Bayesian Methods for Data Analysis in Software Engineering

Probabilistic Sequential Decision Making

Mohan Sridharan[1] Akbar Siami Namin[2]

[1] Stochastic Estimation and Autonomous Robotics (SEAR) Lab
Department of Computer Science
Texas Tech University

[2] AdVanced Empirical Software Testing and Analysis (AVESTA) Research Group
Department of Computer Science
Texas Tech University

May 3, 2010; Cape Town

Session 4

- **9.00–10.30**:
  - Introduction.
  - Statistical analysis; hypothesis testing.
  - Basic probability, Bayes' rule.
- **11.00–12.30**:
  - Bayesian classification.
  - Bayesian regression.
  - Bayesian inference.
- **14.00–15.30**:
  - Information theory.
  - Stochastic sampling.
- **16.00–17.30**:
  - Markov decision processes.
  - Partially observable Markov decision processes.
  - Discussion.

Session 4: MDP

- **9.00–10.30**:
  - Introduction.
  - Statistical analysis; hypothesis testing.
  - Basic probability, Bayes' rule.
- **11.00–12.30**:
  - Bayesian classification.
  - Bayesian regression.
  - Bayesian inference.
- **14.00–15.30**:
  - Information theory.
  - Stochastic sampling.
- **16.00–17.30**:
  - Markov decision processes.
  - Partially observable Markov decision processes.
  - Discussion.

Planning and Decision making

- State estimation typically *not* the end goal.
- Estimated state used to determine appropriate actions.
- Challenges:
  - *Non-determinism*: action outcomes unreliable.
  - *Partial observability*: system state not directly observable.
  - *Computational complexity*: actions are computationally expensive.
- Solution: plan a sequence of actions to perform a given task reliably and efficiently.

Planning and Decision making

- State estimation typically *not* the end goal.
- Estimated state used to determine appropriate actions.

- Challenges:
  - *Non-determinism*: action outcomes unreliable.
  - *Partial observability*: system state not directly observable.
  - *Computational complexity*: actions are computationally expensive.

- Solution: plan a sequence of actions to perform a given task reliably and efficiently.

Planning and Decision making

- State estimation typically *not* the end goal.
- Estimated state used to determine appropriate actions.

- Challenges:
  - *Non-determinism*: action outcomes unreliable.
  - *Partial observability*: system state not directly observable.
  - *Computational complexity*: actions are computationally expensive.

- Solution: plan a sequence of actions to perform a given task reliably and efficiently.

Planning and Decision making

- State estimation typically *not* the end goal.
- Estimated state used to determine appropriate actions.

- Challenges:
  - *Non-determinism*: action outcomes unreliable.
  - *Partial observability*: system state not directly observable.
  - *Computational complexity*: actions are computationally expensive.

- Solution: plan a sequence of actions to perform a given task reliably and efficiently.

Probabilistic Sequential Decision Making

- Probabilistic representation of the non-deterministic actions.
- Used in several applications in many different fields: computer vision, robotics, computer games.
- Different methods such as MDP and POMDP.
- Applicable to software engineering and program analysis!

Probabilistic Sequential Decision Making

- Probabilistic representation of the non-deterministic actions.
- Used in several applications in many different fields: computer vision, robotics, computer games.
- Different methods such as MDP and POMDP.
- Applicable to software engineering and program analysis!

Probabilistic Sequential Decision Making

- Probabilistic representation of the non-deterministic actions.
- Used in several applications in many different fields: computer vision, robotics, computer games.
- Different methods such as MDP and POMDP.
- Applicable to software engineering and program analysis!

Probabilistic Sequential Decision Making

- Probabilistic representation of the non-deterministic actions.
- Used in several applications in many different fields: computer vision, robotics, computer games.
- Different methods such as MDP and POMDP.
- Applicable to software engineering and program analysis!

MDP Formulation

- Defined by the tuple: $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$.
- $\mathcal{S} = \{s_1, \ldots, s_N\}$ is the set of states.
- $\mathcal{A} = \{a_1, \ldots, a_M\}$ is the set of actions.
- $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S}' \rightarrow [0, 1]$ is the state transition matrix. $\mathcal{T}(s, a, s') = \mathcal{P}^a_{ss'} = p(s_{t+1} = s' | s_t = s, a_t = a)$.
- $\mathcal{R} : \mathcal{S} \times \mathcal{A} \rightarrow \Re$ is the reward specification. Provides expected utility of an action: $\mathcal{R}(s, a) = \mathcal{R}^a_{ss'} = E\{r_{t+1} | s_t = s, a_t = a, s_{t+1} = s'\}$.

MDP Formulation

- Defined by the tuple: $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$.
- $\mathcal{S} = \{s_1, \ldots, s_N\}$ is the set of states.
- $\mathcal{A} = \{a_1, \ldots, a_M\}$ is the set of actions.
- $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S}' \rightarrow [0, 1]$ is the state transition matrix. $\mathcal{T}(s, a, s') = \mathcal{P}^a_{ss'} = p(s_{t+1} = s' | s_t = s, a_t = a)$.
- $\mathcal{R} : \mathcal{S} \times \mathcal{A} \rightarrow \Re$ is the reward specification. Provides expected utility of an action: $\mathcal{R}(s, a) = \mathcal{R}^a_{ss'} = E\{r_{t+1} | s_t = s, a_t = a, s_{t+1} = s'\}$.

MDP Formulation

- Defined by the tuple: $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$.
- $\mathcal{S} = \{s_1, \ldots, s_N\}$ is the set of states.
- $\mathcal{A} = \{a_1, \ldots, a_M\}$ is the set of actions.
- $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S}' \rightarrow [0, 1]$ is the state transition matrix. $\mathcal{T}(s, a, s') = \mathcal{P}^a_{ss'} = p(s_{t+1} = s' | s_t = s, a_t = a)$.
- $\mathcal{R} : \mathcal{S} \times \mathcal{A} \rightarrow \Re$ is the reward specification. Provides expected utility of an action: $\mathcal{R}(s, a) = \mathcal{R}^a_{ss'} = E\{r_{t+1} | s_t = s, a_t = a, s_{t+1} = s'\}$.

MDP Formulation

- Defined by the tuple: $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$.
- $\mathcal{S} = \{s_1, \ldots, s_N\}$ is the set of states.
- $\mathcal{A} = \{a_1, \ldots, a_M\}$ is the set of actions.
- $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S}' \rightarrow [0, 1]$ is the state transition matrix. $\mathcal{T}(s, a, s') = \mathcal{P}^a_{ss'} = p(s_{t+1} = s' | s_t = s, a_t = a)$.
- $\mathcal{R} : \mathcal{S} \times \mathcal{A} \rightarrow \Re$ is the reward specification. Provides expected utility of an action: $\mathcal{R}(s, a) = \mathcal{R}^a_{ss'} = E\{r_{t+1} | s_t = s, a_t = a, s_{t+1} = s'\}$.

Rewards

- Expected cumulative (discounted) reward:

$$\mathcal{R}_t = E\left\{ \sum_{k=0}^{T} \gamma^k \, r_{t+k+1} \right\}; \quad \gamma \in [0, 1] \tag{1}$$

- Discount factor: future rewards are worth less.
- $T = 1$: greedy policy.
- $1 < T < \infty$: finite-horizon policy.
- $T = \infty$: infinite-horizon policy.

Rewards

- Expected cumulative (discounted) reward:

$$\mathcal{R}_t = E\left\{ \sum_{k=0}^{T} \gamma^k \, r_{t+k+1} \right\}; \quad \gamma \in [0, 1] \tag{1}$$

- Discount factor: future rewards are worth less.
- $T = 1$: greedy policy.
- $1 < T < \infty$: finite-horizon policy.
- $T = \infty$: infinite-horizon policy.

Value Function and Policy

- Policy maps states to actions:

$$\pi : s \rightarrow a; \quad s \in \mathcal{S}, a \in \mathcal{A}(s) \tag{2}$$

- $\pi(s, a)$ is probability of taking action a in state s.
- Value function is the value of state s under policy π:

$$\mathcal{V}^{\pi}(s) = E_{\pi}\{\mathcal{R}_t | s_t = s\} = E_{\pi}\left\{ \sum_{k=0}^{\infty} \gamma^k \, r_{t+k+1} | s_t = s \right\} \tag{3}$$

- Action-value function is the value of action a in state s:

$$\mathcal{Q}^{\pi}(s, a) = E_{\pi}\{\mathcal{R}_t | s_t = s, a_t = a\} \tag{4}$$

$$= E_{\pi}\left\{ \sum_{k=0}^{\infty} \gamma^k \, r_{t+k+1} | s_t = s, a_t = a \right\}$$

Value Function and Policy

- Policy maps states to actions:

$$\pi : s \rightarrow a; \quad s \in \mathcal{S}, a \in \mathcal{A}(s) \tag{2}$$

- $\pi(s, a)$ is probability of taking action a in state s.
- Value function is the value of state s under policy π:

$$\mathcal{V}^{\pi}(s) = E_{\pi}\{\mathcal{R}_t | s_t = s\} = E_{\pi}\left\{ \sum_{k=0}^{\infty} \gamma^k \, r_{t+k+1} | s_t = s \right\} \tag{3}$$

- Action-value function is the value of action a in state s:

$$\mathcal{Q}^{\pi}(s, a) = E_{\pi}\{\mathcal{R}_t | s_t = s, a_t = a\} \tag{4}$$

$$= E_{\pi}\left\{ \sum_{k=0}^{\infty} \gamma^k \, r_{t+k+1} | s_t = s, a_t = a \right\}$$

Value Function and Policy

- Policy maps states to actions:

$$\pi : s \rightarrow a; \quad s \in \mathcal{S}, a \in \mathcal{A}(s) \tag{2}$$

- $\pi(s, a)$ is probability of taking action a in state s.
- Value function is the value of state s under policy π:

$$\mathcal{V}^{\pi}(s) = E_{\pi}\{\mathcal{R}_t | s_t = s\} = E_{\pi}\left\{ \sum_{k=0}^{\infty} \gamma^k \, r_{t+k+1} | s_t = s \right\} \tag{3}$$

- Action-value function is the value of action a in state s:

$$\mathcal{Q}^{\pi}(s, a) = E_{\pi}\{\mathcal{R}_t | s_t = s, a_t = a\} \tag{4}$$

$$= E_{\pi}\left\{ \sum_{k=0}^{\infty} \gamma^k \, r_{t+k+1} | s_t = s, a_t = a \right\}$$

Value Function and Policy

- Policy maps states to actions:

$$\pi : s \rightarrow a; \quad s \in \mathcal{S}, a \in \mathcal{A}(s) \tag{2}$$

- $\pi(s, a)$ is probability of taking action a in state s.
- Value function is the value of state s under policy π:

$$\mathcal{V}^{\pi}(s) = E_{\pi}\{\mathcal{R}_t | s_t = s\} = E_{\pi}\left\{ \sum_{k=0}^{\infty} \gamma^k \, r_{t+k+1} | s_t = s \right\} \tag{3}$$

- Action-value function is the value of action a in state s:

$$\mathcal{Q}^{\pi}(s, a) = E_{\pi}\{\mathcal{R}_t | s_t = s, a_t = a\} \tag{4}$$
$$= E_{\pi}\left\{ \sum_{k=0}^{\infty} \gamma^k \, r_{t+k+1} | s_t = s, a_t = a \right\}$$

Greedy Policy

- Greedy policy: take action that maximizes reward in the current step.

$$\pi_1(s) = \underset{a}{\text{argmax}}\, \mathcal{R}(s, a) \tag{5}$$

- Value function for greedy policy:

$$\mathcal{V}^{\pi_1}(s) = \max_a \mathcal{R}(s, a) \tag{6}$$

Greedy Policy

- Greedy policy: take action that maximizes reward in the current step.

$$\pi_1(s) = \underset{a}{\text{argmax}}\, \mathcal{R}(s, a) \tag{5}$$

- Value function for greedy policy:

$$\mathcal{V}^{\pi_1}(s) = \max_{a} \mathcal{R}(s, a) \tag{6}$$

Recursive Estimation and Optimal Policy

- Bellman equation for $\mathcal{V}^{\pi}$:

$$\begin{aligned} \mathcal{V}^{\pi}(s) &= E_{\pi}\{\mathcal{R}_t | s_t = s\} \\ &= \sum_{a} \pi(s,a) \sum_{s'} \mathcal{P}^{a}_{ss'} \left[\mathcal{R}^{a}_{ss'} + \gamma \mathcal{V}^{\pi}(s')\right] \end{aligned} \tag{7}$$

- Optimal state-value functions:

$$\mathcal{V}^{*}(s) = \max_{\pi} \mathcal{V}^{\pi}(s) \tag{8}$$

$$\mathcal{Q}^{*}(s,a) = \max_{\pi} \mathcal{Q}^{\pi}(s,a) = E\left\{r_{t+1} + \gamma \mathcal{V}^{*}(s_{t+1}) | s_t = s, a_t = a\right\}$$

Recursive Estimation and Optimal Policy

- Bellman equation for $\mathcal{V}^{\pi}$:

$$\begin{aligned} \mathcal{V}^{\pi}(s) &= E_{\pi}\{\mathcal{R}_t | s_t = s\} \\ &= \sum_{a} \pi(s,a) \sum_{s'} \mathcal{P}^{a}_{ss'} \left[\mathcal{R}^{a}_{ss'} + \gamma \mathcal{V}^{\pi}(s')\right] \end{aligned} \tag{7}$$

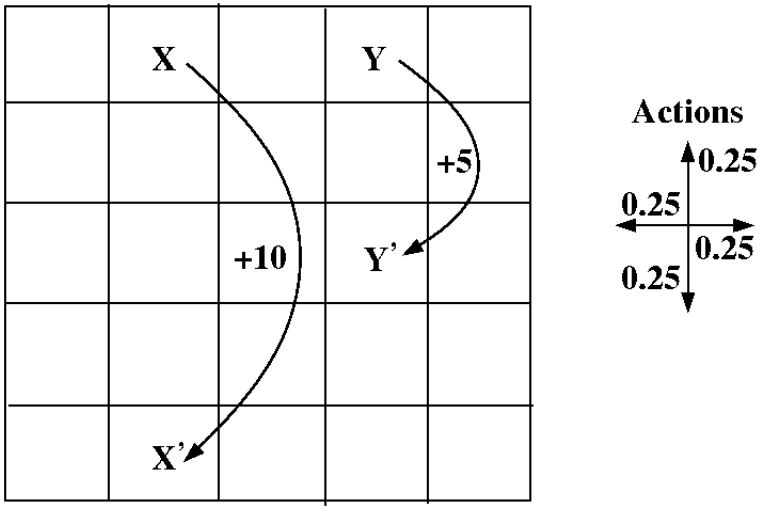
- Optimal state-value functions:

$$\mathcal{V}^{*}(s) = \max_{\pi} \mathcal{V}^{\pi}(s) \tag{8}$$

$$\mathcal{Q}^{*}(s,a) = \max_{\pi} \mathcal{Q}^{\pi}(s,a) = E\left\{r_{t+1} + \gamma \mathcal{V}^{*}(s_{t+1}) | s_t = s, a_t = a\right\}$$

Gridworld Example – Model

- $\mathcal{S} = \{s_i : i \in [1, 25]\}$ and $\mathcal{A} = \{\text{up}, \text{down}, \text{left}, \text{right}\}$.
- All actions from X lead to X' with $r = 10$; all actions from Y lead to Y' with $r = 5$; other actions have $r = -1$.
- Equi-probable policy: $\forall a \in \mathcal{A},\ \pi(*, a) = 0.25$.



Gridworld Example – Values

- Assume $\mathcal{V}_0(s) = 0 \ \forall s \in \mathcal{S}$. Recursive computation.

$$\mathcal{V}(s_1) = \sum_a \pi(s_1, a) \sum_{s'} \mathcal{P}^a_{s_1 s'} \left[\mathcal{R}^a_{s_1 s'} + \gamma \mathcal{V}^\pi(s') \right] \quad (9)$$

$$= \sum_{a=1}^{4} \frac{1}{4} \{1 \cdot (-1 + 0)\} = -1$$

3.3	8.8	4.4	5.3	1.5
1.5	3.0	2.3	1.9	0.5
0.1	0.7	0.7	0.4	−0.4
−1.0	−0.4	−0.4	−0.6	−1.2
−1.9	−1.3	−1.2	−1.4	−2.0

Optimality Equations

- Bellman optimality equation for $\mathcal{V}^*$:

$$\mathcal{V}^*(s) = \max_{a \in \mathcal{A}(s)} \mathcal{Q}^{\pi^*}(s,a) \tag{10}$$
$$= \max_{a \in \mathcal{A}(s)} \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma \mathcal{V}^*(s') \right]$$

- Bellman optimality equation for $\mathcal{Q}^*$:

$$\mathcal{Q}^*(s,a) = \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma \max_{a'} \mathcal{Q}^*(s',a') \right] \tag{11}$$

Optimality Equations

- Bellman optimality equation for $\mathcal{V}^*$:

$$\mathcal{V}^*(s) = \max_{a \in \mathcal{A}(s)} \mathcal{Q}^{\pi^*}(s,a) \tag{10}$$

$$= \max_{a \in \mathcal{A}(s)} \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma \mathcal{V}^*(s') \right]$$

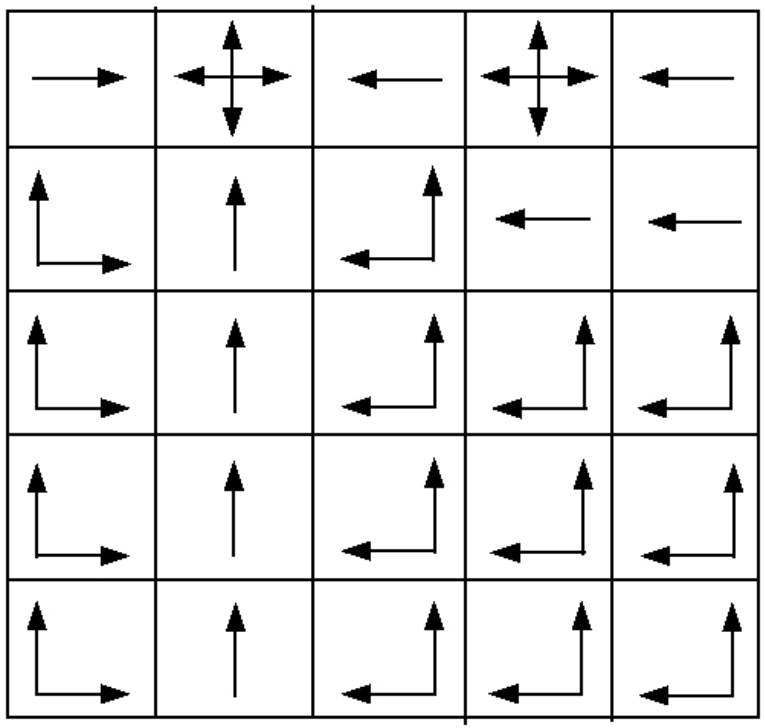
- Bellman optimality equation for $\mathcal{Q}^*$:

$$\mathcal{Q}^*(s,a) = \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma \max_{a'} \mathcal{Q}^*(s',a') \right] \tag{11}$$

Gridworld Example Again

- Optimal value function $\mathcal{V}^*$ and optimal policy π^*.

22.0	24.4	22.0	19.4	17.5
19.8	22.0	19.8	17.8	16.0
17.8	19.8	17.8	16.0	14.4
16.0	17.8	16.0	14.4	13.0
14.4	16.0	14.4	13.0	11.7



Reinforcement Learning

- RL problem: learning from interaction to achieve a goal.

- The agent is the learner and decision-maker.
- The agent interacts with the environment through actions and receives numerical rewards.

- A task is a complete specification of an environment.
- A method suitable to solve a RL problem is a RL method.
- A RL task that satisfies the Markov property is a MDP.

Reinforcement Learning

- RL problem: learning from interaction to achieve a goal.
- The agent is the learner and decision-maker.
- The agent interacts with the environment through actions and receives numerical rewards.
- A task is a complete specification of an environment.
- A method suitable to solve a RL problem is a RL method.
- A RL task that satisfies the Markov property is a MDP.

Reinforcement Learning

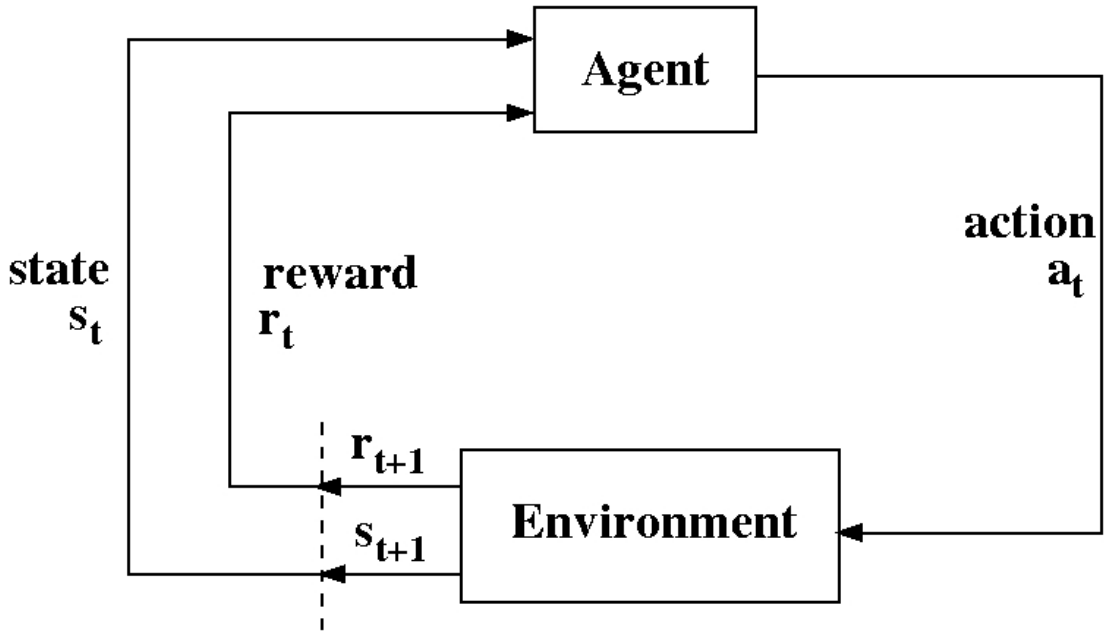
- RL problem: learning from interaction to achieve a goal.

- The agent is the learner and decision-maker.
- The agent interacts with the environment through actions and receives numerical rewards.

- A task is a complete specification of an environment.
- A method suitable to solve a RL problem is a RL method.
- A RL task that satisfies the Markov property is a MDP.

Pictorial Representation

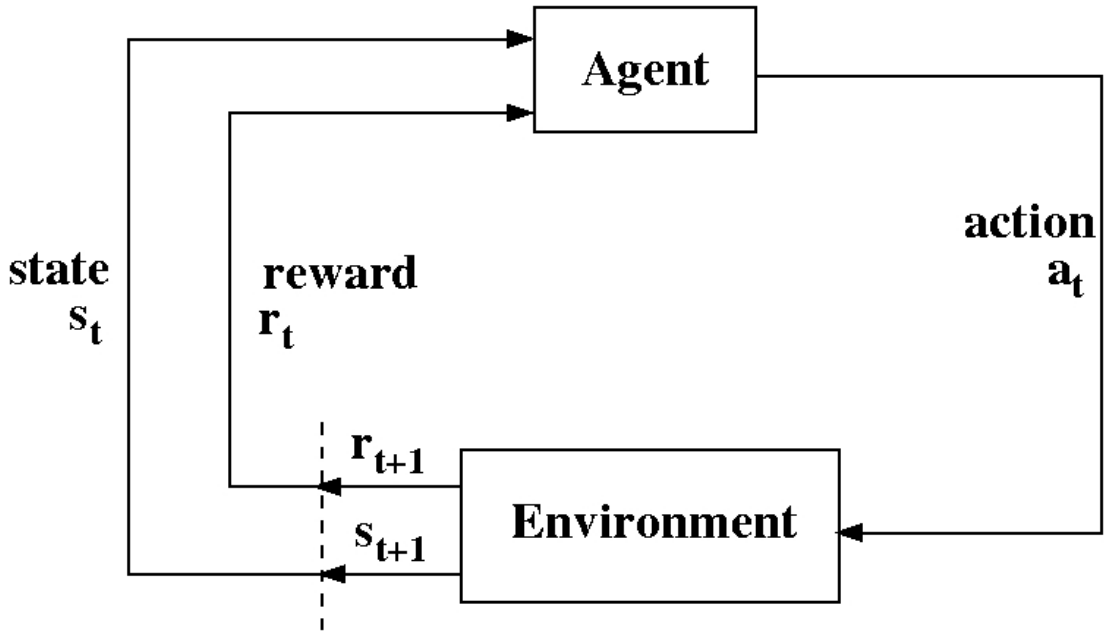
- Agent-environment interaction:



- Given state $s_t \in \mathcal{S}$ at time-step t, agent chooses action $a_t \in \mathcal{A}(s_t)$.
- The agent receives reward r_{t+1} at time $t+1$ and the resulting state is s_{t+1}.

Pictorial Representation

- Agent-environment interaction:



- Given state $s_t \in \mathcal{S}$ at time-step t, agent chooses action $a_t \in \mathcal{A}(s_t)$.
- The agent receives reward r_{t+1} at time $t+1$ and the resulting state is s_{t+1}.

Solution Methods

- Simple solution methods:
  - Dynamic programming.
  - Monte-Carlo methods.
  - Temporal-difference learning.
- Extensions:
  - Eligibility traces.
  - Function approximation.

Policy Iteration and Value Iteration

- Policy iteration:

$$\pi_0 \xrightarrow{E} \mathcal{V}^{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} \mathcal{V}^{\pi_1} \xrightarrow{I} \pi_2 \xrightarrow{E} \ldots \xrightarrow{I} \pi^* \xrightarrow{E} \mathcal{V}^*$$

- Each iteration of PI involves iterative policy evaluation: computationally expensive!
- Value iteration:

$$\mathcal{V}_{k+1}(s) = \max_a E\{r_{t+1} + \gamma \mathcal{V}_k(s_{t+1}) | s_t = s, a_t = a\} \quad (12)$$
$$= \max_a \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma \mathcal{V}_k(s')\right]$$

- Each sweep of VI combines a sweep each of policy evaluation and policy improvement: faster convergence.

Policy Iteration and Value Iteration

- Policy iteration:

$$\pi_0 \xrightarrow{E} \mathcal{V}^{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} \mathcal{V}^{\pi_1} \xrightarrow{I} \pi_2 \xrightarrow{E} \ldots \xrightarrow{I} \pi^* \xrightarrow{E} \mathcal{V}^*$$

- Each iteration of PI involves iterative policy evaluation: computationally expensive!
- Value iteration:

$$\mathcal{V}_{k+1}(s) = \max_a E\{r_{t+1} + \gamma \mathcal{V}_k(s_{t+1}) | s_t = s, a_t = a\} \quad (12)$$
$$= \max_a \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma \mathcal{V}_k(s')\right]$$

- Each sweep of VI combines a sweep each of policy evaluation and policy improvement: faster convergence.

Policy Iteration and Value Iteration

- Policy iteration:

$$\pi_0 \xrightarrow{E} \mathcal{V}^{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} \mathcal{V}^{\pi_1} \xrightarrow{I} \pi_2 \xrightarrow{E} \ldots \xrightarrow{I} \pi^* \xrightarrow{E} \mathcal{V}^*$$

- Each iteration of PI involves iterative policy evaluation: computationally expensive!
- Value iteration:

$$\mathcal{V}_{k+1}(s) = \max_a E\{r_{t+1} + \gamma \mathcal{V}_k(s_{t+1}) | s_t = s, a_t = a\} \quad (12)$$
$$= \max_a \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma \mathcal{V}_k(s')\right]$$

- Each sweep of VI combines a sweep each of policy evaluation and policy improvement: faster convergence.

Policy Iteration and Value Iteration

- Policy iteration:

$$\pi_0 \xrightarrow{E} \mathcal{V}^{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} \mathcal{V}^{\pi_1} \xrightarrow{I} \pi_2 \xrightarrow{E} \ldots \xrightarrow{I} \pi^* \xrightarrow{E} \mathcal{V}^*$$

- Each iteration of PI involves iterative policy evaluation: computationally expensive!
- Value iteration:

$$\begin{aligned} \mathcal{V}_{k+1}(s) &= \max_a E\{r_{t+1} + \gamma \mathcal{V}_k(s_{t+1}) | s_t = s, a_t = a\} \quad (12) \\ &= \max_a \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma \mathcal{V}_k(s') \right] \end{aligned}$$

- Each sweep of VI combines a sweep each of policy evaluation and policy improvement: faster convergence.

Summary of Solution Methods

- Dynamic programming-based *search*:
  - Require accurate environmental model.
  - State values updated based on estimated values of successor states: *bootstrapping*.

- Monte-Carlo methods:
  - No model required: use *sample* episodes of states, actions and rewards from on-line or simulated interactions.
  - Do not bootstrap.
  - More robust to violations of Markov assumption.

Summary of Solution Methods

- Dynamic programming-based *search*:
  - Require accurate environmental model.
  - State values updated based on estimated values of successor states: *bootstrapping*.
- Monte-Carlo methods:
  - No model required: use *sample* episodes of states, actions and rewards from on-line or simulated interactions.
  - Do not bootstrap.
  - More robust to violations of Markov assumption.

Summary of Solution Methods

- Temporal-Difference learning:
  - Central idea of reinforcement learning.
  - Does not require accurate environmental model.
  - Performs bootstrapping: online and incremental.
  - Example methods: Q-learning, Sarsa, Actor-critic.
  - Eligibility traces: $TD(\lambda)$, $Q(\lambda)$, $Sarsa(\lambda)$.
  - Used in various domains: finance, elections, weather forecasting, power distribution.

Summary of Solution Methods

- Temporal-Difference learning:
  - Central idea of reinforcement learning.
  - Does not require accurate environmental model.
  - Performs bootstrapping: online and incremental.
  - Example methods: Q-learning, Sarsa, Actor-critic.
  - Eligibility traces: $TD(\lambda)$, $Q(\lambda)$, $Sarsa(\lambda)$.
  - Used in various domains: finance, elections, weather forecasting, power distribution.

Summary of Solution Methods

- Temporal-Difference learning:
  - Central idea of reinforcement learning.
  - Does not require accurate environmental model.
  - Performs bootstrapping: online and incremental.
  - Example methods: Q-learning, Sarsa, Actor-critic.
  - Eligibility traces: $TD(\lambda)$, $Q(\lambda)$, $Sarsa(\lambda)$.
  - Used in various domains: finance, elections, weather forecasting, power distribution.

Summary

- Introduced probabilistic sequential decision-making.
- Described MDP formulation and policy estimation.
- Introduced reinforcement learning problem domain.
- Summarized popular solution techniques for RL problems.
- How can MDPs be used to determine optimal action sequences in software engineering applications?

Summary

- Introduced probabilistic sequential decision-making.
- Described MDP formulation and policy estimation.

- Introduced reinforcement learning problem domain.
- Summarized popular solution techniques for RL problems.

- How can MDPs be used to determine optimal action sequences in software engineering applications?

Summary

- Introduced probabilistic sequential decision-making.
- Described MDP formulation and policy estimation.

- Introduced reinforcement learning problem domain.
- Summarized popular solution techniques for RL problems.

- How can MDPs be used to determine optimal action sequences in software engineering applications?

Summary

- Introduced probabilistic sequential decision-making.
- Described MDP formulation and policy estimation.

- Introduced reinforcement learning problem domain.
- Summarized popular solution techniques for RL problems.

- How can MDPs be used to determine optimal action sequences in software engineering applications?

Summary

- Introduced probabilistic sequential decision-making.
- Described MDP formulation and policy estimation.

- Introduced reinforcement learning problem domain.
- Summarized popular solution techniques for RL problems.

- How can MDPs be used to determine optimal action sequences in software engineering applications?

Session 4: POMDP

- **9.00–10.30**:
  - Introduction.
  - Statistical analysis; hypothesis testing.
  - Basic probability, Bayes' rule.
- **11.00–12.30**:
  - Bayesian classification.
  - Bayesian regression.
  - Bayesian inference.
- **14.00–15.30**:
  - Information theory.
  - Stochastic sampling.
- **16.00–17.30**:
  - Markov decision processes.
  - Partially observable Markov decision processes.
  - Discussion.

Why POMDP?

- MDP assumes that the state of the system is known.
- True state of the system typically not directly observable: partial observability.
- Agent can only perform actions, whose outcomes are used to update the agent's belief of the state of the system.
- POMDPs elegantly encapsulate the partial observability and non-determinism.

Why POMDP?

- MDP assumes that the state of the system is known.
- True state of the system typically not directly observable: partial observability.
- Agent can only perform actions, whose outcomes are used to update the agent's belief of the state of the system.
- POMDPs elegantly encapsulate the partial observability and non-determinism.

Why POMDP?

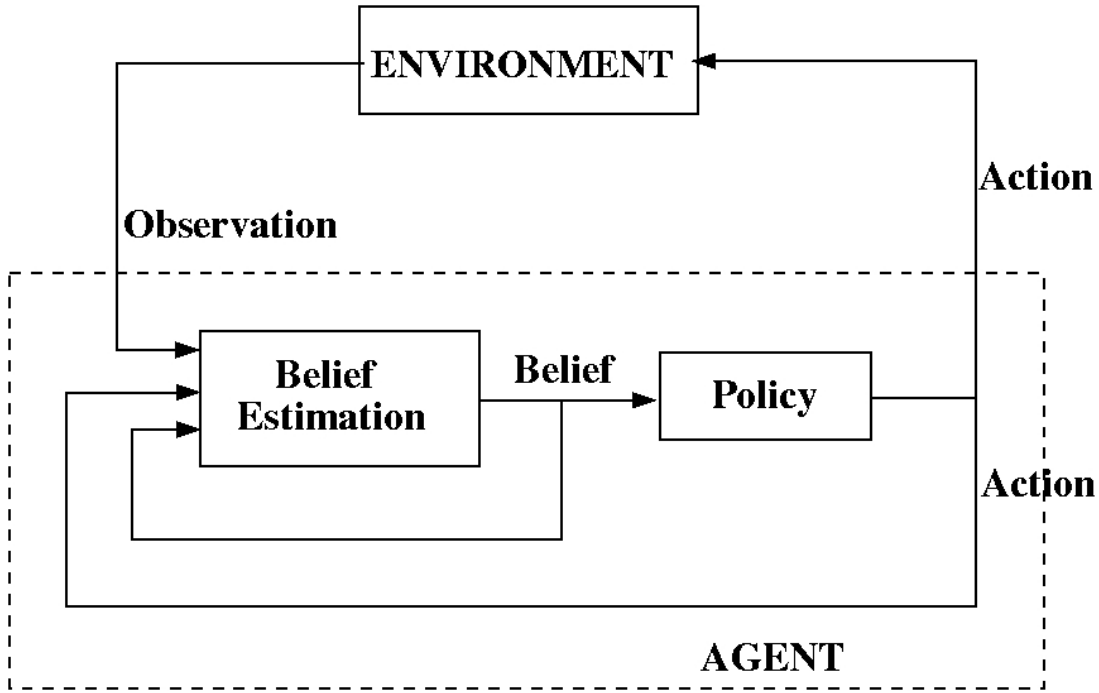
- MDP assumes that the state of the system is known.
- True state of the system typically not directly observable: partial observability.
- Agent can only perform actions, whose outcomes are used to update the agent's belief of the state of the system.
- POMDPs elegantly encapsulate the partial observability and non-determinism.

Why POMDP?

- MDP assumes that the state of the system is known.
- True state of the system typically not directly observable: partial observability.
- Agent can only perform actions, whose outcomes are used to update the agent's belief of the state of the system.
- POMDPs elegantly encapsulate the partial observability and non-determinism.

POMDP Model

- Similar to MDP, but consider belief $b \in \mathcal{B}$ instead of state $s \in \mathcal{S}$:



POMDP Formulation

- Defined by the tuple: $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{Z}, \mathcal{O}, \mathcal{R} \rangle$.
- $\mathcal{S} = \{s_1, \ldots, s_N\}$ is the set of states, which cannot be observed directly.
- $\mathcal{A} = \{a_1, \ldots, a_M\}$ is the set of actions.
- $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S}' \rightarrow [0, 1]$ is the state transition matrix.
- $\mathcal{Z} = \{z_1, \ldots, z_K\}$ is the set of observations.
- $\mathcal{O} : \mathcal{S} \times \mathcal{A} \times \mathcal{Z} \rightarrow [0, 1]$ is the observation function that provides the probability of different observations for each state-action combination.
- $\mathcal{R} : \mathcal{S} \times \mathcal{A} \rightarrow \Re$ is the reward specification.

POMDP Formulation

- Defined by the tuple: $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{Z}, \mathcal{O}, \mathcal{R} \rangle$.

- $\mathcal{S} = \{s_1, \ldots, s_N\}$ is the set of states, which cannot be observed directly.
- $\mathcal{A} = \{a_1, \ldots, a_M\}$ is the set of actions.
- $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S}' \rightarrow [0, 1]$ is the state transition matrix.
- $\mathcal{Z} = \{z_1, \ldots, z_K\}$ is the set of observations.
- $\mathcal{O} : \mathcal{S} \times \mathcal{A} \times \mathcal{Z} \rightarrow [0, 1]$ is the observation function that provides the probability of different observations for each state-action combination.
- $\mathcal{R} : \mathcal{S} \times \mathcal{A} \rightarrow \Re$ is the reward specification.

POMDP Formulation

- Defined by the tuple: $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{Z}, \mathcal{O}, \mathcal{R} \rangle$.

- $\mathcal{S} = \{s_1, \ldots, s_N\}$ is the set of states, which cannot be observed directly.
- $\mathcal{A} = \{a_1, \ldots, a_M\}$ is the set of actions.
- $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S}' \rightarrow [0, 1]$ is the state transition matrix.
- $\mathcal{Z} = \{z_1, \ldots, z_K\}$ is the set of observations.
- $\mathcal{O} : \mathcal{S} \times \mathcal{A} \times \mathcal{Z} \rightarrow [0, 1]$ is the observation function that provides the probability of different observations for each state-action combination.
- $\mathcal{R} : \mathcal{S} \times \mathcal{A} \rightarrow \Re$ is the reward specification.

POMDP Formulation

- Defined by the tuple: $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{Z}, \mathcal{O}, \mathcal{R} \rangle$.

- $\mathcal{S} = \{s_1, \ldots, s_N\}$ is the set of states, which cannot be observed directly.
- $\mathcal{A} = \{a_1, \ldots, a_M\}$ is the set of actions.
- $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S}' \rightarrow [0, 1]$ is the state transition matrix.
- $\mathcal{Z} = \{z_1, \ldots, z_K\}$ is the set of observations.
- $\mathcal{O} : \mathcal{S} \times \mathcal{A} \times \mathcal{Z} \rightarrow [0, 1]$ is the observation function that provides the probability of different observations for each state-action combination.
- $\mathcal{R} : \mathcal{S} \times \mathcal{A} \rightarrow \Re$ is the reward specification.

POMDP Formulation

- Defined by the tuple: $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{Z}, \mathcal{O}, \mathcal{R} \rangle$.

- $\mathcal{S} = \{s_1, \ldots, s_N\}$ is the set of states, which cannot be observed directly.
- $\mathcal{A} = \{a_1, \ldots, a_M\}$ is the set of actions.
- $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S}' \rightarrow [0, 1]$ is the state transition matrix.
- $\mathcal{Z} = \{z_1, \ldots, z_K\}$ is the set of observations.
- $\mathcal{O} : \mathcal{S} \times \mathcal{A} \times \mathcal{Z} \rightarrow [0, 1]$ is the observation function that provides the probability of different observations for each state-action combination.
- $\mathcal{R} : \mathcal{S} \times \mathcal{A} \rightarrow \Re$ is the reward specification.

POMDP Formulation

- Defined by the tuple: $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{Z}, \mathcal{O}, \mathcal{R} \rangle$.

- $\mathcal{S} = \{s_1, \ldots, s_N\}$ is the set of states, which cannot be observed directly.
- $\mathcal{A} = \{a_1, \ldots, a_M\}$ is the set of actions.
- $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S}' \rightarrow [0, 1]$ is the state transition matrix.
- $\mathcal{Z} = \{z_1, \ldots, z_K\}$ is the set of observations.
- $\mathcal{O} : \mathcal{S} \times \mathcal{A} \times \mathcal{Z} \rightarrow [0, 1]$ is the observation function that provides the probability of different observations for each state-action combination.
- $\mathcal{R} : \mathcal{S} \times \mathcal{A} \rightarrow \Re$ is the reward specification.

Belief and Policy

- Each belief is a probability distribution over the underlying states:

$$b_i = \{p(s_1), \ldots, p(s_N)\} \tag{13}$$

- The policy now maps from belief states to actions: $\pi : b \longrightarrow a$.

- Policy is computed by searching over the belief space.
- Plan execution corresponds to policy execution.

Belief and Policy

- Each belief is a probability distribution over the underlying states:

$$b_i = \{p(s_1), \ldots, p(s_N)\} \tag{13}$$

- The policy now maps from belief states to actions: $\pi : b \rightarrow a$.

- Policy is computed by searching over the belief space.
- Plan execution corresponds to policy execution.

Belief and Policy

- Each belief is a probability distribution over the underlying states:

$$b_i = \{p(s_1), \ldots, p(s_N)\} \tag{13}$$

- The policy now maps from belief states to actions: $\pi : b \rightarrow a$.
- Policy is computed by searching over the belief space.
- Plan execution corresponds to policy execution.

Belief and Policy

- Each belief is a probability distribution over the underlying states:

$$b_i = \{p(s_1), \ldots, p(s_N)\} \tag{13}$$

- The policy now maps from belief states to actions: $\pi : b \rightarrow a$.

- Policy is computed by searching over the belief space.
- Plan execution corresponds to policy execution.

Solution Methods

- Optimal solvers:
  - Based on linear programming.
  - Computationally expensive.
- Approximate solvers:
  - Sample from the belief space (MCMC-based).
  - Computationally more tractable.
- Challenge: worst-case complexity exponential in state-space dimensions.
- Solution: use hierarchical planning architecture whose levels match the cognitive requirements of the task.

Solution Methods

- Optimal solvers:
  - Based on linear programming.
  - Computationally expensive.
- Approximate solvers:
  - Sample from the belief space (MCMC-based).
  - Computationally more tractable.
- Challenge: worst-case complexity exponential in state-space dimensions.
- Solution: use hierarchical planning architecture whose levels match the cognitive requirements of the task.

Solution Methods

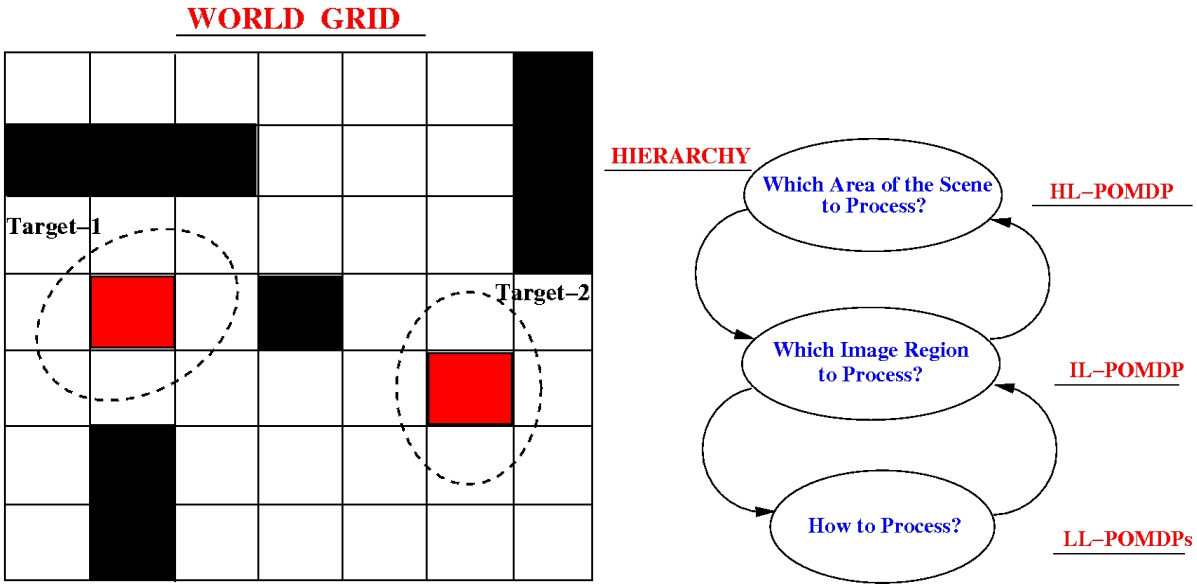
- Optimal solvers:
  - Based on linear programming.
  - Computationally expensive.
- Approximate solvers:
  - Sample from the belief space (MCMC-based).
  - Computationally more tractable.
- Challenge: worst-case complexity exponential in state-space dimensions.
- Solution: use hierarchical planning architecture whose levels match the cognitive requirements of the task.

Solution Methods

- Optimal solvers:
  - Based on linear programming.
  - Computationally expensive.
- Approximate solvers:
  - Sample from the belief space (MCMC-based).
  - Computationally more tractable.
- Challenge: worst-case complexity exponential in state-space dimensions.
- Solution: use hierarchical planning architecture whose levels match the cognitive requirements of the task.

Visual Search

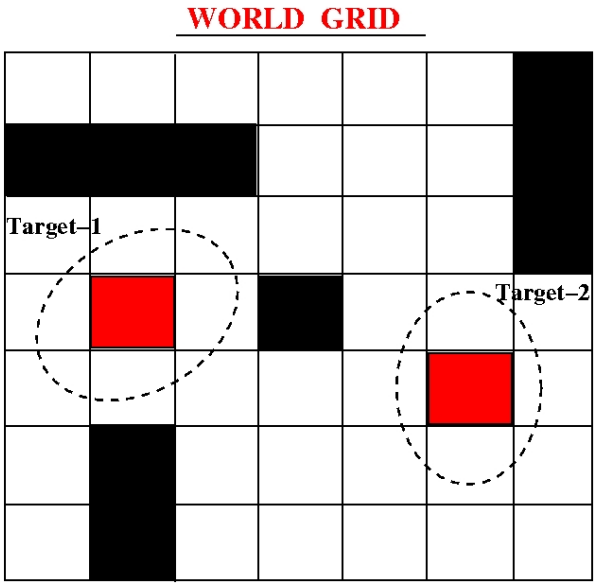
- Goal: enable mobile robot to find target object in an indoor environment.
- Challenges: large dynamic area; too many actions and states.

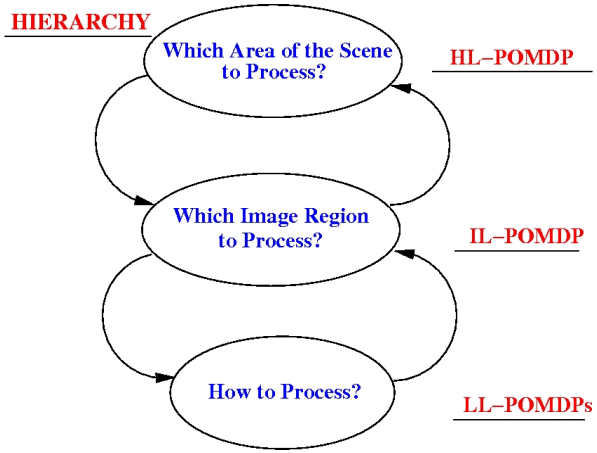


- Three levels: where to look?, what to look for? and how to process data?

Visual Search

- Goal: enable mobile robot to find target object in an indoor environment.
- Challenges: large dynamic area; too many actions and states.

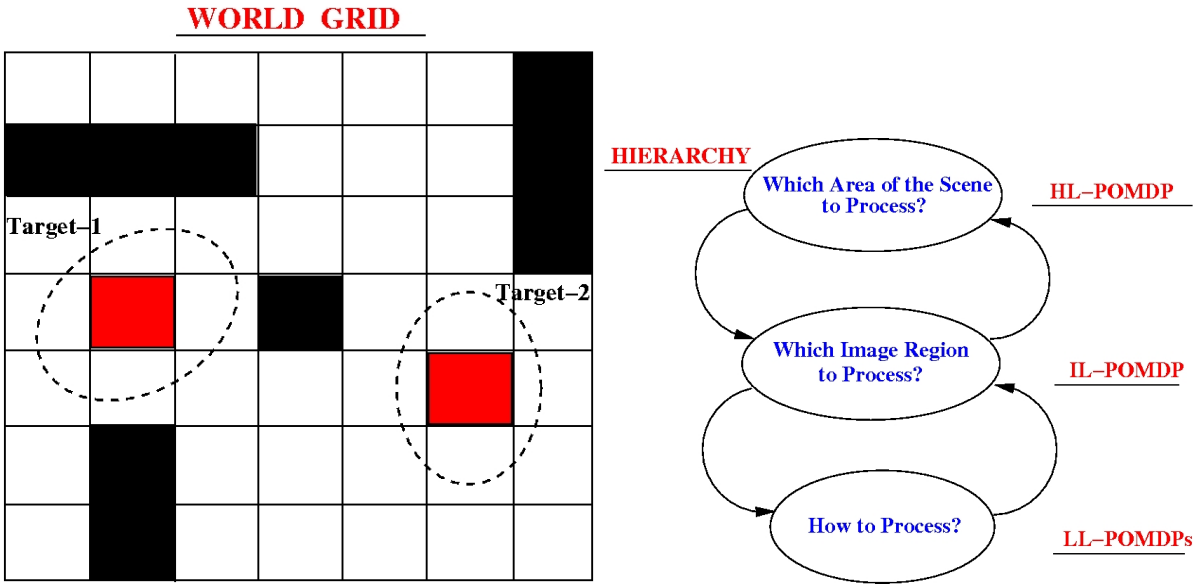




- Three levels: where to look?, what to look for? and how to process data?

Visual Search

- Goal: enable mobile robot to find target object in an indoor environment.
- Challenges: large dynamic area; too many actions and states.



- Three levels: where to look?, what to look for? and how to process data?

Summary

- Introduced the basics of the probabilistic sequential decision making.
- Visual search used as an illustrative example.
- POMDPs suitable for sequential planning in real-world domains.
- How can it be used in software engineering applications?

Summary

- Introduced the basics of the probabilistic sequential decision making.
- Visual search used as an illustrative example.
- POMDPs suitable for sequential planning in real-world domains.
- How can it be used in software engineering applications?

Summary

- Introduced the basics of the probabilistic sequential decision making.
- Visual search used as an illustrative example.
- POMDPs suitable for sequential planning in real-world domains.
- How can it be used in software engineering applications?

Summary

- Introduced the basics of the probabilistic sequential decision making.
- Visual search used as an illustrative example.
- POMDPs suitable for sequential planning in real-world domains.
- How can it be used in software engineering applications?

Further Reading


R. L. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.


L. Kaelbling and M. Littman and A. Cassandra. *Planning and Acting in Partially Observable Stochastic Domains*, Artificial Intelligence, vol. 101, pp. 99-134, 1998.


M. Ghallab and D. Nau and P. Traverso. *Automated Planning: Theory and Practice*. Morgan Kaufmann, 2004.


S. Thrun and W. Burgard and D. Fox. *Probabilistic Robotics*. MIT Press, 2005.

Overall Summary

- Described statistical hypothesis testing, classification and regression.
- Described Bayesian formulation and its application to classification and regression.
- Introduced Bayesian inference and the basic Bayesian filtering approach.
- Described stochastic importance sampling and its application to mutation testing.
- Presented probabilistic sequential decision making techniques: MDPs and POMDPs.

Overall Summary

- Described statistical hypothesis testing, classification and regression.
- Described Bayesian formulation and its application to classification and regression.
- Introduced Bayesian inference and the basic Bayesian filtering approach.
- Described stochastic importance sampling and its application to mutation testing.
- Presented probabilistic sequential decision making techniques: MDPs and POMDPs.

Overall Summary

- Described statistical hypothesis testing, classification and regression.
- Described Bayesian formulation and its application to classification and regression.
- Introduced Bayesian inference and the basic Bayesian filtering approach.
- Described stochastic importance sampling and its application to mutation testing.
- Presented probabilistic sequential decision making techniques: MDPs and POMDPs.

Overall Summary

- Described statistical hypothesis testing, classification and regression.
- Described Bayesian formulation and its application to classification and regression.
- Introduced Bayesian inference and the basic Bayesian filtering approach.
- Described stochastic importance sampling and its application to mutation testing.
- Presented probabilistic sequential decision making techniques: MDPs and POMDPs.

Overall Summary

- Described statistical hypothesis testing, classification and regression.
- Described Bayesian formulation and its application to classification and regression.
- Introduced Bayesian inference and the basic Bayesian filtering approach.
- Described stochastic importance sampling and its application to mutation testing.
- Presented probabilistic sequential decision making techniques: MDPs and POMDPs.