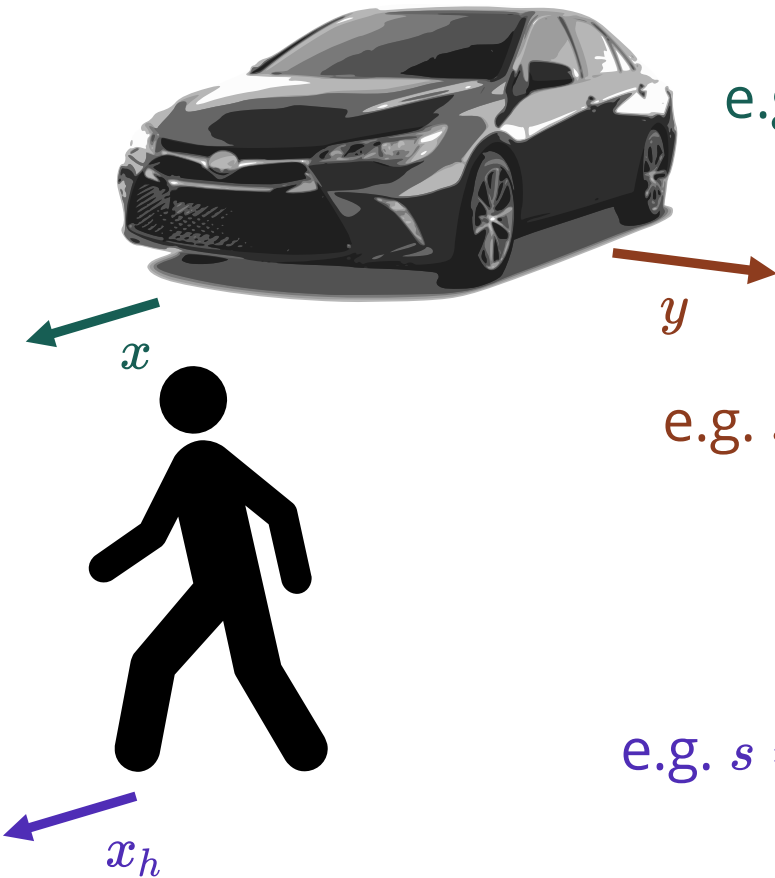
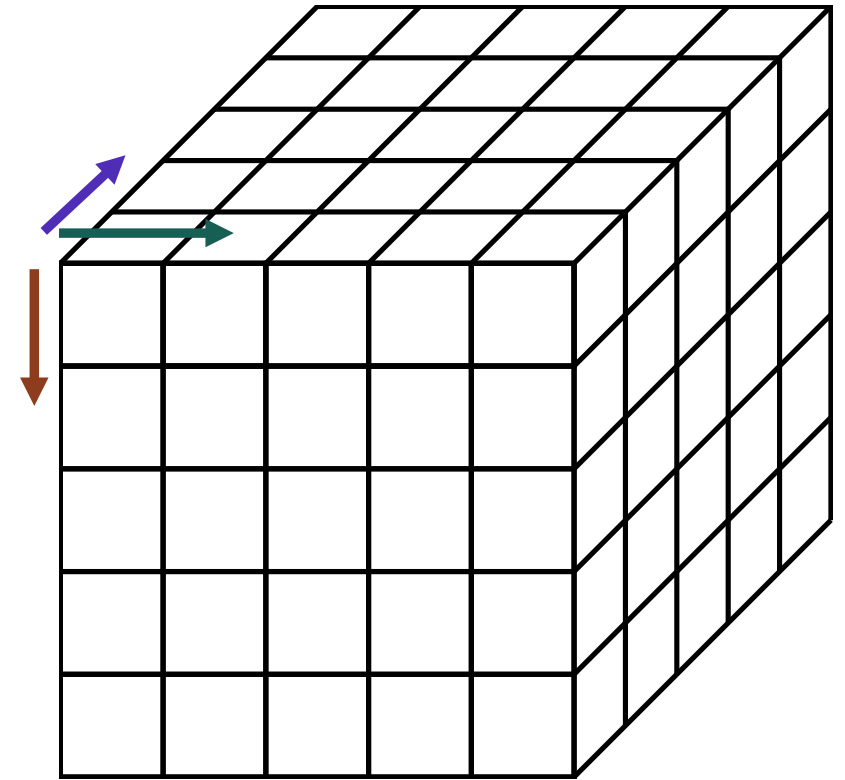
3 dimensions

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

$$|S| = 125$$

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

(Discretize each dimension into 5 segments)



Offline vs Online Solutions

Offline

Online

Offline vs Online Solutions

Offline

- Before Execution: find V^*/Q^*

Online

Offline vs Online Solutions

Offline

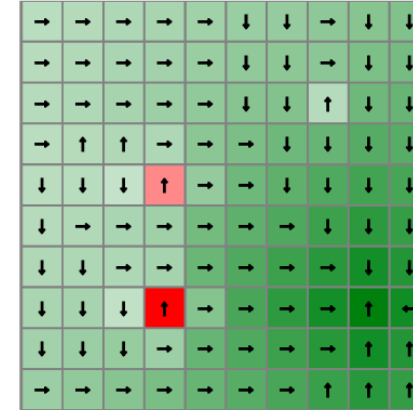
- Before Execution: find V^*/Q^*
- During Execution: $\pi^*(s) = \operatorname{argmax} Q^*(s, a)$

Online

Offline vs Online Solutions

Offline

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

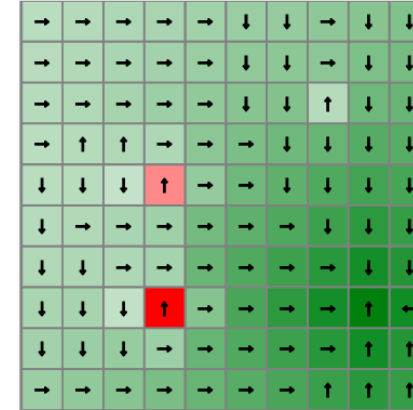


Online

Offline vs Online Solutions

Offline

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



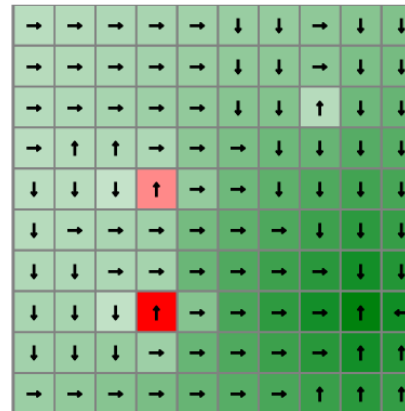
Online

- Before Execution: <nothing>

Offline vs Online Solutions

Offline

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



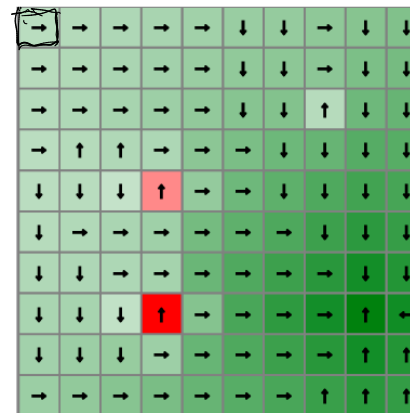
Online

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

Offline vs Online Solutions

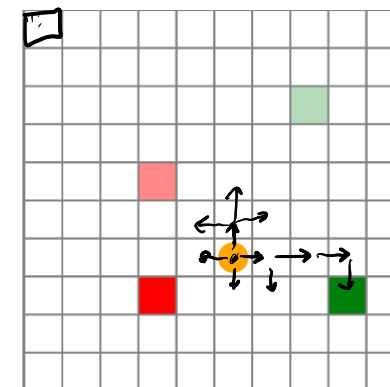
Offline

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



Online

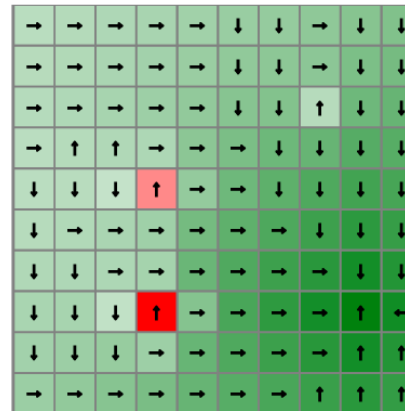
- Before Execution: <nothing>
- During Execution: Consider actions and their consequences (everything)



Offline vs Online Solutions

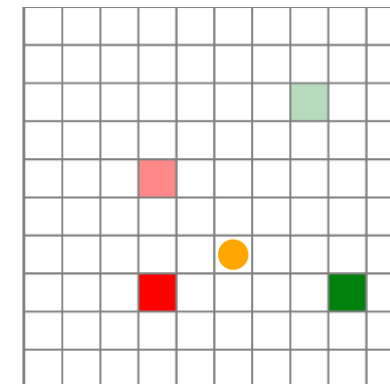
Offline

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



Online

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

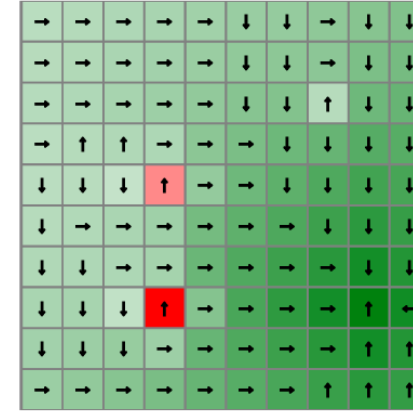


- Why?

Offline vs Online Solutions

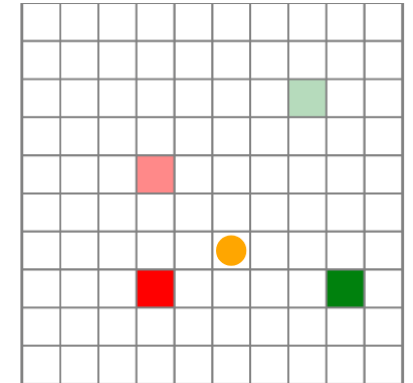
Offline

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



Online

- Before Execution: <nothing>
- During Execution: Consider actions and their consequences (everything)
- Why?
- Online methods are insensitive to the size of S !

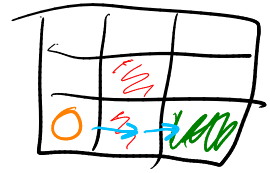
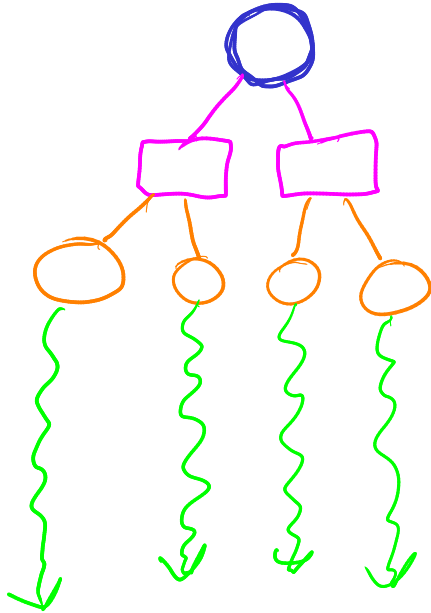


Rollout Lookahead

```

struct MDP
  γ # discount factor
  S # state space
  A # action space
  T # transition function
  R # reward function
  TR # sample transition and reward
end
 $s', r \leftarrow G(s, a)$ 

```



```

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) U(s)
  U(s) = rollout(π.P, s, π.π, π.d)
  return greedy(π.P, U, s).a
end

```

```

function greedy(P::MDP, U, s)
  u, a = findmax(a → lookahead(P, U, s, a), P.A)
  return (a=a, u=u)
end

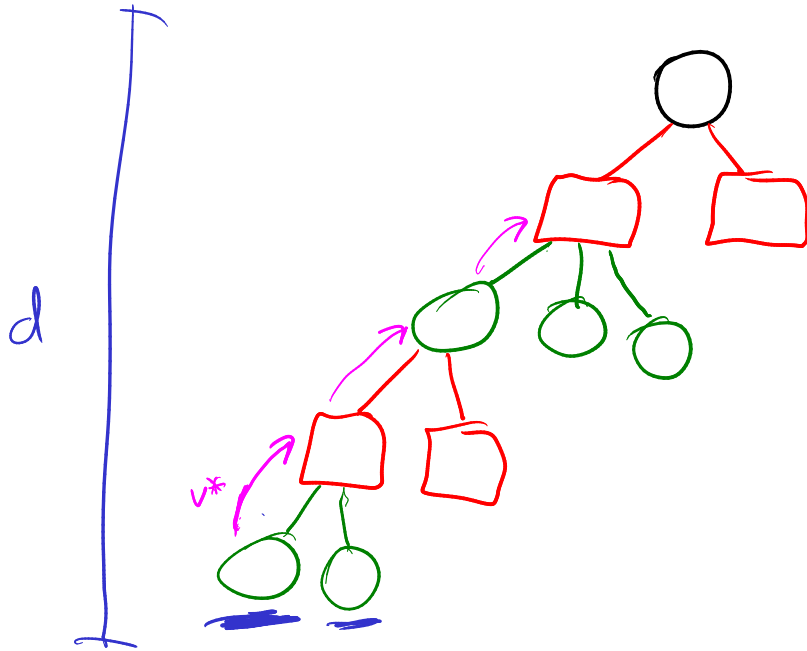
```

```

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)
end

```

Forward Search



Algorithm 4.6 Forward search

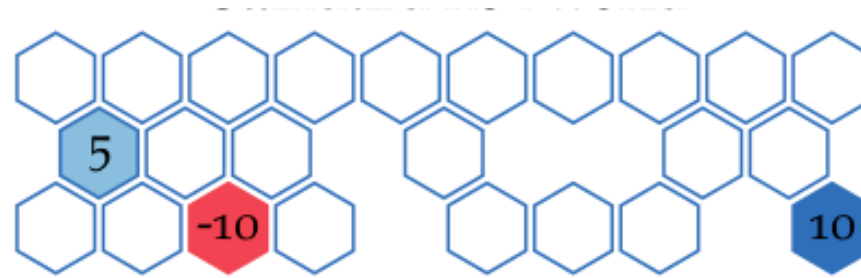
```

1: function SELECTACTION( $s$ ,  $d$ )
2:   if  $d = 0$ 
3:     return (NIL, 0)
4:    $(a^*, v^*) \leftarrow (\text{NIL}, -\infty)$ 
5:   for  $a \in A(s)$ 
6:      $v \leftarrow R(s, a)$ 
7:     for  $s' \in S(s, a)$ 
8:        $(a', v') \leftarrow \text{SELECTACTION}(s', d - 1)$ 
9:        $v \leftarrow v + \gamma T(s' | s, a) v'$ 
10:    if  $v > v^*$ 
11:       $(a^*, v^*) \leftarrow (a, v)$ 
12:  return  $(a^*, v^*)$ 
  
```

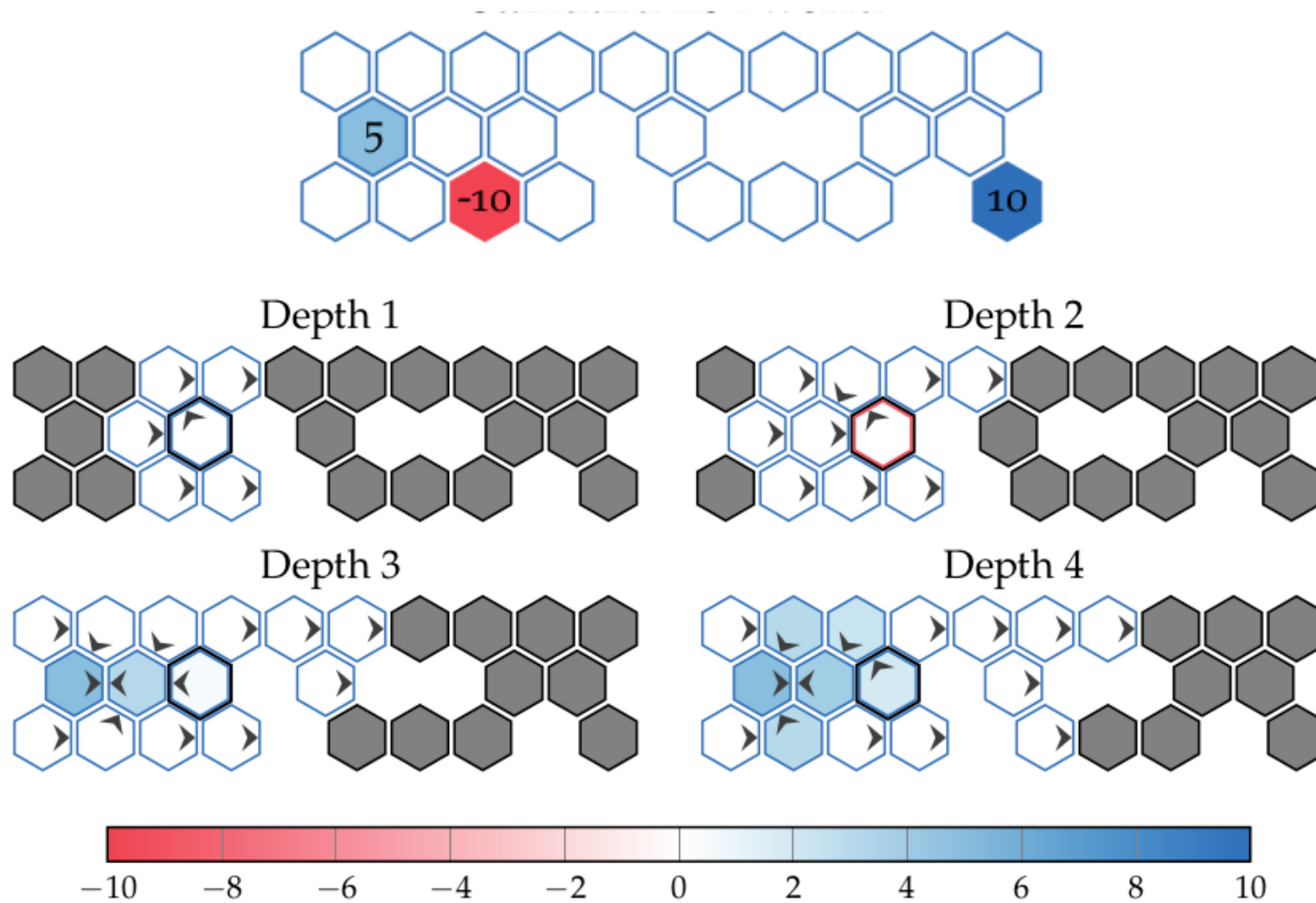
$$\underline{(|S| \times |A|)^d}$$

Forward Search depth

Forward Search depth

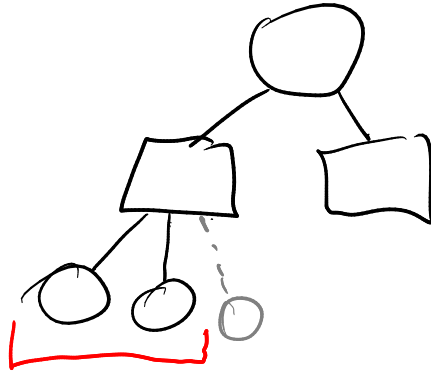


Forward Search depth



Sparse Sampling

$n = 2$



Algorithm 4.8 Sparse sampling

```

1: function SELECTACTION( $s, d$ )
2:   if  $d = 0$ 
3:     return (NIL, 0)
4:    $(a^*, v^*) \leftarrow (\text{NIL}, -\infty)$ 
5:   for  $a \in A(s)$ 
6:      $v \leftarrow 0$ 
7:     for  $i \leftarrow 1$  to  $n$ 
8:        $(s', r) \sim G(s, a)$ 
9:        $(a', v') \leftarrow \text{SELECTACTION}(s', d - 1)$ 
10:       $v \leftarrow v + (r + \gamma v') / n$ 
11:   if  $v > v^*$ 
12:      $(a^*, v^*) \leftarrow (a, v)$ 
13:   return  $(a^*, v^*)$ 
  
```

$(n|A|)^d$
Size of tree

$$\underbrace{|V^{\text{SS}}(s) - V^*(s)| \leq \epsilon}$$

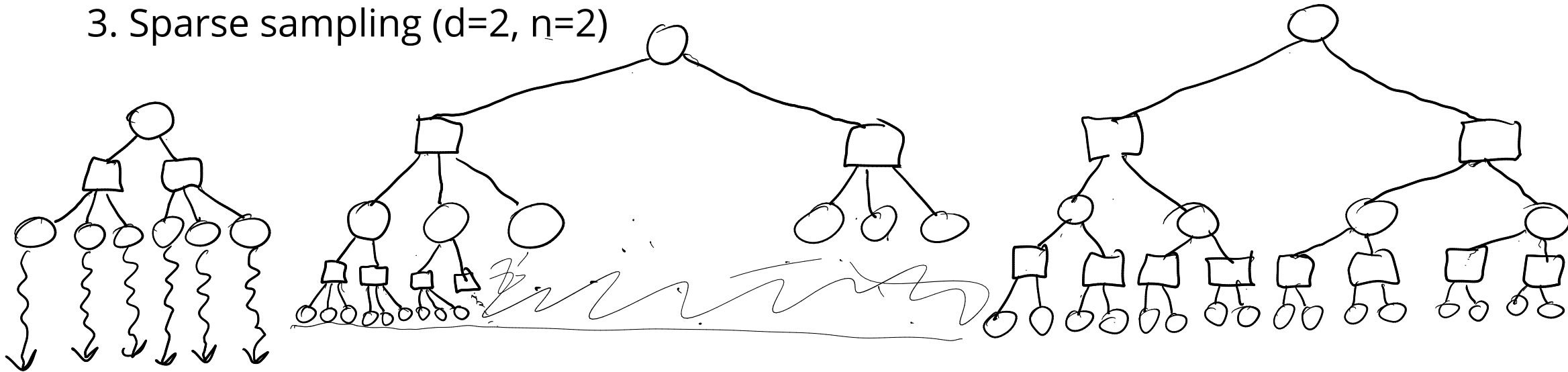
n, ϵ , and d related, but independent of $|S|$

<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)

Keep track of:

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

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

Monte Carlo Tree Search (MCTS/UCT)

Keep track of:

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

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

$$\underbrace{Q(s, a)} + \underbrace{c \sqrt{\frac{\log N(s)}{N(s, a)}}}_{\text{Exploration}}$$

Monte Carlo Tree Search (MCTS/UCT)

Keep track of:

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

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

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

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

Monte Carlo Tree Search (MCTS/UCT)

Keep track of:

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

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

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

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

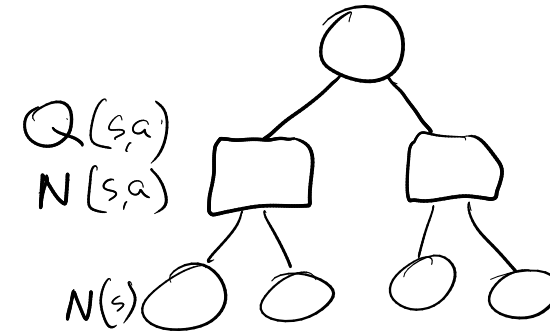
start with $c = 2(\bar{V} - \underline{V})$, $\beta = 1/4$

Monte Carlo Tree Search (MCTS/UCT)

Keep track of:

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

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



Original

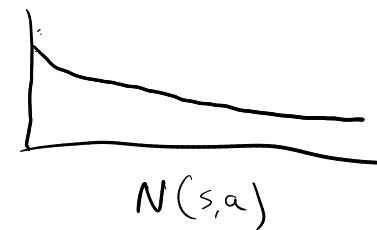
$$Q(s, a) + c \sqrt{\frac{\log N(s)}{N(s, a)}}$$

New PolyUCT

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

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)

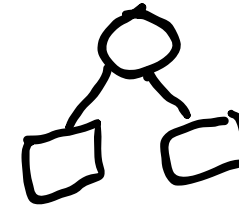
```
function ( $\pi$ ::MonteCarloTreeSearch)(s)
  for k in 1: $\pi$ .m
    simulate!( $\pi$ , s)
  end
  return argmax(a $\rightarrow$  $\pi$ .Q[(s,a)],  $\pi$ . $\mathcal{P}$ . $\mathcal{A}$ )
end
```

```
function simulate!( $\pi$ ::MonteCarloTreeSearch, s, d= $\pi$ .d)
  if d  $\leq$  0
    return  $\pi$ .U(s)
  end
   $\mathcal{P}$ , N, Q, c =  $\pi$ . $\mathcal{P}$ ,  $\pi$ .N,  $\pi$ .Q,  $\pi$ .c
   $\mathcal{A}$ , TR,  $\gamma$  =  $\mathcal{P}$ . $\mathcal{A}$ ,  $\mathcal{P}$ .TR,  $\mathcal{P}$ . $\gamma$ 
  if !haskey(N, (s, first( $\mathcal{A}$ )))
    for a in  $\mathcal{A}$ 
      N[(s,a)] = 0
      Q[(s,a)] = 0.0
    end
    return  $\pi$ .U(s)
  end
  a = explore( $\pi$ , s)
  s', r = TR(s,a)
  q = r +  $\gamma$ *simulate!( $\pi$ , s', d-1)
  N[(s,a)] += 1
  Q[(s,a)] += (q-Q[(s,a)])/N[(s,a)]
  return q
end
```

Monte Carlo Tree Search (MCTS/UCT)

```
function ( $\pi$ ::MonteCarloTreeSearch)(s)
  for k in 1: $\pi$ .m
    simulate!( $\pi$ , s)
  end
  return argmax(a $\rightarrow$  $\pi$ .Q[(s,a)],  $\pi$ . $\mathcal{P}$ . $\mathcal{A}$ )
end
```

```
function simulate!( $\pi$ ::MonteCarloTreeSearch, s, d= $\pi$ .d)
  if d  $\leq$  0
    return  $\pi$ .U(s)
  end
   $\mathcal{P}$ , N, Q, c =  $\pi$ . $\mathcal{P}$ ,  $\pi$ .N,  $\pi$ .Q,  $\pi$ .c
   $\mathcal{A}$ , TR,  $\gamma$  =  $\mathcal{P}$ . $\mathcal{A}$ ,  $\mathcal{P}$ .TR,  $\mathcal{P}$ . $\gamma$ 
  if !haskey(N, (s, first( $\mathcal{A}$ )))
    for a in  $\mathcal{A}$ 
      N[(s,a)] = 0
      Q[(s,a)] = 0.0
    end
    return  $\pi$ .U(s)
  end
  a = explore( $\pi$ , s)
  s', r = TR(s,a)
  q = r +  $\gamma$ *simulate!( $\pi$ , s', d-1)
  N[(s,a)] += 1
  Q[(s,a)] += (q-Q[(s,a)])/N[(s,a)]
  return q
end
```



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
```

m iterations

```

function simulate!( $\pi$ : MonteCarloTreeSearch, s, d= $\pi$ .d)
    if  $d \leq 0$ 
        return  $\pi.U(s)$ 
    end
     $\mathcal{P}, N, Q, c = \pi.\mathcal{P}, \pi.N, \pi.Q, \pi.c$ 
     $\mathcal{A}, TR, \gamma = \mathcal{P}.\mathcal{A}, \mathcal{P}.TR, \mathcal{P}.\gamma$ 
    if !haskey( $N$ , (s, first( $\mathcal{A}$ )))
        for a in  $\mathcal{A}$ 
             $\rightarrow N[(s,a)] = 0$ 
             $\rightarrow Q[(s,a)] = 0.0$ 
        end
    end
    return  $\pi.U(s)$ 
end

s,a) end
a = explore( $\pi, s$ )
 $s', r = TR(s,a)$ 
q = r +  $\gamma * \text{simulate!}(\pi, s', d-1)$ 
 $N[(s,a)] += 1$ 
 $Q[(s,a)] += (q - Q[(s,a)]) / N[(s,a)]$ 
return q
end

```

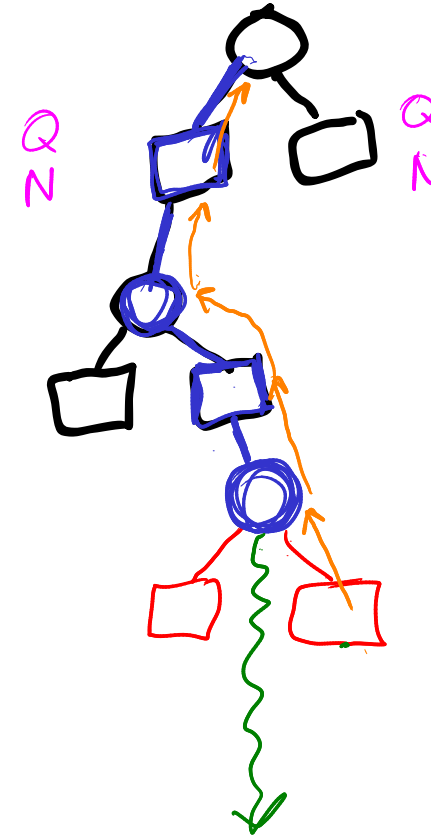
Handwritten annotations on the right side of the code:

- 2. expansion (in red, pointing to the expansion of new nodes)
- 3. Rollout (in blue, pointing to the simulation step)
- 4. Backup (in orange, pointing to the backup step)

Handwritten notes on the left side of the code:

- $s,a)$ (in black, next to the end of the function)
- $Q(s,a) + c \sqrt{\frac{\log N(s)}{N(s)}}$ (in green, next to the exploration formula)

want $U(s)$
rollout gives a sample
estimate of U



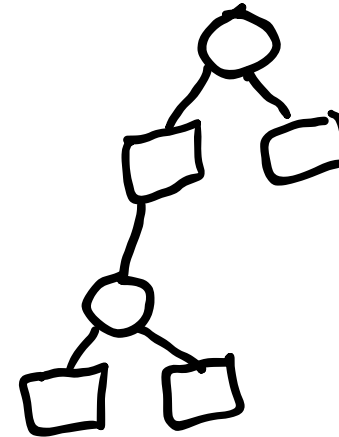
Monte Carlo Tree Search (MCTS/UCT)

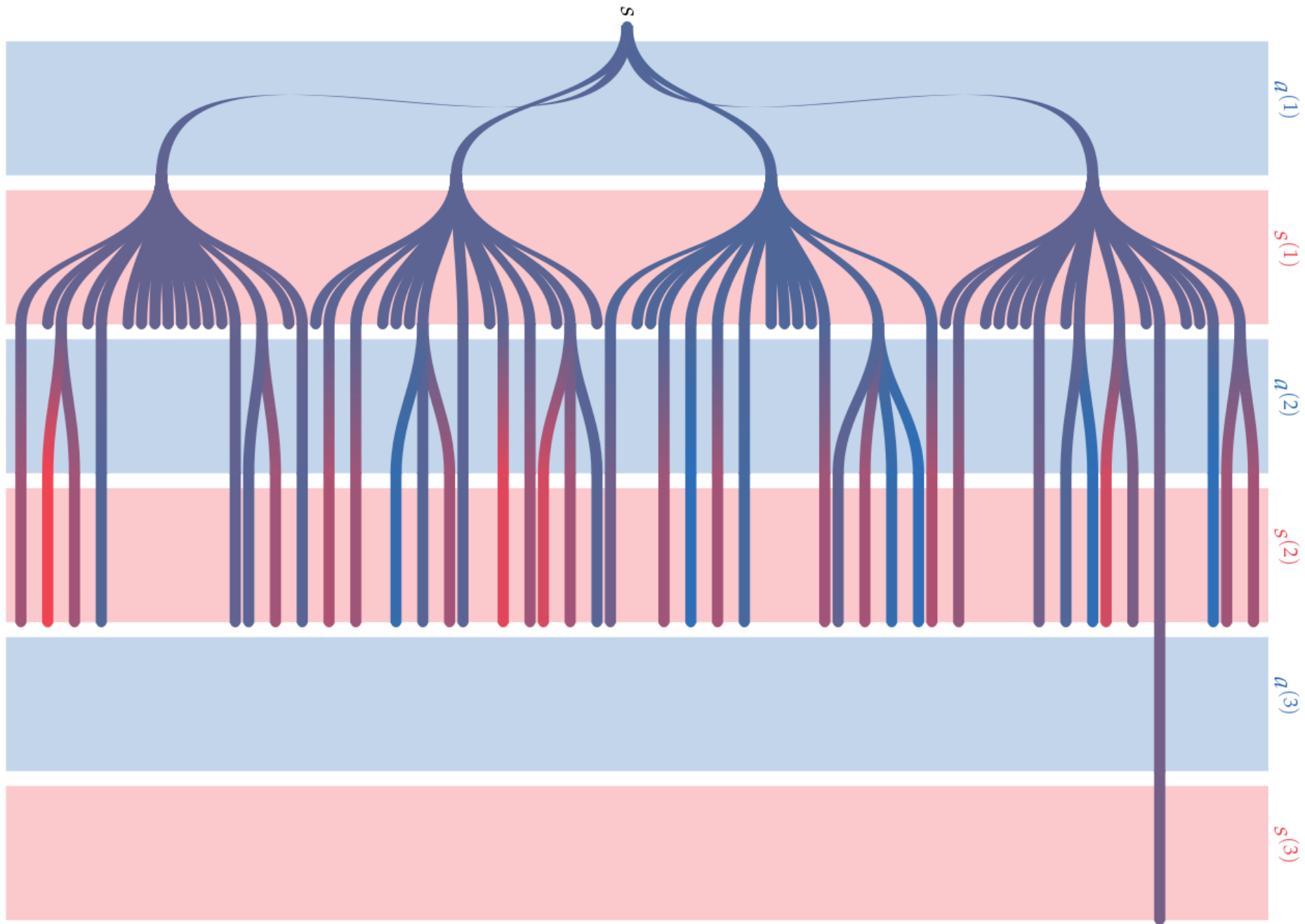
Algorithm 4.9 Monte Carlo tree search

```
1: function SELECTACTION( $s, d$ )
2:   loop
3:     SIMULATE( $s, d, \pi_0$ )
4:   return  $\arg \max_a Q(s, a)$ 
5: function SIMULATE( $s, d, \pi_0$ )
6:   if  $d = 0$ 
7:     return 0
8:   if  $s \notin T$ 
9:     for  $a \in A(s)$ 
10:       $(N(s, a), Q(s, a)) \leftarrow (N_0(s, a), Q_0(s, a))$ 
11:      $T = T \cup \{s\}$ 
12:     return ROLLOUT( $s, d, \pi_0$ )
13:    $a \leftarrow \arg \max_{a \in A(s)} \left[ Q(s, a) + c \sqrt{\frac{\log N(s)}{N(s, a)}} \right]$ 
14:    $(s', r) \sim G(s, a)$ 
15:    $q \leftarrow r + \gamma \text{SIMULATE}(s', d - 1, \pi_0)$ 
16:    $N(s, a) \leftarrow N(s, a) + 1$ 
17:    $Q(s, a) \leftarrow Q(s, a) + \frac{q - Q(s, a)}{N(s, a)}$ 
18:   return  $q$ 
```

Algorithm 4.10 Rollout evaluation

```
1: function ROLLOUT( $s, d, \pi_0$ )
2:   if  $d = 0$ 
3:     return 0
4:    $a \sim \pi_0(s)$ 
5:    $(s', r) \sim G(s, a)$ 
6:   return  $r + \gamma \text{ROLLOUT}(s', d - 1, \pi_0)$ 
```





Using Online Methods in a Simulation

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 $0 \dots T - 1$

$a \leftarrow \pi(s)$

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

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

$s \leftarrow s'$

return \hat{u}

Create tree (run more imaginary rollouts)
Choose best action based on tree

take step in real environment

Guiding Questions

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

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

Things it keeps track of:

$Q(s, a)$: 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**

$$L^d(s, a) = U^d(s, a) = R(s, a), \forall a$$

$$L^d(s) = U^d(s) = \max_a R(s, a)$$

else if $n_{sd} = 0$ **then**

for each $a \in A$ **do**

$$L^d(s, a) = V_{\min}$$

$$U^d(s, a) = V_{\max}$$

for C times **do**

$$s' \sim T(s, a, \cdot)$$

$$L^{d-1}(s') = V_{\min}$$

$$U^{d-1}(s') = V_{\max}$$

$$K^d(s, a) = K^d(s, a) \cup \{s'\}$$

$$a^* = \operatorname{argmax}_a U^d(s, a)$$

$$s^* = \max_{s' \in K^d(s, a^*)} (U^{d-1}(s') - L^{d-1}(s'))$$

FSSS($s^*, d - 1$)

$$n_{sd} = n_{sd} + 1$$

$$L^d(s, a^*) = R(s, a^*) + \gamma \sum_{s' \in K^d(s, a^*)} L^{d-1}(s') / C$$

$$U^d(s, a^*) = R(s, a^*) + \gamma \sum_{s' \in K^d(s, a^*)} U^{d-1}(s') / C$$

$$L^d(s) = \max_a L^d(s, a)$$

$$U^d(s) = \max_a U^d(s, a)$$

Forward Search Sparse Sampling

Algorithm 3 FSSS(s, d)

if $d = 1$ (leaf) **then**

$$L^d(s, a) = U^d(s, a) = R(s, a), \forall a$$

$$L^d(s) = U^d(s) = \max_a R(s, a)$$

else if $n_{sd} = 0$ **then**

for each $a \in A$ **do**

$$L^d(s, a) = V_{\min}$$

$$U^d(s, a) = V_{\max}$$

for C times **do**

$$s' \sim T(s, a, \cdot)$$

$$L^{d-1}(s') = V_{\min}$$

$$U^{d-1}(s') = V_{\max}$$

$$K^d(s, a) = K^d(s, a) \cup \{s'\}$$

$$a^* = \operatorname{argmax}_a U^d(s, a)$$

$$s^* = \max_{s' \in K^d(s, a^*)} (U^{d-1}(s') - L^{d-1}(s'))$$

FSSS($s^*, d - 1$)

$$n_{sd} = n_{sd} + 1$$

$$L^d(s, a^*) = R(s, a^*) + \gamma \sum_{s' \in K^d(s, a^*)} L^{d-1}(s') / C$$

$$U^d(s, a^*) = R(s, a^*) + \gamma \sum_{s' \in K^d(s, a^*)} U^{d-1}(s') / C$$

$$L^d(s) = \max_a L^d(s, a)$$

$$U^d(s) = \max_a U^d(s, a)$$

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.