Chapter 6

Deliberation with Probabilistic Domain Models

Dana S. Nau
University of Maryland
Motivation

- Situations where actions have multiple possible outcomes and each outcome has a probability

- Several possible action representations
  - Bayes nets, probabilistic actions, ...

- Book doesn’t commit to any representation
  - Mainly concentrates on the underlying semantics

roll-die(d)
pre: holding(d) = true
eff:
1/6:  top(d) ← 1
1/6:  top(d) ← 2
1/6:  top(d) ← 3
1/6:  top(d) ← 4
1/6:  top(d) ← 5
1/6:  top(d) ← 6
Probabilistic planning domain

Definitions
\[ \Sigma = (S, A, \gamma, \Pr, \text{cost}) \]
- \( S = \{\text{states}\} \)
- \( A = \{\text{actions}\} \)
- \( \gamma : S \times A \rightarrow 2^S \)
- \( \Pr(s' | s, a) = \text{probability of going to state } s' \text{ if we apply } a \text{ in } s \)
  - \( \Pr(s' | s, a) \neq 0 \text{ iff } s' \in \gamma(s, a) \)
- \( \text{cost} : S \times A \rightarrow \mathbb{R}_{\geq 0} \)
  - \( \text{cost}(s, a) = \text{cost of action } a \text{ in state } s \)
  - may omit, default is \( \text{cost}(s, a) = 1 \)
- \( \text{Applicable}(s) = \{a | \gamma(s, a) \neq \emptyset\} \)

Example
- Start at \( d1 \), want to get to \( d4 \)
- Some roads are one-way, some are two-way
- Unreliable steering, especially on hills
  - may slip and go elsewhere
- Simplified state and action names:
  - write \( \{\text{loc}(r1) = d2\} \) as \( d2 \)
  - write \( \text{move}(r1, d2, d3) \) as \( m23 \)
- \( \gamma(d1, m12) = \{d2\} \)
  - \( \Pr(d2 | d1, m12) = 1 \)
- \( m21, m34, m41, m43, m45, m52, m54: \)
  - like \( m12 \)
- \( \gamma(d1, m14) = \{d1, d4\} \)
  - \( \Pr(d4 | d1, m14) = 0.5 \)
  - \( \Pr(d1 | d1, m14) = 0.5 \)
- \( \gamma(d2, m23) = \{d3, d5\} \)
  - \( \Pr(d3 | d2, m23) = 0.8 \)
  - \( \Pr(d5 | d2, m23) = 0.2 \)
- No \( m11, \) no \( m25 \)
Probabilistic planning domain

Definitions
\[ \Sigma = (S, A, \gamma, \Pr, \text{cost}) \]
- \( S = \{\text{states}\} \)
- \( A = \{\text{actions}\} \)
- \( \gamma : S \times A \rightarrow 2^S \)
- \( \Pr(s' | s, a) = \text{probability of going to state } s' \text{ if we apply } a \text{ in } s \)
  - \( \Pr(s' | s, a) \neq 0 \iff s' \in \gamma(s,a) \)
- \( \text{cost: } S \times A \rightarrow \mathbb{R}_{\geq 0} \)
  - \( \text{cost}(s,a) = \text{cost of action } a \text{ in state } s \)
  - may omit, default is cost\( (s,a) = 1 \)
- \( \text{Applicable}(s) = \{a | \gamma(s,a) \neq \emptyset \} \)

Example
- \( \gamma(d1,m12) = \{d2\} \)
  - \( \Pr(d2 | d1,m12) = 1 \)
- \( m21, m34, m41, m43, m45, m52, m54: \)
  - like m12
- \( \gamma(d1,m14) = \{d1,d4\} \)
  - \( \Pr(d4 | d1,m14) = 0.5 \)
  - \( \Pr(d1 | d1,m14) = 0.5 \)
- \( \gamma(d2,m23) = \{d3,d5\} \)
  - \( \Pr(d3 | d2,m23) = 0.8 \)
  - \( \Pr(d5 | d2,m23) = 0.2 \)
- there’s no m25

Poll: Can a plan (sequence of actions) be a solution for this problem?
1. yes
2. no
Policies, Problems

- Same as in Chapter 5:

- **Policy**
  - partial function $\pi : S \rightarrow A$ such that
    - for every $s \in \text{Dom}(\pi) \subseteq S$,
      $\pi(s) \in \text{Applicable}(s)$

- **Transitive closure**
  - $\hat{\gamma}(s, \pi) = \{s \text{ and all states reachable from } s \text{ using } \pi\}$

- Graph($s, \pi$) = rooted graph induced by $\pi$ at $s$
  - nodes: $\hat{\gamma}(s, \pi)$
  - edges: state transitions

- $\text{leaves}(s, \pi) = \hat{\gamma}(s, \pi) \setminus \text{Dom}(\pi)$

$p_i = \{(d_1, m_{12}), (d_2, m_{23}), (d_3, m_{34})\}$

$\text{Dom}(p_i) = \{d_1, d_2, d_3\}$

$\hat{\gamma}(d_1, p_i) = \{d_1, d_2, d_3, d_4, d_5\}$

$\text{leaves}(d_1, p_i) = \hat{\gamma}(d_1, p_i) \setminus \text{Dom}(p_i)$

$= \{d_4, d_5\}$
Solutions

- **Stochastic shortest path (SSP) problem:**
  - a triple \((\Sigma, s_0, S_g)\)

- **Solution** for \((\Sigma, s_0, S_g)\):
  - A policy \(\pi\) for \(\Sigma\) such that \(\hat{\gamma}(s_0, \pi) \cap S_g \neq \emptyset\)

- Unlike Chapter 5, don’t require \(\pi\) to end at \(S_g\)

- A solution policy \(\pi\) is **closed** if it doesn’t stop at non-goal states unless there’s no way to continue
  - for every state in \(\hat{\gamma}(s, \pi)\), either
    - \(s \in \text{Dom}(\pi)\) (i.e., \(\pi(s)\) is defined)
    - or \(s \in S_g\)
    - or Applicable\((s) = \emptyset\)

- For the rest of this chapter we require all solutions to be closed

\[
\pi_1 = \{(d1, m12), (d2, m23), (d3, m34)\}
\]

\[
\pi_2 = \{(d1, m12), (d2, m23), (d3, m34), (d5, m54)\}
\]

\[
\pi_3 = \{(d1, m12), (d2, m23), (d3, m34), (d5, m56)\}
\]

\[
\pi_4 = \{(d1, m12), (d2, m23), (d3, m34), (d5, m57), (d7, m75)\}
\]
Histories

- **History**: sequence of states \( \sigma = \langle s_0, s_1, s_2, \ldots \rangle \)
  - May be finite or infinite
    \( \langle d_1, d_2, d_3, d_4 \rangle \)
    \( \langle d_1, d_2, d_1, d_2, \ldots \rangle \)

- Let \( H(s, \pi) = \) set of all possible histories if we start at \( s \) and follow \( \pi \)
  - Stop if we reach a state \( s' \) such that \( s' \notin \text{Dom}(\pi) \) or \( s' \in S_g \)

- If \( \sigma \in H(s, \pi) \) then
  - \( \Pr(\sigma | s, \pi) = \prod_{s_i, s_{i+1} \in \sigma} \Pr(s_{i+1} | s_i, \pi(s_i)) \)
  - Product of the probabilities of the state transitions
  - \( \sum_{\sigma \in H(s, \pi)} \Pr(\sigma | s, \pi) = 1 \)

\[\begin{align*}
\pi_3 &= \{(d_1,m_{12}), (d_2,m_{23}), (d_3,m_{34}), (d_5,m_{56})\} \\
H(s_0, \pi_3) &= \{\sigma_1, \sigma_2\}, \text{ where:} \\
&\quad \sigma_1 = \langle d_1, d_2, d_3, d_4 \rangle \\
&\quad \sigma_2 = \langle d_1, d_2, d_5, d_6 \rangle \\
&\quad \Pr(\sigma_1 | s_0, \pi_3) = 1 \times 0.8 \times 1 = 0.8 \\
&\quad \Pr(\sigma_2 | s_0, \pi_3) = 1 \times 0.2 \times 1 = 0.2
\end{align*}\]
Unsafe Solutions

- Probability of reaching a goal state:
  \[ \Pr(S_g \mid s, \pi) = \sum_{\sigma \in H(s, \pi)} \Pr(\sigma \mid s, \pi) \mid \sigma \text{ ends at a state in } S_g \}

- Formula in book is equivalent but more complicated

- A solution is unsafe if \( 0 < \Pr(S_g \mid s_0, \pi) < 1 \)

- \( \pi_3 = \{(d1,m12), (d2,m23), (d3,m34), (d5,m56)\} \)

- \( H(s_0, \pi_3) = \{\sigma_1, \sigma_2\} \), where:
  - \( \sigma_1 = \langle d1,d2,d3,d4 \rangle \)
  - \( \sigma_2 = \langle d1,d2,d5,d6 \rangle \)

- \( \Pr(\sigma_1 \mid s_0, \pi_3) = 1 \times 0.8 \times 1 = 0.8 \)
- \( \Pr(\sigma_2 \mid s_0, \pi_3) = 1 \times 0.2 \times 1 = 0.2 \)

- \( d6 \) is an explicit dead end
  - no applicable actions

- \( \Pr(S_g \mid s_0, \pi_3) = \Pr(\sigma_1 \mid s_0, \pi_1) = 0.8 \)
Unsafe Solutions

- Probability of reaching a goal state:
  - $\Pr(S_g \mid s, \pi) = \sum_{\sigma \in H(s, \pi)} \{\Pr(\sigma \mid s, \pi) \mid \sigma \text{ ends at a state in } S_g\}$

- Formula in book is equivalent but more complicated

- A solution is unsafe if $0 < \Pr(S_g \mid s_0, \pi) < 1$

- $\pi_4 = \{(d1,m12), (d2,m23), (d3,m34), (d5,m57), (d7,m75)\}$

- $H(s_0, \pi_3) = \{\sigma_1, \sigma_2\}$, where:
  - $\sigma_1 = \langle d1,d2,d3,d4 \rangle$
  - $\sigma_3 = \langle d1,d2,d5,d6,d5,d6,\ldots \rangle$

- $\Pr(\sigma_1 \mid s_0, \pi_2) = 1 \times .8 \times 1 = .8$
- $\Pr(\sigma_3 \mid s_0, \pi_2) = 1 \times .2 \times 1 \times 1 \times 1 \times 1 \times \ldots = .2$

- $\langle d5, d6, d5, d6, \ldots \rangle$ is an implicit dead end
  - Applicable actions, but no way to reach goal

- $\Pr(S_g \mid s_0, \pi_3) = \Pr(\sigma_1 \mid s_0, \pi_1) = 0.8$
Safe Solutions

• A solution is safe if $\Pr(S_g | s_0, \pi) = 1$

• An acyclic safe solution:
  ▷ $\pi_2 = \{(d_1, m_{12}), (d_2, m_{23}), (d_3, m_{34}), (d_5, m_{54})\}$

• $H(s_0, \pi_2)$ contains two histories:
  ▷ $\sigma_1 = (d_1, d_2, d_3, d_4)$  \hspace{1cm} $\Pr(\sigma_1 | s_0, \pi_2) = 1 \times .8 \times 1 = .8$
  ▷ $\sigma_4 = (d_1, d_2, d_5, d_4)$  \hspace{1cm} $\Pr(\sigma_4 | s_0, \pi_2) = 1 \times .2 \times 1 = .2$

• $\Pr(S_g | s_0, \pi_2) = .8 + .2 = 1$
\textbf{Safe Solutions}

- A solution is \textit{safe} if $\Pr(S_g|s_0, \pi) = 1$

- A cyclic safe solution:
  - $\pi_5 = \{d1, m14\}$

- $H(\pi_5)$ contains infinitely many histories:
  - $\sigma_5 = \langle d1, d4 \rangle$ \hspace{1cm} $\Pr(\sigma_5|s_0, \pi_5) = \frac{1}{2}$
  - $\sigma_6 = \langle d1, d1, d4 \rangle$ \hspace{1cm} $\Pr(\sigma_6|s_0, \pi_5) = \left(\frac{1}{2}\right)^2 = \frac{1}{4}$
  - $\sigma_7 = \langle d1, d1, d1, d4 \rangle$ \hspace{1cm} $\Pr(\sigma_7|s_0, \pi_5) = \left(\frac{1}{2}\right)^3 = \frac{1}{8}$
  - $\sigma_\infty = \langle d1, d1, d1, d1, d1, \ldots \rangle$

- $\Pr(S_g|s_0, \pi_5) = \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \ldots = 1$

\textbf{Poll: what is $\Pr(\sigma_\infty|s_0, \pi_5)$?}
1. 1
2. 0
3. a number between 0 and 1
4. undefined
Safe Solutions

- A solution is *safe* if $\Pr(S_g|s_0, \pi) = 1$

- Another cyclic safe solution:
  $\pi_6 = \{(d1, m54), (d4, m41)\}$

- We stop when we reach a goal, so $(d4, m41)$ doesn’t matter
  - Same histories and probabilities as for $\pi_5$
- \( \text{cost}(s,a) = \text{cost of using } a \text{ in } s \)
- **Example:**
  - each “horizontal” action costs 1
  - each “vertical” action costs 100
- Let \( \sigma = \langle s_0, s_1, s_2, \ldots \rangle \in H(s_0, \pi) \)
  - \( \text{cost}(\sigma \mid s_0, \pi) = \sum \{ \text{cost}(s_i, \pi(s_i)) \mid s_i, \pi(s_i) \in \sigma \} \)
- Let \( \pi \) be a safe solution
- At each state \( s \in \text{Dom}(\pi) \), expected cost of following \( \pi \) to goal:
  - Weighted sum of history costs:
    - \( V_{\pi}(s) = \sum_{\sigma \in H(s, \pi)} \Pr(\sigma \mid s, \pi) \text{cost}(\sigma \mid s, \pi) \)
  - Recursive equation
    - \( V_{\pi}(s) = \begin{cases} 
    0, & \text{if } s \in S_g \\
    \text{cost}(s, \pi(s)) + \sum_{s' \in \gamma(s, \pi(s))} \Pr(s' \mid s, \pi(s)) V_{\pi}(s'), & \text{otherwise}
    \end{cases} \)

**Poll:** Which is correct?
1. weighted sum of history costs
2. recursive equation
3. both
4. neither

**Poll:** Is this needed?
1. yes
2. no

**Goal:**
\( S_g = \{d_4\} \)
Example

- \( \pi_3 = \{(d_1, m_{12}), (d_2, m_{23}), (d_3, m_{34}), (d_5, m_{54})\} \)

- Weighted sum of history costs:
  - \( \sigma_1 = \langle d_1, d_2, d_3, d_4 \rangle \)
    - \( \Pr(\sigma_1 | s_0, \pi_3) = 0.8 \)
    - \( \text{cost}(\sigma_1 | s_0, \pi_3) = 100 + 1 + 100 = 201 \)
  - \( \sigma_2 = \langle d_1, d_2, d_5, d_4 \rangle \)
    - \( \Pr(\sigma_2 | s_0, \pi_3) = 0.2 \)
    - \( \text{cost}(\sigma_2 | s_0, \pi_3) = 100 + 1 + 100 = 201 \)

- \( V^{\pi_3}(d_1) = 0.8(201) + 0.2(201) = 201 \)

- Recursive equation:
  \[
  V^{\pi_3}(d_1) = 100 + 1(V^{\pi_3}(d_2)) \\
  = 100 + 1 + 0.8(V^{\pi_3}(d_3)) + 0.2(V^{\pi_3}(d_5)) \\
  = 100 + 1 + 0.8(100) + 0.2(100) \\
  = 201
  \]
### Example

- \( \pi_7 = \{ (d1, m14), (d2, m23), (d3, m34), (d5, m54) \} \)

- Weighted sum of history costs:
  - \( \sigma_5 = \langle d1, d4 \rangle \)
    \[
    \begin{align*}
    \Pr(\sigma_5 | \pi_7) &= \frac{1}{2} \\
    \text{cost}(\sigma_5 | \pi_7) &= 1
    \end{align*}
    \]
  - \( \sigma_6 = \langle d1, d1, d4 \rangle \)
    \[
    \begin{align*}
    \Pr(\sigma_6 | \pi_7) &= \left(\frac{1}{2}\right)^2 \\
    \text{cost}(\sigma_6 | \pi_7) &= 2
    \end{align*}
    \]
  - \( \sigma_7 = \langle d1, d1, d1, d4 \rangle \)
    \[
    \begin{align*}
    \Pr(\sigma_7 | \pi_7) &= \left(\frac{1}{2}\right)^3 \\
    \text{cost}(\sigma_7 | \pi_7) &= 3
    \end{align*}
    \]
  
  ...  

- \( V^{\pi_7}(d1) = (\frac{1}{2})1 + (\frac{1}{2})^2 2 + (\frac{1}{2})^3 3 + \ldots = 2 \)

- Recursive equation:
  \[
  V^{\pi_7}(d1) = 1 + \frac{1}{2}(0) + \frac{1}{2}(V^{\pi_7}(d1))
  \]
  
  \( \frac{1}{2}V^{\pi_7}(d1) = 1 \)
  
  \( V^{\pi_7}(d1) = 2 \)

---

- Recursive equation is easier computationally
- Given safe solution \( \pi \),
  - Compute \( V^{\pi} \) by solving \( n \) linear equations, \( n \) unknowns
  - \( n = |\hat{\gamma}(s_0,\pi)| \)
Planning as Optimization

- Let $\pi$ and $\pi'$ be safe solutions
  - $\pi$ dominates $\pi'$ if at every state $s$ where they’re both defined (i.e., $s \in \text{Dom}(\pi) \cap \text{Dom}(\pi')$), $V^\pi(s) \leq V^{\pi'}(s)$
  - On the previous two slides
    - $\pi_3 = \{(d1,m12), (d2,m23), (d3,m34), (d5,m54)\}$
    - $\pi_7 = \{(d1, m14), (d2, m23), (d3, m34), (d5, m54)\}$
    - $d1$ is the only state where they differ
      - $V^{\pi_3}(d1) = 201$; $V^{\pi_7}(d1) = 2$
    - $\pi_7$ dominates $\pi_3$

- $\pi$ is optimal if $\pi$ dominates every safe solution
- If $\pi$ and $\pi'$ are both optimal, then $V^\pi(s) = V^{\pi'}(s)$ at every state where they’re both defined
  - On slide 11, $\pi_5 = \{(d1,m14)\}$
  - $\pi_5$ and $\pi_7$ both dominate each other
Planning as Optimization

- Recall:
  \[ V^\pi(s) = \begin{cases} 
  0, & \text{if } s \text{ is a goal} \\
  \text{cost}(s, \pi(s)) + \sum_{s' \in \gamma(s, \pi(s))} \Pr(s' | s, \pi(s)) V^\pi(s'), & \text{otherwise} 
\end{cases} \]

- Let \( V^*(s) \) = expected cost using an optimal safe solution

- *Optimality principle* (Bellman’s theorem):

  \[ V^*(s) = \begin{cases} 
  0, & \text{if } s \text{ is a goal} \\
  \min_{a \in \text{Applicable}(s)} \left\{ \text{cost}(s, a) + \sum_{s' \in \gamma(s, a)} \Pr(s' | s, a) V^*(s') \right\}, & \text{otherwise} 
\end{cases} \]

- Intuition:
  - At state d1
    - Applicable actions m12 and m14
  - Suppose we know
    - \( V^*(d2), V^*(d4), V^*(d6) \)
  - What’s the best choice at d1?

\[ V^*(d2) = 115 \]
\[ V^*(d6) = 1 \]
Cost to Go

- Let \((\Sigma, s_0, S_g)\) be a safe SSP
  - i.e., \(S_g\) is reachable from every state
  - same as safely exploriable in Chapter 5

- Let \(\pi\) be a safe solution that’s defined at all non-goal states
  - i.e., \(\text{Dom}(\pi) = S \setminus S_g\)

- Compute \(V^\pi\) as \(|S|\) equations, \(|S|\) unknowns

- Let \(s \in S, a \in \text{Applicable}(s)\)
  - Cost-to-go:
    - expected cost at \(s\) if we first use \(a\),
      then use \(\pi\) afterward
  - \(Q^\pi(s,a) = \text{cost}(s,a) + \sum_{s' \in \gamma(s,a')} \Pr(s' | s,a') \cdot V^\pi(s')\)

- For every \(s \in S \setminus S_g\)
  let \(\pi'(s) \in \arg\min_{a \in \text{Applicable}(s)} Q^\pi(s,a')\)

Poll: Does \(\pi'\) dominate \(\pi\)?
1. always
2. sometimes
3. never
Example

\[ \pi = \{ (d_1,m_{12}), (d_2,m_{23}), (d_3,m_{34}), (d_5,m_{54}) \} \]

\[ V^{\pi}(d_4) = 0 \]
\[ V^{\pi}(d_3) = 100 + V^{\pi}(d_4) = 100 \]
\[ V^{\pi}(d_5) = 100 + V^{\pi}(d_4) = 100 \]
\[ V^{\pi}(d_2) = 1 + (0.8 V^{\pi}(d_3) + 0.2 V^{\pi}(d_5)) \]
\[ = 101 \]
\[ V^{\pi}(d_1) = 100 + V^{\pi}(d_2) = 201 \]

\[ Q^{\pi}(d_1,m_{12}) = 100 + 101 = 201 \]
\[ Q^{\pi}(d_1,m_{14}) = 1 + \frac{1}{2}(201) + \frac{1}{2}(0) = 101.5 \]
\[ \arg\min_a Q^{\pi}(d_1,a) = m_{14} \]
\[ Q^{\pi}(d_2,m_{23}) = 1 + (0.8(100) + 0.2(100)) = 101 \]
\[ Q^{\pi}(d_2,m_{21}) = 1 + 101 = 301 \]
\[ \arg\min_a Q^{\pi}(d_2,a) = m_{23} \]
\[ Q^{\pi}(d_3,m_{34}) = 100 + 0 = 100 \]
\[ Q^{\pi}(d_3,m_{32}) = 1 + 101 = 102 \]
\[ \arg\min_a Q^{\pi}(d_3,a) = m_{34} \]
\[ Q^{\pi}(d_5,m_{54}) = 100 + 0 = 100 \]
\[ Q^{\pi}(d_5,m_{52}) = 1 + 101 = 102 \]
\[ \arg\min_a Q^{\pi}(d_5,a) = m_{54} \]

\[ \pi' = \{ (d_1,m_{14}), (d_2,m_{23}), (d_3,m_{34}), (d_5,m_{54}) \} \]

\[ V^{\pi'}(d_4) = 0 \]
\[ V^{\pi'}(d_3) = 100 + V^{\pi'}(d_4) = 100 \]
\[ V^{\pi'}(d_5) = 100 + V^{\pi'}(d_4) = 100 \]
\[ V^{\pi'}(d_2) = 1 + (0.8 V^{\pi'}(d_3) + 0.2 V^{\pi'}(d_5)) \]
\[ = 101 \]
\[ V^{\pi'}(d_1) = 1 + \frac{1}{2} V^{\pi'}(d_1) + \frac{1}{2} V^{\pi'}(d_4) \]
\[ \Rightarrow V^{\pi'}(d_1) = 2 \]
Policy Iteration

- \( \text{PI}(\Sigma, s_0, S_g, \pi_0) \)
  \[
  \pi \leftarrow \pi_0
  \]
  loop
  compute \( \{ V^\pi(s) \mid s \in S \} \) \( n \) equations, \( n \) unknowns, where \( n = |S| \)
  for every \( s \in S \setminus S_g \) do
    \( \pi'(s) \leftarrow \text{argmin}_{a \in \text{Applicable}(s)} Q^\pi(s, a) \)
    \( E(\text{cost of using } a \text{ then } \pi) \)
    if \( \pi' = \pi \) then
      return \( \pi \)
  \( \pi \leftarrow \pi' \)

- Converges in a finite number of iterations
- Example:

\[
\begin{align*}
\text{Start: } s_0 &= d1 \\
\text{Goal: } S_g &= \{d4\}
\end{align*}
\]
Value Iteration

- $V_0$ is a heuristic function
  - e.g., adapt an $h$ heuristic from Chapter 2
  - for each $s \in S$
    - $V_0(s)$ is an estimate of the expected cost of getting from $s$ to a goal state
    - must have $V_0(s) = 0$ for every $s \in S_g$
- $\eta > 0$: for testing approximate convergence

- $V$ is an array
  - $V(s)$ = updated estimate of expected cost of getting from $s$ to a goal state
- $\pi$ = policy computed from $V$

- Difference from the book:
  - In the book, $VI$ computes $r$ as a separate step, not in $Bellman$-$Update$

$\text{VI}(\Sigma, s_0, S_g, V_0, \eta)$
- global $V \leftarrow V_0$
- global $\pi \leftarrow \emptyset$
- loop
  - $r \leftarrow \max \{\text{Bellman-Update}(s) \mid s \in S \setminus S_g\}$
  - if $r \leq \eta$ then return $\pi$

$\text{Bellman-Update}(s)$
- global $V, \pi$
- $v_{old} \leftarrow V(s)$
- for every $a \in \text{Applicable}(s)$ do
  - $Q(s,a) \leftarrow \text{cost}(s,a) + \sum_{s' \in S} \Pr(s'|s,a) V(s')$
  - $V(s) \leftarrow \min_{a \in \text{Applicable}(s)} Q(s,a)$
  - $\pi(s) \leftarrow \arg\min_{a \in \text{Applicable}(s)} Q(s,a)$
- return $|V(s) - v_{old}|$
VI(\(\Sigma, s_0, S_g, V_0, \eta\))

global \(V \leftarrow V_0\)
global \(\pi \leftarrow \emptyset\)

loop

\(r \leftarrow \max\{\text{Bellman-Update}(s) \mid s \in S \setminus S_g\}\)

if \(r \leq \eta\) then return \(\pi\)

Bellman-Update\((s)\)

global \(V, \pi\)

\(v_{\text{old}} \leftarrow V(s)\)

for every \(a \in \text{Applicable}(s)\) do

\(Q(s,a) \leftarrow \text{cost}(s,a) + \sum_{s' \in S} \Pr(s'|s,a) V(s')\)

\(V(s) \leftarrow \min_{a \in \text{Applicable}(s)} Q(s,a)\)

\(\pi(s) \leftarrow \arg\min_{a \in \text{Applicable}(s)} Q(s,a)\)

return \(|V(s) - v_{\text{old}}|\)
Iteration 2

\[ Q(d_1, m_{12}) = 100 + 1 = 101 \]
\[ Q(d_1, m_{14}) = 1 + \left( \frac{1}{2}(1) + \frac{1}{2}(0) \right) = 1 \]
\[ V(d_1) = 1; \pi(d_1) = m_{14} \]

\[ Q(d_2, m_{21}) = 100 + 1 \frac{1}{2} = 101 \frac{1}{2} \]
\[ Q(d_2, m_{23}) = 1 + .8(2) + .2(2) = 3 \]
\[ V(d_2) = 3; \pi(d_2) = m_{23} \]

\[ Q(d_3, m_{32}) = 1 + 3 = 4 \]
\[ Q(d_3, m_{34}) = 100 + 0 = 100 \]
\[ V(d_3) = 4; \pi(d_3) = m_{32} \]

\[ Q(d_5, m_{52}) = 1 + 3 = 4 \]
\[ Q(d_5, m_{54}) = 100 + 0 = 100 \]
\[ V(d_5) = 4; \pi(d_5) = m_{52} \]

\[ r = \max(1 \frac{1}{2} - 1, 3 - 1, \quad 4 - 2, 4 - 2) = 2 \]

\[ \eta = 0.2 \]
\[ V(d_1) = 1 \]
\[ V(d_2) = 1 \]
\[ V(d_3) = 2 \]
\[ V(d_5) = 2 \]
\[ \pi(d_1) = m_{14} \]
\[ \pi(d_2) = m_{23} \]
\[ \pi(d_2) = m_{32} \]
\[ \pi(d_5) = m_{52} \]
\[ V(\Sigma, s_0, S_g, V_0, \eta) \]

global \( V \leftarrow V_0 \)
global \( \pi \leftarrow \emptyset \)

loop

\[ r \leftarrow \max \{ \text{Bellman-Update}(s) \mid s \in S \setminus S_g \} \]
if \( r \leq \eta \) then return \( \pi \)

Bellman-Update(\( s \))

global \( V, \pi \)
\( v_{\text{old}} \leftarrow V(s) \)

for every \( a \in \text{Applicable}(s) \) do

\[ Q(s,a) \leftarrow \text{cost}(s,a) + \sum_{s' \in S} \Pr(s'|s,a) V(s') \]

\( V(s) \leftarrow \min_{a \in \text{Applicable}(s)} Q(s,a) \)

\( \pi(s) \leftarrow \text{argmin}_{a \in \text{Applicable}(s)} Q(s,a) \)

return \( |V(s) - v_{\text{old}}| \)

---

**Iteration 3**

\[ \eta = 0.2 \]
\[ V(d1) = 1\frac{1}{2} \]
\[ V(d2) = 3 \]
\[ V(d3) = 4 \]
\[ V(d5) = 4 \]
\[ \pi(d1) = m14 \]
\[ \pi(d2) = m23 \]
\[ \pi(d2) = m32 \]
\[ \pi(d5) = m52 \]

\[ Q(d1,m12) = 100 + 3 = 103 \]
\[ Q(d1,m14) = 1 + (\frac{1}{2}(1\frac{1}{2}) + \frac{1}{2}(0)) = 1\frac{3}{4} \]
\[ V(d1) = 1\frac{3}{4}; \pi(d1) = m14 \]

\[ Q(d2,m21) = 100 + 1\frac{3}{4} = 101\frac{3}{4} \]
\[ Q(d2,m23) = 1 + .8(4) + .2(4) = 5 \]
\[ V(d2) = 5; \pi(d2) = m23 \]

\[ Q(d3,m32) = 1 + 5 = 6 \]
\[ Q(d3,m34) = 100 + 0 = 100 \]
\[ V(d3) = 6; \pi(d3) = m32 \]

\[ Q(d5,m52) = 1 + 5 = 6 \]
\[ Q(d5,m54) = 100 + 0 = 100 \]
\[ V(d5) = 6; \pi(d5) = m52 \]

\[ r = \max(1\frac{3}{4} - 1\frac{1}{2}, 5 - 3, 6 - 4, 6 - 4) = 2 \]
\[ Q(d_1, m_{12}) = 100 + 5 = 105 \]
\[ Q(d_1, m_{14}) = 1 + \left( \frac{1}{2} \left( \frac{3}{4} \right) + \frac{1}{2} (0) \right) = \frac{17}{8} \]
\[ V(d_1) = \frac{17}{8} ; \pi(d_1) = m_{14} \]
\[ Q(d_2, m_{21}) = 100 + \frac{17}{8} = 101\frac{7}{8} \]
\[ Q(d_2, m_{23}) = 1 + .8(6) + .2(6) = 7 \]
\[ V(d_2) = 7 ; \pi(d_2) = m_{23} \]
\[ Q(d_3, m_{32}) = 1 + 7 = 8 \]
\[ Q(d_3, m_{34}) = 100 + 0 = 100 \]
\[ V(d_3) = 8 ; \pi(d_3) = m_{32} \]
\[ Q(d_5, m_{52}) = 1 + 7 = 8 \]
\[ Q(d_5, m_{54}) = 100 + 0 = 100 \]
\[ V(d_5) = 8 ; \pi(d_5) = m_{52} \]
\[ r = \max(1\frac{7}{8} - 1\frac{3}{4}, 7 - 5, 8 - 6, 8 - 6) = 2 \]
Iteration 1, with a better $V_0$

$V(I(s_0, S_g, V_0, \eta))$

global $V \leftarrow V_0$
global $\pi \leftarrow \emptyset$

loop

$r \leftarrow \max \{\text{Bellman-Update}(s) \mid s \in S \setminus S_g\}$

if $r \leq \eta$ then return $\pi$

Bellman-Update($s$)

global $V, \pi$

$v_{\text{old}} \leftarrow V(s)$

for every $a \in \text{Applicable}(s)$ do

$Q(s,a) \leftarrow \text{cost}(s,a) + \sum_{s' \in S} \Pr(s'|s,a) \; V(s')$

$V(s) \leftarrow \min_{a \in \text{Applicable}(s)} Q(s,a)$

$\pi(s) \leftarrow \text{argmin}_{a \in \text{Applicable}(s)} Q(s,a)$

return $|V(s) - v_{\text{old}}|$

$\eta = 0.2$

$V(d1) = 2$
$V(d2) = 101$
$V(d3) = 100$
$V(d5) = 100$

$Q(d1,m12) = 100 + 101 = 201$

$Q(d1,m14) = 1 + (0.5(0) + 0.5(2)) = 2$

$V(d1) = 2; \; \pi(d1) = m14$

$Q(d2,m21) = 100 + 2 = 102$

$Q(d2,m23) = 1 + 0.8(100) + 0.2(100) = 101$

$V(d2) = 101; \; \pi(d2) = m23$

$Q(d3,m32) = 1 + 101 = 102$

$Q(d3,m34) = 100 + 0 = 100$

$V(d3) = 100; \; \pi(d3) = m34$

$Q(d5,m52) = 1 + 101 = 102$

$Q(d5,m54) = 100 + 0 = 100$

$V(d5) = 100; \; \pi(d5) = m54$

$r = \max(0, 0, 0, 0) = 0 < \eta$

$V(I)$ returns $\pi$

Nau – Lecture slides for Automated Planning and Acting
Discussion

- Policy iteration computes new $\pi$ in each iteration; computes $V^\pi$ from $\pi$
  - More work per iteration than value iteration
    - Needs to solve a set of simultaneous equations
    - Usually converges in a smaller number of iterations
- Value iteration
  - Computes new $V$ in each iteration; chooses $\pi$ based on $V$
    - New $V$ is a revised set of heuristic estimates
      - Not $V^\pi$ for $\pi$ or any other policy
  - Less work per iteration: doesn’t need to solve a set of equations
  - Usually takes more iterations to converge

- At each iteration, both algorithms need to examine the entire state space
  - Number of iterations polynomial in $|S|$, but $|S|$ may be quite large
- Next: use search techniques to avoid searching the entire space
Suppose \( \text{Frontier} = \{s_4, s_6\} \) and \( f(s_4) < f(s_6) \)

- expand \( s_4 \)
- if its children have larger \( f \) values than \( f(s_6) \)
  then expand \( s_6 \) next
Equivalent Approach Using Policies

- At each state $s$,
  - $\pi(s) = $ the action that currently looks best
  - $V(s) = $ estimated cost of following $\pi$ to goal

\[
\begin{align*}
V(s_5) &= c(a_6) + V(s_6) \\
V(s_6) &= h(s_6)
\end{align*}
\]

- Expand $s_4$, update $V(s_4)$, update $V(s_3)$
- If $c(a_3) + V(s_3) > c(a_5) + V(s_5)$, then revise $\pi(s_2)$
- Update $V(s_2)$, $V(s_1)$, $V(s_0)$

- AO*: generalization of A* for acyclic SSPs
  - Updating like above, but trees rather than paths
AO* Search

- An acyclic SSP can be represented as an AND/OR graph
  - OR nodes: choose an action
  - AND nodes: action’s outcomes
    - $V(s_0) = c(a_1) + V(s_1) + V(s_2)$
    - $V(s_1) = c(a_3) + V(s_5) + V(s_6)$
    - $V(s_2) = c(a_4) + V(s_7) + V(s_8)$

- leaves($s_0, \pi$) = \{s_5, s_6, s_7, s_8\}
- Expand one of them, e.g., s_5
- Going bottom-up:
  - Update $V$ and $\pi$ values for $s_5$ and its ancestors
AO*($\Sigma, s_0, S_g, V_0$)

global $\pi \leftarrow \emptyset$

global $Envelope \leftarrow \{s_0\}$

global $V; V(s_0) \leftarrow V_0(s_0)$

while ($\gamma(s_0, \pi) \setminus S_g) \cap Fringe \neq \emptyset$ do

select $s \in (\gamma(s_0, \pi) \setminus S_g) \cap Fringe$

for all $a \in Applicable(s)$ and $s' \in \gamma(s, a)$ do

if $s' \in \gamma(s, a) \notin Envelope$ then

add $s'$ to $Envelope$

$V(s') \leftarrow V_0(s')$

AO-Update($s$)

return $\pi$

AO-Update($s$)  // update $V$ and $\pi$ values of $s$ and its ancestors

$Z \leftarrow \{s\}$  // nodes that need updating

while $Z \neq \emptyset$ do

select $s \in Z$ such that $\gamma(s, \pi(s)) \cap Z = \{s\}$

remove $s$ from $Z$

Bellman-Update($s$)

$Z \leftarrow Z \cup \{s' \in Envelope \mid s \in \gamma(s', \pi)\}$

add the states “just above” $s$

Bellman-Update($s$)

global $V, \pi$

$v_{old} \leftarrow V(s)$

for every $a \in Applicable(s)$ do

$Q(s, a) \leftarrow cost(s, a) + \sum_{s' \in S} Pr(s'|s, a) V(s')$

$V(s) \leftarrow \min_{a \in Applicable(s)} Q(s, a)$

$\pi(s) \leftarrow \arg\min_{a \in Applicable(s)} Q(s, a)$

return $|V(s) - v_{old}|$

AO* is similar to Expanded $\cup$ Frontier in A*

Fringe $\equiv$ Envelope $\setminus$ Dom($\pi$)

• like Frontier in A*, but updated in Bellman-Update

AO-Update($s$)  // update $V$ and $\pi$ values of $s$ and its ancestors

$Z \leftarrow \{s\}$  // nodes that need updating

while $Z \neq \emptyset$ do

select $s \in Z$ such that $\gamma(s, \pi(s)) \cap Z = \{s\}$

remove $s$ from $Z$

Bellman-Update($s$)

$Z \leftarrow Z \cup \{s' \in Envelope \mid s \in \gamma(s', \pi)\}$

add the states “just above” $s$

Bellman-Update($s$)

global $V, \pi$

$v_{old} \leftarrow V(s)$

for every $a \in Applicable(s)$ do

$Q(s, a) \leftarrow cost(s, a) + \sum_{s' \in S} Pr(s'|s, a) V(s')$

$V(s) \leftarrow \min_{a \in Applicable(s)} Q(s, a)$

$\pi(s) \leftarrow \arg\min_{a \in Applicable(s)} Q(s, a)$

return $|V(s) - v_{old}|$

AO* (Sigma, s0, Sg, V0)

local pi <- ∅

local Envelope <- \{s0\}

local V; V(s0) <- V0(s0)

while (γ(s0, π) \ Sg) \ Cap Fringe \= ∅ do

select s ∈ (γ(s0, π) \ Sg) \ Cap Fringe

for all a ∈ Applicable(s) and s’ ∈ γ(s, a) do

if s’ ∈ γ(s, a) \ Envelope then

add s’ to Envelope

V(s’) <- V0(s’)

AO-Update(s)

return π

AO-Update(s)  // update V and π values of s and its ancestors

Z <- \{s\}  // nodes that need updating

while Z \= ∅ do

select s ∈ Z such that γ(s, π(s)) \ Cap Z = \{s\}

remove s from Z

Bellman-Update(s)

Z <- Z \∪ \{s’ ∈ Envelope \mid s ∈ γ(s’, π)\}

add the states “just above” s

Bellman-Update(s)

global V, π

v_{old} <- V(s)

for every a ∈ Applicable(s) do

Q(s, a) <- cost(s, a) + ∑_{s’ ∈ S} Pr(s’|s, a) V(s’)

V(s) <- min_{a ∈ Applicable(s)} Q(s, a)

π(s) <- argmin_{a ∈ Applicable(s)} Q(s, a)

return |V(s) - v_{old}|

not needed this time

• like Frontier in A*, but updated in Bellman-Update

Fringe ≡ Envelope \ Dom(π)

AO* (Sigma, s0, Sg, V0)

local pi <- ∅

local Envelope <- \{s0\}

local V; V(s0) <- V0(s0)

while (γ(s0, π) \ Sg) \ Cap Fringe \= ∅ do

select s ∈ (γ(s0, π) \ Sg) \ Cap Fringe

for all a ∈ Applicable(s) and s’ ∈ γ(s, a) do

if s’ ∈ γ(s, a) \ Envelope then

add s’ to Envelope

V(s’) <- V0(s’)

AO-Update(s)

return π

AO-Update(s)  // update V and π values of s and its ancestors

Z <- \{s\}  // nodes that need updating

while Z \= ∅ do

select s ∈ Z such that γ(s, π(s)) \ Cap Z = \{s\}

remove s from Z

Bellman-Update(s)

Z <- Z \∪ \{s’ ∈ Envelope \mid s ∈ γ(s’, π)\}

add the states “just above” s

Bellman-Update(s)

global V, π

v_{old} <- V(s)

for every a ∈ Applicable(s) do

Q(s, a) <- cost(s, a) + ∑_{s’ ∈ S} Pr(s’|s, a) V(s’)

V(s) <- min_{a ∈ Applicable(s)} Q(s, a)

π(s) <- argmin_{a ∈ Applicable(s)} Q(s, a)

return |V(s) - v_{old}|

not needed this time

• like Frontier in A*, but updated in Bellman-Update

Fringe ≡ Envelope \ Dom(π)

AO* (Sigma, s0, Sg, V0)

local pi <- ∅

local Envelope <- \{s0\}

local V; V(s0) <- V0(s0)

while (γ(s0, π) \ Sg) \ Cap Fringe \= ∅ do

select s ∈ (γ(s0, π) \ Sg) \ Cap Fringe

for all a ∈ Applicable(s) and s’ ∈ γ(s, a) do

if s’ ∈ γ(s, a) \ Envelope then

add s’ to Envelope

V(s’) <- V0(s’)

AO-Update(s)

return π

AO-Update(s)  // update V and π values of s and its ancestors

Z <- \{s\}  // nodes that need updating

while Z \= ∅ do

select s ∈ Z such that γ(s, π(s)) \ Cap Z = \{s\}

remove s from Z

Bellman-Update(s)

Z <- Z \∪ \{s’ ∈ Envelope \mid s ∈ γ(s’, π)\}

add the states “just above” s

Bellman-Update(s)

global V, π

v_{old} <- V(s)

for every a ∈ Applicable(s) do

Q(s, a) <- cost(s, a) + ∑_{s’ ∈ S} Pr(s’|s, a) V(s’)

V(s) <- min_{a ∈ Applicable(s)} Q(s, a)

π(s) <- argmin_{a ∈ Applicable(s)} Q(s, a)

return |V(s) - v_{old}|

not needed this time

• like Frontier in A*, but updated in Bellman-Update

Fringe ≡ Envelope \ Dom(π)
AO* $(\Sigma, s_0, S_g, V_0)$

global $\pi \leftarrow \emptyset$

global $Envelope \leftarrow \{s_0\}$

global $V$; $V(s_0) \leftarrow V_0(s_0)$

while $(\hat{\gamma}(s_0, \pi) \setminus S_g) \cap Fringe \neq \emptyset$ do

select $s \in (\hat{\gamma}(s_0, \pi) \setminus S_g) \cap Fringe$

for all $a \in \text{Applicable}(s)$ and $s' \in \gamma(s, a)$ do

if $s' \in \gamma(s, a) \notin Envelope$ then

add $s'$ to $Envelope$

$V(s') \leftarrow V_0(s')$

AO-Update($s$)

return $\pi$

AO-Update($s$) // update $V$ and $\pi$ values of $s$ and its ancestors

$Z \leftarrow \{s\}$ // nodes that need updating

while $Z \neq \emptyset$ do

select $s \in Z$ such that $\hat{\gamma}(s, \pi(s)) \cap Z = \{s\}$

remove $s$ from $Z$

Bellman-Update($s$)

$Z \leftarrow Z \cup \{s' \in Envelope | s \in \gamma(s', \pi)\}$

Bellman-Update($s$)

global $V, \pi$

$v_{old} \leftarrow V(s)$

for every $a \in \text{Applicable}(s)$ do

$Q(s, a) \leftarrow \text{cost}(s, a) + \sum_{s' \in S} \text{Pr}(s'|s, a) \cdot V(s')$

$V(s) \leftarrow \min_{a \in \text{Applicable}(s)} Q(s, a)$

$\pi(s) \leftarrow \text{argmin}_{a \in \text{Applicable}(s)} Q(s, a)$

return $|V(s) - v_{old}|$

Example: $V_0(s) = 0$ for all $s$

Start: $s_0 = d1$

Goal: $S_g = \{d4\}$
**AO***(\(\Sigma, s_0, S_g, V_0\))

- Global \(\pi \leftarrow \emptyset\)
- Global \(\text{Envelope} \leftarrow \{s_0\}\)
- Global \(V; V(s_0) \leftarrow V_0(s_0)\)

While \((\gamma(s_0, \pi) \setminus S_g) \cap \text{Fringe} \neq \emptyset\) do

- Select \(s \in (\gamma(s_0, \pi) \setminus S_g) \cap \text{Fringe}\)
- For all \(a \in \text{Applicable}(s)\) and \(s' \in \gamma(s, a)\) do
  - If \(s' \in \gamma(s, a) \notin \text{Envelope}\) then
    - Add \(s'\) to \(\text{Envelope}\)
    - \(V(s') \leftarrow V_0(s')\)
- \(\text{AO-Update}(s)\)

**AO-Update(s)** // update \(V\) and \(\pi\) values of \(s\) and its ancestors

- \(Z \leftarrow \{s\}\) // nodes that need updating
- While \(Z \neq \emptyset\) do
  - Select \(s \in Z\) such that \(\gamma(s, \pi(s)) \cap Z = \{s\}\)
  - Remove \(s\) from \(Z\)
  - Bellman-Update(s)
  - \(Z \leftarrow Z \cup \{s' \in \text{Envelope} | s \in \gamma(s', \pi)\}\)

Bellman-Update(s)

- Global \(V, \pi\)
- \(v_{\text{old}} \leftarrow V(s)\)
- For every \(a \in \text{Applicable}(s)\) do
  - \(Q(s, a) \leftarrow \text{cost}(s, a) + \sum_{s' \in S} \text{Pr}(s'|s, a) \cdot V(s')\)
  - \(V(s) \leftarrow \min_{a \in \text{Applicable}(s)} Q(s, a)\)
- \(\pi(s) \leftarrow \text{argmin}_{a \in \text{Applicable}(s)} Q(s, a)\)
- Return \(|V(s) - v_{\text{old}}|\)

**Example:** \(V_0(s) = 0\) for all \(s\)

Start: \(s_0 = d1\)

Goal: \(S_g = \{d4\}\)
AO* ($\Sigma, s_0, S_g, V_0$)

- `global $\pi \leftarrow \emptyset$`
- `global $Envelope \leftarrow \{s_0\}$`
- `global $V$; $V(s_0) \leftarrow V_0(s_0)$`

while ($\hat{\gamma}(s_0, \pi) \setminus S_g \cap Fringe \neq \emptyset$) do
  select $s \in (\hat{\gamma}(s_0, \pi) \setminus S_g \cap Fringe$
  for all $a \in \text{Applicable}(s)$ and $s' \in \gamma(s, a)$ do
    if $s' \in \gamma(s, a) \notin Envelope$ then
      add $s'$ to $Envelope$
      $V(s') \leftarrow V_0(s')$
  AO-Update($s$)

return $\pi$

AO-Update($s$) // update $V$ and $\pi$ values of $s$ and its ancestors

$Z \leftarrow \{s\}$ // nodes that need updating

while $Z \neq \emptyset$ do
  select $s \in Z$ such that $\hat{\gamma}(s, \pi(s)) \cap Z = \{s\}$
  remove $s$ from $Z$
  Bellman-Update($s$)

$Z \leftarrow Z \cup \{s' \in Envelope \mid s \in \gamma(s', \pi)\}$

Bellman-Update($s$)

- `global $V, \pi$`
- `$v_{old} \leftarrow V(s)$`

for every $a \in \text{Applicable}(s)$ do
  $Q(s, a) \leftarrow \text{cost}(s, a) + \sum_{s' \in S} \text{Pr}(s' | s, a) \times V(s')$
  $V(s) \leftarrow \min_{a \in \text{Applicable}(s)} Q(s, a)$
  $\pi(s) \leftarrow \arg\min_{a \in \text{Applicable}(s)} Q(s, a)$

return $|V(s) - v_{old}|$

Example: $V_0(s) = 0$ for all $s$

Start: $s_0 = d_1$

Goal: $S_g = \{d_4\}$

\[ V(d_1) = 10 \]
\[ \pi(d_1) = m_{14} \]
\[ V(d_2) = 15 \]
\[ \pi(d_2) = m_{23} \]
\[ V(d_3) = 0 \]
\[ V(d_4) = 0 \]
\[ V(d_5) = 0 \]
\[ V(d_6) = 0 \]
AO* ($\Sigma, s_0, S_g, V_0$)

- Global $\pi \leftarrow \emptyset$
- Global $Envelope \leftarrow \{ s_0 \}$
- Global $V$; $V(s_0) \leftarrow V_0(s_0)$
- While $(\hat{\gamma}(s_0, \pi) \setminus S_g) \cap Fringe \neq \emptyset$
  - Select $s \in (\hat{\gamma}(s_0, \pi) \setminus S_g) \cap Fringe$
  - For all $a \in \text{Applicable}(s)$ and $s' \in \gamma(s, a)$ do
    - If $s' \in \gamma(s, a) \notin Envelope$ then
      - Add $s'$ to $Envelope$
      - $V(s') \leftarrow V_0(s')$
  - AO-Update($s$)

AO-Update($s$)  // update $V$ and $\pi$ values of $s$ and its ancestors
- $Z \leftarrow \{ s \}$  // nodes that need updating
- While $Z \neq \emptyset$
  - Select $s \in Z$ such that $\hat{\gamma}(s, \pi(s)) \cap Z = \{ s \}$
  - Remove $s$ from $Z$
  - Bellman-Update($s$)
- $Z \leftarrow Z \cup \{ s' \in Envelope \mid s \in \gamma(s', \pi) \}$

Bellman-Update($s$)

- Global $V, \pi$
- $v_{old} \leftarrow V(s)$
- For every $a \in \text{Applicable}(s)$ do
  - $Q(s, a) \leftarrow \text{cost}(s, a) + \sum_{s' \in S} \Pr(s'|s, a) \cdot V(s')$
  - $V(s) \leftarrow \min_{a \in \text{Applicable}(s)} Q(s, a)$
  - $\pi(s) \leftarrow \arg\min_{a \in \text{Applicable}(s)} Q(s, a)$
- Return $|V(s) - v_{old}|$

Example:

- $V(d_1) = 15$
- $\pi(d_1) = m_{23}$
- $V(d_2) = 15$
- $\pi(d_2) = m_{23}$
- $V(d_3) = 0$
- $\pi(d_3) = m_{23}$
- $V(d_4) = 0$
- $\pi(d_4) = m_{14}$
- $V(d_5) = 0$
- $\pi(d_5) = m_{12}$

Start: $s_0 = d_1$
Goal: $S_g = \{ d_4 \}$

Example: $V_0(s) = 0$ for all $s$
What to do about dead ends?

AO* \((\Sigma, s_0, S_g, V_0)\)

- global \(\pi \leftarrow \emptyset\)
- global \(Envelope \leftarrow \{s_0\}\)
- global \(V; V(s_0) \leftarrow V_0(s_0)\)

while \((\hat{\gamma}(s_0, \pi) \setminus S_g) \cap Fringe \neq \emptyset\) do
  - select \(s \in (\hat{\gamma}(s_0, \pi) \setminus S_g) \cap Fringe\)
  - for all \(a \in \text{Applicable}(s)\) and \(s' \in \gamma(s,a)\) do
    - if \(s' \in \gamma(s,a) \notin Envelope\) then
      - add \(s'\) to \(Envelope\)
      - \(V(s') \leftarrow V_0(s')\)
  - AO-Update(s)

return \(\pi\)

AO-Update(s) // update \(V\) and \(\pi\) values of \(s\) and its ancestors

\(Z \leftarrow \{s\}\) // nodes that need updating

while \(Z \neq \emptyset\) do
  - select \(s \in Z\) such that \(\hat{\gamma}(s, \pi(s)) \cap Z = \{s\}\)
  - remove \(s\) from \(Z\)
  - Bellman-Update(s)

\(Z \leftarrow Z \cup \{s' \in Envelope \mid s \in \gamma(s', \pi)\}\)

Recall how Guided-Find-Safe-Solution (Chap. 5) made actions inapplicable if they went to dead ends? AO* can be modified to do this too.
Heuristics through Determinization

What to use for $V_0$?

- classical heuristic function $h$ on the determinized domain $\Sigma_d$

- If $h$ is admissible for $\Sigma_d$ then also admissible for $\Sigma$

- Why:
  - Let $\pi$ be any optimal solution for $\Sigma$
  - Let $p$ be any acyclic execution of $\pi$
    - $p$ is a plan in $\Sigma_d$
    - $h(s_0) \leq \text{cost}(p) \leq \text{cost}(\pi)$
Heuristics through Determinization

What to use for $V_0$?

- Call classical planner on the determinized problem
  - Get plan $p = \langle a_1, a_2, \ldots, a_n \rangle$
  - Return $V_0(s) = \text{cost}(p)$

- If the classical planner always returns optimal plans, then $V_0$ is admissible

- Outline of proof:
  - Let $\pi$ be any solution in $\Sigma$
  - Every execution of $\pi$ corresponds to a solution plan $p'$ in $\Sigma_d$
  - Suppose $\text{cost}(p') < V_0(s)$
    - then the planner would have chosen $p'$ instead of $p$
  - Thus $\text{cost}(p') \geq V_0(s)$

\[ m_{23} = 0.2 \quad m_{23_1} = 15 \quad m_{23_2} = 0.8 \]
\[ m_{12} = c = 10 \quad m_{12} = 100 \quad m_{14} = 0.5 \quad m_{14_1} = 0.5 \quad m_{14_2} = 20 \]
\[ m_{34} = c = 1 \quad m_{34} = 1 \quad m_{54} = c = 100 \]
LAO* \((\Sigma, s_0, S_g, V_0)\)

\[
global \pi \leftarrow \emptyset \\
global Envelope \leftarrow \{s_0\} \\
global V; V(s_0) \leftarrow V_0(s_0) \\
while (\hat{\gamma}(s_0, \pi) \setminus S_g) \cap Fringe \neq \emptyset do \\
select s \in (\hat{\gamma}(s_0, \pi) \setminus S_g) \cap Fringe \\
for all \ a \in \text{Applicable}(s) and s' \in \gamma(s, a) do \\
if s' \in \gamma(s, a) \notin Envelope then \\
add s' to Envelope \\
V(s') \leftarrow V_0(s') \\
LAO-Update(s) \\
return \pi
\]

LAO-Update(s)

\[
Z \leftarrow \{s\} \cup \{s' \in Envelope \mid s \in \hat{\gamma}(s', \pi)\} \\
loop until \ r \leq \eta \ or \ new \ states \ added \ to \ (\hat{\gamma}(s_0, \pi) \setminus S_g) \cap Fringe \\
r \leftarrow \max \{\text{Bellman-Update}(s) \mid s \in Z\}
\]

Bellman-Update(s)

\[
global V, \pi \\
v_{\text{old}} \leftarrow V(s) \\
for every \ a \in \text{Applicable}(s) \ do \\
Q(s, a) \leftarrow \text{cost}(s, a) + \sum_{s' \in S} \Pr(s' | s, a) \ V(s') \\
V(s) \leftarrow \min_{a \in \text{Applicable}(s)} Q(s, a) \\
\pi(s) \leftarrow \arg\min_{a \in \text{Applicable}(s)} Q(s, a) \\
return |V(s) - v_{\text{old}}|
\]

Example: \(V_0(s) = 0\) for all \(s\)

Start: \(s_0 = d1\)

Goal: \(S_g = \{d4\}\)
Example

LAO* ($\Sigma, s_0, S_g, V_0$)

Global $\pi \leftarrow \emptyset$

Global $\text{Envelope} \leftarrow \{s_0\}$

Global $V$; $V(s_0) \leftarrow V_0(s_0)$

while ($\hat{\gamma}(s_0, \pi) \setminus S_g$) $\cap$ Fringe $\neq \emptyset$ do

select $s \in (\hat{\gamma}(s_0, \pi) \setminus S_g) \cap$ Fringe

for all $a \in \text{Applicable}(s)$ and $s' \in \gamma(s, a)$ do

if $s' \in \gamma(s, a) \notin \text{Envelope}$ then

add $s'$ to $\text{Envelope}$

$V(s') \leftarrow V_0(s')$

end

end

LAO-Update(s)

return $\pi$

LAO-Update(s)

$Z \leftarrow \{s\} \cup \{s' \in \text{Envelope} \mid s \in \hat{\gamma}(s', \pi)\}$

loop until $r \leq \eta$ or new states added to $\hat{\gamma}(s_0, \pi) \cap$ Fringe

$r \leftarrow \max\{\text{Bellman-Update}(s) \mid s \in Z\}$

Bellman-Update(s)

Global $V$, $\pi$; $v_{old} \leftarrow V(s)$

for every $a \in \text{Applicable}(s)$ do

$Q(s, a) \leftarrow \text{cost}(s, a) + \sum_{s' \in S} \Pr (s'|s, a) \cdot V(s')$

$V(s) \leftarrow \min_{a \in \text{Applicable}(s)} Q(s, a)$

$\pi(s) \leftarrow \text{argmin}_{a \in \text{Applicable}(s)} Q(s, a)$

return $|V(s) - v_{old}|$

return $\pi$

$\eta = 0.2$

$V_0(s) = 0$ for all $s$

Envelope = $\{d_1\}$

Iteration 1 of LAO*'s loop:

select $s = d_1$

Applicable(d1) = $\{m_{12}, m_{14}\}$

add $d_2$ to $\text{Envelope}$; $V(d_2) \leftarrow 0$

add $d_4$ to $\text{Envelope}$; $V(d_4) \leftarrow 0$

Call LAO-Update(d1)

$\pi$ is empty, so $Z = \{d_1\}$

Iteration 1 of LAO-Update's loop:

Call Bellman-update(d1):

$Q(d_1, m_{12}) = 100 + 0 = 100$

$Q(d_1, m_{14}) = 1 + (\frac{1}{2}(0) + \frac{1}{2}(0)) = 1$

$V(d_1) = 1; \pi(d_1) = m_{14}$

$r = V(d_1) - 0 = 1$

Keep iterating until $r \leq 0.2$

$V(d_1) = 1.875; r = 0.0625$

Iteration 2 of LAO*'s loop:

($\hat{\gamma}(s_0, \pi) \setminus S_g$) $\cap$ Fringe $= \emptyset$

so return $\pi = \{(d_1, m_{14})\}$
Nau – Lecture slides for Automated Planning and Acting

\[ \text{Modified Example} \]

\[ \text{Goal: } S_g = \{d4\} \]

**Iteration 1 of L AO \(^*\) 's loop:**
- select \( s = d1 \); Applicable\( (d1) = \{m12, m14\} \)
- \( V(d2) \leftarrow 0 \); add \( d2 \) to \( \text{Envelope} \)
- \( V(d4) \leftarrow 0 \); add \( d4 \) to \( \text{Envelope} \)

Call L AO-Update\( (d1) \)
- \( \pi \) is empty, so \( Z = \{d1\} \)

**Iteration 1 of L AO-Update 's loop:**
- Call Bellman-update\( (d1) \):
  - \( Q(d1,m12) = 100 + 0 = 100 \)
  - \( Q(d1,m14) = 80 + \left( \frac{1}{2}(0) + \frac{1}{2}(0) \right) = 80 \)
  - \( V(d1) = 80; \pi(d1) = m14 \)
- \( r = V(d1) - 0 = 80 \)

**Iteration 2 of L AO-Update 's loop:**
- Call Bellman-update\( (d1) \):
  - \( Q(d1,m12) = 100 + 0 = 100 \)
  - \( Q(d1,m14) = 80 + \left( \frac{1}{2}(80) + \frac{1}{2}(0) \right) = 120 \)
  - \( V(d1) = 100; \pi(d1) = m12 \)
- \( r = V(d1) - 80 = 20 \)

State \( d2 \) added to \( (\hat{\gamma}(s_0,\pi) \setminus S_g) \cap \text{Fringe} \)

L AO-Update returns

After more iterations, L AO\(^*\) eventually returns
- \( \pi = \{d1,m12\}, \{d2,m23\}, \{d3,m34\} \)
Skipping Ahead

- Skipping ILAO*, HDP, LDFS$_{a}$, LRTDP, SLATE
  - I’ll come back to these if there’s time
Planning and Acting

Differences:
- Takes explicit starting state $s_0$
  - Not necessary, could observe $s_0$ instead
- Doesn’t abstract $s$ (to simplify the presentation)
- Lookahead returns an action instead of a plan

What to use for Lookahead?
- AO*, LAO*, …
  - Modify to search part of the space
- Classical planner searching a determinized domain
  - next page
- Stochastic sampling algorithms

Run-Lookahead($\Sigma, g$) // Chapter 2
$s \leftarrow$ abstraction of observed state $\xi$
while $s \neq g$ do
  $\pi \leftarrow$ Lookahead($\Sigma, s, g$)
  if $\pi =$ failure then return failure
  $a \leftarrow$ pop-first-action($\pi$); perform($a$)
  $s \leftarrow$ abstraction of observed state $\xi$

Run-Lookahead($\Sigma, s_0, S_g$) // Chapter 6
$s \leftarrow s_0$
while $s \not\in S_g$ and Applicable($s$) $\neq \emptyset$ do
  $a \leftarrow$ Lookahead($s, \theta$)
  perform action $a$
  $s \leftarrow$ observe resulting state
Planning and Acting

FS-Replan($\Sigma, s, S_g$)

$\pi_d \leftarrow \emptyset$

while $s \not\in S_g$ and Applicable($s$) $\neq \emptyset$ do

if $\pi_d(s)$ is undefined then do

$\pi_d \leftarrow \text{Plan2policy(Forward-search ($\Sigma_d, s, S_g$))}$

if $\pi_d = \text{failure}$ then return failure

perform action $\pi_d(s)$

$s \leftarrow \text{observe resulting state}$

- FS-Replan (Chapter 5)
  - Run-Lazy-lookahead, with Lookahead = classical planner on determinized domain
  - Generalization of FF-Replan (which used FastForward)
- Problem: classical planner may choose a plan that depends on low-probability outcome
- RFF algorithm (see book) attempts to alleviate this
Multi-Arm Bandit Problem

- Statistical model of sequential experiments
  - Name comes from a traditional slot machine (one-armed bandit)
- Multiple actions $a_1, a_2, \ldots, a_n$
  - Each $a_i$ provides a reward from an unknown probability distribution $p_i$
  - Assume each $p_i$ is stationary
    - Same every time, regardless of history
- Objective: maximize expected utility of a sequence of actions
- Exploitation vs exploration dilemma:
  - *Exploitation*: choose action that has given you high rewards in the past
  - *Exploration*: choose action that’s less familiar, in hopes that it might produce a higher reward
UCB (Upper Confidence Bound) Algorithm

- Assume all rewards are between 0 and 1
  - If they aren’t, normalize them
- For each action $a$, let
  - $r(a) =$ average reward you’ve gotten from $a$
  - $n(a) =$ number of times you’ve tried $a$
  - $n_t = \sum_a n(a)$
  - $Q(a) = r(a) + \sqrt{2 \ln n_t / n(a)}$

UCB:
  - if there are untried actions:
    - $\tilde{a} \leftarrow$ any untried action
  - else:
    - $\tilde{a} \leftarrow \operatorname{argmax}_a Q(a)$
  - perform $\tilde{a}$
  - update $r(\tilde{a}), n(\tilde{a}), n_t, Q(a)$

Theorem (given some assumptions):
As the number of calls to UCB $\to \infty$,
UCB’s choice at each call $\to$ optimal choice

<table>
<thead>
<tr>
<th>Actions:</th>
<th>a1</th>
<th>a2</th>
<th>a3</th>
</tr>
</thead>
<tbody>
<tr>
<td>No. of tries:</td>
<td>n(a1) = 5</td>
<td>n(a2) = 3</td>
<td>n(a3) = 2</td>
</tr>
<tr>
<td>Rewards:</td>
<td>r(a1) = 0.4</td>
<td>0.3333</td>
<td>0</td>
</tr>
<tr>
<td>Q values:</td>
<td>Q(a1) = 1.35971</td>
<td>1.5723</td>
<td>1.5174</td>
</tr>
<tr>
<td>Payoffs:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1st</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2nd</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3rd</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4th</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5th</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6th</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7th</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8th</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9th</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10th</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
UCT Algorithm

- **UCT\((s,h)\)**: recursive UCB computation on an SSP
  - Adapted for minimization rather than maximization
- **Monte Carlo rollout**:
  - At \(s\), choose action \(\tilde{a}\) using UCB computation
    - Perform \(\tilde{a}\), get state \(s'\)
    - Do the same thing recursively at \(s'\)
    - Continue until reaching a goal, dead end, or depth \(h\)
  - At each state visited, keep statistics on choices, utilities
Using UCT Offline

\[
\text{loop} \\
\quad \text{call } UCT(s_0, \infty) \\
\quad \forall s, \pi(s) \leftarrow \text{argmax}\{Q(s, a) | a \in \text{Applicable}(s)\}
\]

- As number of calls to UCT $\rightarrow \infty$, $\pi$ converges to optimal
  - Problem: finding optimal $\pi$ may take many iterations
- Better for online planning (e.g., Run-Lookahead)
  - $\pi(s_0)$ approaches optimal must faster than the rest of $\pi$
Using UCT Online

- Lookahead procedure for Run-Lookahead:
  - call UCT\((s, h)\) multiple times at current state \(s\)
  - e.g., until
    - allotted time runs out, or
    - \(\max\{\text{last change in } Q(s, a) \mid a \in \text{Applicable}(s)\} \leq \eta\)
  - return \(\arg\max\{Q(s, a) \mid a \in \text{Applicable}(s)\}\)

\[
\text{UCT}(s, h) \\
\text{if } s \in S_g \text{ then return } 0 \text{ if } h = 0 \text{ then return } V_0(s) \text{ if } s \notin \text{Envelope} \text{ then do} \text{ add } s \text{ to } \text{Envelope} \quad n(s) \leftarrow 0 \\
\quad \text{for all } a \in \text{Applicable}(s) \text{ do} \quad Q(s, a) \leftarrow 0; \quad n(s, a) \leftarrow 0 \quad Untried \leftarrow \{a \in \text{Applicable}(s) \mid n(s, a) = 0\} \quad \text{if } Untried \neq \emptyset \text{ then } \tilde{a} \leftarrow \text{Choose}(\text{Untried}) \quad \text{else } \tilde{a} \leftarrow \arg\min_{a \in \text{Applicable}(s)} \{Q(s, a) - C \times \lfloor \log(n(s))/n(s, a) \rfloor^{1/2} \} \quad s' \leftarrow \text{Sample}(\Sigma, s, \tilde{a}) \quad \text{cost-rollout} \leftarrow \text{cost}(s, \tilde{a}) + \text{UCT}(s', h - 1) \quad Q(s, \tilde{a}) \leftarrow [n(s, \tilde{a}) \times Q(s, \tilde{a}) + \text{cost-rollout}] / (1 + n(s, \tilde{a})) \quad n(s) \leftarrow n(s) + 1 \quad n(s, \tilde{a}) \leftarrow n(s, \tilde{a}) + 1 \quad \text{return cost-rollout}
\]
Using UCT Online

- Lookahead procedure for Run-Lazy-Lookaahead:
  - loop:
    - call UCT many times at current state
    - At state \( s \) visited, \( \pi(s) \leftarrow \) action with highest expected utility
  - Problem: the farther you follow \( \pi \), the less likely that \( \pi(s) \) is optimal
    - Near the bottom of the tree, \( \pi(s) \) might be \( \approx \) random choice
  - Possible workaround?
    - Modify Run-Lazy-Lookahead to call UCT more frequently

\[
\text{UCT}(s, h)
\]
if \( s \in S_g \) then return 0
if \( h = 0 \) then return \( V_0(s) \)
if \( s \notin Envelope \) then do
  add \( s \) to \( Envelope \)
  \( n(s) \leftarrow 0 \)
  for all \( a \in \text{Applicable}(s) \) do
    \( Q(s, a) \leftarrow 0; \ n(s, a) \leftarrow 0 \)
  \( \text{Untried} \leftarrow \{ a \in \text{Applicable}(s) \mid n(s, a) = 0 \} \)
  if \( \text{Untried} \neq \emptyset \) then \( \tilde{a} \leftarrow \text{Choose}(\text{Untried}) \)
  else \( \tilde{a} \leftarrow \text{argmin}_{a \in \text{Applicable}(s)} \{ Q(s, a) - C \times \left[ \log(n(s)) / n(s, a) \right]^{\frac{1}{2}} \} \)
  \( s' \leftarrow \text{Sample}(\Sigma, s, \tilde{a}) \)
  \( \text{cost-rollout} \leftarrow \text{cost}(s, \tilde{a}) + \text{UCT}(s', h - 1) \)
  \( Q(s, \tilde{a}) \leftarrow [n(s, \tilde{a}) \times Q(s, \tilde{a}) + \text{cost-rollout}] / (1 + n(s, \tilde{a})) \)
  \( n(s) \leftarrow n(s) + 1 \)
  \( n(s, \tilde{a}) \leftarrow n(s, \tilde{a}) + 1 \)
return \( \text{cost-rollout} \)
Using UCT with a Simulator

- Suppose you don’t know the probabilities and costs
  - But you have a fast, accurate simulator for the environment
- Lookahead procedure (see previous slides)
  - Run UCT multiple times in the simulated environment
  - Learn state-transition probabilities, expected utilities

\[
\text{UCT}(s, h) \\
\text{if } s \in S_g \text{ then return } 0 \\
\text{if } h = 0 \text{ then return } V_0(s) \\
\text{if } s \notin \text{Envelope} \text{ then do} \\
\quad \text{add } s \text{ to } \text{Envelope} \\
\quad n(s) \leftarrow 0 \\
\quad \text{for all } a \in \text{Applicable}(s) \text{ do} \\
\quad \quad Q(s, a) \leftarrow 0; \; n(s, a) \leftarrow 0 \\
\quad \text{Untried} \leftarrow \{a \in \text{Applicable}(s) \mid n(s, a) = 0\} \\
\text{if } \text{Untried} \neq \emptyset \text{ then } \tilde{a} \leftarrow \text{Choose}(\text{Untried}) \\
\text{else } \tilde{a} \leftarrow \text{argmin}_{a \in \text{Applicable}(s)} \{Q(s, a) - C \times [\log(n(s))/n(s, a)]^{1/2}\} \\
\quad s' \leftarrow \text{Sample}(\Sigma, s, \tilde{a}) \\
\quad \text{cost-rollout} \leftarrow \text{cost}(s, \tilde{a}) + \text{UCT}(s', h - 1) \\
\quad Q(s, \tilde{a}) \leftarrow [n(s, \tilde{a}) \times Q(s, \tilde{a}) + \text{cost-rollout}]/(1 + n(s, \tilde{a})) \\
\quad n(s) \leftarrow n(s) + 1 \\
\quad n(s, \tilde{a}) \leftarrow n(s, \tilde{a}) + 1 \\
\text{return cost-rollout }
\]
Using UCT for Exploration

- Suppose you don’t know the probabilities and costs
- Suppose you can restart your actor as many times as you want
  - Caveat: usually not very feasible in real environments
- Can modify UCT to be an acting procedure, use it to explore the environment

\[
\text{UCT}(s, h) \\
\text{if } s \in S_g \text{ then return } 0 \\
\text{if } h = 0 \text{ then return } V_0(s) \\
\text{if } s \notin Envelope \text{ then do} \\
\quad \text{add } s \text{ to } Envelope \\
\quad n(s) \leftarrow 0 \\
\quad \text{for all } a \in \text{Applicable}(s) \text{ do} \\
\quad \quad Q(s, a) \leftarrow 0; \ n(s, a) \leftarrow 0 \\
\quad \text{Untried} \leftarrow \{a \in \text{Applicable}(s) \mid n(s, a) = 0\} \\
\text{if } \text{Untried} \neq \emptyset \text{ then } \tilde{a} \leftarrow \text{Choose(Untried)} \\
\text{else } \tilde{a} \leftarrow \text{argmin}_{a \in \text{Applicable}(s)} \{Q(s, a) - C \times \log(n(s))/n(s, a)\}^{1/2} \\
\text{s'} \leftarrow \text{Sample}(\Sigma, s, \tilde{a}) \\
\text{cost-rollout} \leftarrow \text{cost}(s, \tilde{a}) + \text{UCT}(s', h - 1) \\
\text{Q}(s, \tilde{a}) \leftarrow [n(s, \tilde{a}) \times Q(s, \tilde{a}) + \text{cost-rollout}]/(1 + n(s, \tilde{a})) \\
\text{n}(s) \leftarrow n(s) + 1 \\
\text{n}(s, \tilde{a}) \leftarrow n(s, \tilde{a}) + 1 \\
\text{return } \text{cost-rollout}
\]
UCT in Two-Player Games

- Generate Monte Carlo rollouts using a modified version of UCT
- Main differences:
  - Instead of choosing actions that minimize accumulated cost, choose actions that maximize payoff at the end of the game
  - UCT for player 1 recursively calls UCT for player 2
    - Choose opponent’s action
  - UCT for player 2 recursively calls UCT for player 1
- First competent computer programs for go
  - \( \approx 2008-2012 \)
- Monte Carlo rollout techniques similar to UCT were used to train AlphaGo
Summary

- SSPs
- solutions, closed solutions, histories
- unsafe solutions, acyclic safe solutions, cyclic safe solutions
- expected cost, planning as optimization
- policy iteration
- value iteration (asynchronous version)
  - Bellman-update
- AO*, LAO*
- Planning and Acting
  - Run-Lookahead
  - FS-Replan
- UCB, UCT