Online Methods

Last Time

- Value Iteration
- Policy Iteration

$$V'(s) = \max_{a \in A} \left(R(s,a) + \gamma \sum_{s' \in S} T(s'|s,a) V(s') \right)$$

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?

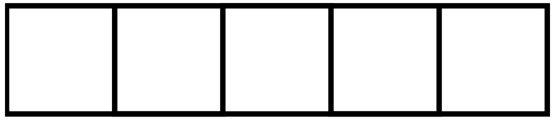
- Problems Policy and Value Iteration may struggle with?
  - Path planning across the country, or interplanetary
  - More realistic car dynamics (continuous states)
- Why are these problems hard?
  - State Space is massive (or infinite)

Curse of Dimensionality

Curse of Dimensionality

1 dimension, 5 segments

$$|\mathcal{S}| = 5$$



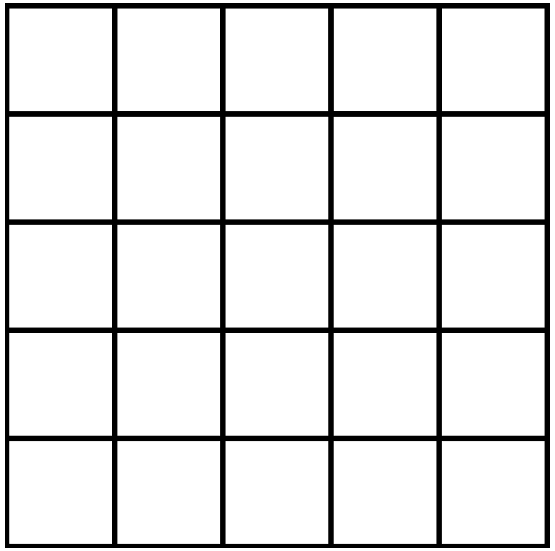
Curse of Dimensionality

1 dimension, 5 segments

$$|\mathcal{S}| = 5$$

2 dimensions, 5 segments

$$|\mathcal{S}| = 25$$



Curse of Dimensionality

1 dimension, 5 segments

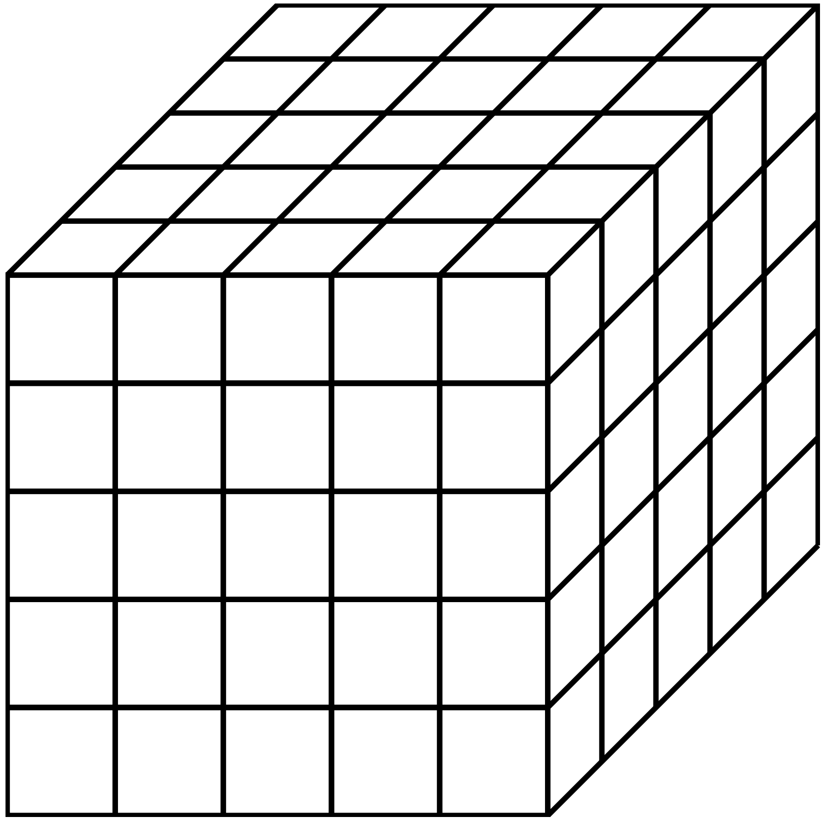
$$|\mathcal{S}| = 5$$

2 dimensions, 5 segments

$$|\mathcal{S}| = 25$$

3 dimensions, 5 segments

$$|\mathcal{S}| = 125$$



Curse of Dimensionality

1 dimension, 5 segments

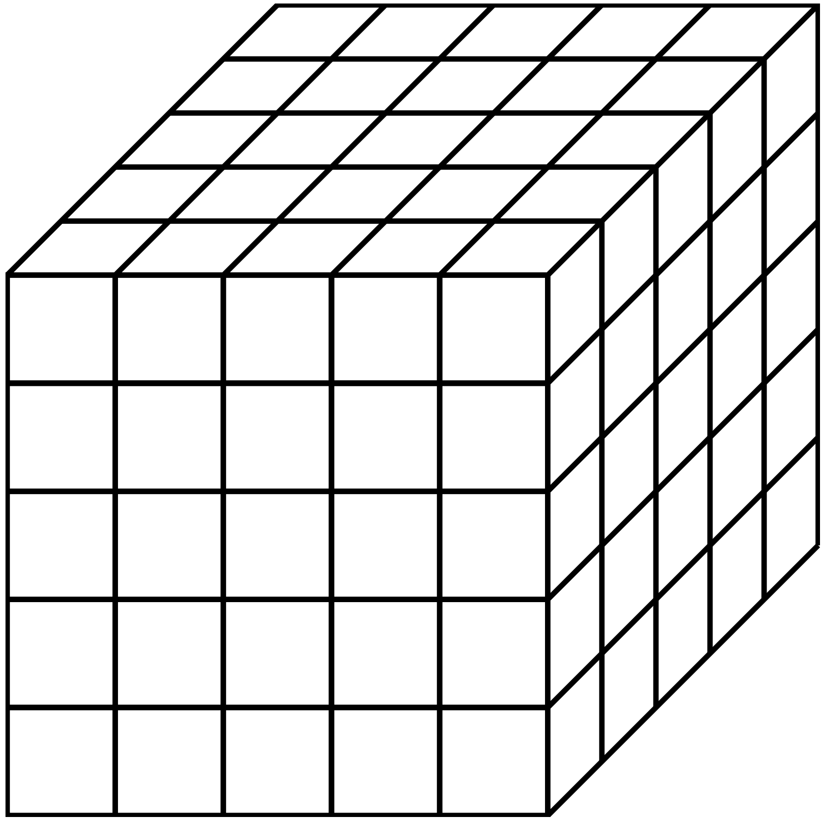
$$|\mathcal{S}| = 5$$

2 dimensions, 5 segments

$$|\mathcal{S}| = 25$$

3 dimensions, 5 segments

$$|\mathcal{S}| = 125$$



n dimensions, k segments $\rightarrow |\mathcal{S}| = k^n$

Offline vs Online Solutions

Offline

Online

Offline vs Online Solutions

Offline

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

Online

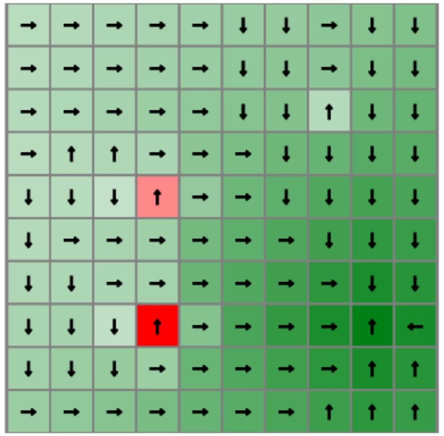
Offline vs Online Solutions

Offline

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

Online

Offline vs Online Solutions

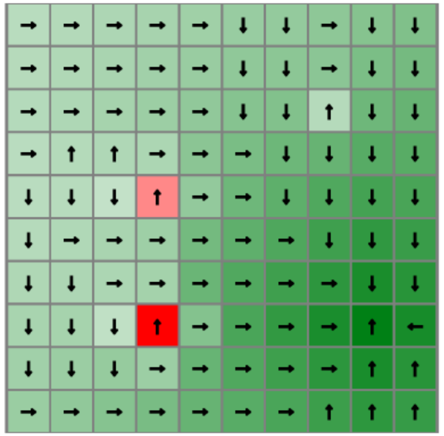


Offline

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

Online

Offline vs Online Solutions



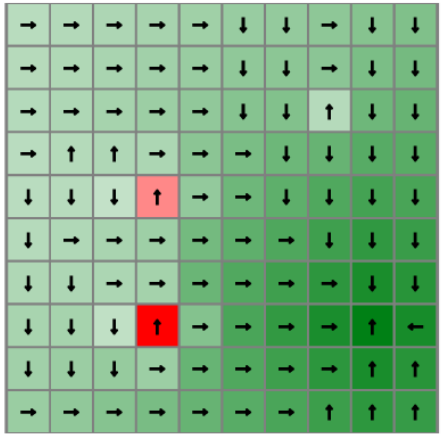
Offline

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

Online

- Before Execution: <nothing>

Offline vs Online Solutions



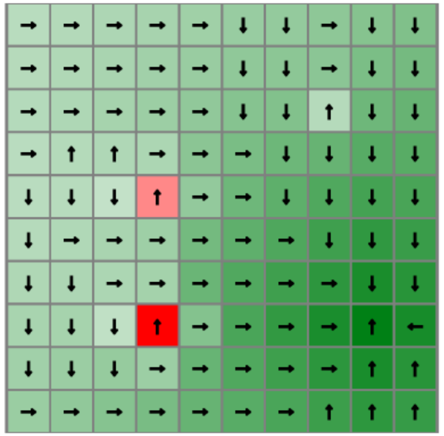
Offline

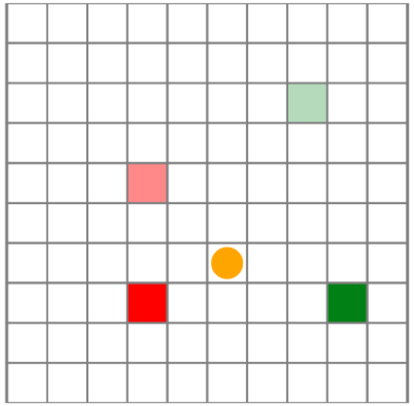
- Before Execution: find V^*/Q^*
- During Execution: $\pi^*(s) = \operatorname{argmax} Q^*(s,a)$

Online

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

Offline vs Online Solutions





Offline

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

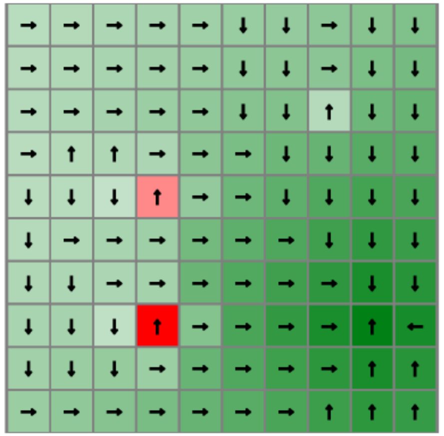
Online

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

Offline vs Online Solutions

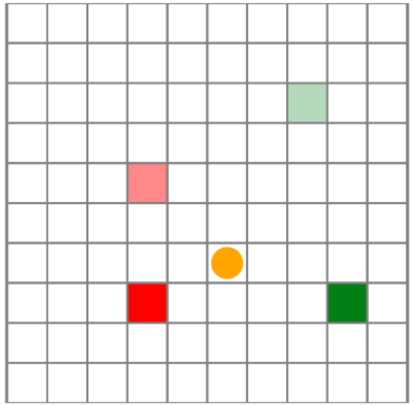
<u>Offline</u>

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



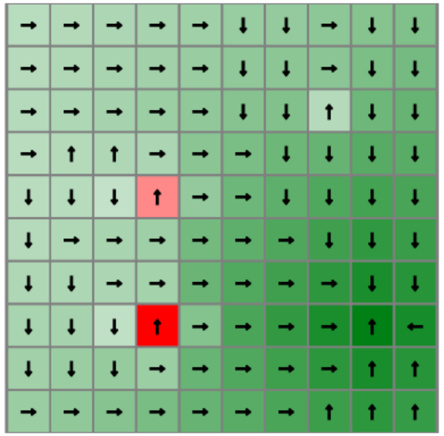
<u>Online</u>

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



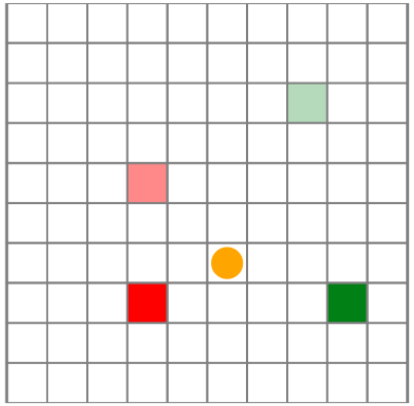
- Why?

Offline vs Online Solutions



Offline

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



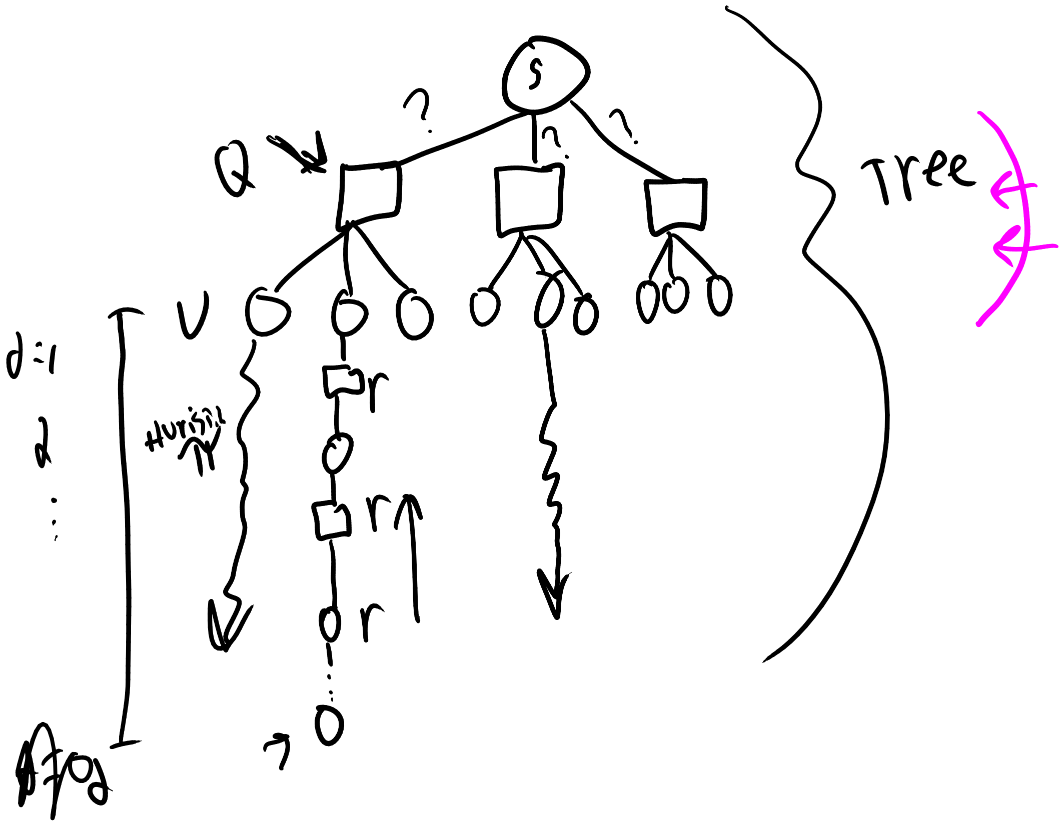
Online

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

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

|S|=3
|A|=3

One Step Lookahead




```
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
```

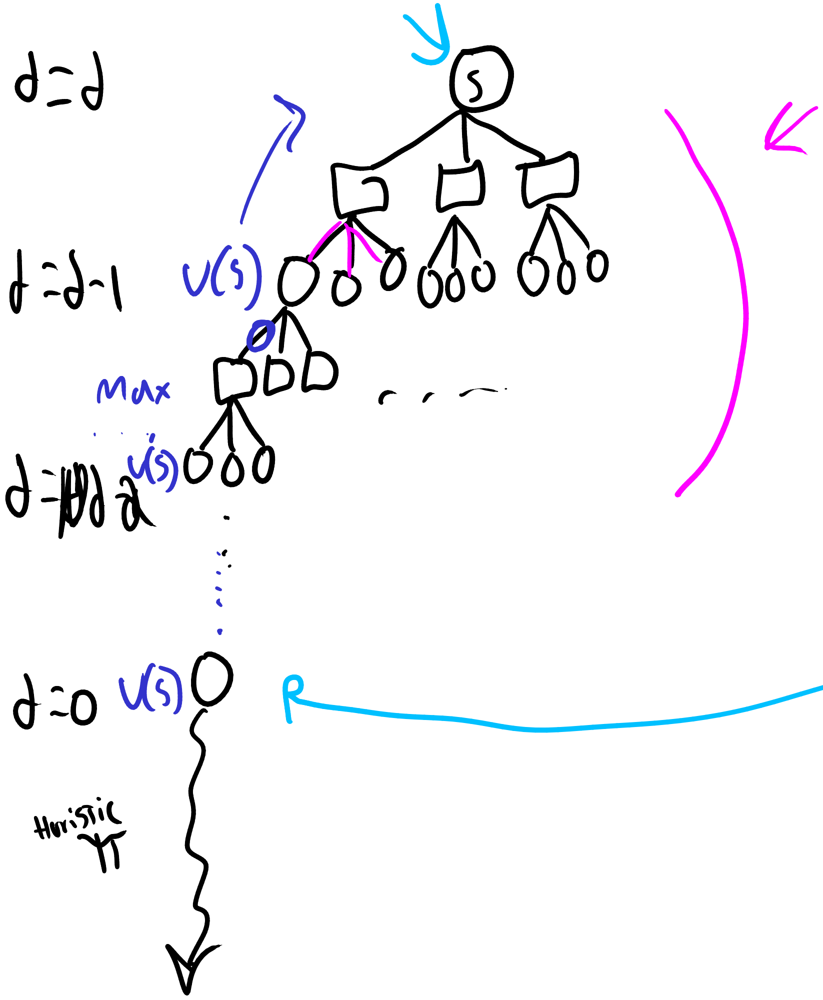
T(s'|s,a)

```
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)
```

|A|=3

Forward Search




```
function forward_search(𝒫, s, d, U)
    if d ≤ 0
        return (a=nothing, u=U(s))
    end
    best = (a=nothing, u=-Inf)
    U′(s) = forward_search(𝒫, s, d-1, U).u
    for a in 𝒫.𝒜
        u = lookahead(𝒫, U′, s, a)
        if u > best.u
            best = (a=a, u=u)
        end
    end
    return best
end
```

rollout

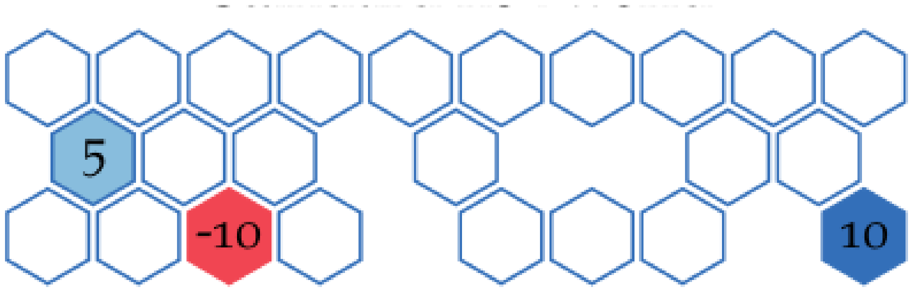
```
function lookahead(𝒫::MDP, U, s, a)
    𝒮, T, R, γ = 𝒫.𝒮, 𝒫.T, 𝒫.R, 𝒫.γ
    return R(s,a) + γ*sum(T(s,a,s′)*U(s′) for s′ in 𝒮)
```

$$O\left((|S| \times |A|)^d\right)$$

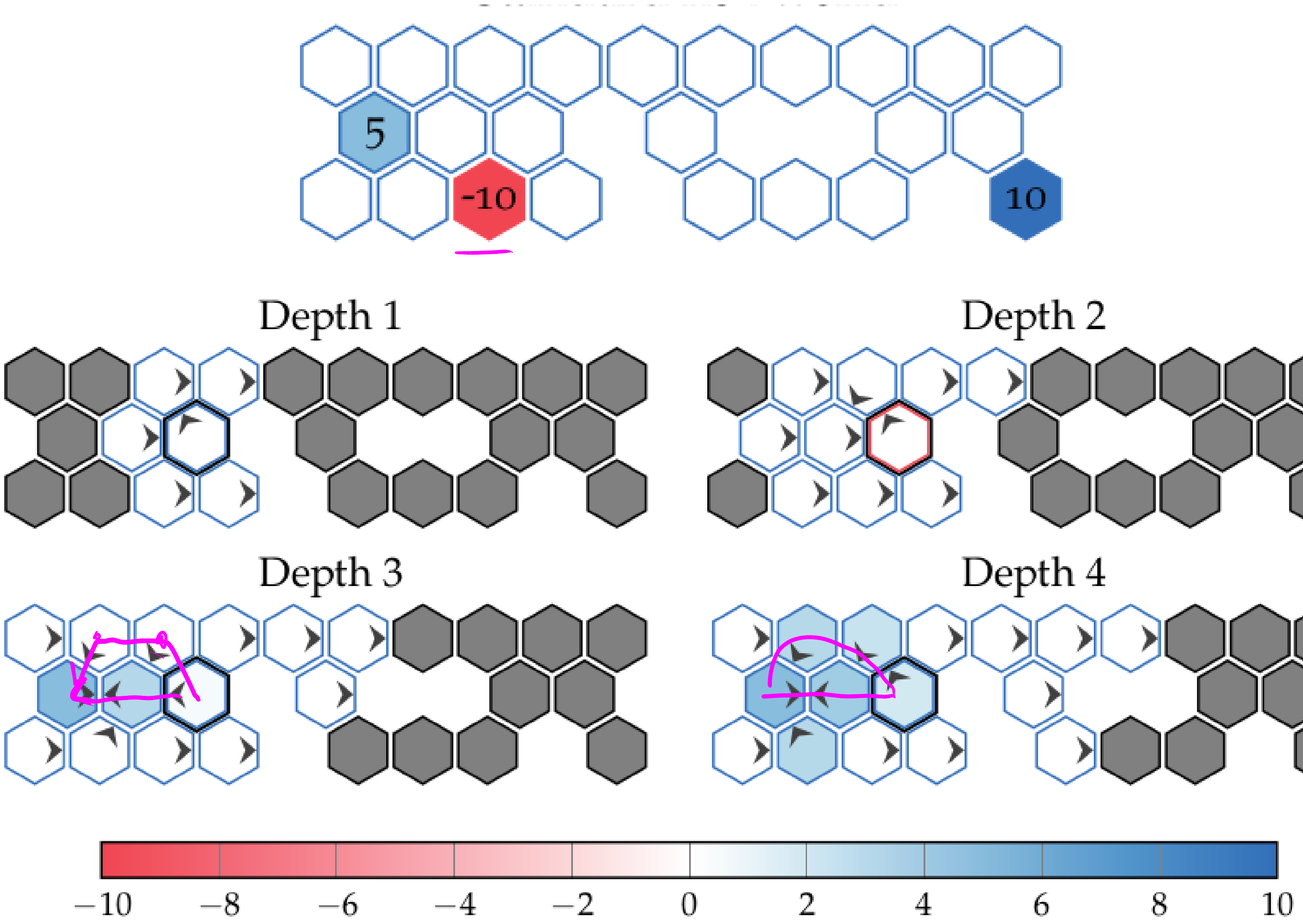
forward_search(d=d-1)

Forward Search depth

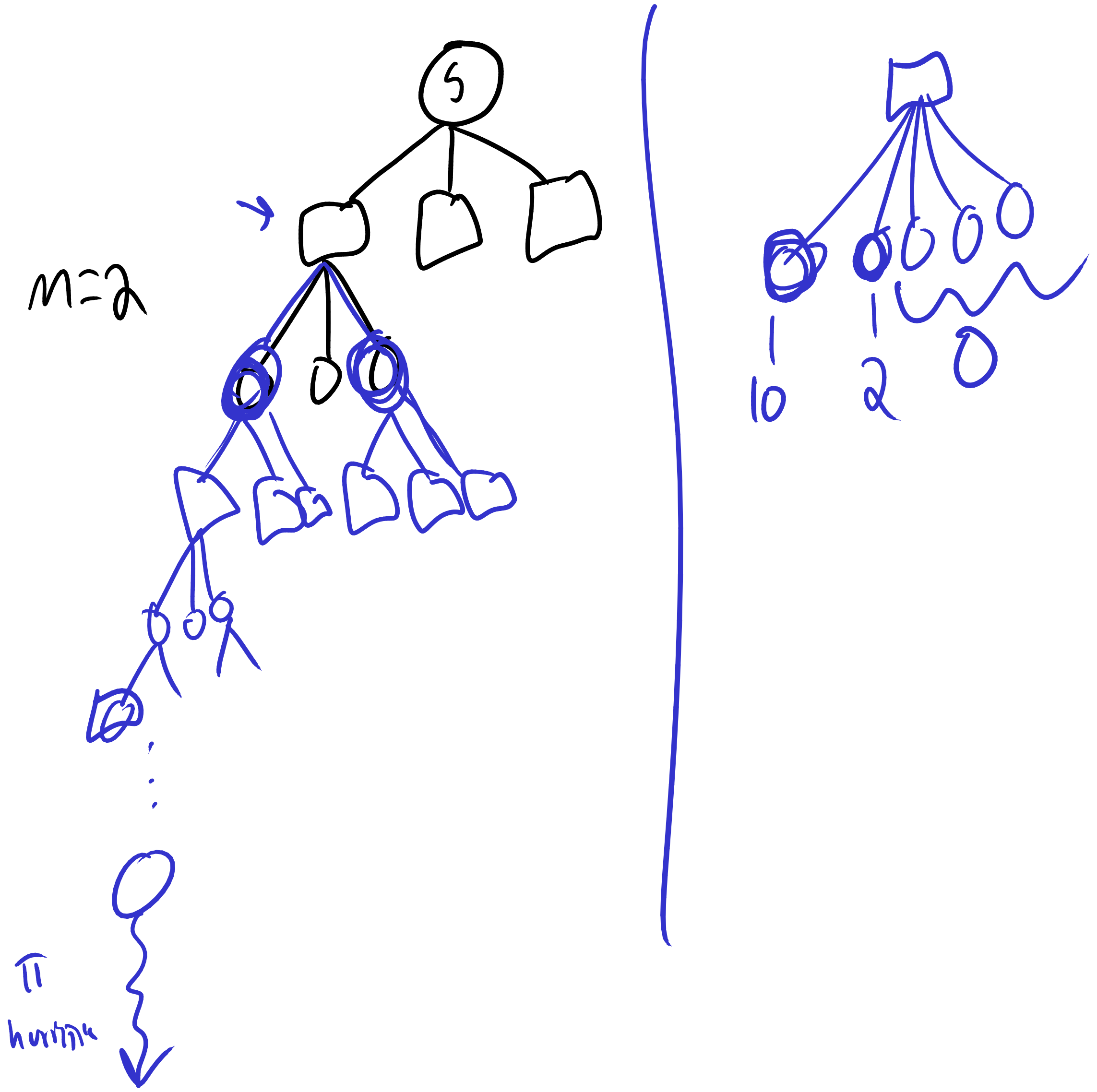
Forward Search depth



Forward Search depth



Sparse Sampling




```
function sparse_sampling(𝒫, s, d, m, U)
    if d ≤ 0
        return (a=nothing, u=U(s))
    end
    best = (a=nothing, u=-Inf)
    for a in 𝒫.𝒜
        u = 0.0
        for i in 1:m
            s′, r = randstep(𝒫, s, a)
            a′, u′ = sparse_sampling(𝒫, s′, d-1, m, U)
            u += (r + 𝒫.γ*u′) / m
        end
        if u > best.u
            best = (a=a, u=u)
        end
    end
    return best
end
```

Sparse Sampling

```
function sparse_sampling(𝒫, s, d, m, U)
    if d ≤ 0
        return (a=nothing, u=U(s))
    end
    best = (a=nothing, u=-Inf)
    for a in 𝒫.𝒜
        u = 0.0
        for i in 1:m
            s′, r = randstep(𝒫, s, a)
            a′, u′ = sparse_sampling(𝒫, s′, d-1, m, U)
            u += (r + 𝒫.γ*u′) / m
        end
        if u > best.u
            best = (a=a, u=u)
        end
    end
    return best
end
```

$$O\left((m|A|)^d\right)$$

Sparse Sampling

```
function sparse_sampling(𝒫, s, d, m, U)
    if d ≤ 0
        return (a=nothing, u=U(s))
    end
    best = (a=nothing, u=-Inf)
    for a in 𝒫.𝒜
        u = 0.0
        for i in 1:m
            s′, r = randstep(𝒫, s, a)
            a′, u′ = sparse_sampling(𝒫, s′, d-1, m, U)
            u += (r + 𝒫.γ*u′) / m
        end
        if u > best.u
            best = (a=a, u=u)
        end
    end
    return best
end
```

$O\left((m|A|)^d\right)$ $\quad$ $|V^{\mathrm{SS}}(s) - V^*(s)| \leq \epsilon$

Sparse Sampling

```
function sparse_sampling(𝒫, s, d, m, U)
    if d ≤ 0
        return (a=nothing, u=U(s))
    end
    best = (a=nothing, u=-Inf)
    for a in 𝒫.𝒜
        u = 0.0
        for i in 1:m
            s′, r = randstep(𝒫, s, a)
            a′, u′ = sparse_sampling(𝒫, s′, d-1, m, U)
            u += (r + 𝒫.γ*u′) / m
        end
        if u > best.u
            best = (a=a, u=u)
        end
    end
    return best
end
```

$O\left((m|A|)^d\right)$ $\quad |V^{\mathrm{SS}}(s) - V^*(s)| \leq \epsilon$ $\quad m$, ϵ, and d related, but independent of $|S|$

Sparse Sampling

```
function sparse_sampling(𝒫, s, d, m, U)
    if d ≤ 0
        return (a=nothing, u=U(s))
    end
    best = (a=nothing, u=-Inf)
    for a in 𝒫.𝒜
        u = 0.0
        for i in 1:m
            s′, r = randstep(𝒫, s, a)
            a′, u′ = sparse_sampling(𝒫, s′, d-1, m, U)
            u += (r + 𝒫.γ*u′) / m
        end
        if u > best.u
            best = (a=a, u=u)
        end
    end
    return best
end
```

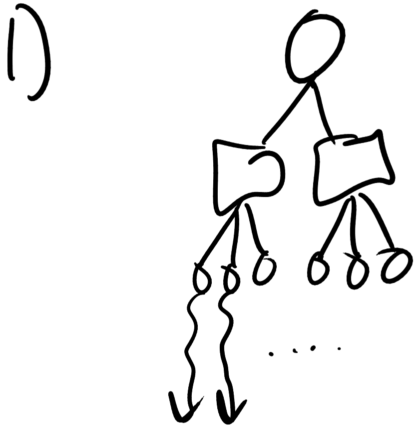
$O\left((m|A|)^d\right)$ $\quad |V^{\text{SS}}(s) - V^*(s)| \leq \epsilon$ $\quad m$, ϵ, and d related, but independent of $|S|$

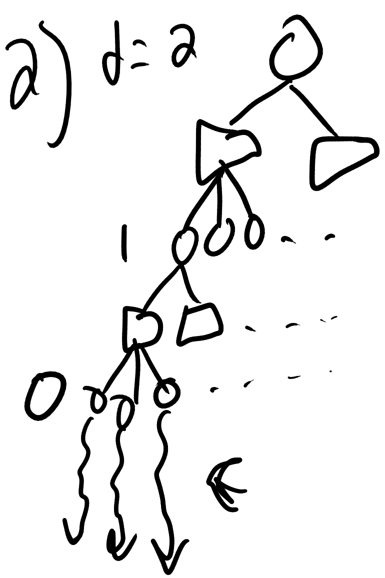
https://www.cis.upenn.edu/~mkearns/papers/sparsesampling-journal.pdf

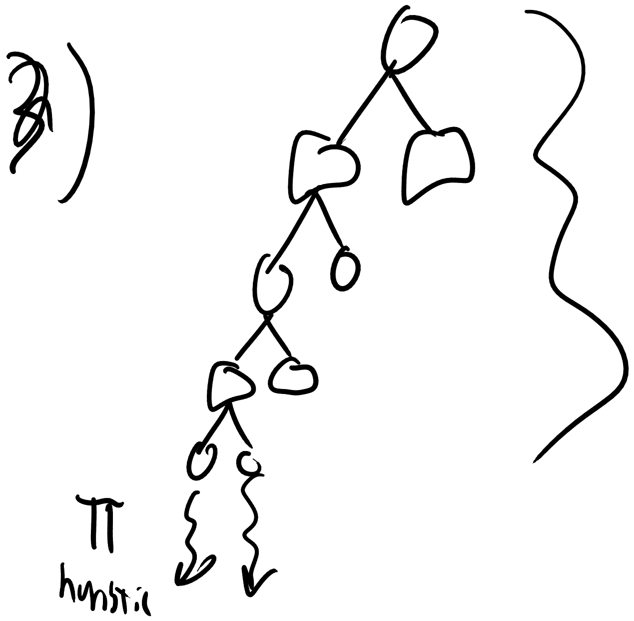
Break

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, m=2)







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

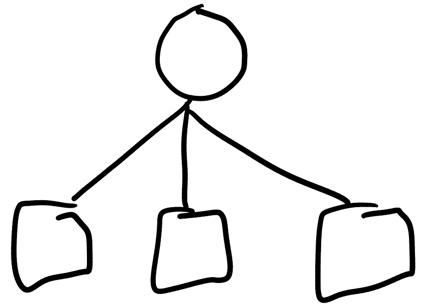
$L(s)$: Lower bound for value of state s

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

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

S_1: V:100 L:-50 Q:75

S_2: V:79 L:70 Q:75



A_1: V=100 L=-5 Q=75

A_2: V=79 L=70 Q=75

Forward Search Sparse Sampling

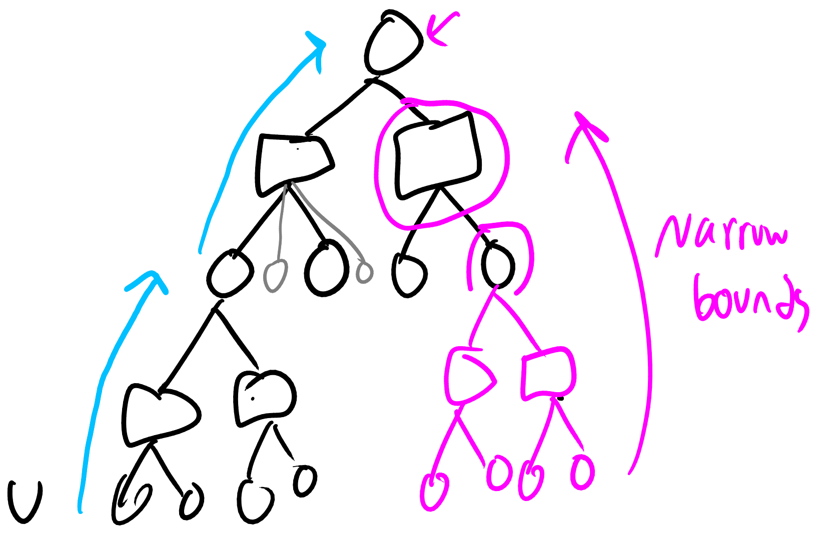
Algorithm 3 FSSS(s, d)

if $d = 1$ (leaf) **then** — exit case
  $L^d(s,a) = U^d(s,a) = R(s,a), \forall a$
  $L^d(s) = U^d(s) = \max_a R(s,a)$
else if $n_{sd} = 0$ **then** — if new state, pre-allocate
  for each $a \in A$ **do**
    $L^d(s,a) = V_{\min}$
    $U^d(s,a) = V_{\max}$
    for C times do
      $s' \sim T(s,a,\cdot)$
      $L^{d-1}(s') = V_{\min}$
      $U^{d-1}(s') = V_{\max}$
      $K^d(s,a) = K^d(s,a) \cup \{s'\}$ ← Reachable states
$a^* = \text{argmax}_a U^d(s,a)$ ← & ↑V
$s^* = \max_{s' \in K^d(s,a^*)} (U^{d-1}(s') - L^{d-1}(s'))$ ←
FSSS(s^*, $d-1$)
$n_{sd} = n_{sd} + 1$ ← how many times visiting
$L^d(s,a^*) = R(s,a^*) + \gamma \sum_{s' \in K^d(s,a^*)} L^{d-1}(s')/C$
$U^d(s,a^*) = R(s,a^*) + \gamma \sum_{s' \in K^d(s,a^*)} U^{d-1}(s')/C$ — update
$L^d(s) = \max_a L^d(s,a)$
$U^d(s) = \max_a U^d(s,a)$

C=2

explore

update



explore: a w\ ↑ V
out of $s' \in K(s,a)$
s' w\ ↑ $|U-L|$ ≈ uncertainty

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.

Monte Carlo Tree Search (MCTS/UCT)

- FSSS, but with less to keep track of

<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

Monte Carlo Tree Search (MCTS/UCT)

- FSSS, but with less to keep track of

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

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

Monte Carlo Tree Search (MCTS/UCT)

- FSSS, but with less to keep track of

<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

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

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

Monte Carlo Tree Search (MCTS/UCT)

- FSSS, but with less to keep track of

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)

- FSSS, but with less to keep track of

<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

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

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)

- FSSS, but with less to keep track of

```
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
```

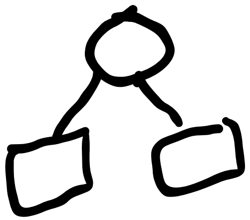
```
function simulate!(π::MonteCarloTreeSearch, s, d=π.d)
    if d ≤ 0
        return π.U(s)
    end
    𝒫, N, Q, c = π.𝒫, π.N, π.Q, π.c
    𝒜, TR, γ = 𝒫.𝒜, 𝒫.TR, 𝒫.γ
    if !haskey(N, (s, first(𝒜)))
        for a in 𝒜
            N[(s,a)] = 0
            Q[(s,a)] = 0.0
        end
        return π.U(s)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

Monte Carlo Tree Search (MCTS/UCT)

- FSSS, but with less to keep track of

```
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
```

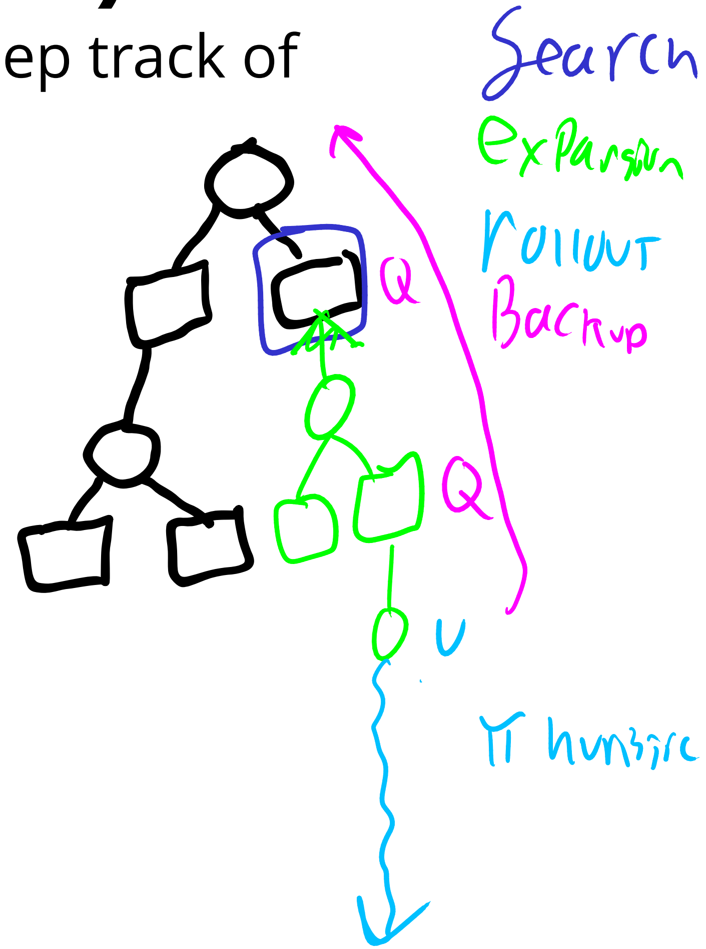


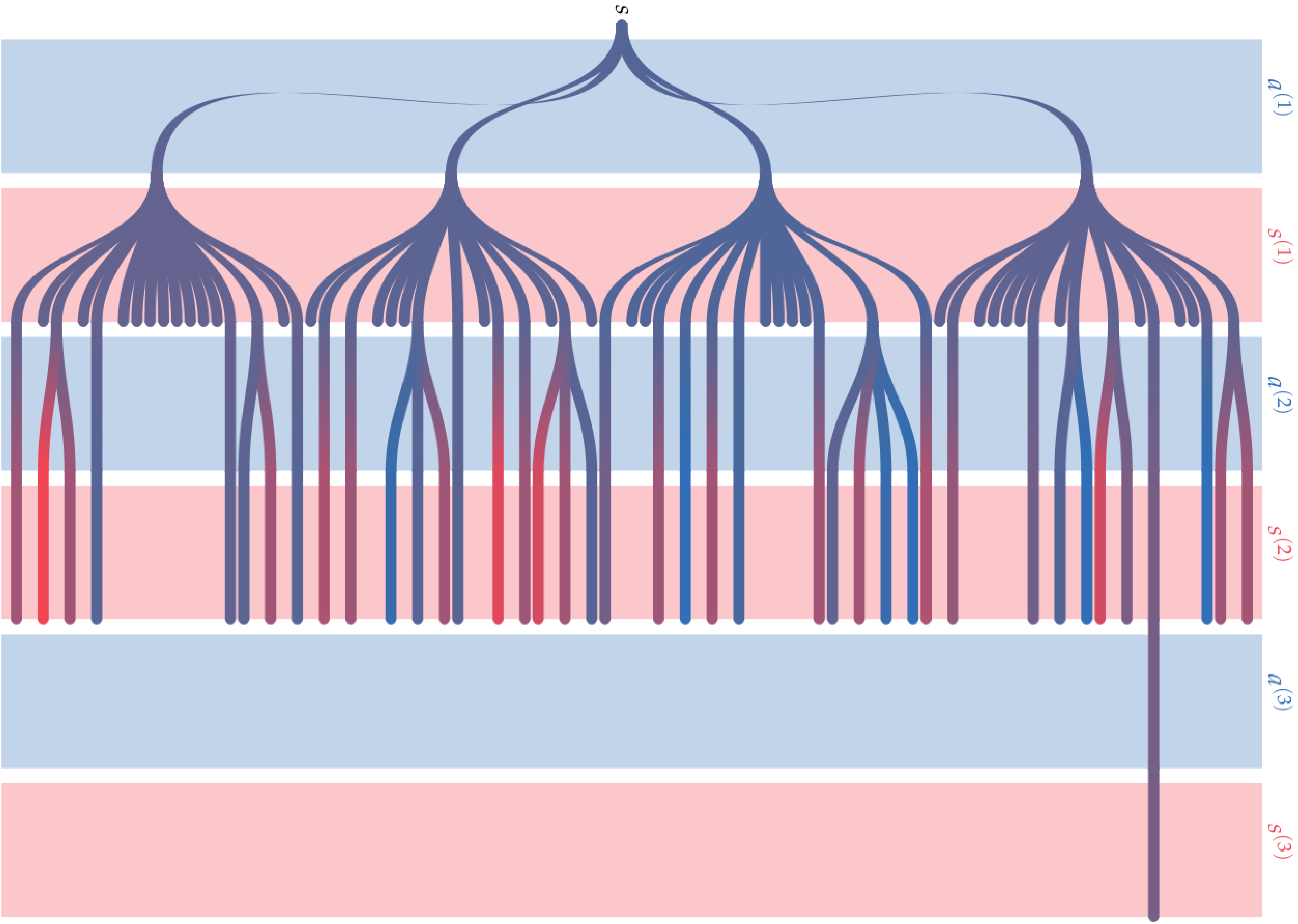
Monte Carlo Tree Search (MCTS/UCT)

- FSSS, but with less to keep track of

```julia
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
```

```julia
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)

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?