Last Time

Last Time

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

Yes! Because B is a contraction

any number of optimal policies

every optimal policy achieves U^*

π^1 $U^{\pi^1} = U^*$

π^2 $U^{\pi^2} = U^*$

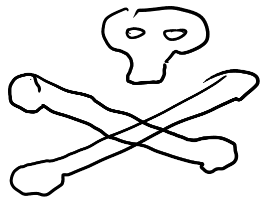
$A = \{left, right1, right2\}$

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?

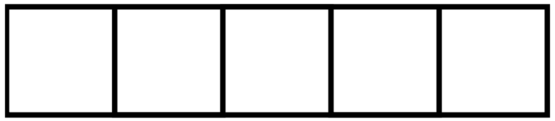
Curse of Dimensionality



Curse of Dimensionality

1 dimension, 5 segments

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



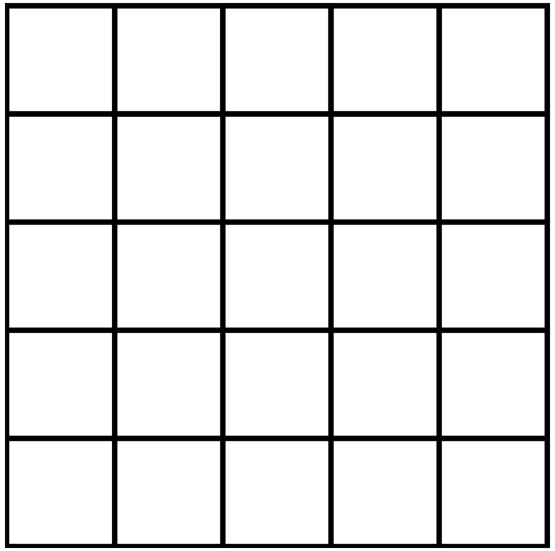
Curse of Dimensionality

1 dimension, 5 segments

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

2 dimensions, 5 segments

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



Curse of Dimensionality

1 dimension, 5 segments

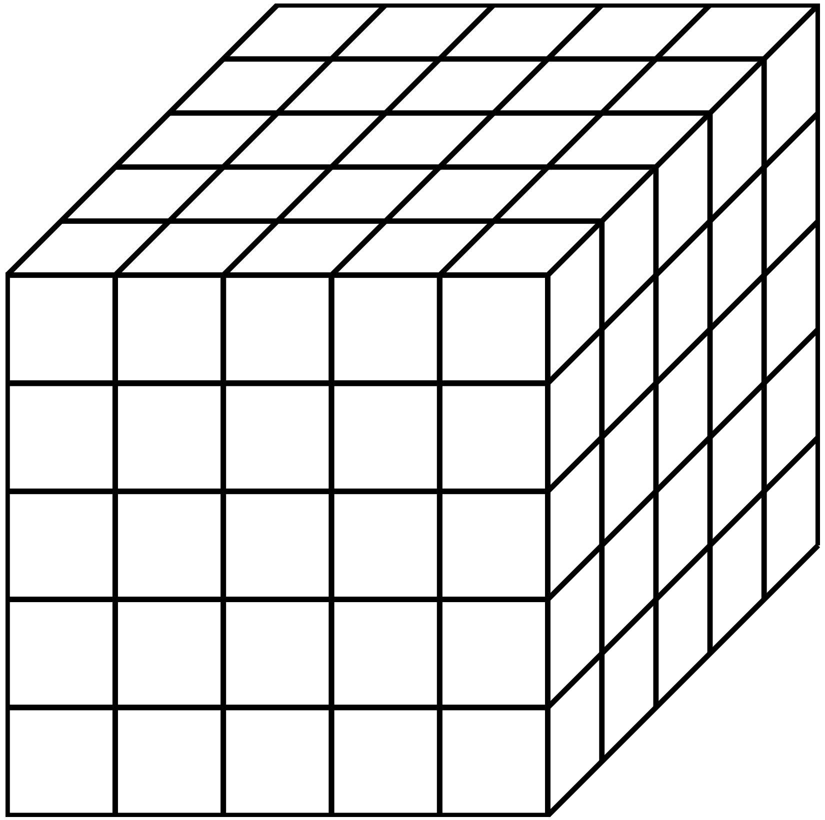
$$|\mathcal{S}| = 5$$

2 dimensions, 5 segments

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

3 dimensions, 5 segments

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



Curse of Dimensionality

1 dimension, 5 segments

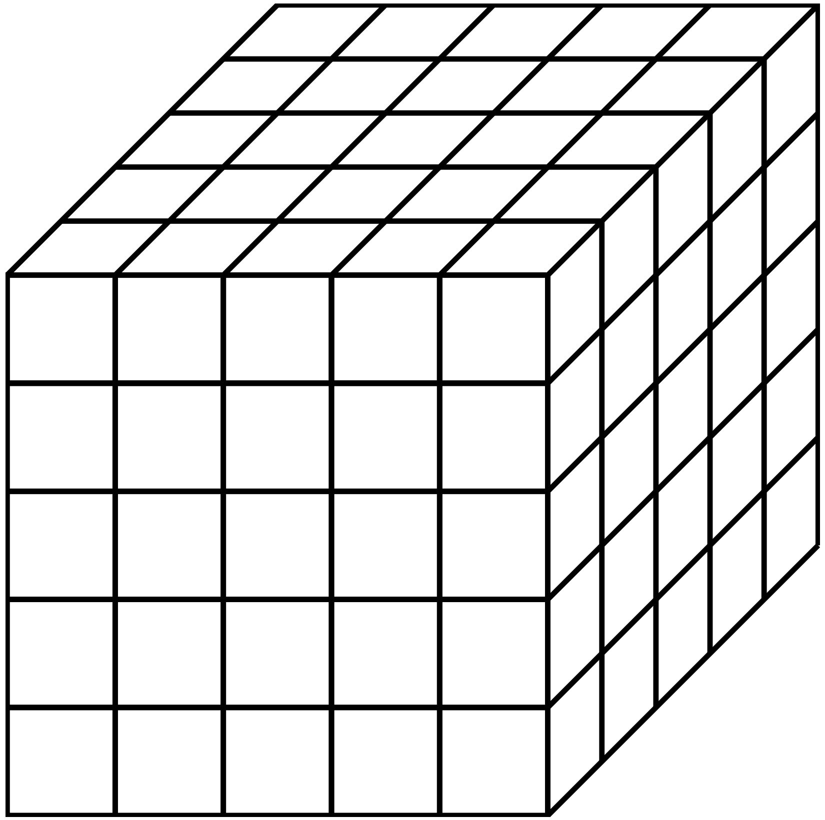
$|\mathcal{S}| = 5$

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

<u>Offline</u>

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

<u>Online</u>

Offline vs Online Solutions

Offline

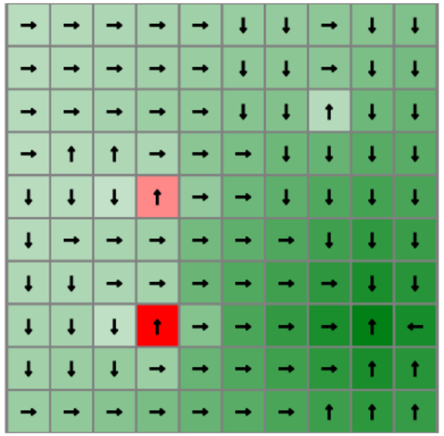
- Before Execution: find V^*/Q^*
- During Execution: $\pi^*(s) = \operatorname{argmax} Q^*(s, a)$

Online

Offline vs Online Solutions

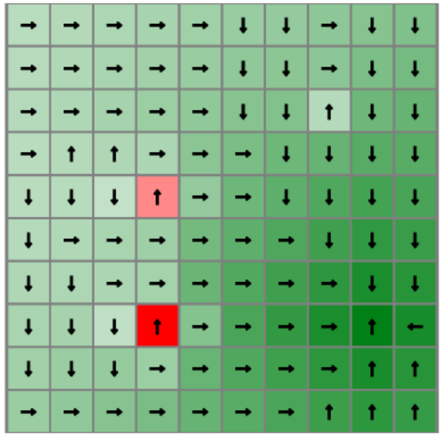
Offline

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



Online

Offline vs Online Solutions



Offline

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

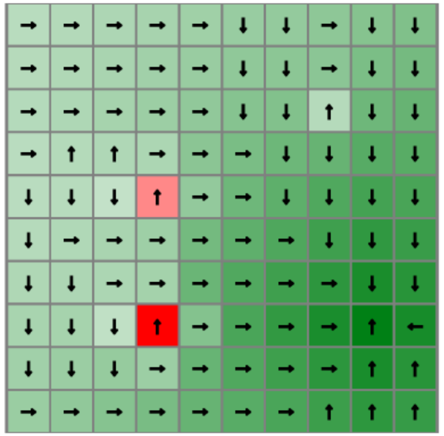
Online

- Before Execution: <nothing>

Offline vs Online Solutions

<u>Offline</u>

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



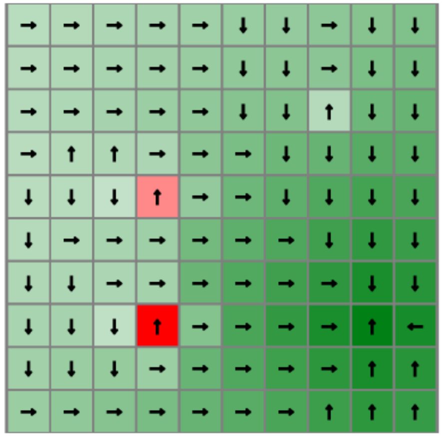
<u>Online</u>

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

Offline vs Online Solutions

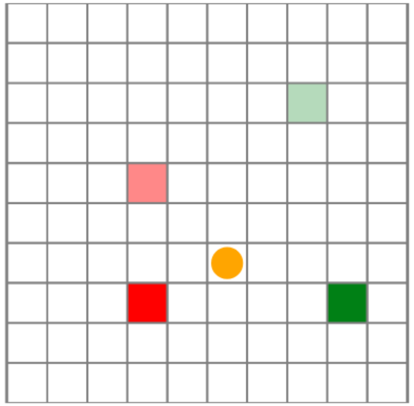
<u>Offline</u>

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

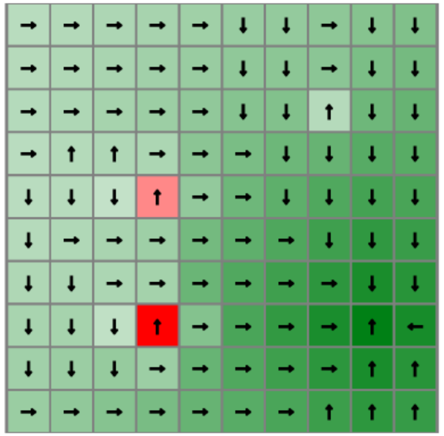


<u>Online</u>

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



Offline vs Online Solutions

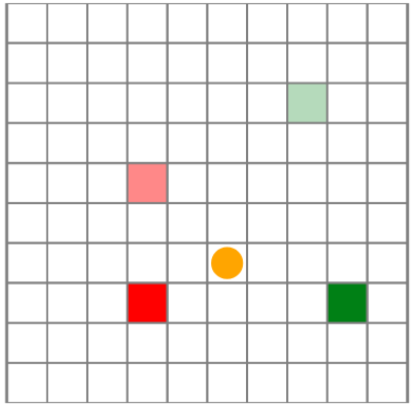


Offline

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

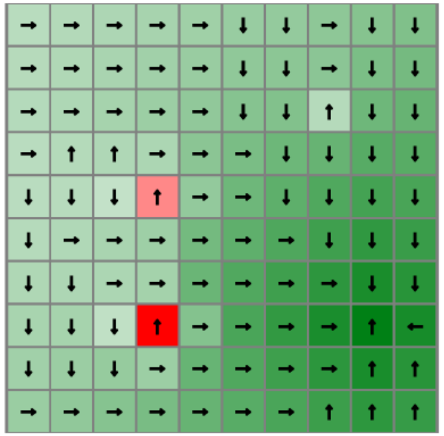
Online

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



- Why?

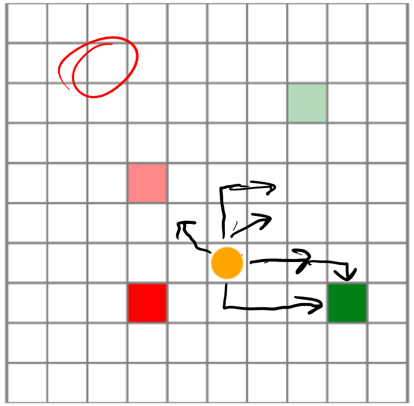
Offline vs Online Solutions



Offline

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

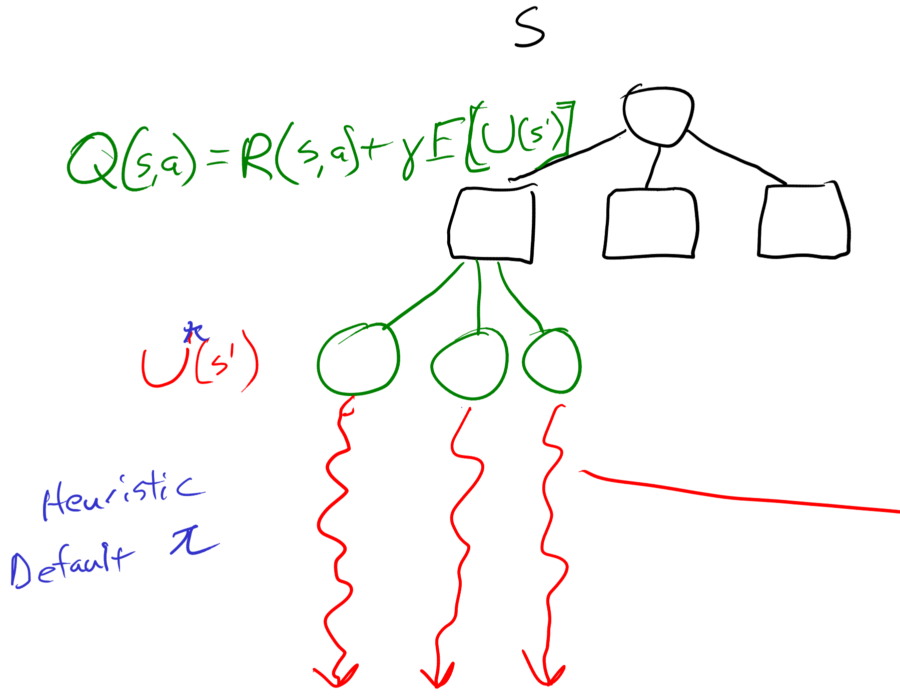
Online



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

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

One Step Lookahead



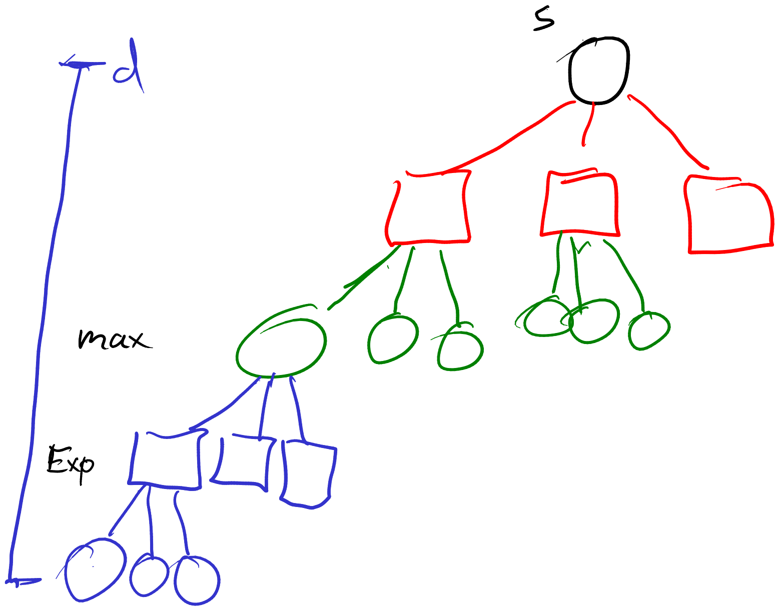

```julia
randstep(𝒫::MDP, s, a) = 𝒫.TR(s, a)

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
```

Forward Search



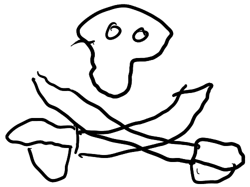
$Q(s,a)$

$U = \max Q(s,a)$

$Q = R(s,a) + \gamma E[U(s')]$

U

```julia
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
```

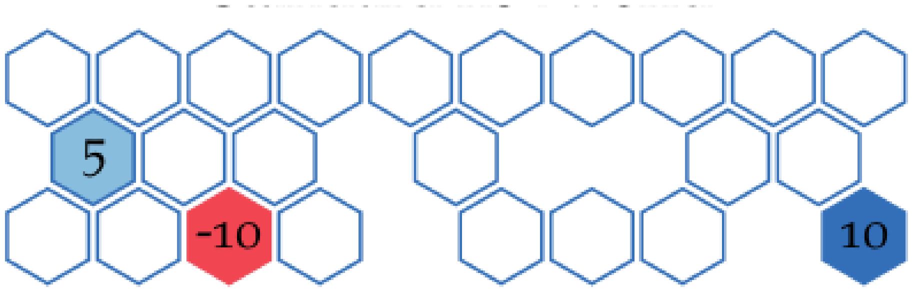


$$O((|S| \times |A|)^{d})$$

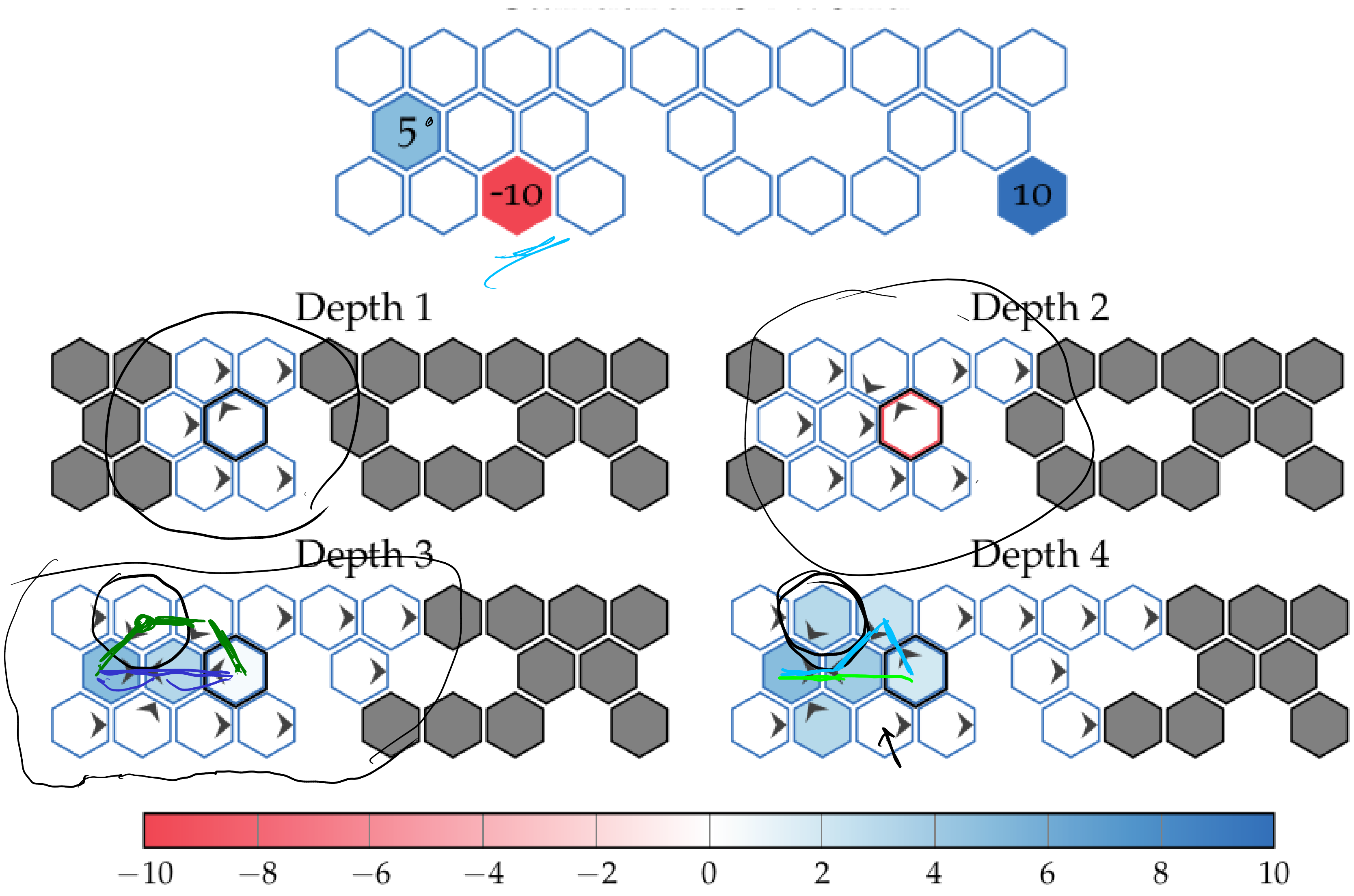
sad!

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
```

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

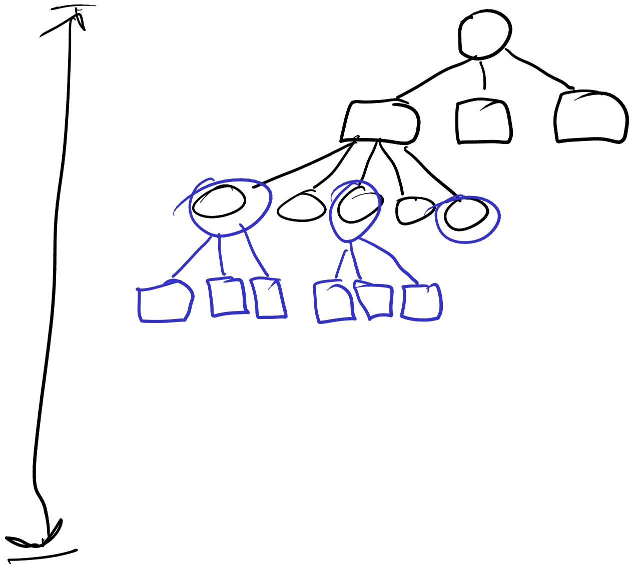
$|V^{\text{SS}}(s) - V^*(s)| \leq \epsilon$

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
```

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

Sparse Sampling



m=3

```
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 $|S|$

mad!

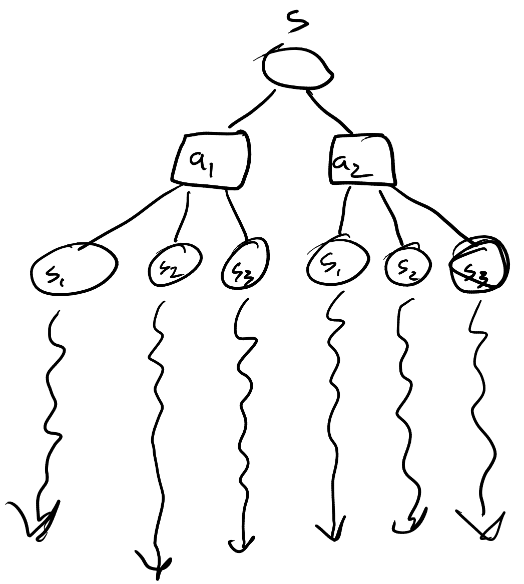
https://www.cis.upenn.edu/~mkearns/papers/sparsesampling-journal.pdf

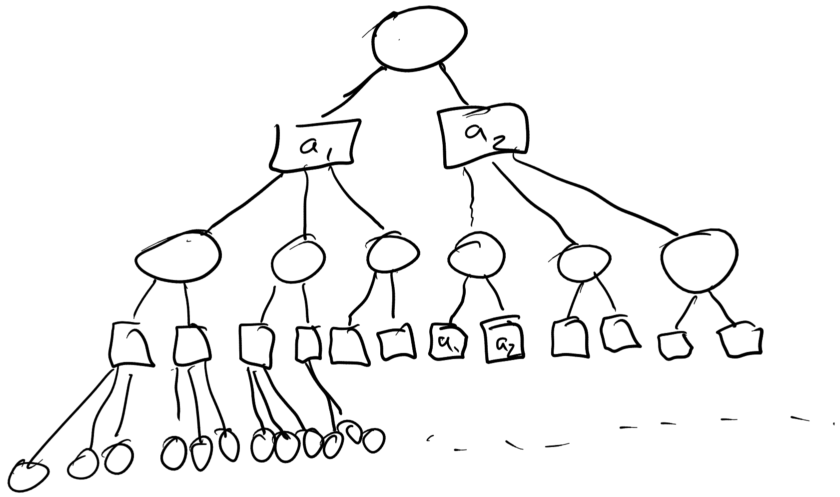
not on exam

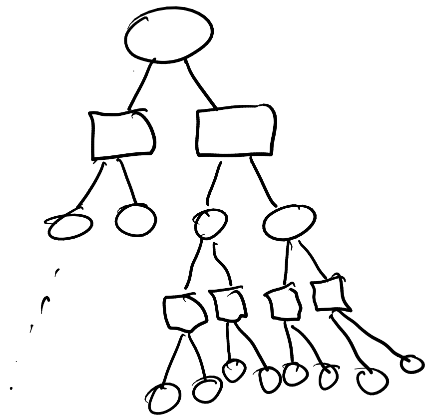
Break

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)







Branch and Bound

Assume you have $\underline{V}(s)$ and $\bar{Q}(s, a)$

```
function branch_and_bound(𝒫, s, d, Ulo, Qhi)
    if d ≤ 0
        return (a=nothing, u=Ulo(s))
    end
    U′(s) = branch_and_bound(𝒫, s, d-1, Ulo, Qhi).u
    best = (a=nothing, u=-Inf)
    for a in sort(𝒫.𝒜, by=a->Qhi(s,a), rev=true)
        if Qhi(s, a) < best.u
            return best # safe to prune
        end
        u = lookahead(𝒫, U′, s, a)
        if u > best.u
            best = (a=a, u=u)
        end
    end
    return best
end
```

Branch and Bound

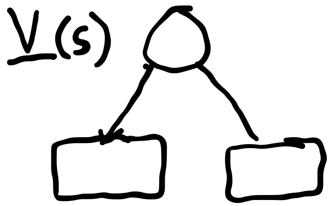
Assume you have $\underline{V}(s)$ and $\bar{Q}(s, a)$

$\underline{V}(s)$ O

```
function branch_and_bound(𝒫, s, d, Ulo, Qhi)
    if d ≤ 0
        return (a=nothing, u=Ulo(s))
    end
    U′(s) = branch_and_bound(𝒫, s, d-1, Ulo, Qhi).u
    best = (a=nothing, u=-Inf)
    for a in sort(𝒫.𝒜, by=a->Qhi(s,a), rev=true)
        if Qhi(s, a) < best.u
            return best # safe to prune
        end
        u = lookahead(𝒫, U′, s, a)
        if u > best.u
            best = (a=a, u=u)
        end
    end
    return best
end
```

Branch and Bound

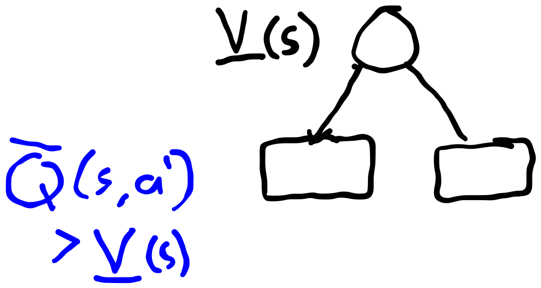
Assume you have $\underline{V}(s)$ and $\bar{Q}(s,a)$




```
function branch_and_bound(𝒫, s, d, Ulo, Qhi)
    if d ≤ 0
        return (a=nothing, u=Ulo(s))
    end
    U′(s) = branch_and_bound(𝒫, s, d-1, Ulo, Qhi).u
    best = (a=nothing, u=-Inf)
    for a in sort(𝒫.𝒜, by=a->Qhi(s,a), rev=true)
        if Qhi(s, a) < best.u
            return best # safe to prune
        end
        u = lookahead(𝒫, U′, s, a)
        if u > best.u
            best = (a=a, u=u)
        end
    end
    return best
end
```

Branch and Bound

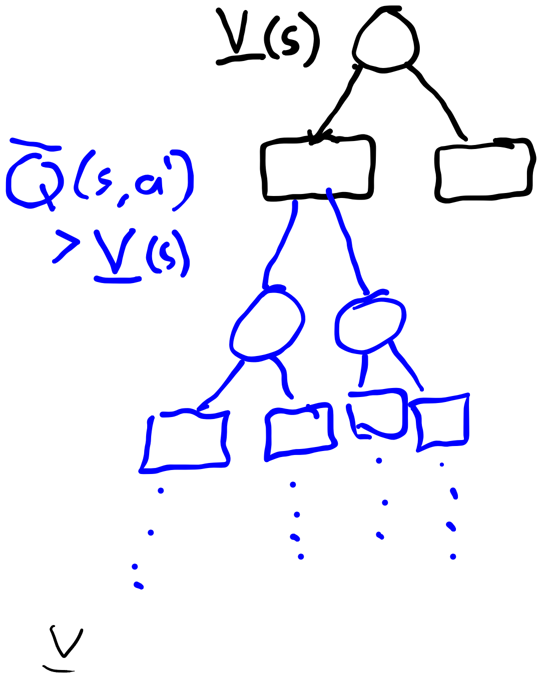
Assume you have $\underline{V}(s)$ and $\bar{Q}(s, a)$




```
function branch_and_bound(𝒫, s, d, Ulo, Qhi)
    if d ≤ 0
        return (a=nothing, u=Ulo(s))
    end
    U′(s) = branch_and_bound(𝒫, s, d-1, Ulo, Qhi).u
    best = (a=nothing, u=-Inf)
    for a in sort(𝒫.𝒜, by=a->Qhi(s,a), rev=true)
        if Qhi(s, a) < best.u
            return best # safe to prune
        end
        u = lookahead(𝒫, U′, s, a)
        if u > best.u
            best = (a=a, u=u)
        end
    end
    return best
end
```

Branch and Bound

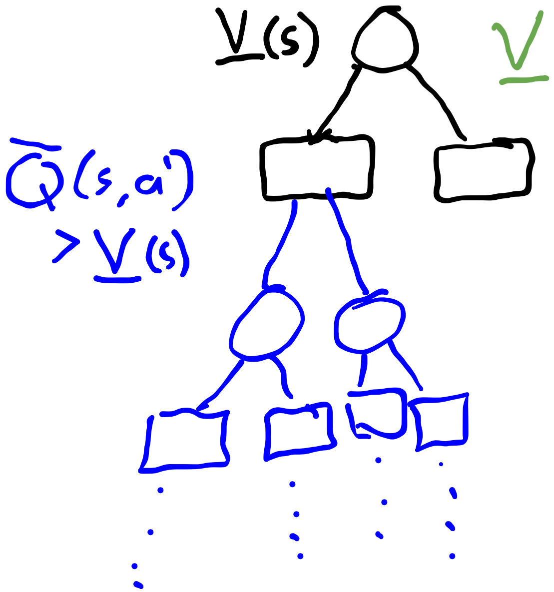
Assume you have $\underline{V}(s)$ and $\bar{Q}(s,a)$




```
function branch_and_bound(𝒫, s, d, Ulo, Qhi)
    if d ≤ 0
        return (a=nothing, u=Ulo(s))
    end
    U′(s) = branch_and_bound(𝒫, s, d-1, Ulo, Qhi).u
    best = (a=nothing, u=-Inf)
    for a in sort(𝒫.𝒜, by=a->Qhi(s,a), rev=true)
        if Qhi(s, a) < best.u
            return best # safe to prune
        end
        u = lookahead(𝒫, U′, s, a)
        if u > best.u
            best = (a=a, u=u)
        end
    end
    return best
end
```

Branch and Bound

Assume you have $\underline{V}(s)$ and $\bar{Q}(s,a)$

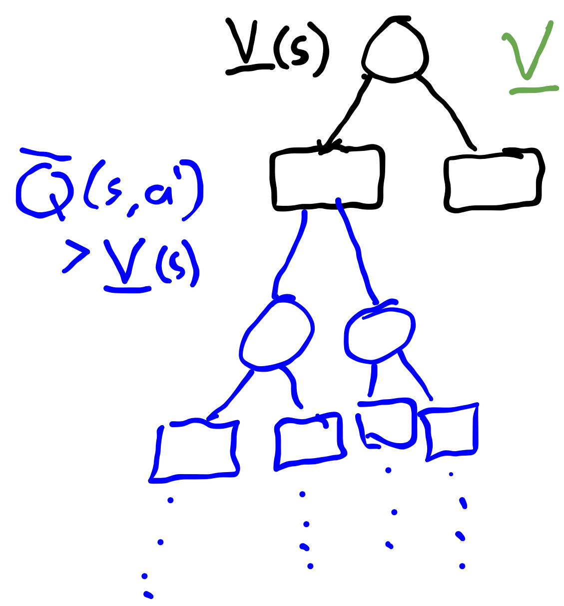


$\underline{V}(s) = \max(\underline{V}(s), Q(s,a'))$

```
function branch_and_bound(𝒫, s, d, Ulo, Qhi)
    if d ≤ 0
        return (a=nothing, u=Ulo(s))
    end
    U′(s) = branch_and_bound(𝒫, s, d-1, Ulo, Qhi).u
    best = (a=nothing, u=-Inf)
    for a in sort(𝒫.𝒜, by=a->Qhi(s,a), rev=true)
        if Qhi(s, a) < best.u
            return best # safe to prune
        end
        u = lookahead(𝒫, U′, s, a)
        if u > best.u
            best = (a=a, u=u)
        end
    end
    return best
end
```

Branch and Bound

Assume you have $\underline{V}(s)$ and $\bar{Q}(s,a)$



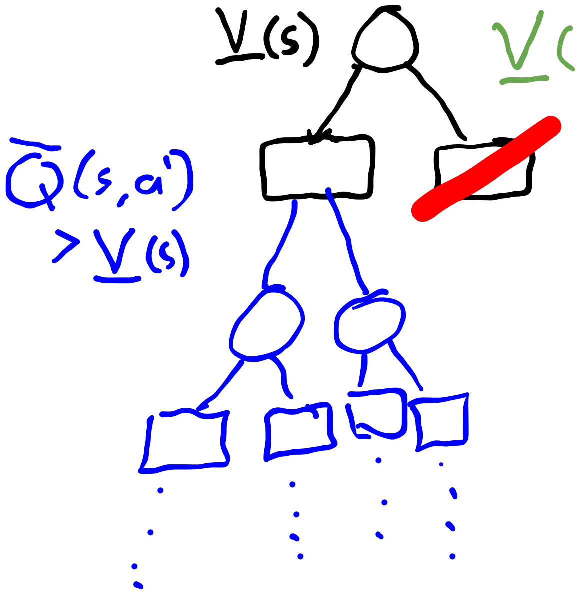
$\underline{V}(s) = \max(\underline{V}(s), Q(s,a'))$

$\bar{Q}(s,a^2) < \underline{V}(s)$

```
function branch_and_bound(𝒫, s, d, Ulo, Qhi)
    if d ≤ 0
        return (a=nothing, u=Ulo(s))
    end
    U′(s) = branch_and_bound(𝒫, s, d-1, Ulo, Qhi).u
    best = (a=nothing, u=-Inf)
    for a in sort(𝒫.𝒜, by=a->Qhi(s,a), rev=true)
        if Qhi(s, a) < best.u
            return best # safe to prune
        end
        u = lookahead(𝒫, U′, s, a)
        if u > best.u
            best = (a=a, u=u)
        end
    end
    return best
end
```

Branch and Bound

Assume you have $\underline{V}(s)$ and $\bar{Q}(s,a)$



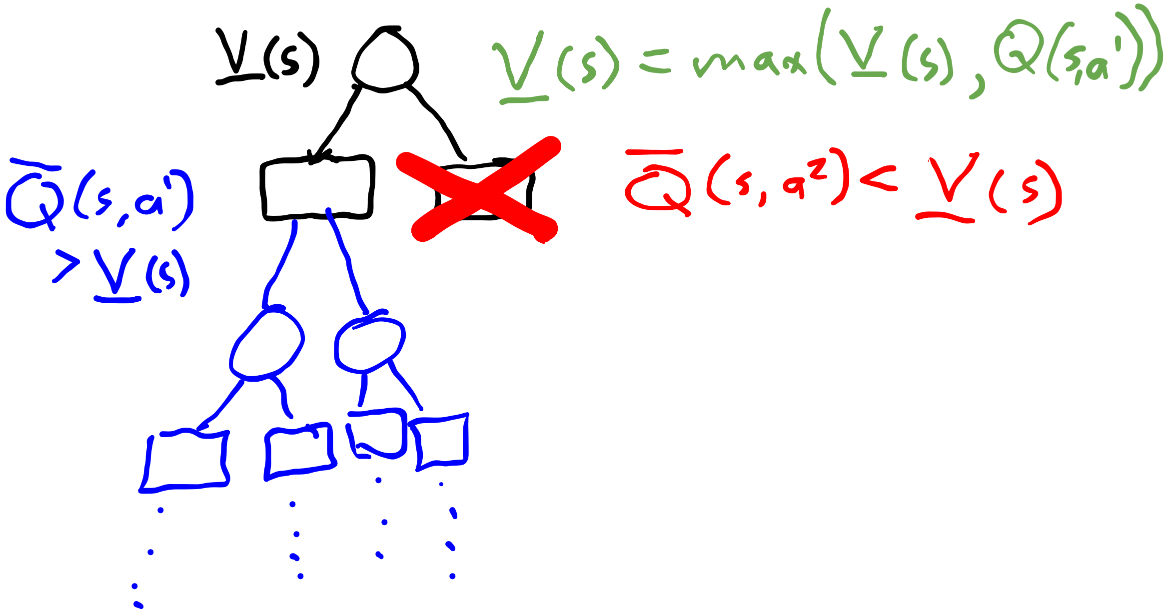
$\underline{V}(s) = \max(\underline{V}(s), Q(s,a'))$

$\bar{Q}(s,a^2) < \underline{V}(s)$

```
function branch_and_bound(𝒫, s, d, Ulo, Qhi)
    if d ≤ 0
        return (a=nothing, u=Ulo(s))
    end
    U′(s) = branch_and_bound(𝒫, s, d-1, Ulo, Qhi).u
    best = (a=nothing, u=-Inf)
    for a in sort(𝒫.𝒜, by=a->Qhi(s,a), rev=true)
        if Qhi(s, a) < best.u
            return best # safe to prune
        end
        u = lookahead(𝒫, U′, s, a)
        if u > best.u
            best = (a=a, u=u)
        end
    end
    return best
end
```

Branch and Bound

Assume you have $\underline{V}(s)$ and $\bar{Q}(s, a)$




```
function branch_and_bound(𝒫, s, d, Ulo, Qhi)
    if d ≤ 0
        return (a=nothing, u=Ulo(s))
    end
    U′(s) = branch_and_bound(𝒫, s, d-1, Ulo, Qhi).u
    best = (a=nothing, u=-Inf)
    for a in sort(𝒫.𝒜, by=a->Qhi(s,a), rev=true)
        if Qhi(s, a) < best.u
            return best # safe to prune
        end
        u = lookahead(𝒫, U′, s, a)
        if u > best.u
            best = (a=a, u=u)
        end
    end
    return best
end
```

Monte Carlo Tree Search (MCTS/UCT)

Monte Carlo Tree Search (MCTS/UCT)

Search

Monte Carlo Tree Search (MCTS/UCT)

Search Expansion

Monte Carlo Tree Search (MCTS/UCT)

Search Expansion Rollout

Monte Carlo Tree Search (MCTS/UCT)

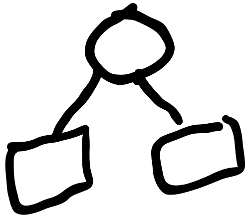
Search Expansion Rollout Backup

Monte Carlo Tree Search (MCTS/UCT)

Search Expansion Rollout Backup

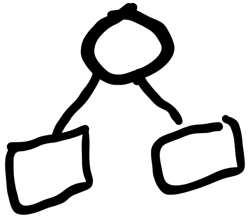
Monte Carlo Tree Search (MCTS/UCT)

Search | Expansion | Rollout | Backup

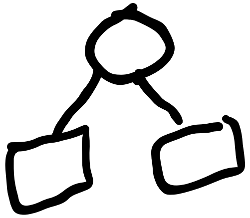


Monte Carlo Tree Search (MCTS/UCT)

Search



Expansion

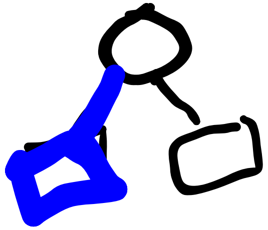


Rollout

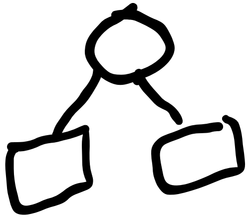
Backup

Monte Carlo Tree Search (MCTS/UCT)

Search



Expansion



Rollout

Backup

more common

$$\overbrace{Q(s,a) + c\sqrt{\frac{\log N(s)}{N(s,a)}}} \quad \text{or} \quad \overbrace{Q(s,a) + c\frac{N(s)^{\beta}}{\sqrt{N(s,a)}}}$$

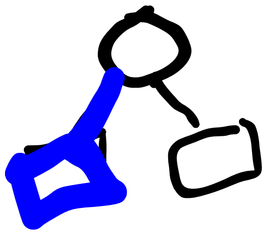
more theoretically justified

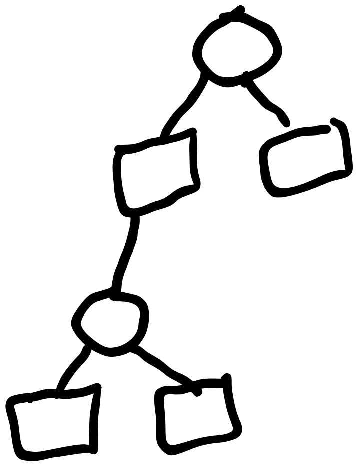
low $N(s,a)/N(s)$ = high bonus

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

Monte Carlo Tree Search (MCTS/UCT)

Search | Expansion | Rollout | Backup





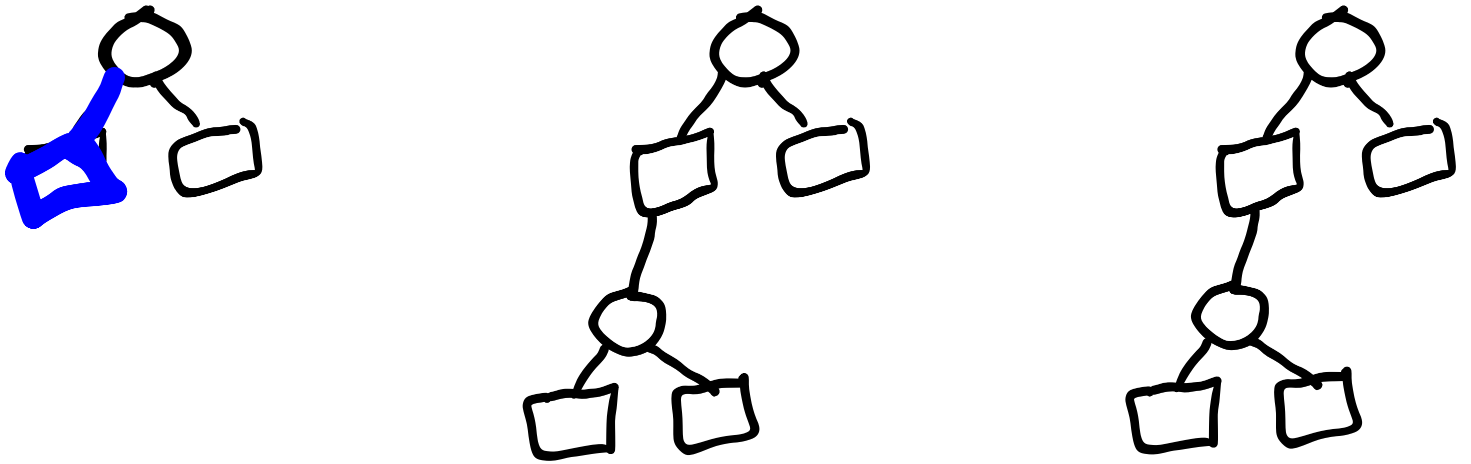
$$Q(s,a) + c\sqrt{\frac{\log N(s)}{N(s,a)}} \quad \text{or} \quad Q(s,a) + c\frac{N(s)^\beta}{\sqrt{N(s,a)}}$$

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

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

Monte Carlo Tree Search (MCTS/UCT)

Search | Expansion | Rollout | Backup



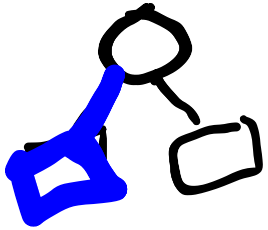
$Q(s,a) + c\sqrt{\frac{\log N(s)}{N(s,a)}}$ or $Q(s,a) + c\frac{N(s)^{\beta}}{\sqrt{N(s,a)}}$

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

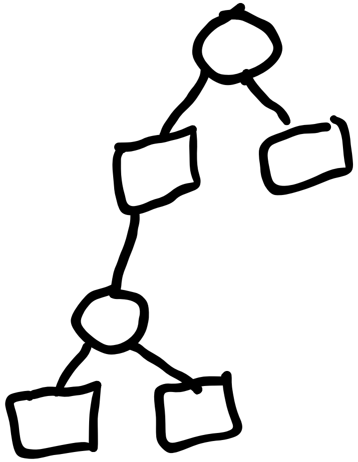
start with $c = 2(\bar{V} - \underline{V})$, $\beta = 1/4$

Monte Carlo Tree Search (MCTS/UCT)

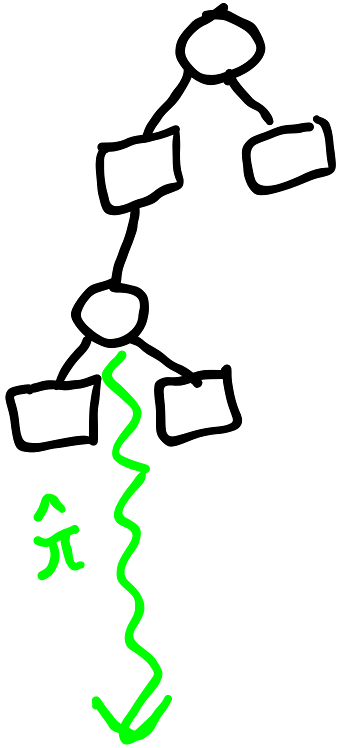
Search



Expansion



Rollout



Backup

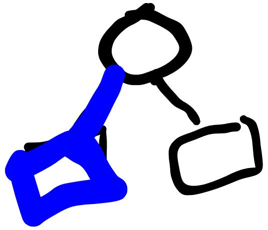
$$Q(s,a) + c\sqrt{\frac{\log N(s)}{N(s,a)}} \quad \text{or} \quad Q(s,a) + c\frac{N(s)^{\beta}}{\sqrt{N(s,a)}}$$

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

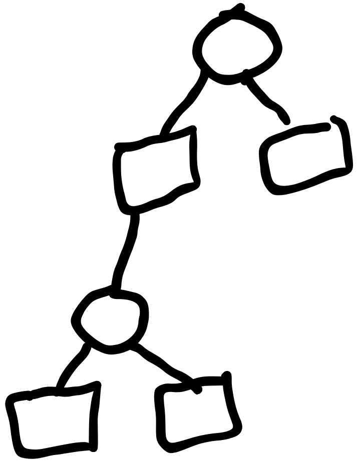
start with $c = 2(\bar{V} - \underline{V})$, $\beta = 1/4$

Monte Carlo Tree Search (MCTS/UCT)

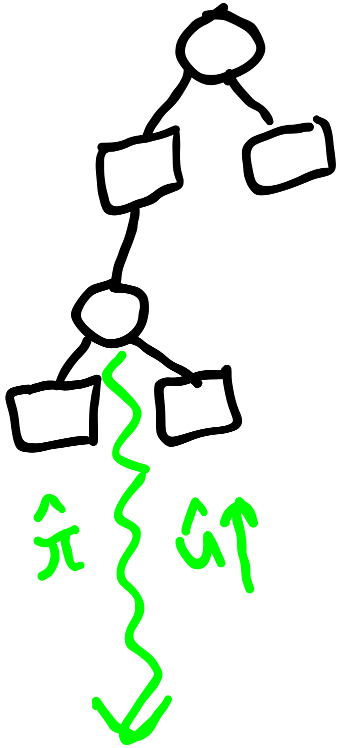
Search



Expansion



Rollout



Backup

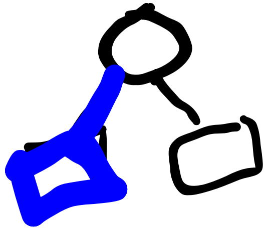
$$Q(s,a) + c\sqrt{\frac{\log N(s)}{N(s,a)}} \quad \text{or} \quad Q(s,a) + c\frac{N(s)^{\beta}}{\sqrt{N(s,a)}}$$

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

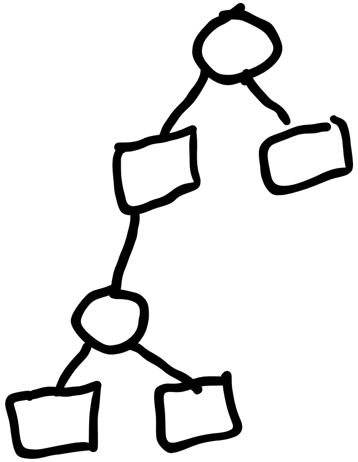
start with $c = 2(\bar{V} - \underline{V})$, $\beta = 1/4$

Monte Carlo Tree Search (MCTS/UCT)

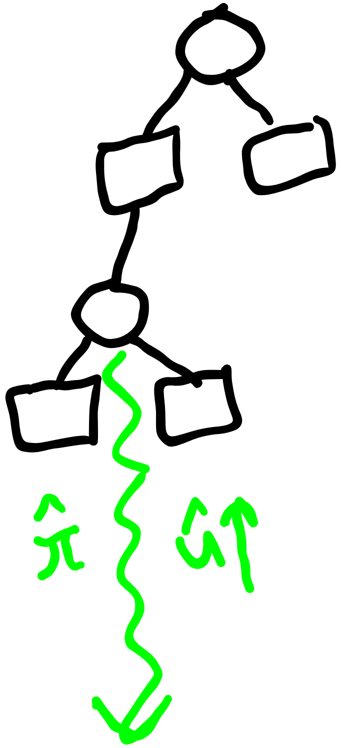
Search



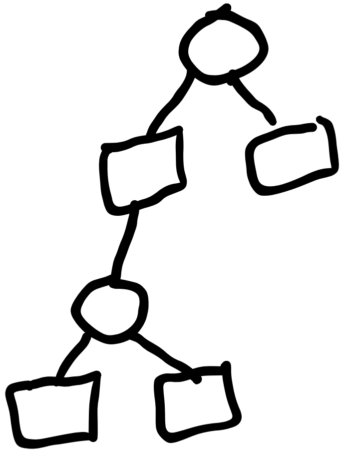
Expansion



Rollout



Backup



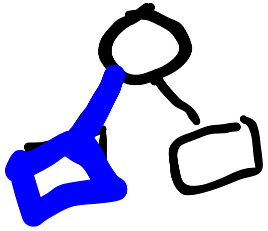
$$Q(s,a) + c\sqrt{\frac{\log N(s)}{N(s,a)}} \quad \text{or} \quad Q(s,a) + c\frac{N(s)^{\beta}}{\sqrt{N(s,a)}}$$

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

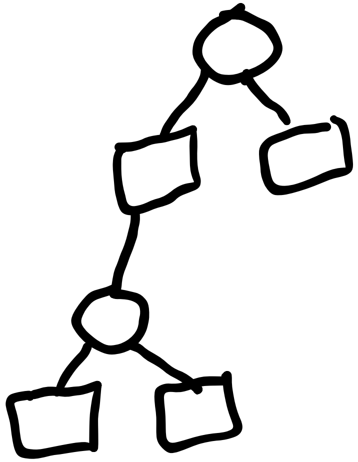
start with $c = 2(\bar{V} - \underline{V})$, $\beta = 1/4$

Monte Carlo Tree Search (MCTS/UCT)

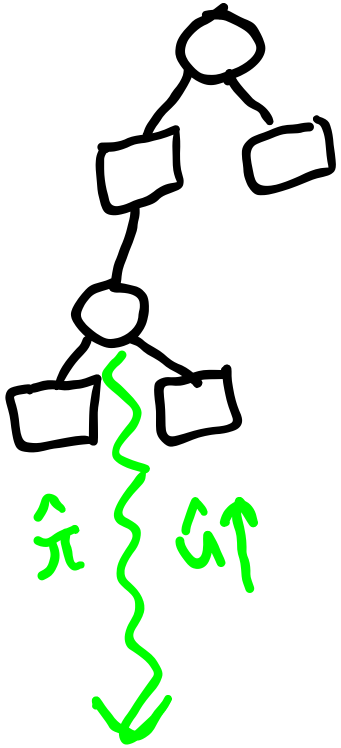
Search



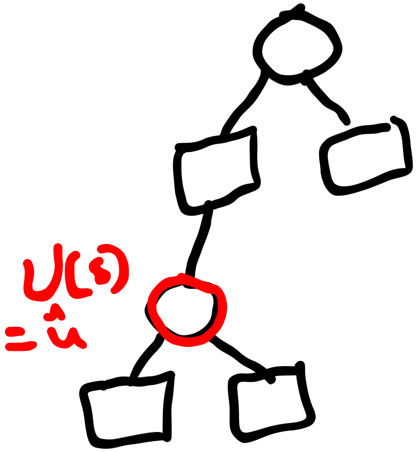
Expansion



Rollout



Backup



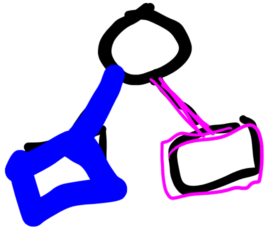
$$Q(s,a) + c\sqrt{\frac{\log N(s)}{N(s,a)}} \quad \text{or} \quad Q(s,a) + c\frac{N(s)^\beta}{\sqrt{N(s,a)}}$$

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

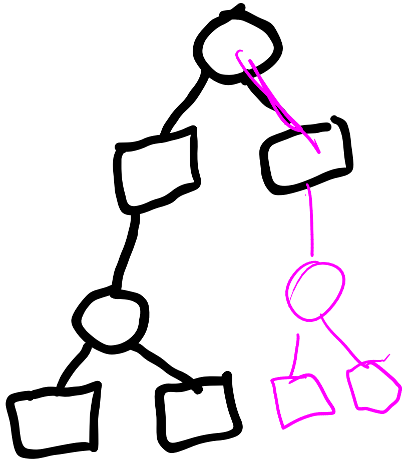
start with $c = 2(\bar{V} - \underline{V})$, $\beta = 1/4$

Monte Carlo Tree Search (MCTS/UCT)

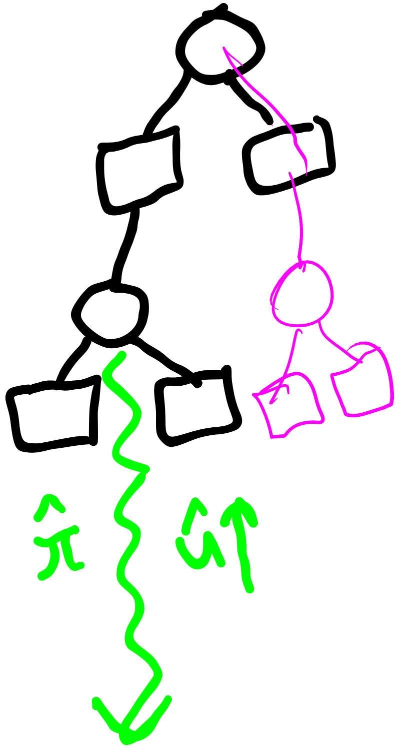
Search



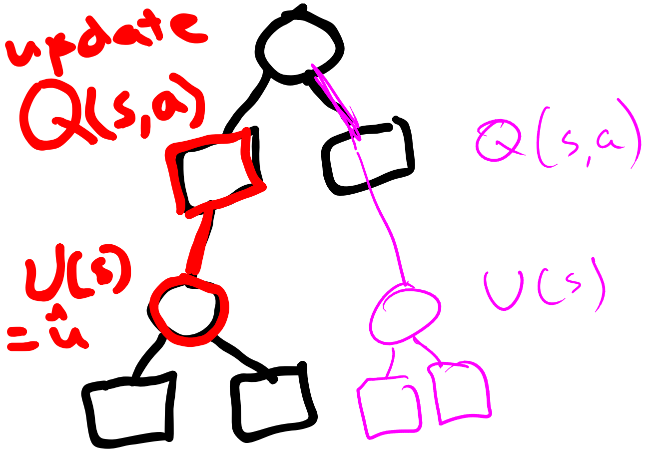
Expansion



Rollout



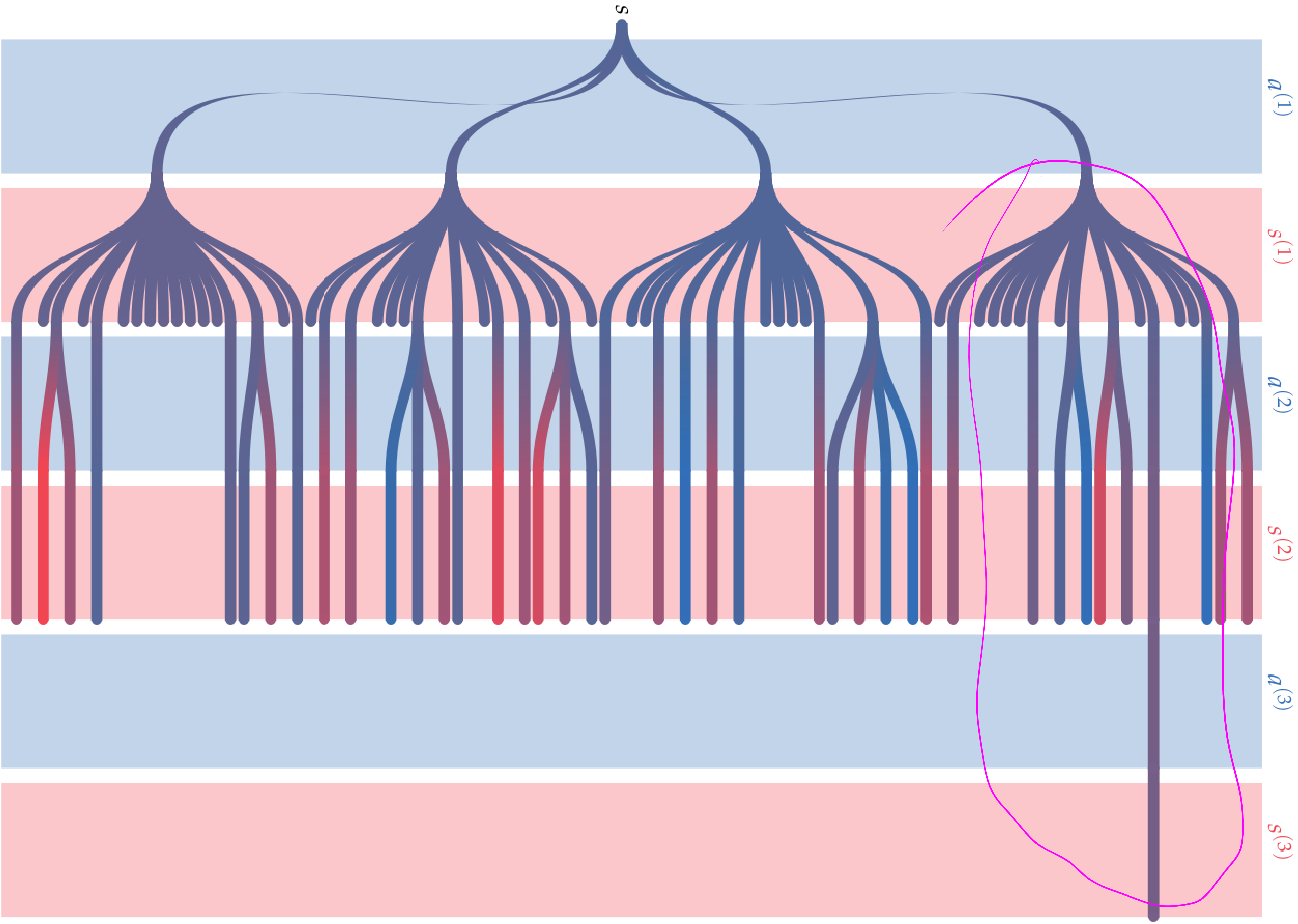
Backup



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

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

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


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?