Automated Planning and Scheduling

Lecture 12: Acting and Planning under Uncertainty

Tomáš Balyo, Dominik Schreiber | January 15, 2020
Acting and Planning under Uncertainty: “Select Chapters”

- Admitting non-determinism: Markov Decision Processes (MDP)
- Admitting partial observability: Partially Observable MDP (POMDP)

(A bit) More information: [GNT04]
Dimensions of Uncertainty

Assumptions we made at the start:

- State transitions (i.e. actions) are deterministic
- Each state is perfectly well-known (fully observable world)
- Goals are a fixed subset of states to reach
Dimensions of Uncertainty

Assumptions we made at the start:

- State transitions (i.e. actions) are deterministic
- Each state is perfectly well-known (fully observable world)
- Goals are a fixed subset of states to reach

What about the following scenarios?

- It might rain or it might not rain.
- Picking up a package fails in 1 out of 1000 attempts, resulting the package to break.
- I want to maximize the overall amount of made cocktails.
Non-deterministic Planning (1)

Stochastic System

A stochastic system is a tuple $\Sigma = (S, A, P)$, where $S$ are states, $A$ are actions, and $P_a(s'|s)$ is a probability distribution over $s'$, i.e. $\sum_{s' \in S} P_a(s'|s) = 1$ for all $(s, a)$. 
Non-deterministic Planning (1)

Stochastic System

A **stochastic system** is a tuple $\Sigma = (S, A, P)$, where $S$ are states, $A$ are actions, and $P_a(s'|s)$ is a probability distribution over $s'$, i.e. $\sum_{s'\in S} P_a(s'|s) = 1$ for all $(s, a)$.

Example: $s := \{t@A, p1@A\}$, $a := \text{load-p1}$

- $P_{\text{load-p1}}(\{t@A, p1@t\}|s) := 0.999$ (package is properly loaded)
- $P_{\text{load-p1}}(\{t@A\}|s) := 0.001$ (package breaks, “vanishes”)
- $P_{\text{load-p1}} \equiv 0$ in all other cases
Non-deterministic Planning (2)

Applicable actions

Given a stochastic system \( \Sigma = (S, A, P) \) and a state \( s \), we call
\[
A(s) := \{ a \in A \mid \exists s' \in S : P_a(s' | s) \neq 0 \}
\]
the applicable actions in state \( s \).

- All actions with some transition probability are applicable
Non-deterministic Planning (2)

Applicable actions

Given a stochastic system $\Sigma = (S, A, P)$ and a state $s$, we call $A(s) := \{a \in A \mid \exists s' \in S : P_a(s'|s) \neq 0\}$ the applicable actions in state $s$.

- All actions with some transition probability are applicable
- Implicit preconditions (hidden in probability distributions)
- All the world’s non-determinism is based on action effects
From Plans to Policies

What meaning would a classical goal have in stochastic systems?

- Find any (shortest?) path to goal with prob. $> 0$? $\geq 0.5$? $= 1$?
- Eventually reach the goal (infinite plan with finite description)^
- ...?
From Plans to Policies

What meaning would a classical goal have in stochastic systems?

- Find any (shortest?) path to goal with prob. $> 0$? $\geq 0.5$? $= 1$?
- Eventually reach the goal (infinite plan with finite description)?
- ...?

Instead: Depart from classical goal. What we actually want:

**Policy**

A *policy* $\pi$ maps each state $s \in S$ of a stochastic system to an action $a$. 
From Plans to Policies

What meaning would a classical goal have in stochastic systems?

- Find any (shortest?) \textit{path to goal} with prob. $> 0$? $\geq 0.5$? $= 1$?
- Eventually reach the \textit{goal} (infinite plan with finite description)?
- \ldots?

Instead: Depart from classical goal. What we actually want:

\textbf{Policy}

A \textit{policy} $\pi$ maps each state $s \in S$ of a stochastic system to an action $a$.

- Policies are \textit{deterministic}, unlike the world they are executed in
- Fits general “\textit{stimulus–response}” \textit{model} of autonomous agents:
  Sense a state $\Rightarrow$ Act $\Rightarrow$ Sense next state $\Rightarrow$ \ldots
Histories

Policy Execution, History

Executing a policy $\pi$ from a state $s_0$ probabilistically leads to some infinite state sequence $\langle s_0, s_1, s_2, \ldots \rangle$ called a history.

- Some histories are more likely than others
Histories

Policy Execution, History

Executing a policy $\pi$ from a state $s_0$ probabilistically leads to some infinite state sequence $\langle s_0, s_1, s_2, \ldots \rangle$ called a history.

- Some histories are more likely than others:
  - Calculate probability of a history $h = \langle s_0, s_1, \ldots \rangle$ given a policy $\pi$ by
    \[
    P(h|\pi) = \prod_{i \geq 0} P_{\pi(s_i)}(s_{i+1}|s_i)
    \]
- Product of all transition probabilities when applying actions as decided by policy
Example: Unreliable Wagon

- States: Location of a wagon \((x_0, \ldots, x_6)\)
- Two actions: \textit{steer-left}, \textit{steer-right}
Example: Unreliable Wagon

- States: Location of a wagon \((x_0, \ldots, x_6)\)
- Two actions: steer-left, steer-right
- For each junction origin \(\Rightarrow \{\text{left}, \text{right}\}\):

  \[
  P_{\text{steer-left}}(\text{at(left)}|\text{at(origin)}) = 0.9, \\
  P_{\text{steer-left}}(\text{at(right)}|\text{at(origin)}) = 0.1, \\
  P_{\text{steer-right}}(\text{at(left)}|\text{at(origin)}) = 0.1, \\
  P_{\text{steer-right}}(\text{at(right)}|\text{at(origin)}) = 0.9,
  \]

Policy \(\pi\): "always steer left"; 
\(\forall s \in S: \pi(s) = \text{steer-left}\)
Example: Unreliable Wagon

- States: Location of a wagon (x0, ..., x6)
- Two actions: steer-left, steer-right
- For each junction origin ⇒ {left, right}:
  \[ P_{\text{steer-left}}(\text{at(left)}|\text{at(origin)}) = 0.9, \]
  \[ P_{\text{steer-left}}(\text{at(right)}|\text{at(origin)}) = 0.1, \]
  \[ P_{\text{steer-right}}(\text{at(left)}|\text{at(origin)}) = 0.1, \]
  \[ P_{\text{steer-right}}(\text{at(right)}|\text{at(origin)}) = 0.9, \]
- Policy \( \pi \): “always steer left”;
  \[ \forall s \in S : \pi(s) = \text{steer-left} \]
- History \( h_1 = \langle x_0, x_1, x_2, x_4, x_4, x_4, ... \rangle \)
Example: Unreliable Wagon

- States: Location of a wagon \((x_0, \ldots, x_6)\)
- Two actions: steer-left, steer-right
- For each junction origin \(\Rightarrow \{\text{left, right}\}:\)
  \[
  P_{\text{steer-left}}(\text{at(left)}|\text{at(origin)}) = 0.9, \\
  P_{\text{steer-left}}(\text{at(right)}|\text{at(origin)}) = 0.1, \\
  P_{\text{steer-right}}(\text{at(left)}|\text{at(origin)}) = 0.1, \\
  P_{\text{steer-right}}(\text{at(right)}|\text{at(origin)}) = 0.9,
  \]
- Policy \(\pi\): “always steer left”;
  \(\forall s \in S: \pi(s) = \text{steer-left}\)
- History \(h_1 = \langle x_0, x_1, x_2, x_4, x_4, x_4, \ldots \rangle\)
  \(\Rightarrow P(h_1|\pi) = 0.9 \cdot 0.9 \cdot 0.9 \cdot 1 \cdot 1 \cdot \ldots = 0.729\)
- History \(h_2 = \langle x_0, x_1, x_3, x_6, x_6, x_6, \ldots \rangle\)
Example: Unreliable Wagon

- States: Location of a wagon \((x_0, \ldots, x_6)\)
- Two actions: \text{steer-left}, \text{steer-right}
- For each junction \(\text{origin} \Rightarrow \{\text{left}, \text{right}\}:\)
  \[ P_{\text{steer-left}}(\text{at(left)}|\text{at(origin)}) = 0.9, \]
  \[ P_{\text{steer-left}}(\text{at(right)}|\text{at(origin)}) = 0.1, \]
  \[ P_{\text{steer-right}}(\text{at(left)}|\text{at(origin)}) = 0.1, \]
  \[ P_{\text{steer-right}}(\text{at(right)}|\text{at(origin)}) = 0.9, \]
- Policy \(\pi\): “always steer left”;
  \[ \forall s \in S : \pi(s) = \text{steer-left} \]
- History \(h_1 = \langle x_0, x_1, x_2, x_4, x_4, x_4, \ldots \rangle\)
  \[ \Rightarrow P(h_1|\pi) = 0.9 \cdot 0.9 \cdot 0.9 \cdot 1 \cdot 1 \cdot \ldots = 0.729 \]
- History \(h_2 = \langle x_0, x_1, x_3, x_6, x_6, x_6, \ldots \rangle\)
  \[ \Rightarrow P(h_2|\pi) = 0.9 \cdot 0.1 \cdot 0.9 \cdot 1 \cdot 1 \cdot \ldots = 0.081 \]
Goal of stochastic planning: find a policy
Finding any policy is trivial – what is a good policy?
Rate My State

- Goal of stochastic planning: find a policy
- Finding any policy is trivial – what is a good policy?

**Costs and Rewards**

A **reward function** $R : S \rightarrow \mathbb{R}$ rates a state $s$. A **cost function** $C : S \times A \rightarrow \mathbb{R}$ describes the cost of an action $a$ in a state $s$. 

A utility function $V$ rates a history $h$ from a policy $\pi$ as $V(h | \pi) := \sum_{i \geq 0} \gamma^i (R(s_i) - C(s_i, \pi(s_i)))$ where $0 < \gamma \leq 1$ is a discount factor.
Goal of stochastic planning: find a policy

Finding any policy is trivial – what is a **good policy**?

### Costs and Rewards

A **reward function** $R : S \rightarrow \mathbb{R}$ rates a state $s$. A **cost function** $C : S \times A \rightarrow \mathbb{R}$ describes the cost of an action $a$ in a state $s$.

### Utility

A **utility function** $V$ rates a history $h$ from a policy $\pi$ as

\[
V(h|\pi) := \sum_{i \geq 0} \gamma^i (R(s_i) - C(s_i, \pi(s_i)))
\]

where $0 < \gamma \leq 1$ is a **discount factor**.
Example: Unreliable Wagon (2)

- Cost function $C \equiv 0$;
- Reward function: $R(x) = \begin{cases} 2, & \text{if } x = x_6, \\ 1, & \text{if } x = x_2, \\ 0, & \text{else} \end{cases}$
- History $h_1 = \langle x_0, x_1, x_2, x_4, x_4, x_4, \ldots \rangle$
Example: Unreliable Wagon (2)

- **Cost function** $C \equiv 0$;
- **Reward function**: 
  $$R(x) = \begin{cases} 
  2, & \text{if } x = x_6, \\
  1, & \text{if } x = x_2, \\
  0, & \text{else}
  \end{cases}$$

- **History** $h_1 = \langle x_0, x_1, x_2, x_4, x_4, x_4, \ldots \rangle$
  $$\Rightarrow V(h_1|\pi) = 0 + 0 + 1\gamma^2 + 0 + 0 + \ldots$$

- **History** $h_2 = \langle x_0, x_1, x_3, x_6, x_6, x_6, \ldots \rangle$
Example: Unreliable Wagon (2)

- Cost function $C \equiv 0$;
- Reward function: $R(x) = \begin{cases} 2, & \text{if } x = x_6, \\ 1, & \text{if } x = x_2, \\ 0, & \text{else} \end{cases}$
- History $h_1 = \langle x_0, x_1, x_2, x_4, x_4, x_4, \ldots \rangle$
  \[ \Rightarrow V(h_1|\pi) = 0 + 0 + 1\gamma^2 + 0 + 0 + \ldots \]
- History $h_2 = \langle x_0, x_1, x_3, x_6, x_6, x_6, \ldots \rangle$
  \[ \Rightarrow V(h_2|\pi) = 0 + 0 + 0 + 2\gamma^3 + 2\gamma^4 + 2\gamma^5 + \ldots \]

Purpose of $\gamma$?

- Sooner reward is better than later reward
- Keeps $V$ finite (usually)
Cost function $C \equiv 0$;

Reward function: $R(x) = \begin{cases} 
2, & \text{if } x = x_6, \\
1, & \text{if } x = x_2, \\
0, & \text{else} 
\end{cases}$

History $h_1 = \langle x_0, x_1, x_2, x_4, x_4, x_4, \ldots \rangle$
\[ \Rightarrow V(h_1|\pi) = 0 + 0 + 1\gamma^2 + 0 + 0 + \ldots \]

History $h_2 = \langle x_0, x_1, x_3, x_6, x_6, x_6, \ldots \rangle$
\[ \Rightarrow V(h_2|\pi) = 0 + 0 + 0 + 2\gamma^3 + 2\gamma^4 + 2\gamma^5 + \ldots \]

$\gamma = 0.1$:
Example: Unreliable Wagon (2)

- Cost function $C \equiv 0$;
- Reward function: $R(x) = \begin{cases} 2, & \text{if } x = x_6, \\ 1, & \text{if } x = x_2, \\ 0, & \text{else} \end{cases}$
- History $h_1 = \langle x_0, x_1, x_2, x_4, x_4, x_4, \ldots \rangle$
  $\Rightarrow V(h_1|\pi) = 0 + 0 + 1\gamma^2 + 0 + 0 + \ldots$
- History $h_2 = \langle x_0, x_1, x_3, x_6, x_6, x_6, \ldots \rangle$
  $\Rightarrow V(h_2|\pi) = 0 + 0 + 0 + 2\gamma^3 + 2\gamma^4 + 2\gamma^5 + \ldots$
- $\gamma = 0.1 : V(h_1|\pi) = 0.01 > 0.003 > V(h_2|\pi)$
- $\gamma = 0.5 :$
Example: Unreliable Wagon (2)

Cost function $C \equiv 0$;

Reward function: $R(x) = \begin{cases} 
2, & \text{if } x = x_6, \\
1, & \text{if } x = x_2, \\
0, & \text{else} 
\end{cases}$

History $h_1 = \langle x_0, x_1, x_2, x_4, x_4, x_4, \ldots \rangle$

$\Rightarrow V(h_1|\pi) = 0 + 0 + 1\gamma^2 + 0 + 0 + \ldots$

History $h_2 = \langle x_0, x_1, x_3, x_6, x_6, x_6, \ldots \rangle$

$\Rightarrow V(h_2|\pi) = 0 + 0 + 0 + 2\gamma^3 + 2\gamma^4 + 2\gamma^5 + \ldots$

$\gamma = 0.1: V(h_1|\pi) = 0.01 > 0.003 > V(h_2|\pi)$

$\gamma = 0.5: V(h_1|\pi) = 0.25 < 0.5 = V(h_2|\pi)$

Purpose of $\gamma$?
Cost function $C \equiv 0$;

Reward function:

\[
R(x) = \begin{cases} 
2, & \text{if } x = x6, \\
1, & \text{if } x = x2, \\
0, & \text{else}
\end{cases}
\]

History $h_1 = \langle x0, x1, x2, x4, x4, x4, \ldots \rangle$

\[ V(h_1|\pi) = 0 + 0 + 1\gamma^2 + 0 + 0 + \ldots \]

History $h_2 = \langle x0, x1, x3, x6, x6, x6, \ldots \rangle$

\[ V(h_2|\pi) = 0 + 0 + 0 + 2\gamma^3 + 2\gamma^4 + 2\gamma^5 + \ldots \]

$\gamma = 0.1: V(h_1|\pi) = 0.01 > 0.003 > V(h_2|\pi)$

$\gamma = 0.5: V(h_1|\pi) = 0.25 < 0.5 = V(h_2|\pi)$

Purpose of $\gamma$?

- Sooner reward is better than later reward
- Keeps $V$ finite (usually)
Policy Rating

- A good policy should be likely to lead to histories with high utility:

**Expected Utility**

Given a policy $\pi$, the expected utility of $\pi$ is defined as

$$E(\pi) = \sum_h P(h|\pi)V(h|\pi).$$
Policy Rating

- A good policy should be likely to lead to histories with high utility:

Expected Utility

Given a policy $\pi$, the expected utility of $\pi$ is defined as

$$E(\pi) = \sum_h P(h|\pi) V(h|\pi).$$

- Just the definition of expected value for random variable $V(h)$
- Now we can rate and compare policies by expected utility
Example: Unreliable Wagon (3)

- Reward function: \( R(x) = \begin{cases} 
1 & \text{if } x = x_6, \\
0 & \text{else} 
\end{cases} \)

- Policy \( \pi_r \): “always steer right”

- Histories with non-zero utility:
  \[ \langle x_0, x_3, x_6, \ldots \rangle, \langle x_0, x_1, x_3, x_6, \ldots \rangle, \langle x_0, x_3, x_3, x_6, \ldots \rangle, \langle x_0, x_1, x_3, x_3, x_6, \ldots \rangle, \ldots \]
Example: Unreliable Wagon (3)

- Reward function: \( R(x) = \begin{cases} 1 & \text{if } x = x_6, \\ 0 & \text{else} \end{cases} \)
- Policy \( \pi_r \): “always steer right”
- Histories with non-zero utility:
  \[ \langle x_0, x_3, x_6, \ldots \rangle, \langle x_0, x_1, x_3, x_6, \ldots \rangle, \langle x_0, x_3, x_3, x_6, \ldots \rangle, \langle x_0, x_1, x_3, x_3, x_6, \ldots \rangle, \ldots \]
- \( E(\pi_r) = (0.9 \cdot 0.1) \cdot \gamma^2 + (0.1 \cdot 0.9 \cdot 0.1) \cdot \gamma^3 + (0.9 \cdot 0.9 \cdot 0.1) \cdot \gamma^3 + (0.1 \cdot 0.9 \cdot 0.9 \cdot 0.1) \cdot \gamma^4 + \ldots \)

Is this policy better than the policy \( \pi'_r \): \( x_0 \mapsto \text{steer-right}, \ast \mapsto \text{steer-left} \) for very small \( \gamma \), e.g. \( \gamma = 0.001 \)?

For very large \( \gamma \), e.g. \( \gamma = 0.999 \)?
Example: Unreliable Wagon (3)

- Reward function: \( R(x) = \begin{cases} 1 & \text{if } x = x_6, \\ 0 & \text{else} \end{cases} \)
- Policy \( \pi_r \): “always steer right”
- Histories with non-zero utility:
  \( \langle x_0, x_3, x_6, \ldots \rangle, \langle x_0, x_1, x_3, x_6, \ldots \rangle, \langle x_0, x_3, x_3, x_6, \ldots \rangle, \langle x_0, x_1, x_3, x_3, x_6, \ldots \rangle, \ldots \)
- \( E(\pi_r) = (0.9 \cdot 0.1) \cdot \gamma^2 + (0.1 \cdot 0.9 \cdot 0.1) \cdot \gamma^3 + (0.9 \cdot 0.9 \cdot 0.1) \cdot \gamma^3 + (0.1 \cdot 0.9 \cdot 0.9 \cdot 0.1) \cdot \gamma^4 + \ldots \)
- Is this policy better than the policy \( \pi' : x_0 \mapsto \text{steer-right}, * \mapsto \text{steer-left} \):
  - for very small \( \gamma \), e.g. \( \gamma = 0.001 \)?
  - for very large \( \gamma \), e.g. \( \gamma = 0.999 \)?
Assembling everything:

**Markov Decision Process**

A Markov Decision Process (MDP) is a stochastic system \( \Sigma = (S, A, P) \) with a reward function \( R \), a cost function \( C \), and a utility function \( V \). The solution of a MDP is a policy \( \pi^* \) which maximizes \( E(\pi^*) \).

- Special cases:
  - Reward function \( R \equiv 0 \) \( \Rightarrow \) only cost function \( C \) matters
  - Cost function \( C \equiv 0 \) \( \Rightarrow \) only reward function \( R \) matters
Markov Decision Processes

Assembling everything:

Markov Decision Process

A Markov Decision Process (MDP) is a stochastic system $\Sigma = (S, A, P)$ with a reward function $R$, a cost function $C$, and a utility function $V$. The solution of a MDP is a policy $\pi^*$ which maximizes $E(\pi^*)$.

- Special cases:
  - Reward function $R \equiv 0 \implies$ only cost function $C$ matters
  - Cost function $C \equiv 0 \implies$ only reward function $R$ matters

- More abstract than classical planning:
  Result is not a plan, but an “online plan generator” $\pi^*$
A Markov Decision Process (MDP) is a stochastic system $\Sigma = (S, A, P)$ with a reward function $R$, a cost function $C$, and a utility function $V$. The solution of a MDP is a policy $\pi^*$ which maximizes $E(\pi^*)$.

- Special cases:
  - Reward function $R \equiv 0 \Rightarrow$ only cost function $C$ matters
  - Cost function $C \equiv 0 \Rightarrow$ only reward function $R$ matters

- More abstract than classical planning:
  Result is not a plan, but an “online plan generator” $\pi^*$

- Solutions to MDP are defined by optimality:
  - A suboptimal plan is a valid solution for a planning problem
  - A suboptimal policy is not a solution for a MDP
How can we plan an optimal policy in MDPs?

An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision.

— Bellman, 1957 [Bel57]
How can we plan an optimal policy in MDPs?

An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision.

— Bellman, 1957 [Bel57]

Let $E(s)$ be the expected utility from some state $s$

For the expected utility under an optimal policy, $E^*(s)$, we have:

$$E^*(s) = \max_{a \in A} \left( R(s) - C(s, a) + \gamma \sum_{s' \in S} P_a(s'|s) E^*(s') \right)$$

(Bellman Equation)
How can we plan an optimal policy in MDPs?

An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision. — Bellman, 1957 [Bel57]

Let $E(s)$ be the expected utility from some state $s$.

For the expected utility under an optimal policy, $E^*(s)$, we have:

$$E^*(s) = \max_{a \in A} \left( R(s) - C(s, a) + \gamma \sum_{s' \in S} P_a(s' | s) E^*(s') \right)$$

(Bellman Equation)

Recursive definition of $E^*(s)$ $\Rightarrow$ Find fixpoint!
Algorithm 1: Value Iteration

Simple iterative process to find optimal expected utility (and policy):

- **Initialization:** \( \forall s : E^0(s) := 0 \)
Algorithm 1: Value Iteration

Simple iterative process to find optimal expected utility (and policy):

- **Initialization**: \( \forall s : E^0(s) := 0 \)
- **One iteration**:

\[
\forall s : E^{i+1}(s) := \max_a \left( R(s) - C(s, a) + \gamma \sum_{s'} P_a(s'|s) E^i(s') \right)
\]

How to extract optimal policy \( \pi^* \)?
Pick actions which are chosen in above formula in final iteration
\[ \pi^*(s, a) := \arg\max_a \left( R(s) - C(s, a) + \gamma \sum_{s'} P_a(s'|s) E^*(s') \right) \]

Can stop calculation at some \( i \) such that \( \forall s : |E^i(s) - E^*(s)| < \epsilon \)
\( \Rightarrow \epsilon \)-approximation of \( E^* \)
Algorithm 1: Value Iteration

Simple iterative process to find optimal expected utility (and policy):

- **Initialization**: \( \forall s : E^0(s) := 0 \)
- **One iteration**:

\[
\forall s : E^{i+1}(s) := \max_a \left( R(s) - C(s, a) + \gamma \sum_{s'} P_a(s'|s) E^i(s') \right)
\]

- \( E^i \) converges to \( E^* \) for \( i \to \infty \) (bounded in practice)
- How to extract optimal policy \( \pi^* \)?
Algorithm 1: Value Iteration

Simple iterative process to find optimal expected utility (and policy):

- **Initialization**: \( \forall s : E^0(s) := 0 \)
- **One iteration**:

\[
\forall s : E^{i+1}(s) := \max_a \left( R(s) - C(s, a) + \gamma \sum_{s'} P_a(s'|s) E^i(s') \right)
\]

- \( E^i \) converges to \( E^* \) for \( i \to \infty \) (bounded in practice)
- **How to extract optimal policy** \( \pi^* \)?
  - Pick actions which are chosen in above formula in final iteration
  - \( \pi^*(s, a) := \arg\max_a \left( R(s) - C(s, a) + \gamma \sum_{s'} P_a(s'|s) E^*(s') \right) \)
Algorithm 1: Value Iteration

Simple iterative process to find optimal expected utility (and policy):

- **Initialization**: \( \forall s : E^0(s) := 0 \)
- **One iteration**:

\[
\forall s : E^{i+1}(s) := \max_a \left( R(s) - C(s, a) + \gamma \sum_{s'} P_a(s'|s) E^i(s') \right)
\]

- \( E^i \) converges to \( E^* \) for \( i \to \infty \) (bounded in practice)
- **How to extract optimal policy** \( \pi^* \)?
  - Pick actions which are chosen in above formula in final iteration
  - \( \pi^*(s, a) := \arg\max_a \left( R(s) - C(s, a) + \gamma \sum_{s'} P_a(s'|s) E^*(s') \right) \)
- Can stop calculation at some \( i \) such that \( \forall s : |E^i(s) - E^*(s)| < \varepsilon \)
  \( \Rightarrow \varepsilon\)-approximation of \( E^* \)
Algorithm 2: Policy Iteration

Alternative procedure: Incrementally update policy $\pi$ instead of utility

- Initialization: $\forall s : \pi^0(s) = [\text{some random action}]$
Algorithm 2: Policy Iteration

Alternative procedure: Incrementally update policy $\pi$ instead of utility

- **Initialization:** $\forall s : \pi^0(s) = [\text{some random action}]$
- **One iteration:** Solve system of $|S|$ linear equations

$$E(s) = R(s) - C(s, \pi^i(s)) + \gamma \sum_{s' \in S} P_{\pi^i(s)}(s' | s) E(s')$$

for each $E(s), s \in S$
Algorithm 2: Policy Iteration

Alternative procedure: Incrementally update policy $\pi$ instead of utility

- **Initialization:** $\forall s : \pi^0(s) = \text{[some random action]}$
- **One iteration:** Solve system of $|S|$ linear equations

$$E(s) = R(s) - C(s, \pi^i(s)) + \gamma \sum_{s' \in S} P_{\pi^i(s)}(s'|s) \cdot E(s')$$

for each $E(s), s \in S$. Then update policy for each $s$:

$$\pi^{i+1}(s) := \arg\max_{a \in A} \left( R(s) - C(s, a) + \gamma \sum_{s' \in S} P_a(s'|s) \cdot E(s') \right)$$
Algorithm 2: Policy Iteration

Alternative procedure: Incrementally update policy $\pi$ instead of utility

- **Initialization:** $\forall s : \pi^0(s) = [\text{some random action}]$
- **One iteration:** Solve system of $|S|$ linear equations

$$E(s) = R(s) - C(s, \pi^i(s)) + \gamma \sum_{s' \in S} P_{\pi^i(s)}(s'|s) \ E(s')$$

for each $E(s), s \in S$. Then update policy for each $s$:

$$\pi^{i+1}(s) := \arg\max_{a \in A} \left( R(s) - C(s, a) + \gamma \sum_{s' \in S} P_a(s'|s) \ E(s') \right)$$

- More work per iteration, but can converge faster than value iteration
Algorithm 2: Policy Iteration

Alternative procedure: Incrementally update policy $\pi$ instead of utility

- **Initialization:** $\forall s : \pi^0(s) = [\text{some random action}]$
- **One iteration:** Solve system of $|S|$ linear equations

$$E(s) = R(s) - C(s, \pi^i(s)) + \gamma \sum_{s' \in S} P_{\pi^i(s)}(s'|s) E(s')$$

for each $E(s), s \in S$. Then update policy for each $s$:

$$\pi^{i+1}(s) := \arg\max_{a \in A} \left( R(s) - C(s, a) + \gamma \sum_{s' \in S} P_a(s'|s) E(s') \right)$$

- More work per iteration, but can converge faster than value iteration
- Instead of expensive argmax, can also use any action which brings an improvement over current $\pi$
Iterative Procedures: Discussion

Difference between value iteration and policy iteration?

- Value It.: Computes next iteration based on last iteration (direct computation)
- Policy It.: Computes next iteration based on current iteration (requires to solve a LSE)
Iterative Procedures: Discussion

Difference between value iteration and policy iteration?

- Value It.: Computes next iteration **based on last iteration** (direct computation)
- Policy It.: Computes next iteration **based on current iteration** (requires to solve a LSE)

Drawbacks to both approaches?
Iterative Procedures: Discussion

Difference between value iteration and policy iteration?

- Value It.: Computes next iteration based on last iteration (direct computation)
- Policy It.: Computes next iteration based on current iteration (requires to solve a LSE)

Drawbacks to both approaches?

- Iterates repeatedly over entire state space
Iterative Procedures: Discussion

Difference between value iteration and policy iteration?

- Value It.: Computes next iteration based on last iteration (direct computation)
- Policy It.: Computes next iteration based on current iteration (requires to solve a LSE)

Drawbacks to both approaches?

- Iterates repeatedly over entire state space
- But state space is huge!
- Mostly impractical for realistic problems

More feasible alternatives?

#neverforget
Idea: Real-Time Value Iteration

- Abandon explicit calculation over entire state space: back to heuristic search!
- Pick set of initial states, (set of goal states)
Idea: Real-Time Value Iteration

- Abandon explicit calculation over entire state space: back to heuristic search!
- Pick set of initial states, (set of goal states)
- Heuristic function: Choose action which maximizes expected utility
- General procedure:
  1. Visit a state \( s \)
  2. Perform one-step lookahead to learn more about expected utilities
  3. Update utility for \( s \)
  4. Transition to next state
Idea: Real-Time Value Iteration

- Abandon explicit calculation over entire state space: back to heuristic search!
- Pick set of initial states, (set of goal states)
- Heuristic function: Choose action which maximizes expected utility
- General procedure:
  1. Visit a state \( s \)
  2. Perform one-step lookahead to learn more about expected utilities
  3. Update utility for \( s \)
  4. Transition to next state

- Hidden in 4.: Exploitation vs. Exploration problem
  (pick best state known so far, or least known state?)
Idea: Real-Time Value Iteration

- Abandon explicit calculation over entire state space: back to heuristic search!
- Pick set of initial states, (set of goal states)
- Heuristic function: Choose action which maximizes expected utility
- General procedure:
  1. Visit a state $s$
  2. Perform one-step lookahead to learn more about expected utilities
  3. Update utility for $s$
  4. Transition to next state

- Hidden in 4.: Exploitation vs. Exploration problem
  (pick best state known so far, or least known state?)
- Guaranteed termination only under certain restrictions;
  no promise on optimality of final policy
Towards Partial Observability

- So far: Probabilistic transitions, but full observability
- Counter example: Realistic robot navigation
Towards Partial Observability

- So far: Probabilistic transitions, but full observability
- Counter example: Realistic robot navigation
  - Robot never knows exactly where it is
  - **Indicators:** GPS, cameras, collision sensors, ...
Towards Partial Observability

- So far: Probabilistic transitions, but full observability

- Counter example: Realistic robot navigation
  - Robot never knows exactly where it is
  - Indicators: GPS, cameras, collision sensors, ...
  - Robot senses environment, infers a belief state: probability distribution over possible states
  - Robot then moves according to most likely scenario
A Partially Observable Markov Decision Process (POMDP) is a stochastic system $\Sigma = (S, A, P)$ with a finite set of observations $O$ with probabilities $P_a(o|s)$ for $a \in A$, $s \in S$, $o \in O$. It must hold that $\sum_{o \in O} P_a(o|s) = 1$. 

Given you are in state $s$ and apply action $a$, what is the probability that you make the observation $o$?
A Partially Observable Markov Decision Process (POMDP) is a stochastic system \( \Sigma = (S, A, P) \) with a finite set of observations \( O \) with probabilities \( P_a(o|s) \) for \( a \in A, s \in S, o \in O \). It must hold that \( \sum_{o \in O} P_a(o|s) = 1 \).

- \( P_a(o|s) \) := “Given you are in state \( s \) and apply action \( a \), what is the probability that you make the observation \( o \)?”
A Partially Observable Markov Decision Process (POMDP) is a stochastic system $\Sigma = (S, A, P)$ with a finite set of observations $O$ with probabilities $P_a(o|s)$ for $a \in A$, $s \in S$, $o \in O$. It must hold that $\sum_{o \in O} P_a(o|s) = 1$.

- $P_a(o|s) := \text{“Given you are in state } s \text{ and apply action } a, \text{ what is the probability that you make the observation } o?\text{”}$
- Agent does not know current state $s$; can just sense observations and, on that base, apply actions
- Given an observation, can infer likelihood of being in some state
A belief state $b$ is a probability distribution over $S$, i.e. $\sum_{s \in S} b(s) = 1$.

- Initial belief state $b(s)$ is usually known, e.g.
  - $b(s) = 1$ for some known initial state $s$
  - Uniform distribution over all states (absolute uncertainty)
Belief States

Belief State

A belief state $b$ is a probability distribution over $S$, i.e. $\sum_{s \in S} b(s) = 1$.

- Initial belief state $b(s)$ is usually known, e.g.
  - $b(s) = 1$ for some known initial state $s$
  - Uniform distribution over all states (absolute uncertainty)

- Compute probability of an observation $o$ when applying action $a$:

$$P_a(o) = \sum_{s \in S} P_a(o|s) b(s)$$
Belief States

A belief state $b$ is a probability distribution over $S$, i.e. $\sum_{s \in S} b(s) = 1$.

- Initial belief state $b(s)$ is usually known, e.g.
  - $b(s) = 1$ for some known initial state $s$
  - Uniform distribution over all states (absolute uncertainty)

- Compute probability of an observation $o$ when applying action $a$:
  \[ P_a(o) = \sum_{s \in S} P_a(o|s) \cdot b(s) \]

- Calculate new belief state $b_a$ after applying an action $a$:
  \[ \forall s : b_a(s) := \sum_{s' \in S} P_a(s|s') \cdot b(s') \]
Example: Probabilistic Wagon (1)

Partially observable 1D movement of a wagon (discrete integer locations)

- Initial belief state: $b(2) = 0.8, \quad b(1) = b(3) = 0.1$
  (e.g. given by a GPS sensor)
Example: Probabilistic Wagon (1)

Partially observable 1D movement of a wagon (discrete integer locations)

- Initial belief state: $b(2) = 0.8$, $b(1) = b(3) = 0.1$
  (e.g. given by a GPS sensor)
- Action $a = \text{move-right}$: For each location $x$:
  $P_a(x + 2|x) = 0.8$, $P_a(x + 1|x) = P_a(x + 3|x) = 0.1$
Example: Probabilistic Wagon (1)

Partially observable 1D movement of a wagon (discrete integer locations)

- Initial belief state: $b(2) = 0.8$, $b(1) = b(3) = 0.1$
  (e.g. given by a GPS sensor)

- Action $a = \text{move-right}$: For each location $x$:
  
  \[
P_a(x + 2 | x) = 0.8, \quad P_a(x + 1 | x) = P_a(x + 3 | x) = 0.1
  \]

- Belief state after applying $a$:
  
  \[
b_a(x) = 0.1 \cdot b(x - 1) + 0.8 \cdot b(x - 2) + 0.1 \cdot b(x - 3)
  \]

  \[
  \Rightarrow b_a(2) = b_a(6) = 0.01, \quad b_a(3) = b_a(5) = 0.16, \quad b_a(4) = 0.66
  \]
Agent knows current belief state $b$, action $a$ and observation $o$:
How to compute new belief state $b'(s)$?

Just compute $b'(s) := b(a(s))$.

Update belief based on previous belief and state transition probabilities:

Ignores observation.

Problem: Belief will get more and more uncertain!

Better: Use likelihood of made observation after computing $b(a(s))$:

$b'(s) := P(a(o|s)) = P(a(o|s)) b(a(s)) P(a(o))$ (Bayes rule \[BPC63\])

Allows to transform a prior belief and the likelihood and overall probability of an observation into a posterior belief.

Central equation for Artificial Intelligence!
Bayesian Inference

- Agent knows current belief state $b$, action $a$ and observation $o$: How to compute new belief state $b'(s)$?
- Just compute $b'(s) := b_a(s)$?
  - Update belief based on previous belief and state transition probabilities
  - Ignores observation
  - Problem: Belief will get more and more uncertain!
Bayesian Inference

- Agent knows current belief state $b$, action $a$ and observation $o$:
  - How to compute new belief state $b'(s)$?
- Just compute $b'(s) := b_a(s)$?
  - Update belief based on previous belief and state transition probabilities
  - Ignores observation
  - Problem: Belief will get more and more uncertain!
- Better: Use likelihood of made observation after computing $b_a(s)$
Bayesian Inference

- Agent knows current belief state $b$, action $a$ and observation $o$: How to compute new belief state $b'(s)$?
- Just compute $b'(s) := b_a(s)$?
  - Update belief based on previous belief and state transition probabilities
  - Ignores observation
  - Problem: Belief will get more and more uncertain!
- Better: Use likelihood of made observation after computing $b_a(s)$

\[
b'(s) := P_a(s|o) = \frac{P_a(o|s) b_a(s)}{P_a(o)} \quad \text{(Bayes rule [BPC63])}
\]

- Allows to transform a prior belief and the likelihood and overall probability of an observation into a posterior belief
- Central equation for Artificial Intelligence!
Prior belief (right after applying $a$):

$$b_a(2) = b_a(6) = 0.01, \quad b_a(3) = b_a(5) = 0.16, \quad b_a(4) = 0.66$$
Example: Probabilistic Wagon (2)

- Prior belief (right after applying $a$):
  \[ b_a(2) = b_a(6) = 0.01, \quad b_a(3) = b_a(5) = 0.16, \quad b_a(4) = 0.66 \]

- GPS observation $o := \text{“I am at location 3”}$
  \[ P_a(o|2) := P_a(o|4) := 0.1, \quad P_a(o|3) := 0.8 \quad \text{(GPS is noisy!)} \]
  Probability of $o$: \[ P_a(o) = \sum_x b_a(x) P_a(o|x) \]
  \[ = 0.01 \cdot 0.1 + 0.16 \cdot 0.8 + 0.66 \cdot 0.1 = 0.195 \]
Example: Probabilistic Wagon (2)

- **Prior belief** (right after applying \( a \)):
  \[ b_a(2) = b_a(6) = 0.01, \quad b_a(3) = b_a(5) = 0.16, \quad b_a(4) = 0.66 \]

- **GPS observation** \( o := \text{"I am at location 3"} \)
  \[ P_a(o|2) := P_a(o|4) := 0.1, \quad P_a(o|3) := 0.8 \quad \text{(GPS is noisy!)} \]
  Probability of \( o \):
  \[ P_a(o) = \sum_x b_a(x) P_a(o|x) \]
  \[ = 0.01 \cdot 0.1 + 0.16 \cdot 0.8 + 0.66 \cdot 0.1 = 0.195 \]

- **Posterior belief**:
  \[ b'(x) = P_a(o|x) b_a(x) / P_a(o) \]
  \[ \Rightarrow b'(2) \approx 0.005, \quad b'(3) \approx 0.656, \quad b'(4) \approx 0.338 \]
POMDP in Action

Procedure of an acting agent in a POMDP:

1. Start with some initial belief state $b$
2. Based on $b$, apply some action $a$ and make an observation $o$
3. Calculate prior belief state $b_a$
4. Infer new belief state $b'$
5. Back to (2)
POMDP in Action

Procedure of an acting agent in a POMDP:

1. Start with some initial belief state $b$
2. Based on $b$, apply some action $a$ and make an observation $o$
3. Calculate prior belief state $b_a$
4. Infer new belief state $b'$
5. Back to (2)

Where would a policy come into play?

- Step (2): Map a belief state to an action
- Robotic navigation example:
  Policy is usually specifically written for robot’s task
- Can we still domain-independently find an optimal policy?
Forward Search in POMDP?

As always, let’s try to generalize SSS . . .

- Basically, interpret POMDP as a MDP where $S$ is set of belief states
As always, let’s try to generalize SSS . . .

- Basically, interpret POMDP as a MDP where \( S \) is set of belief states
  - Renders problem effectively fully observable
Forward Search in POMDP?

As always, let’s try to generalize SSS . . .

- Basically, interpret POMDP as a MDP where $S$ is set of belief states
  - Renders problem effectively fully observable
  - State space becomes infinite and continuous

Tomáš Balyo, Dominik Schreiber – Planning and Scheduling
January 15, 2020
Forward Search in POMDP?

As always, let’s try to generalize SSS . . .

- Basically, interpret POMDP as a MDP where $S$ is set of belief states
  - Renders problem effectively fully observable
  - State space becomes infinite and continuous
- Use adjusted version of MDP forward search
  - Traverse belief states instead of actual states
  - Explore expected utility of next possible belief states (one step look-ahead)
  - Pick action promising best utility
Forward Search in POMDP?

As always, let’s try to generalize SSS . . .

- Basically, interpret POMDP as a MDP where $S$ is set of belief states
  - Renders problem effectively fully observable
  - State space becomes infinite and continuous
- Use adjusted version of MDP forward search
  - Traverse belief states instead of actual states
  - Explore expected utility of next possible belief states
    (one step look-ahead)
  - Pick action promising best utility
- No optimality, termination not guaranteed
From Planning to Learning

The more uncertainty, the more we leave the area of planning.
From Planning to Learning

The more uncertainty, the more we leave the area of planning.

- What if not even probability of observations, \( P_a(o|s) \), is given?
  \( \Rightarrow \) Can only infer “rules” of the world by trial and error
From Planning to Learning

The more uncertainty, the more we leave the area of planning.

- What if not even probability of observations, $P_a(o|s)$, is given?
  - Can only infer “rules” of the world by trial and error
- Planning always requires a clear model to build upon!
- Rather use techniques from research area of (Machine) Learning:
  - Reinforcement Learning
  - Monte Carlo Tree Search methods
  - Bayesian Belief Networks
  - Regression, Clustering, Neural Networks, ...
Stay tuned!

Next lecture: Automated Scheduling
