Automated Planning and Acting
Probabilistic Models

Institute of Information Systems

Dr. Mattis Hartwig
1. Planning and Acting with Deterministic Models
2. Planning and Acting with Refinement Methods
3. Planning and Acting with Temporal Models
4. Planning and Acting with Nondeterministic Models
5. Standard Decision Making
6. Planning and Acting with Probabilistic Models
   a. Stochastic Shortest-Path Problems
   b. Heuristic Search Algorithms
   c. Online Approaches Including Reinforcement Learning
7. Advanced Decision Making
8. Human-aware Planning
Acknowledgements

- Automated Planning and Acting Chapter 6
- Slides based on material provided by Dana Nau, Ralf Möller, and Shengyu Zhang
- Adapted by Tanya Braun
Outline

- **6.2 Stochastic shortest path problems**
  - Safe/unsafe policies
  - Optimality
  - Policy iteration, value iteration
- **6.3 Heuristic search algorithms (omitted)**
- **6.4 Online probabilistic planning**
  - Lookahead
  - Reinforcement learning
Probabilistic Planning Domain

- $\Sigma = (S, A, \gamma, P, \text{cost})$
  - $S$ = set of states
  - $A$ = set of actions
  - $\gamma : S \times A \rightarrow 2^S$ a transition function
  - $P(s' | s, a)$ = probability of going to state $s'$ if we perform $a$ in $s$
    - Require $P(s' | s, a) \neq 0$ iff $s' \in \gamma(s, a)$
  - $\text{cost} : S \times A \rightarrow \mathbb{R}^{>0}$
    - $\text{cost}(s, a)$ = cost of action $a$ in state $s$
    - may omit, default is $\text{cost}(s, a) = 1$

**Difference in syntax:** MDPs do not have an explicit transition function $\gamma$, only a set of applicable actions $A(s)$ per state and the transition model $P(s' | s, a)$

Instead of maximizing expected utility as before: **Minimize expected cost**
Example

- Robot $r_1$ starts at $d_1$
- Objective: get to $d_4$
- Simplified state names: write $\{\text{loc}(r_1) = d_2\}$ as $d_2$
- Simplified action names: write $\text{move}(r_1,d_2,d_3)$ as $m_{23}$
- $r_1$ has unreliable steering, especially on hills
  - May slip and go elsewhere

- $m_{14}$: $P(d_4 \mid d_1, m_{14}) = 0.5$
  - $P(d_1 \mid d_1, m_{14}) = 0.5$
- $m_{23}$: $P(d_3 \mid d_2, m_{23}) = 0.8$
  - $P(d_5 \mid d_2, m_{23}) = 0.2$
- $m_{21}$: $P(d_2 \mid d_1, m_{21}) = 1$
- $m_{34}, m_{41}, m_{43}, m_{45}, m_{52}, m_{54}$: like $m_{21}$
Policies, Problems, Solutions

- **Stochastic shortest path (SSP) problem:**
  - a triple \((\Sigma, s_0, S_g)\)
- **Policy:**
  - partial function \(\pi : S \rightarrow A\) s.t.
    - for every \(s \in \text{Dom}(\pi) \subseteq S\), \(\pi(s) \in \text{Applicable}(s)\)
- **Solution for \((\Sigma, s_0, S_g)\):**
  - a policy \(\pi\) s.t.
    - \(s_0 \in \text{Dom}(\pi)\) and
    - \(\hat{\pi}(s_0, \pi) \cap S_g \neq \emptyset\)

\[
\begin{align*}
\Sigma & : m14: P(d4 | d1, m14) = 0.5 \\
& \quad P(d1 | d1, m14) = 0.5 \\
& m23: P(d3 | d1, m23) = 0.8 \\
& \quad P(d5 | d1, m23) = 0.2
\end{align*}
\]
Notation and Terminology

• As before:
  • Transitive closure
    • \( \hat{\gamma}(s, \pi) = \{s \text{ and all states reachable from } s \text{ using } \pi\} \)
  • \textit{Graph}(s, \pi) = \text{rooted graph induced by } \pi \text{ at } s
    • Nodes: \( \hat{\gamma}(s, \pi) \)
    • Edges: state transitions
    • \( \text{leaves}(s, \pi) = \hat{\gamma}(s, \pi) \setminus \text{Dom}(\pi) \)

• A solution policy \( \pi \) is \textbf{closed} if it does not stop at non-goal states unless there is no way to continue
  • for every state \( s \in \hat{\gamma}(s, \pi) \), either
    • \( s \in \text{Dom}(\pi) \) (i.e., \( \pi \) specifies an action at \( s \)),
    • \( s \in S_g \) (i.e., \( s \) is a goal state), or
    • \( \text{Applicable}(s) = \emptyset \) (i.e., there are no applicable actions at \( s \))
Dead Ends

• Dead end
  • A state or set of states from which the goal is unreachable

Explicit dead end

Start: \( s_0 = d_1 \)

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

Implicit dead end

Start: \( s_0 = d_1 \)

Goal: \( S_g = \{d_4\} \)
Histories

- **History**: sequence of states \( \sigma = \langle s_0, s_1, s_2, \ldots \rangle \)
  - May be finite or infinite
    - \( \sigma = \langle d_1, d_2, d_3, d_4 \rangle \)
    - \( \sigma = \langle d_1, d_2, d_1, d_2, \ldots \rangle \)
  - \( H(s, \pi) = \{ \text{all possible histories if we start at } s \text{ and follow } \pi, \text{ stopping if } \pi(s) \text{ is undefined or if we reach a goal state} \} \)
  - If \( \sigma \in H(s, \pi) \), then
    \[
    P(\sigma | s, \pi) = \prod_i P(s_{i+1} | s_i, \pi(s_i))
    \]
  - Thus
    \[
    \sum_{\sigma \in H(s, \pi)} P(\sigma | s, \pi) = 1
    \]

Probability of reaching a goal:

\[
P(S_g | s, \pi) = \sum_{\sigma \in H(s, \pi)} P(\sigma | s, \pi)
\]
Do you have an idea for a definition of an unsafe solution?
Unsafe Solutions

- Unsafe solution: \( 0 < P(S_g | s_0, \pi) < 1 \)
- Example:
  - \( \pi_1 = \{(d1, m12), (d2, m23), (d3, m34)\} \)
  - \( H(s_0, \pi_1) \) contains two histories:
    - \( \sigma_1 = \{d1, d2, d3, d4\} \)
    - \( P(\sigma_1 | s_0, \pi_1) = 1 \cdot 0.8 \cdot 1 = 0.8 \)
    - \( \sigma_2 = \{d1, d2, d5\} \)
    - \( P(\sigma_2 | s_0, \pi_1) = 1 \cdot 0.2 = 0.2 \)
    - \( P(S_g | s_0, \pi_1) = 0.8 \)
Unsafe Solutions

- Unsafe solution: $0 < P(S_g | s_0, \pi) < 1$

- Example:
  - $\pi_2 = \{ (d1, m12), (d2, m23), (d3, m34), (d5, m56), (d6, m65) \}$
  - $H(s_0, \pi_2)$ contains two histories:
    - $\sigma_1 = \{ d1, d2, d3, d4 \}$
    - $P(\sigma_1 | s_0, \pi_2)$
      $\quad = 1 \cdot 0.8 \cdot 1 = 0.8$
    - $\sigma_3 = \{ d1, d2, d5, d6, \ldots \}$
    - $P(\sigma_3 | s_0, \pi_2)$
      $\quad = 1 \cdot 0.2 \cdot 1 \cdot \cdots = 0.2$
    - $P(S_g | s_0, \pi_2)$
      $\quad = 0.8$
Safe Solutions

• Safe solution: \( P(S_g | s_0, \pi) = 1 \)

• An acyclic safe solution:
  • \( \pi_3 = \{(d1, m12), (d2, m23), (d3, m34), (d5, m54)\} \)
  • \( H(s_0, \pi_3) \) contains two histories:
    • \( \sigma_1 = \{d1, d2, d3, d4\} \)
    • \( P(\sigma_1 | s_0, \pi_3) \)
      \( = 1 \cdot 0.8 \cdot 1 = 0.8 \)
  • \( \sigma_4 = \{d1, d2, d5, d4\} \)
  • \( P(\sigma_4 | s_0, \pi_3) \)
    \( = 1 \cdot 0.2 \cdot 1 = 0.2 \)
  • \( P(S_g | s_0, \pi_3) \)
    \( = 0.8 + 0.2 = 1 \)
Safe Solutions

- Safe solution: \( P(S_g|s_0, \pi) = 1 \)

- A cyclic safe solution:
  - \( \pi_4 = \{ (d1, m14) \} \)
  - \( H(s_0, \pi_4) \) contains infinitely many histories:
    - \( \sigma_5 = \{ d1, d4 \} \)
    - \( P(\sigma_5|s_0, \pi_4) = 0.5 = \left( \frac{1}{2} \right)^1 \)
    - \( \sigma_6 = \{ d1, d1, d4 \} \)
    - \( P(\sigma_6|s_0, \pi_4) \)
      \[ = 0.5 \cdot 0.5 = \left( \frac{1}{2} \right)^2 \]
    - ...
    - \( P(S_g|s_0, \pi_4) \)
      \[ = \frac{1}{2} + \frac{1}{4} + ... = 1 \]
Safe Solutions

- Safe solution: \( P(S_g|s_0, \pi) = 1 \)

- Another cyclic safe solution:
  - \( \pi_5 = \{(d1, m14), (d4, m41)\} \)
  - \( H(s_0, \pi_5) = H(s_0, \pi_4): \)
  - \( \sigma_5 = \langle d1, d4 \rangle \)
  - \( P(\sigma_5|s_0, \pi_5) = 0.5 = \left(\frac{1}{2}\right)^1 \)
  - \( \sigma_6 = \langle d1, d1, d4 \rangle \)
  - \( P(\sigma_6|s_0, \pi_6) \)
    \[ = 0.5 \cdot 0.5 = \left(\frac{1}{2}\right)^2, \]
  - ...  
  - \( P(S_g|s_0, \pi_5) \)
    \[ = \frac{1}{2} + \frac{1}{4} + \ldots = 1 \]
Expected Cost

- $cost(s, a) = \text{cost of using } a \text{ in } s$
- **Example**
  - Each “horizontal” action costs 1
  - Each “vertical” action costs 100
- **Costs of a history**
  \[ \sigma = (s_0, s_1, s_2, \ldots) \]
- $cost(\sigma | s_0, \pi) = \sum_{i \in \sigma} cost(s_i, \pi(s_i))$

![Graph](image)
Expected Cost

- Let $\pi$ be a safe solution
- At each state $s \in Dom(\pi)$, expected cost of following $\pi$ to goal:
  - Weighted sum of history costs:
    \[
    V^\pi(s) = \text{cost}(s, \pi(s)) + \sum_{\sigma \in H(s, \pi), \sigma' = \sigma \setminus \{s\}} P(\sigma'|s, \pi) \text{cost}(\sigma'|s, \pi)
    \]
  - Recursive formulation
    \[
    V^\pi(s) = \begin{cases} 
    0 & \text{if } s \in S_g \\
    \text{cost}(s, \pi(s)) + \sum_{s' \in \gamma(s, \pi(s))} P(s'|s, \pi(s)) V^\pi(s') & \text{otherwise}
    \end{cases}
    \]

Compare policy evaluation of the policy iteration algorithm of the previous topic
• \( \pi_3 = \{(d1, m12), (d2, m23), (d3, m34), (d5, m54)\} \)

• Weighted sum of history cost:
  • \( \sigma_1 = \langle d1, d2, d3, d4 \rangle \)
  • \( P(\sigma_1|s_0, \pi_3) = 0.8 \)
  • \(\text{cost}(\sigma_1|s_0, \pi_3) = 100 + 1 + 100 = 201 \)

  • \( \sigma_4 = \langle d1, d2, d5, d4 \rangle \)
  • \( P(\sigma_4|s_0, \pi_3) = 0.2 \)
  • \( \text{cost}(\sigma_4|s_0, \pi_3) = 100 + 1 + 100 = 201 \)

• \( V^{\pi_3}(d1) \)
  \( = 0.8(201) + 0.2(201) \)
  \( = 201 \)

• Recursive equation
  • \( V^{\pi_3}(d1) \)
    \( = 100 + V^{\pi_3}(d2) \)
    \( = 100 + 1 + 0.8V^{\pi_3}(d3) + 0.2V^{\pi_3}(d5) \)
    \( = 100 + 1 + 0.8(100) + 0.2(100) \)
    \( = 201 \)
Safe Solutions

- $\pi_4 = \{(d1, m14)\}$
- Weighted sum of history cost:
  - $\sigma_5 = \langle d1, d4 \rangle$
  - $P(\sigma_5 | s_0, \pi_4) = \left(\frac{1}{2}\right)^1$
  - $cost(\sigma_5 | s_0, \pi_4) = 1$
  - $\sigma_6 = \langle d1, d1, d4 \rangle$
  - $P(\sigma_6 | s_0, \pi_4) = \left(\frac{1}{2}\right)^2$
  - $cost(\sigma_6 | s_0, \pi_4) = 2$
  - ...
  - $V^{\pi_4}(d1) = \frac{1}{2}(1) + \frac{1}{4}(2) + ... = 2$

- Recursive equation
  - $V^{\pi_4}(d1) = 1 + 0.5(0) + 0.5(V^{\pi_4}(d1))$
  - $\Leftrightarrow 0.5V^{\pi_4}(d1) = 1$
  - $\Leftrightarrow V^{\pi_4}(d1) = 2$
Planning as Optimisation

- Let $\pi$ and $\pi'$ be safe solutions
  - $\pi$ dominates $\pi'$ if $\forall s \in \text{Dom}(\pi) \cap \text{Dom}(\pi') : V^\pi(s) \leq V^{\pi'}(s)$
  - $\pi$ 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 are both defined
- $V^*(s) = \text{expected cost of getting to the goal using an optimal safe solution}$
- Recall expected cost of following $\pi$ to goal starting in $s$
  \[
  V^\pi(s) = \begin{cases} 
  0 & \text{if } s \in S_g \\
  \text{cost}(s, \pi(s)) + \sum_{s' \in \gamma(s, \pi(s))} P(s'|s, \pi(s))V^\pi(s') & \text{otherwise}
  \end{cases}
  \]

- Optimality principle (Bellman’s theorem):
  \[
  V^*(s) = \begin{cases} 
  0 & \text{if } s \in S_g \\
  \min_{a \in \text{Applicable}(s)} \left\{ \text{cost}(s, \pi(s)) + \sum_{s' \in \gamma(s, \pi(s))} P(s'|s, \pi(s))V^*(s') \right\} & \text{otherwise}
  \end{cases}
  \]
Cost to Go

- Let $\left( \Sigma, s_0, S_g \right)$ be a safe SSP
  - i.e., $S_g$ is reachable from every state
  - Same as safely explorable in non-deterministic models
- Let $\pi$ be a safe solution that is defined at all non-goal states
  - i.e., $\text{Dom}(\pi) = S \setminus S_g$
- Let $a \in \text{Applicable}(s)$
- Cost-to-go
  \[ Q^\pi(s, a) = \text{cost}(s, a) + \sum_{s' \in \gamma(s, a)} P(s'|s, a) V^\pi(s') \]
  - Expected cost if we start at $s$, use $a$, and use $\pi$ afterward
- For every $s \in S \setminus S_g$, let
  \[ \pi'(s) \in \arg \min_{a \in \text{Applicable}(s)} Q^\pi(s, a) \]
Policy Iteration

- Inputs
  - SSP problem \((\Sigma, s_0, S_g)\)
  - Initial policy \(\pi_0\)
- Finds an optimal policy
- Converges in a finite number of steps

\[
\text{policy-iteration}(\Sigma, s_0, S_g, \pi_0) \\
\begin{align*}
\pi &\leftarrow \pi_0 \\
\text{loop} &\quad \text{compute}\{V^\pi(s) | s \in S\} \\
\text{for every state } s \in S \setminus S_g &\quad \text{do} \\
A &\leftarrow \arg\min_{a \in \text{Applicable}(s)} Q^\pi(s, a) \\
\text{if } \pi(s) &\in A \text{ then} \\
\pi'(s) &\leftarrow \pi(s) \\
\text{else} &\quad \pi'(s) \leftarrow \text{any action in } A \\
\text{if } \pi' &\neq \pi \text{ then} \\
\text{return } \pi &\quad \pi \leftarrow \pi'
\end{align*}
\]

\(n\) equations,
\(n\) unknowns,
where \(n = |S|\)
Example

• Start with
  • \( \pi = \pi_0 = \{(d1, m12), (d2, m23), (d3, m34), (d5, m54)\} \)
• Expected cost
  • \( V^\pi(d4) = 0 \)
  • \( V^\pi(d3) = 100 + 1 \cdot V^\pi(d4) = 100 \)
  • \( V^\pi(d5) = 100 + 1 \cdot V^\pi(d4) = 100 \)
  • \( V^\pi(d2) = 1 + (0.8 \cdot V^\pi(d3) + 0.2 \cdot V^\pi(d5)) = 101 \)
  • \( V^\pi(d1) = 100 + 1 \cdot V^\pi(d2) = 201 \)
• Cost-to-go
  • \( Q(d1, m12) = 100 + 1(101) = 201 \)
  • \( Q(d1, m14) = 1 + 0.5(201) + 0.5(0) = 101.5 \)
    • \( \text{argmin} = m14 \)
  • \( Q(d2, m23) = 1 + (0.8(100) + 0.2(100)) = 101 \)
  • \( Q(d2, m21) = 100 + 201 = 301 \)
    • \( \text{argmin} = m23 \)

• Cost-to-go continued
  • \( Q(d3, m34) = 100 + 0 = 100 \)
  • \( Q(d3, m32) = 1 + 101 = 102 \)
    • \( \text{argmin} = m34 \)
  • \( Q(d5, m54) = 100 + 0 = 100 \)
  • \( Q(d5, m52) = 1 + 101 = 102 \)
    • \( \text{argmin} = m54 \)
Example

- Continue with
  - $\pi = \{(d1, m14), (d2, m23), (d3, m34), (d5, m54)\}$
- Expected cost
  - $V_{\pi}(d4) = 0$
  - $V_{\pi}(d3) = 100 + V_{\pi}(d4) = 100$
  - $V_{\pi}(d5) = 100 + V_{\pi}(d4) = 100$
  - $V_{\pi}(d2) = 1 + (0.8V_{\pi}(d3) + 0.2V_{\pi}(d5)) = 101$
  - $V_{\pi}(d1) = 1 + (0.5V_{\pi}(d1) + 0.5V_{\pi}(d4)) = 2$
- Cost-to-go
  - $Q(d1, m12) = 100 + 101 = 201$
  - $Q(d1, m14) = 1 + 0.5(2) + 0.5(0) = 2$
    - $\text{argmin} = m14$
  - $Q(d2, m23) = 1 + (0.8(100) + 0.2(100)) = 101$
  - $Q(d2, m21) = 100 + 2 = 102$
  - $\text{argmin} = m23$

\[\pi\text{ unchanged}\]

- Cost-to-go continued
  - $Q(d3, m34) = 100 + 0 = 100$
  - $Q(d3, m32) = 100 + 101 = 201$
    - $\text{argmin} = m34$
  - $Q(d5, m54) = 100 + 0 = 100$
  - $Q(d5, m54) = 100 + 101 = 201$
    - $\text{argmin} = m54$

Start: $s_0 = d1$
Goal: $S_g = \{d4\}$
Value Iteration

- **Inputs**
  - SSP problem $\Sigma, s_0, S_g$
  - Convergence criterion $\eta > 0$
  - $V_0$ is a heuristic for initial values
  - E.g., adapt a heuristics from Ch. 2

- **Outputs**
  - Returns optimal policy $\pi_i$
  - $V_i$ = values computed at $i$'th iteration
  - $\pi_i$ = policy computed from $V_i$

- **Synchronous**: computes $V_i$ and $\pi_i$ from old $V_{i-1}$ and $\pi_{i-1}$
- **Asynchronous**: update $V$ and $\pi$ in place
  - New values available immediately
  - More efficient than synchronous version

**Sync-value-iteration($\Sigma, s_0, S_g, V_0, \eta$)**

\[
\text{for } i = 1, 2, \ldots \text{ do }
\]

\[
\text{for every state } s \in S \setminus S_g \text{ do }
\]

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

\[
Q(s, a) \leftarrow \text{cost}(s, a) + \sum_{s' \in S} P(s'|s, a) V_{i-1}(s')
\]

\[
V_i(s) \leftarrow \min_{a \in \text{Applicable}(s)} Q(s, a)
\]

\[
\pi_i(s) \leftarrow \arg\min_{a \in \text{Applicable}(s)} Q(s, a)
\]

\[
\text{if } \max_{s \in S} |V_i(s) - V_{i-1}(s)| \leq \eta \text{ then }
\]

\[
\text{return } \pi_i
\]

**Async-value-iteration($\Sigma, s_0, S_g, V_0, \eta$)**

\[
\text{global } \pi \leftarrow \emptyset
\]

\[
\text{global } V(s) \leftarrow V_0(s) \forall s
\]

\[
\text{loop}
\]

\[
\text{if } r \leq \eta \text{ then }
\]

\[
\text{return } \pi
\]

\[
\text{Bellman-Update}(s)
\]

\[
\nu_{\text{old}} \leftarrow V(s)
\]

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

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

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

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

\[
\text{return } |V(s) - \nu_{\text{old}}|
\]
**Synchronous**

- $Q(d1, m12) = 100 + 0 = 100$
- $Q(d1, m14) = 1 + (0.5(0) + 0.5(0)) = 1$
  - $V_1(d1) = 1; \pi_1(d1) = m14$
- $Q(d2, m21) = 100 + 0 = 100$
- $Q(d2, m23) = 1 + (0.2(0) + 0.8(0)) = 1$
  - $V_1(d2) = 1; \pi_1(d2) = m23$
- $Q(d3, m32) = 1 + 0 = 1$
- $Q(d3, m34) = 100 + 0 = 100$
  - $V_1(d3) = 1; \pi_1(d3) = m32$
- $Q(d5, m52) = 1 + 0 = 1$
- $Q(d5, m54) = 100 + 0 = 100$
  - $V_1(d5) = 1; \pi_1(d5) = m52$
- $r = \text{max}(1 - 0, 1 - 0.1 - 0.1 - 0) = 1$

**Asynchronous**

- $Q(d1, m12) = 100 + 0 = 100$
- $Q(d1, m14) = 1 + (0.5(0) + 0.5(0)) = 1$
  - $V(d1) = 1; \pi(d1) = m14$
- $Q(d2, m21) = 100 + 1 = 101$
- $Q(d2, m23) = 1 + (0.2(0) + 0.8(0)) = 1$
  - $V(d2) = 1; \pi(d2) = m23$
- $Q(d3, m32) = 1 + 1 = 2$
- $Q(d3, m34) = 100 + 0 = 100$
  - $V(d3) = 2; \pi(d3) = m32$
- $Q(d5, m52) = 1 + 1 = 2$
- $Q(d5, m54) = 100 + 0 = 100$
  - $V(d5) = 2; \pi(d5) = m52$
- $r = \text{max}(1 - 0, 1 - 0, 2 - 0, 2 - 0) = 2$
Synchronous

- $Q(d1, m12) = 100 + 1 = 101$
- $Q(d1, m14) = 1 + (0.5(1) + 0.5(0)) = 1.5$
- $V_1(d1) = 1.5; \pi_1(d1) = m14$
- $Q(d2, m21) = 100 + 1 = 101$
- $Q(d2, m23) = 1 + (0.2(1) + 0.8(1)) = 2$
- $V_1(d2) = 2; \pi_1(d2) = m23$
- $Q(d3, m32) = 1 + 1 = 2$
- $Q(d3, m34) = 100 + 0 = 100$
- $V_1(d3) = 2; \pi_1(d3) = m32$
- $Q(d5, m52) = 1 + 1 = 2$
- $Q(d5, m54) = 100 + 0 = 100$
- $V_1(d5) = 1; \pi_1(d5) = m52$

- $r = \max(1.5 - 1, 2 - 1.2 - 1.2 - 1) = 1$

Asynchronous

- $Q(d1, m12) = 100 + 1 = 101$
- $Q(d1, m14) = 1 + (0.5(1) + 0.5(0)) = 1.5$
- $V(d1) = 1.5; \pi(d1) = m14$
- $Q(d2, m21) = 100 + 1.5 = 101.5$
- $Q(d2, m23) = 1 + (0.2(2) + 0.8(2)) = 3$
- $V(d2) = 3; \pi(d2) = m23$
- $Q(d3, m32) = 1 + 3 = 4$
- $Q(d3, m34) = 100 + 0 = 100$
- $V(d3) = 4; \pi(d3) = m32$

- $Q(d5, m52) = 1 + 3 = 4$
- $Q(d5, m54) = 100 + 0 = 100$
- $V(d5) = 4; \pi(d5) = m52$

- $r = \max(1.5 - 1, 3 - 1, 4 - 2, 4 - 2) = 2$
Synchronous

- $Q(d1, m12) = 100 + 2 = 102$
- $Q(d1, m14) = 1 + (0.5(1.5) + 0.5(0)) = 1.75$
- $V_1(d1) = 1.75; \pi_1(d1) = m14$
- $Q(d2, m21) = 100 + 1.5 = 101.5$
- $Q(d2, m23) = 1 + (0.2(2) + 0.8(2)) = 3$
- $V_1(d2) = 3; \pi_1(d2) = m23$
- $Q(d3, m32) = 1 + 2 = 3$
- $Q(d3, m34) = 100 + 0 = 100$
- $V_1(d3) = 3; \pi_1(d3) = m32$
- $Q(d5, m52) = 1 + 2 = 3$
- $Q(d5, m54) = 100 + 0 = 100$
- $V_1(d5) = 3; \pi_1(d5) = m52$

- $r = \max(1.75 - 1.5, 3 - 2, 3 - 2, 3 - 2) = 1$

$V(d1) = 1.5$
$V(d2) = 2$
$V(d3) = 2$
$V(d5) = 2$

Asynchronous

- $Q(d1, m12) = 100 + 3 = 103$
- $Q(d1, m14) = 1 + (0.5(1.5) + 0.5(0)) = 1.75$
- $V(d1) = 1.75; \pi(d1) = m14$
- $Q(d2, m21) = 100 + 1.75 = 101.75$
- $Q(d2, m23) = 1 + (0.2(4) + 0.8(4)) = 5$
- $V(d2) = 5; \pi(d2) = m23$
- $Q(d3, m32) = 1 + 5 = 6$
- $Q(d3, m34) = 100 + 0 = 100$
- $V(d3) = 6; \pi(d3) = m32$

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

- $r = \max(1.75 - 1.5, 5 - 3, 6 - 4, 6 - 4) = 2$

$V(d1) = 1.5$
$V(d2) = 3$
$V(d3) = 4$
$V(d5) = 4$
Synchronous

- \( Q(d_1, m_{12}) = 100 + 3 = 103 \)
- \( Q(d_1, m_{14}) = 1 + (0.5 \times 1.75 + 0.5 \times 0) = 1.875 \)
- \( V_1(d_1) = 1.875; \pi_1(d_1) = m_{14} \)
- \( Q(d_2, m_{21}) = 100 + 1.75 = 101.75 \)
- \( Q(d_2, m_{23}) = 1 + (0.2 \times 3 + 0.8 \times 3) = 4 \)
- \( V_1(d_2) = 4; \pi_1(d_2) = m_{23} \)
- \( Q(d_3, m_{32}) = 1 + 3 = 4 \)
- \( Q(d_3, m_{34}) = 100 + 0 = 100 \)
- \( V_1(d_3) = 4; \pi_1(d_3) = m_{32} \)
- \( Q(d_5, m_{52}) = 1 + 3 = 4 \)
- \( Q(d_5, m_{54}) = 100 + 0 = 100 \)
- \( V_1(d_5) = 4; \pi_1(d_5) = m_{52} \)
- \( r = \text{max}(1.875 - 1.75, 4 - 3.4 - 3.4 - 3) = 1 \)

\[
V(d_1) = 1.75 \\
V(d_2) = 3 \\
V(d_3) = 3 \\
V(d_5) = 3
\]

Asynchronous

- \( Q(d_1, m_{12}) = 100 + 5 = 105 \)
- \( Q(d_1, m_{14}) = 1 + (0.5 \times 1.75 + 0.5 \times 0) = 1.875 \)
- \( V_1(d_1) = 1.875; \pi_1(d_1) = m_{14} \)
- \( Q(d_2, m_{21}) = 100 + 1.875 = 101.875 \)
- \( Q(d_2, m_{23}) = 1 + (0.2 \times 6 + 0.8 \times 6) = 7 \)
- \( V_2(d_2) = 7; \pi_2(d_2) = m_{23} \)
- \( Q(d_3, m_{32}) = 1 + 7 = 8 \)
- \( Q(d_3, m_{34}) = 100 + 0 = 100 \)
- \( V_3(d_3) = 8; \pi_3(d_3) = m_{32} \)
  - \( Q(d_5, m_{52}) = 1 + 7 = 8 \)
  - \( Q(d_5, m_{54}) = 100 + 0 = 100 \)
  - \( V_5(d_5) = 8; \pi_5(d_5) = m_{52} \)
- \( r = \text{max}(1.875 - 1.75, 7 - 5, 8 - 6, 8 - 6) = 2 \)

\[
V(d_1) = 1.75 \\
V(d_2) = 5 \\
V(d_3) = 6 \\
V(d_5) = 6
\]
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: does not 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
Summary

- SSPs
- Solutions, closed solutions, histories
- Unsafe solutions, acyclic safe solutions, cyclic safe solutions
- Expected cost, planning as optimization
- Policy iteration
- Value iteration (synchronous, asynchronous)
  - Bellman-update
Outline

• 6.2 Stochastic shortest path problems
  • Safe/unsafe policies
  • Optimality
  • Policy iteration, value iteration
• 6.3 Heuristic search algorithms (omitted)
• 6.4 Online probabilistic planning
  • Lookahead
  • Reinforcement learning
Outline

• 6.2 *Stochastic shortest path problems*
  • Safe/unsafe policies
  • Optimality
  • Policy iteration, value iteration
• 6.3 *Heuristic search algorithms (omitted)*
• 6.4 *Online probabilistic planning*
  • Lookahead
  • Reinforcement learning
Planning and Acting

- Same as in Ch. 2, except $s$ instead of $\xi$
  - Could use $s \leftarrow$ abstraction of $\xi$ as in Ch. 2
  - Inputs: SSP problem $(\Sigma, s_0, S_g)$, vector of parameters $\theta$
- Could also use Run-Lazy-Lookahead or Run-Concurrent-Lookahead
- What to use for Lookahead?
  - AO*, LAO*, ... (in book) $\rightarrow$ Modify to search part of the space
  - Classical planner running on determinised domain
  - Stochastic sampling algorithms

\[
\text{Run-Lookahead}(\Sigma, s_0, S_g, \theta)\\
\begin{align*}
    & s \leftarrow s_0 \\
    \text{while } & s \notin S_g \text{ and } \text{Applicable}(s) \neq \emptyset \text{ do} \\
    & a \leftarrow \text{Lookahead}(s, \theta) \\
    & \text{perform action } a \\
    & s \leftarrow \text{observe resulting state}
\end{align*}
\]
Planning and Acting

- If Lookahead = classical planner on determinized domain
  - ⇒ FS-Replan (Ch. 5)
- Problem: Forward-search may choose a plan that depends on low-probability outcome
- RFF algorithm (see book) attempts to alleviate this

Run-Lookahead($\Sigma, s_0, S_g, \theta$)

\[
s \leftarrow s_0 \\
\text{while } s \notin S_g \text{ and Applicable}(s) \neq \emptyset \text{ do}
\]
\[
a \leftarrow \text{Lookahead}(s, \theta) \\
\text{perform action } a \\
s \leftarrow \text{observe resulting state}
\]

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

\[
\pi_d \leftarrow \emptyset \\
\text{while } s \notin S_g \text{ and Applicable}(s) \neq \emptyset \text{ do}
\]
\[
\text{if } \pi_d \text{ undefined for } s \text{ then}
\]
\[
\pi_d \leftarrow \text{Forward-Search}(\Sigma_d, s, S_g) \\
\text{if } \pi_d = \text{failure then}
\]
\[
\text{return failure} \\
\text{perform action } \pi_d(s) \\
s \leftarrow \text{observe resulting state}
\]
Acting as Reinforcement Learning (RL)

- Agent, placed in an environment, must learn to act optimally in it
- Assume that the world behaves like an MDP, except
  - Agent can act but does not know the transition model
  - Agent observes its current state and its reward but does not know the reward function
- Goal: learn an optimal policy
Factors That Make RL Hard

- Actions have non-deterministic effects
  - which are initially unknown and must be learned
- Rewards / punishments can be infrequent
  - Often at the end of long sequences of actions
  - How does an agent determine what action(s) were really responsible for reward or punishment?
    - Credit assignment problem
    - World is large and complex
Passive vs. Active Learning

- **Passive learning**
  - Agent acts based on a fixed policy $\pi$ and tries to learn how good the policy is by observing the world go by
  - Analogous to policy iteration (without the optimisation part)
- **Active learning**
  - Agent attempts to find an optimal (or at least good) policy by exploring different actions in the world
  - Analogous to solving the underlying MDP
Model-based vs. Model-free RL

- **Model-based** approach to RL
  - Learn the MDP model \((P(s' | s, a)\) and \(R\), or an approximation of it
  - Use it to find the optimal policy

- **Model-free** approach to RL
  - Derive the optimal policy without explicitly learning the model
Passive RL

- Suppose the agent is given a policy
- Wants to determine how good it is

- Given $\pi$:

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Need to learn $U^\pi(s)$:

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td>+1</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td>-1</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>0.812</td>
<td>0.868</td>
<td>0.918</td>
<td>+1</td>
</tr>
<tr>
<td>2</td>
<td>0.762</td>
<td>0.660</td>
<td>-1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0.705</td>
<td>0.611</td>
<td>0.388</td>
<td></td>
</tr>
</tbody>
</table>
Passive RL

- Given policy \( \pi \):
  - Estimate \( U^\pi(s) \)
- Not given
  - Transition model \( P(s'|s,a) \)
  - Reward function \( R(s) \)
- Simply follow the policy for many epochs
  - Epochs: training sequences / trials
    
    \[
    (1,1) \to (1,2) \to (1,3) \to (1,2) \to (1,3) \to (2,3) \to (3,3) \to (3,4) + 1 \\
    (1,1) \to (1,2) \to (1,3) \to (2,3) \to (3,3) \to (3,2) \to (3,3) \to (3,4) + 1 \\
    (1,1) \to (2,1) \to (3,1) \to (3,2) \to (4,2) - 1
    \]
  - Assumption: restart or reset possible (or no terminal states with the end of an epoch given by the receipt of a reward)
Direct Utility Estimation (DUE)

- Model-free approach
  - Estimate $U^\pi(s)$ as average total reward of epochs containing $s$
    - Calculating from $s$ to end of epoch
- Reward-to-go of a state $s$
  - The sum of the (discounted) rewards from that state until a terminal state is reached
- Key: use observed reward-to-go of the state as the direct evidence of the actual expected utility of that state
DUE: Example

- Suppose the agent observes the following trial:
  - \((1,1)_{-0.04} \rightarrow (1,2)_{-0.04} \rightarrow (1,3)_{-0.04} \rightarrow (1,2)_{-0.04} \rightarrow (1,3)_{-0.04} \rightarrow (2,3)_{-0.04} \rightarrow (3,3)_{-0.04} \rightarrow (3,4)_{+1}\)
- The total reward starting at \((1,1)\) is 0.72
  - I.e., a sample of the observed-reward-to-go for \((1,1)\)
- For \((1,2)\), there are two samples of the observed-reward-to-go
  - Assuming \(\gamma = 1\)
    1. \((1,2)_{-0.04} \rightarrow (1,3)_{-0.04} \rightarrow (1,2)_{-0.04} \rightarrow (1,3)_{-0.04} \rightarrow (2,3)_{-0.04} \rightarrow (3,3)_{-0.04} \rightarrow (3,4)_{+1}\)
      [Total: 0.76]
    2. \((1,2)_{-0.04} \rightarrow (1,3)_{-0.04} \rightarrow (2,3)_{-0.04} \rightarrow (3,3)_{-0.04} \rightarrow (3,4)_{+1}\)
      [Total: 0.84]
DUE: Convergence

• Keep a running average of the observed reward-to-go for each state
• E.g., for state (1,2), it stores \( \frac{0.76 + 0.84}{2} = 0.8 \)
• As the number of trials goes to infinity, the sample average converges to the true utility
• Big problem: it converges very slowly!
• Why?
  • Does not exploit the fact that utilities of states are not independent
  • Utilities follow the Bellman equation

\[ U^\pi(s_t) = R(s_t) + \gamma \sum_{s_j} P(s_j|\pi(s_t), s_t) U^\pi(s_j) \]

Dependence on neighbouring states
DUE: Problem

- Using the dependence to your advantage
  - Suppose you know that state (3,3) has a high utility
  - Suppose you are now at (3,2)
  - Bellman equation would be able to tell you that (3,2) is likely to have a high utility because (3,3) is a neighbour
- DUE cannot tell you that until the end of the trial
Adaptive Dynamic Programming (ADP)

- Model-based approach
- Given policy $\pi$:
  - Estimate $U^\pi(s)$
  - All while acting in the environment
- How?
  - Basically learns the transition model $P(s'|s, a)$ and the reward function $R(s)$
  - Takes advantage of constraints in the Bellman equation
  - Based on $P(s'|s, a)$ and $R(s)$, performs policy evaluation (part of policy iteration)
Recap: Policy Iteration

- Pick a policy $\pi_0$ at random
- Repeat:
  - **Policy evaluation**: Compute the utility of each state for $\pi_t$
    
    $$U_t(s_i) = R(s_i) + \gamma \sum_{s_j} P(s_j | \pi_t(s_i)) U_t(s_j)$$

    No longer involves a max operation as action is determined by $\pi_t$
  - **Policy improvement**: Compute the policy $\pi_{t+1}$ given $U_t$
    
    $$\pi_{t+1}(s_i) = \text{argmax}_a \sum_{s_j} P(s_j | \pi_t(s_i)) U_t(s_j)$$

    If $\pi_{t+1} = \pi_t$, then return $\pi_t$

Can be solved in $O(n^3)$, where $n = |S|$

Solve the set of linear equations:

$$U(s_i) = R(s_i) + \gamma \sum_{s_j} P(s_j | \pi(s_i)) U(s_j)$$

(often a sparse system)
ADP: Estimate the Utilities

- Make use of policy evaluation to estimate the utilities of states
- To use policy equation

\[ U_{t+1}(s_i) = R(s_i) + \gamma \sum_{s_j} P(s_j|\pi(s_i), s_i)U_t(s_j) \]

agent needs to learn \( P(s'|s, a) \) and \( R(s) \)
- How?
ADP: Learn the Model

- Learning $R(s)$
  - Easy because it is deterministic
  - Whenever you see a new state, store the observed reward value as $R(s)$
- Learning $P(s'|s, a)$
  - Keep track of how often you get to state $s'$ given that you are in state $s$ and do action $a$
  - E.g., if you are in $s = (1,3)$ and you execute $R$ three times and you end up in $s' = (2,3)$ twice, then $P(s'|R, s) = \frac{2}{3}$
function passive-ADP-agent(percept)
returns an action
input: percept, indicating current state $s'$, reward $r'$
static:
    $\pi$, fixed policy
    mdp, MDP with $P[s'|s,a]$, $R(s)$, $\gamma$
    $U$, table of utilities, initially empty
    $N_{sa}$, table of freq. for $s$-$a$ pairs, initially 0
    $N_{sas'}$, table of freq. for $s$-$a$-$s'$ triples, initially 0
    $s,a$, previous state and action, initially null
if $s'$ is new then
    $U[s'] \leftarrow r'$
    $R[s'] \leftarrow r'$
if $s$ is not null then
    increment $N_{sa}[s,a]$ and $N_{sas'}[s,a,s']$
    for each $t$ s.t. $N_{sas'}[s,a,t] \neq 0$ do
        $P[t|s,a] \leftarrow N_{sas'}[s,a,t] / N_{sa}[s,a]$
    $U \leftarrow \text{Policy-evaluation}(\pi,U,mdp)$
if Terminal?($s'$) then
    $s,a \leftarrow \text{null}$
else
    $s,a \leftarrow s',\pi[s']$
return $a$
ADP: Problem

- Need to solve a system of simultaneous equations – costs $O(n^3)$
- Very hard to do if you have $10^{50}$ states like in Backgammon
- Could make things a little easier with modified policy iteration
- Can the agent avoid the computational expense of full policy evaluation?
Temporal Difference Learning (TD)

- Instead of calculating the exact utility for a state, can the agent approximate it and possibly make it less computationally expensive?
- Yes, it can! Using TD:

\[
U^\pi(s_i) = R(s_i) + \gamma \sum_{s_j} P(s_j|\pi(s_i), s_i) U^\pi(s_j)
\]

- Instead of doing the sum over all successors, only adjust the utility of the state based on the successor observed in the trial
- Does not estimate the transition model – model-free
Suppose you see that $U^\pi(1,3) = 0.84$ and $U^\pi(2,3) = 0.92$

If the transition $(1,3) \rightarrow (2,3)$ happens all the time, you would expect to see:

$$U^\pi(1,3) = R(1,3) + U^\pi(2,3)$$

$$\Rightarrow U^\pi(1,3) = -0.04 + U^\pi(2,3)$$

$$\Rightarrow U^\pi(1,3) = -0.04 + 0.92 = 0.88$$

Since you observe $U^\pi(1,3) = 0.84$ in the first trial and it is a little lower than 0.88, so you might want to “bump” it towards 0.88
Aside: Online Mean Estimation

• Suppose that we want to incrementally compute the mean of a sequence of numbers
  • E.g., to estimate the mean of a random variable from a sequence of samples

\[ \hat{X}_{n+1} = \frac{1}{n+1} \sum_{i=1}^{n+1} x_i = \left( \frac{1}{n+1} \sum_{i=1}^{n} x_i \right) + \frac{1}{n+1} x_{n+1} = \left( \frac{n}{n(n+1)} \sum_{i=1}^{n} x_i \right) + \frac{1}{n+1} x_{n+1} \]

\[ \text{average of } n+1 \text{ samples} = \left( \frac{n+1 - 1}{n(n+1)} \sum_{i=1}^{n} x_i \right) + \frac{1}{n+1} x_{n+1} = \left( \frac{n+1}{n(n+1)} \sum_{i=1}^{n} x_i \right) - \left( \frac{1}{n+1} \sum_{i=1}^{n} x_i \right) + \frac{1}{n+1} x_{n+1} \]

\[ = \left( \frac{1}{n} \sum_{i=1}^{n} x_i \right) - \left( \frac{1}{n+1} \cdot \frac{1}{n} \sum_{i=1}^{n} x_i \right) + \frac{1}{n+1} x_{n+1} = \left( \frac{1}{n} \sum_{i=1}^{n} x_i \right) + \frac{1}{n+1} \left( x_{n+1} - \frac{1}{n} \sum_{i=1}^{n} x_i \right) \]

\[ = \hat{X}_n + \frac{1}{n+1} \left( x_{n+1} - \hat{X}_n \right) \]

• Given a new sample \( x_{n+1} \), the new mean is the old estimate (for \( n \) samples) plus the weighted difference between the new sample and old estimate.
TD Update

- TD update for transition from $s$ to $s'$
  \[ U^\pi(s) = U^\pi(s) + \alpha(R(s) + \gamma U^\pi(s') - U^\pi(s)) \]

  - Similar to one step of value iteration
  - Equation called backup
  - So, the update is maintaining a "mean" of the (noisy) utility samples
  - If the learning rate decreases with the number of samples (e.g., $1/n$), then the utility estimates will eventually converge to true values

  \[ U^\pi(s_i) = R(s_i) + \gamma \sum_{s_j} P(s_j|\pi(s_i), s_i) U^\pi(s_j) \]
TD: Convergence

• Since TD uses the observed successor $s'$ instead of all the successors, what happens if the transition $s \rightarrow s'$ is very rare and there is a big jump in utilities from $s$ to $s'$?
  • How can $U^\pi(s)$ converge to the true equilibrium value?
  • Answer:
    The average value of $U^\pi(s)$ will converge to the correct value
    • This means the agent needs to observe enough trials that have transitions from $s$ to its successors
    • Essentially, the effects of the TD backups will be averaged over a large number of transitions
    • Rare transitions will be rare in the set of transitions observed
Comparison between ADP and TD

- **Advantages of ADP**
  - Converges to true utilities in fewer iterations
  - Utility estimates do not vary as much from the true utilities
- **Advantages of TD**
  - Simpler, less computation per observation
  - Crude but efficient first approximation to ADP
  - Do not need to build a transition model to perform its updates
ADP and TD

• Utility estimates for 4x3 grid
  • ADP, given optimal policy
    • Notice the large changes occurring around the 78\textsuperscript{th} trial—this is the first time that the agent falls into the −1 terminal state at (4,2)

• TD
  • More epochs required
  • Faster runtime per epoch

Figures: AIMA, Russell/Norvig
Overall comparisons

- **DUE (model-free)**
  - Simple to implement
  - Each update is fast
  - Does not exploit Bellman constraints and converges slowly

- **ADP (model-based)**
  - Harder to implement
  - Each update is a full policy evaluation (expensive)
  - Fully exploits Bellman constraints
  - Fast convergence (in terms of epochs)

- **TD (model-free)**
  - Update speed and implementation similar to direct estimation
  - Partially exploits Bellman constraints – adjusts state to “agree” with observed successor
    - Not all possible successors
  - Convergence in between DUE and ADP
Passive Learning: Disadvantage

- Learning $U^\pi(s)$ does not lead to an optimal policy, why?
  - Only evaluated $\pi$ (no optimisation)
  - Models are incomplete/inaccurate
  - Agent has only tried limited actions, cannot gain a good overall understanding of $P(s'|s, a)$
- Solution: Active learning
Goal of Active Learning

• Assume that the agent still has access to some sequence of trials performed by the agent
  • Agent is not following any specific policy
  • Assume for now that the sequences should include a thorough exploration of the space
  • We will talk about how to get such sequences later

• The goal is to learn an optimal policy from such sequences
  • Active RL agents
    • Active ADP agent
    • Q-learner (based on TD algorithm)
Active ADP Agent

- Model-based approach
- Using the data from its trials, agent estimates a transition model \( \hat{T} \) and a reward function \( \hat{R} \)
  - With \( \hat{T}(s, a, s') \) and \( \hat{R}(s) \), it has an estimate of the underlying MDP
  - Like passive ADP using policy evaluation
- Given estimate of the MDP, it can compute the optimal policy by solving the Bellman equations using value or policy iteration
  \[
  U(s) = \hat{R}(s) + \gamma \max_a \sum_{s'} \hat{T}(s, a, s')U(s')
  \]
- If \( \hat{T} \) and \( \hat{R} \) are accurate estimations of the underlying MDP model, agent can find the optimal policy this way
Issues with ADP Approach

- Need to maintain MDP model
- $T$ can be very large, $O(|S|^2 \cdot |A|)$
- Also, finding the optimal action requires solving the Bellman equation – time consuming
- Can the agent avoid this large computational complexity both in terms of time and space?
Q-learning

- So far, focus on utilities for states
  - $U(s) = \text{utility of state } s = \text{expected maximum future rewards}$
- Alternative: store Q-values
  - $Q(a, s) = \text{utility of taking action } a \text{ at state } s$
    - $= \text{expected maximum future reward if action } a \text{ taken at state } s$

- Relationship between $U(s)$ and $Q(a, s)$?
  \[
  U(s) = \max_a Q(a, s)
  \]
Q-learning can be model-free

• Note that after computing $U(s)$, to obtain the optimal policy, the agent needs to compute

$$
\pi(s) = \arg\max_a \sum_{s'} T(s, a, s') U(s')
$$

• Requires $T$, model of the world

• Even if it uses TD learning (model-free), it still needs the model to get the optimal policy

• However, if the agent successfully estimates $Q(a, s)$ for all $a$ and $s$, it can compute the optimal policy without using the model

$$
\pi(s) = \arg\max_a Q(a, s)
$$
Q-learning

- At equilibrium when Q-values are correct, we can write the constraint equation:

\[ Q(a, s) = R(s) + \gamma \sum_{s'} T(s, a, s') \max_{a'} U(s') \]

- Expected value for action-state pair \((a, s)\)
- Reward at state \(s\)
- Expected value averaged over all possible states \(s'\) that can be reached from \(s\) after executing action \(a\)
Q-learning

- At equilibrium when Q-values are correct, we can write the constraint equation:

\[
Q(a, s) = R(s) + \gamma \sum_{s'} T(s, a, s') \max_{a'} Q(a', s')
\]

- Expected value for action-state pair \((a, s)\)
- Reward at state \(s\)
- Best value at the next state = \(\max_{a'}\) over all actions in state \(s'\)
- Expected value averaged over all possible states \(s'\) that can be reached from \(s\) after executing action \(a\)
Q-learning without a Model

- **Q-update**: after moving from state \( s \) to state \( s' \) using action \( a \)
  \[
  Q(a, s) \leftarrow Q(a, s) + \alpha \left( R(s) + \gamma \max_{a'} Q(a', s') - Q(a, s) \right)
  \]

- TD approach
- Transition model does not appear anywhere!
- Once converged, optimal policy can be computed without transition model
- Completely model-free learning algorithm
Q-learning: Convergence

• Guaranteed to converge to true Q-values given enough exploration
• Very general procedure
  • Because it is model-free
• Converges slower than ADP agent
  • Because it is completely model-free and it does not enforce consistency among values through the model
Exploitation vs. Exploration

- Actions are always taken for one of the two following purposes
  - **Exploitation**: Execute the current optimal policy to get high payoff
  - **Exploration**: Try new sequences of (possibly random) actions to improve the agent’s knowledge of the environment even though current model does not show they have a high payoff
- Pure exploitation: gets stuck in a rut
- Pure exploration: not much use if you do not put that knowledge into practice
Multi-Arm Bandit Problem

• So far, we assumed that the agent has a set of epochs of sufficient exploration

• Multi-arm bandit problem: Statistical model of sequential experiments
  • Name comes from a traditional slot machine (one-armed bandit)

• Question: Which machine to play?
Actions

- $n$ arms, each with a fixed but unknown distribution of reward
  - In terms of actions: Multiple actions $a_1, a_2, ..., a_n$
    - Each $a_i$ provides a reward from an unknown (but stationary) probability distribution $p_i$
    - Specifically, expectation $\mu_i$ of machine $i$’s reward unknown
      - If all $\mu_i$’s were known, then the task is easy: just pick $\arg\max_i \mu_i$
  - With $\mu_i$’s unknown, question is which arm to pull
Formal Model

• At each time step $t = 1, 2, ..., T$:
  • Each machine $i$ has a random reward $X_{i,t}$
    • $E[X_{i,t}] = \mu_i$ independent of the past (Markov property again)
  • Pick a machine $I_t$ and get reward $X_{I_t,t}$
  • Other machines’ rewards hidden
• Over $T$ time steps, the agent has a total reward of $\sum_{t=1}^{T} X_{I_t,t}$
  • If all $\mu_i$’s known, it would have selected $\arg\max_i \mu_i$ at each time $t$
    • Expected total reward $T \cdot \max_i \mu_i$
• Agent’s “regret”: $T \cdot \max_i \mu_i - \sum_{t=1}^{T} X_{I_t,t}$
Exploitation vs. Exploration Reprise

- **Exploration**: to find the best
  - Overhead: big loss when trying the bad arms
- **Exploitation**: to exploit what the agent has discovered
  - Weakness: there may be better ones that it has not explored and identified
- **Question**: With a fixed budget, how to balance exploration and exploitation such that the total loss (or regret) is small?
Where Does the Loss Come from?

- If $\mu_i$ is small, trying this arm too many times makes a big loss
  - So the agent should try it less if it finds the previous samples from it are bad
- But how to know whether an arm is good?
  - The more the agent tries an arm $i$, the more information it gets about its distribution
    - In particular, the better estimate to its mean $\mu_i$
Where Does the Loss Come from?

• So the agent wants to estimate each $\mu_i$ precisely, and at the same time, it does not want to try bad arms too often
  • Two competing tasks
    • Exploration vs. exploitation dilemma
  • Rough idea: the agent tries an arm if
    • Either
      it has not tried it often enough
    • Or
      its estimate of $\mu_i$ so far is high
UCB (Upper Confidence Bound) Algorithm

- **Input:** Set of actions \( A \)
- **Assume rewards** between 0 and 1
  - If they are not, normalise them
- **For each action** \( a_i \), let
  - \( r_i \) = average reward from \( a_i \)
  - \( t_i \) = number of times \( a_i \) tried
- \( t = \sum_i t_i \)
- **Confidence interval around** \( r_i \)

\[
\left( r_i - \sqrt{\frac{2 \ln t}{t_i}}, r_i + \sqrt{\frac{2 \ln t}{t_i}} \right)
\]
UCB: Performance

- Theorem: If each distribution of reward has support in \([0,1]\), i.e., rewards are normalised, then the regret of the UCB algorithm is at most

\[
O\left( \sum_{i: \mu_i < \mu^*} \frac{\ln T}{\Delta_i} + \sum_{j \in \{1, \ldots, n\}} \Delta_j \right)
\]

- \(\mu^* = \max_i \mu_i\)
- \(\Delta_i = \mu^* - \mu_i\)
- Expected loss of choosing \(a_i\) once
- [without proof]

- Loss grows very slowly with \(T\)
UCB: Performance

• Uses principle of **optimism in face of uncertainty**
  • Agent does not have a good estimate $\hat{\mu}_i$ of $\mu_i$ before trying it many times
  • Thus give a big confidence interval $[-c_i, c_i]$ for such $i$
    • $c_i = \sqrt{\frac{2 \ln t}{t_i}}$
    • And select an $i$ with maximum $\mu_i + c_i$
  
• If an action has not been tried many times, then the big confidence interval makes it still possible to be tried
  • I.e., in face of uncertainty (of $\mu_i$), the agent acts optimistically by giving chances to those that have not been tried enough

\[
\begin{align*}
  r_i + \sqrt{\frac{2 \ln t}{t_i}}
\end{align*}
\]
UCT Algorithm

- Recursive UCB computation to compute $Q(s, a)$ for cost
- Min ops instead of max
- Planning domain $\Sigma$, state $s$
- Horizon $h$ (steps into the future)
- Anytime algorithm:
  - Call repeatedly until time runs out
  - Then choose action
  - $\arg\min_a Q(s, a)$

---

\[
\text{UCT}(\Sigma, s, h) \\
\text{if } s \in S \text{ then} \\
\quad \text{return } 0 \\
\text{if } h = 0 \text{ then} \\
\quad \text{return } V_0(s) \\
\text{if } s \notin \text{Envelope} \text{ then} \\
\quad \text{add } s \text{ to Envelope} \\
\quad n(s) \leftarrow 0 \\
\quad \text{for all } a \in \text{Applicable}(s) \text{ do} \\
\quad \quad Q(s, a) \leftarrow 0 \\
\quad \quad n(s, a) \leftarrow 0 \\
\quad \text{Untried} \leftarrow \{a \in \text{Applicable}(s) \mid n(s, a)=0\} \\
\text{if } \text{Untried} \neq \emptyset \text{ then} \\
\quad \bar{a} \leftarrow \text{Choose}(\text{Untried}) \\
\text{else} \\
\quad \bar{a} \leftarrow \arg\min_{a \in \text{Applicable}(s)} \{Q(s, a) - C \cdot [\log(n(s))/n(s, a)]^{\nu}\} \\
\quad s' \leftarrow \text{Sample}(\Sigma, s, \bar{a}) \\
\quad \text{cost-rollout} \leftarrow \text{cost}(s, \bar{a}) + \text{UCT}(s', h-1) \\
\quad Q(s, \bar{a}) \leftarrow [n(s, \bar{a}) \cdot Q(s, \bar{a}) + \text{cost-rollout}] / (1 + n(s, \bar{a})) \\
\quad n(s) \leftarrow n(s) + 1 \\
\quad n(s, \bar{a}) \leftarrow n(s, \bar{a}) + 1 \\
\text{return } \text{cost-rollout}
\]
UCT as an Acting Procedure

- Suppose probabilities and costs unknown
- Suppose you can restart your actor as many times as you want
- Can modify UCT to be an acting procedure
  - Use it to explore the environment

UCT(\Sigma, s, h)

if \( s \in S \) then
  return 0
if \( h = 0 \) then
  return \( V_0(s) \)
if \( s \notin \text{Envelope} \) then
  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
  \( \hat{a} \leftarrow \text{Choose}(\text{Untried}) \)
else
  \( \hat{a} \leftarrow \arg\min_{a \in \text{Applicable}(s)} \{ Q(s, a) - C \cdot \log(n(s))/n(s, a) \} \)
\( s' \leftarrow \text{Sample}(\Sigma, s, \hat{a}) \)
\( \text{cost-rollout} \leftarrow \text{cost}(s, \hat{a}) + \text{UCT}(s', h-1) \)
\( Q(s, \hat{a}) \leftarrow [n(s, \hat{a}) \cdot Q(s, \hat{a}) + \text{cost-rollout}] / (1 + n(s, \hat{a})) \)
\( n(s) \leftarrow n(s) + 1 \)
\( n(s, \hat{a}) \leftarrow n(s, \hat{a}) + 1 \)
return \( \text{cost-rollout} \)

perform \( \hat{a} \); observe \( s' \)
UCT as a Learning Procedure

- Suppose probabilities and costs are unknown
- But you have an accurate simulator for the environment
- Run UCT multiple times in the simulated environment
- Learn what actions work best

```
UCT(Σ, s, h)
    if s ∈ S then
        return 0
    if h = 0 then
        return V_0(s)
    if s ∉ Envelope then
        add s to Envelope
        n(s) ← 0
        for all a ∈ Applicable(s) do
            Q(s, a) ← 0
            n(s, a) ← 0
    Untried ← {a ∈ Applicable(s) | n(s, a)=0}
    if Untried ≠ ∅ then
        ā ← Choose(Untried)
    else
        ā ← argmin_a∈Applicable(s) \{Q(s, a) - C[log(n(s))/n(s, a)]^u\}
    s′ ← Sample(Σ, s, ā)
    cost-rollout ← cost(s, ā) + UCT(s′, h-1)
    Q(s, ā) ← [n(s, ā)⋅Q(s, ā)+cost-rollout] / (1+n(s, ā))
    n(s) ← n(s) + 1
    n(s, ā) ← n(s, ā) + 1
    return cost-rollout
```
• Generate Monte Carlo rollouts using a modified version of UCT
  • Rollout: game is played out to very end by selecting moves at random, result of each playout used to weight nodes in game tree
• 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
• Produced the first computer programs to play Go well
  • ≈ 2008–2012
• Monte Carlo rollout techniques similar to UCT were used to train AlphaGo
Intermediate Summary

- Run-Lookahead
- Reinforcement learning
  - Passive learning
    - DUE
    - ADP
    - TD
  - Active learning
    - Active ADP
    - Q-learning
  - Multi-armed bandit problem
    - UCB, UCT
Outline per the Book

6.2 Stochastic shortest path problems
   Safe/unsafe policies
   Optimality
   Policy iteration, value iteration

6.3 Heuristic search algorithms
   Best-first search
   Determinisation

6.4 Online probabilistic planning
   Lookahead
   Reinforcement learning

⇒ Next: More on Decision Making