Robust Markov Decision Processes

Prof. Dr. Nils Jansen

Slides by Marnix Suilen and Dr. Thiago Simao

Model Checking 2025

Radboud University Nijmegen

MDPs: The AI View

Markov decision process (MDP)

The **environment** is described by a Markov decision process (MDP) $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$ (Puterman1994).

- $\mathcal{S}$: set of states
- $\mathcal{A}$: set of actions
- $\mathcal{T}: \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$ transition function
- $\mathcal{R}: \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ reward function

The **agent** describes its behavior with a policy $\pi : \mathcal{S} \to \mathcal{A}$.

The agent environment interaction

Agent
policy π

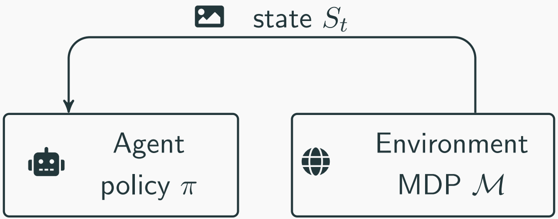
Environment
MDP $\mathcal{M}$

A sequence of interactions between the agent and the environment generates a trajectory (episode).

Trajectory: $S_0, A_0, R_0, S_1, A_1, R_1, \cdots$

where $A_t = \pi(S_t)$, $R_t = \mathcal{R}(S_t, A_t)$, and $S_{t+1} \sim \mathcal{T}(\cdot \mid S_t, A_t)$

The agent environment interaction

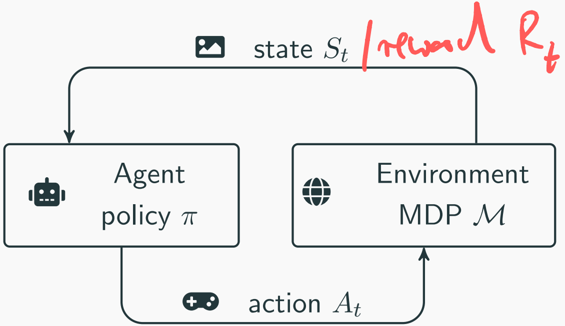


A sequence of interactions between the agent and the environment generates a trajectory (episode).

Trajectory: $S_0, A_0, R_0, S_1, A_1, R_1, \cdots$

where $A_t = \pi(S_t)$, $R_t = \mathcal{R}(S_t, A_t)$, and $S_{t+1} \sim \mathcal{T}(\cdot \mid S_t, A_t)$

The agent environment interaction

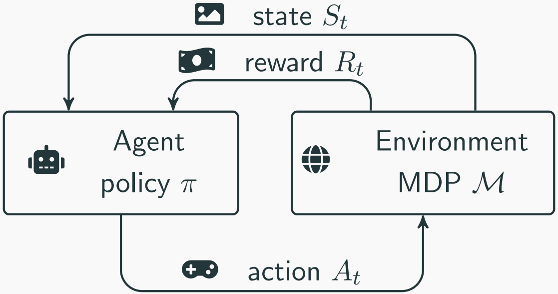


A sequence of interactions between the agent and the environment generates a trajectory (episode).

Trajectory: $S_0, A_0, R_0, S_1, A_1, R_1, \cdots$

where $A_t = \pi(S_t)$, $R_t = \mathcal{R}(S_t, A_t)$, and $S_{t+1} \sim \mathcal{T}(\cdot \mid S_t, A_t)$

The agent environment interaction

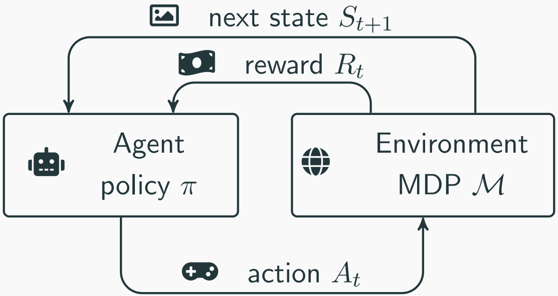


A sequence of interactions between the agent and the environment generates a trajectory (episode).

Trajectory: $S_0, A_0, R_0, S_1, A_1, R_1, \cdots$

where $A_t = \pi(S_t)$, $R_t = \mathcal{R}(S_t, A_t)$, and $S_{t+1} \sim \mathcal{T}(\cdot \mid S_t, A_t)$

The agent environment interaction



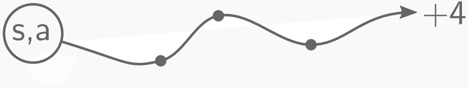
A sequence of interactions between the agent and the environment generates a trajectory (episode).

Trajectory: $S_0, A_0, R_0, S_1, A_1, R_1, \cdots$

where $A_t = \pi(S_t)$, $R_t = \mathcal{R}(S_t, A_t)$, and $S_{t+1} \sim \mathcal{T}(\cdot \mid S_t, A_t)$

Return

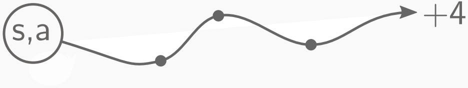
The reward accumulated in a trajectory is called return.



$$\sum_{t=0}^{\infty} R_t$$

Return

The reward accumulated in a trajectory is called return.

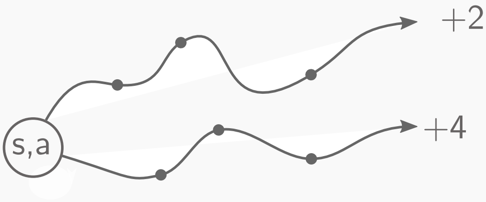


$$\sum_{t=0}^{\infty} \gamma^t R_t$$

The discount factor $\gamma \in [0, 1)$ indicates the present value of future rewards. Intuitively, receiving a reward in the future is worth less than receiving the same reward now.

Return

The reward accumulated in a trajectory is called return.

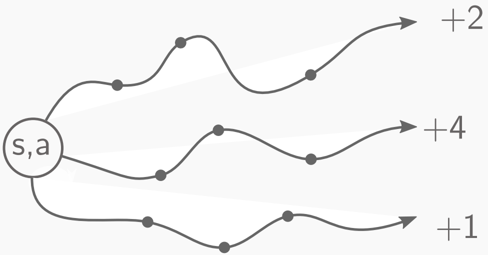


$$\sum_{t=0}^{\infty} \gamma^t R_t$$

The discount factor $\gamma \in [0, 1)$ indicates the present value of future rewards.
Intuitively, receiving a reward in the future is worth less than receiving the same reward now.

Return

The reward accumulated in a trajectory is called return.

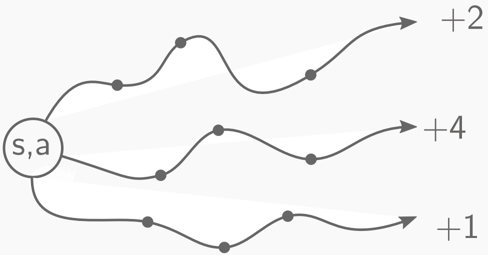


$$\sum_{t=0}^{\infty} \gamma^t R_t$$

The discount factor $\gamma \in [0, 1)$ indicates the present value of future rewards. Intuitively, receiving a reward in the future is worth less than receiving the same reward now.

Return

The reward accumulated in a trajectory is called return.



$$\sum_{t=0}^{\infty} \gamma^t R_t$$

The discount factor $\gamma \in [0, 1)$ indicates the present value of future rewards. Intuitively, receiving a reward in the future is worth less than receiving the same reward now.

The return is a random variable that depends on the policy and the MDP.

Optimality Criterion

Return

$$\sum_{t=0}^{\infty} R_t$$

Discounted Return

$$\sum_{t=0}^{\infty} \gamma^t R_t$$

Optimality Criterion

Expected Discounted Return

$$\mathbb{E}_{\pi,\mathcal{M}}\left[\sum_{t=0}^{\infty}\gamma^t R_t\right]$$

Maximize Expected Discounted Return

$$\arg\max_{\pi} \mathbb{E}_{\pi,\mathcal{M}} \left[\sum_{t=0}^{\infty} \gamma^t R_t \right]$$

Optimality Criterion

Maximize Expected Discounted Return

$$\arg\max_{\pi} \mathbb{E}_{\pi,\mathcal{M}} \left[\sum_{t=0}^{\infty} \gamma^t R_t \right]$$

The Markov decision **problem**

How we solve a Markov decision problem?

Planning

If we have a model of the environment, we can use **planning** to solve the MDP. (Using Value Iteration or Linear Programming)

Policy Iteration

- Computes an optimal policy based on two operations.
- Repeatedly perform
  1. policy evaluation
  2. policy improvement

Policy Evaluation

$V_k^\pi(s)$ indicates the expected value of following the policy π starting on state s for k steps.

$$V_1^\pi(s) = \overbrace{\mathcal{R}(s, \pi(s))}^{\text{immediate reward}} \tag{1}$$

$$\tag{2}$$

Policy Evaluation

$V_k^\pi(s)$ indicates the expected value of following the policy π starting on state s for k steps.

$$V_1^\pi(s) = \overbrace{\mathcal{R}(s, \pi(s))}^{\text{immediate reward}} \tag{1}$$

$$V_{k+1}^\pi(s) = \mathcal{R}(s, \pi(s)) + \gamma \underbrace{\sum_{s' \in \mathcal{S}} \mathcal{T}(s' \mid s, \pi(s)) V_k^\pi(s')}_{\text{expected value of successor state}} \tag{2}$$

Policy Evaluation

$V_k^\pi(s)$ indicates the expected value of following the policy π starting on state s for k steps.

$$V_1^\pi(s) = \overbrace{\mathcal{R}(s, \pi(s))}^{\text{immediate reward}} \tag{1}$$

$$V_{k+1}^\pi(s) = \mathcal{R}(s, \pi(s)) + \gamma \underbrace{\sum_{s' \in \mathcal{S}} \mathcal{T}(s' \mid s, \pi(s)) V_k^\pi(s')}_{\text{expected value of successor state}} \tag{2}$$

At convergence, we have:

$$V^\pi(s) = \mathcal{R}(s, \pi(s)) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{T}(s' \mid s, \pi(s)) V^\pi(s')$$

Robust Markov Decision Processes

Robust MDPs

Robust MDPs extend MDPs by accounting for imprecision or ambiguity in the transition function.

Robust MDPs

Let X be a set of variables. An uncertainty set is a non-empty set of variable assignments subject to some constraints free to choose:

$$\mathcal{U} = \{f \colon X \to \mathbb{R} \mid \text{constraints on } f\}.$$

Definition (Robust MDP)
A robust MDP is a tuple $(S, A, \mathcal{P}, R, \gamma)$ where

- S, A, R and γ are as for standard MDPs,
- $\mathcal{P} \colon \mathcal{U} \to (S \times A \to \mathcal{D}(S))$ is the uncertain transition function.

The word robust

The word **robust** means (according to):

- Cambridge dictionary: (of an object or system) strong and unlikely to break or fail.
- Merriam Webster dictionary: (robust software) capable of performing without failure under a wide range of conditions.
- Oxford Learner's dictionaries: (of a system or an organization) strong and not likely to fail or become weak.

Uncertainty Set

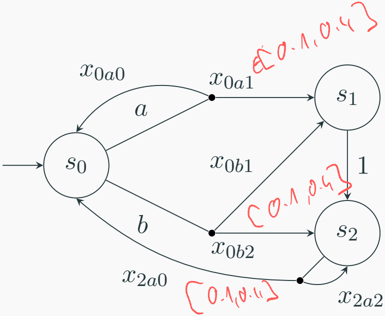
The uncertain transition function $\mathcal{P}$ is a set of standard transition functions $P\colon S \times A \to \mathcal{D}(S)$. We also write $P \in \mathcal{P}$.

Uncertainty Set

The uncertain transition function $\mathcal{P}$ is a set of standard transition functions $P\colon S \times A \to \mathcal{D}(S)$. We also write $P \in \mathcal{P}$.

It is convenient to define the set of variables to have a unique variable for each possible transition of the robust MDP: $X = \{x_{sas'} \mid (s, a, s') \in S \times A \times S\}$.

Example robust MDP with three different uncertainty sets:



$$\mathcal{U}^1 = \{x_{0a1} \in [0.1, 0.9] \wedge x_{0b1} \in [0.1, 0.9] \wedge x_{2a0} \in [0.1, 0.9]\}$$
$$\mathcal{U}^2 = \{x_{0a1} \in [0.1, 0.4] \wedge x_{0b1} = 2x_{0a1} \wedge x_{2a0} \in [0.1, 0.9]\}$$
$$\mathcal{U}^3 = \{x_{0a1} \in [0.1, 0.4] \wedge x_{0b1} = 2x_{0a1} \wedge x_{2a0} = x_{0a1}\}$$

Semantics

agent

Robust MDPs can be viewed as a game between the decision-maker and nature:

- At state s, the decision-maker chooses an action a,
- Nature chooses a transition function $P \in \mathcal{P}$,
- The system moves to state s' with probability $P(s, a)(s')$.

These game semantics are further specified by static and dynamic uncertainty and the rectangularity of the uncertainty set.

Static and Dynamic uncertainty semantics

How nature chooses $P \in \mathcal{P}$ can be done in two different ways:

Static and Dynamic uncertainty semantics

How nature chooses $P \in \mathcal{P}$ can be done in two different ways:

- Static: nature chooses a transition function $P \in \mathcal{P}$ at the start and from then on always uses that P.

Static and Dynamic uncertainty semantics

How nature chooses $P \in \mathcal{P}$ can be done in two different ways:

- Static: nature chooses a transition function $P \in \mathcal{P}$ at the start and from then on always uses that P.
- Dynamic: nature is always free to choose a new $P \in \mathcal{P}$ at every step.

Note that this difference is only relevant in models with cycles, where the same state (and action) can be visited multiple times.

Rectangularity

Rectangularity concerns independence between variables and their constraints in $\mathcal{U}$.

(s, a)-Rectangularity: the variables that occur at (s, a) are unique for that state-action pair and share no constraints with other (s', a').

Rectangularity

Rectangularity concerns independence between variables and their constraints in $\mathcal{U}$.

(s, a)-Rectangularity: the variables that occur at (s, a) are unique for that state-action pair and share no constraints with other (s', a').

The uncertainty set factorizes over state-action pairs: $\mathcal{U} = \bigotimes_{s,a} \mathcal{U}_{s,a}$.

Instead of choosing transition functions $P \in \mathcal{P}$, nature may equivalently choose individual probability distributions $P(s, a) \in \mathcal{P}(s, a)$.

Rectangularity

Rectangularity concerns independence between variables and their constraints in $\mathcal{U}$.

(s, a)-Rectangularity: the variables that occur at (s, a) are unique for that state-action pair and share no constraints with other (s', a').

The uncertainty set factorizes over state-action pairs: $\mathcal{U} = \bigotimes_{s,a} \mathcal{U}_{s,a}$.

Instead of choosing transition functions $P \in \mathcal{P}$, nature may equivalently choose individual probability distributions $P(s, a) \in \mathcal{P}(s, a)$.

Other forms of rectangularity are:

- s-rectangularity: Independence between states, but possible dependencies between different actions at a state.
- Non-rectangularity: Possible dependencies between nature's choice across states. Refer to parametric MDPs.

Solving robust MDPs

The decision-maker wants to maximize the expected discounted reward $\mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t\right]$.

How do we know which $P \in \mathcal{P}$ nature chooses? Assume the worst (or best):

Solving robust MDPs

The decision-maker wants to maximize the expected discounted reward $\mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t\right]$.

How do we know which $P \in \mathcal{P}$ nature chooses? Assume the worst (or best):

- Worst-case: pessimistic; nature 'works against' the decision-maker.
  - Objective: $\max_\pi \min_P \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t\right]$.
  - The resulting expected reward and policy are robust: when we use this policy in practice, the result can only be better than the worst-case.

Solving robust MDPs

The decision-maker wants to maximize the expected discounted reward $\mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t\right]$.

How do we know which $P \in \mathcal{P}$ nature chooses? Assume the worst (or best):

- Worst-case: pessimistic; nature 'works against' the decision-maker.
  - Objective: $\max_\pi \min_P \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t\right]$.
  - The resulting expected reward and policy are robust: when we use this policy in practice, the result can only be better than the worst-case.
- Best-case: optimistic; nature 'helps' in maximizing the reward.
  - Objective: $\max_\pi \max_P \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t\right]$.
  - Resulting expected reward and policy are optimistic: in practice, the result can only be worse than the best-case.

Solving robust MDPs

The decision-maker wants to maximize the expected discounted reward $\mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t\right]$.

How do we know which $P \in \mathcal{P}$ nature chooses? Assume the worst (or best):

- Worst-case: pessimistic; nature 'works against' the decision-maker.
  - Objective: $\max_\pi \min_P \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t\right]$.
  - The resulting expected reward and policy are robust: when we use this policy in practice, the result can only be better than the worst-case.
- Best-case: optimistic; nature 'helps' in maximizing the reward.
  - Objective: $\max_\pi \max_P \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t\right]$.
  - Resulting expected reward and policy are optimistic: in practice, the result can only be worse than the best-case.

Game perspective: adversarial versus cooperative!

Rectangularity makes things easier

For MDPs, memoryless deterministic policies are optimal for discounted reward.

Rectangularity makes things easier

For MDPs, memoryless deterministic policies are optimal for discounted reward.

For robust MDPs, Wiesemann (2013) shows:

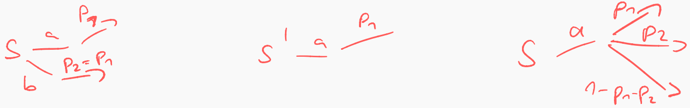
Uncertainty set & rectangularity		Optimal policy class	Policy evaluation
	(s,a)-rectangular	memoryless, deterministic	Polynomial
Convex	s-rectangular	memoryless, randomized	Polynomial
	non-rectangular	memory, randomized	NP-hard

Rectangularity makes things easier

For MDPs, memoryless deterministic policies are optimal for discounted reward.

For robust MDPs, Wiesemann (2013) shows:

Uncertainty set & rectangularity		Optimal policy class	Policy evaluation
	(s, a)-rectangular	memoryless, deterministic	Polynomial
Convex	s-rectangular	memoryless, randomized	Polynomial
	non-rectangular	memory, randomized	NP-hard
	(s, a)-rectangular	memoryless, deterministic	NP-hard
Nonconvex	s-rectangular	memory, randomized	NP-hard
	non-rectangular	memory, randomized	NP-hard



(s, a)-Rectangularity makes things even easier

What about the difference between static and dynamic uncertainty?

Iyengar (2005) shows that in (s, a)-rectangular robust MDPs static and dynamic uncertainty semantics coincide.

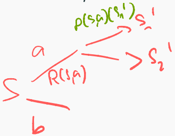
Theorem
Let M be an (s, a)-rectangular robust MDP. Let π_s^* and π_d^* be the optimal memoryless deterministic policies for M under static (s) and dynamic (d) semantics. Then the robust values of these two policies are the same:

$$\min_P \mathbb{E}_{\pi_d^*}\left[\sum_{t=0}^{\infty} \gamma^t r_t\right] = \min_P \mathbb{E}_{\pi_s^*}\left[\sum_{t=0}^{\infty} \gamma^t r_t\right].$$

Robust dynamic programming

Under (s, a)-rectangularity, we can extend value iteration!

Recall, for standard MDPs, we have:



$$V_{n+1}(s) = \max_{a \in A} \left\{ R(s, a) + \gamma \sum_{s' \in S} P(s, a)(s') V_n(s') \right\}.$$

Robust dynamic programming

Under (s, a)-rectangularity, we can extend value iteration!

Recall, for standard MDPs, we have:

$$V_{n+1}(s) = \max_{a \in A} \left\{ R(s,a) + \gamma \sum_{s' \in S} P(s,a)(s') V_n(s') \right\}.$$

Now we need to place the worst-case P in the equation above:

$$V_{n+1}(s) = \overbrace{\max_{a \in A} \Bigg\{ R(s,a) + \gamma \underbrace{\inf_{P(s,a) \in \mathcal{P}(s,a)}}_{\text{inner problem}} \Bigg\{ \overbrace{\sum_{s' \in S} P(s,a)(s') V_n(s')}^{\text{update!}} \Bigg\}}^{\text{outer problem}} \Bigg\}.$$

Note that we use (s, a)-rectangularity.

Finding the worst-case

How do we find $\inf\limits_{P(s,a)\in\mathcal{P}(s,a)} \left\{\sum_{s'\in S} P(s,a)(s')V_n(s')\right\}$? Convexity!

Finding the worst-case

How do we find $\inf_{P(s,a) \in \mathcal{P}(s,a)} \left\{ \sum_{s' \in S} P(s,a)(s') V_n(s') \right\}$? Convexity!

When $\mathcal{P}(s,a)$ is convex, this inner problem is a convex optimization problem.

Can be solved in polynomial time via the interior point method.

Finding the worst-case

How do we find $\inf_{P(s,a)\in\mathcal{P}(s,a)} \left\{\sum_{s'\in S} P(s,a)(s')V_n(s')\right\}$? Convexity!

When $\mathcal{P}(s,a)$ is convex, this inner problem is a convex optimization problem.

Can be solved in polynomial time via the interior point method.

Resulting value and policy will be robust against any choice of nature.

The optimal robust policy is still found by storing the maximizing action at each state.

Finding the best-case

What about the best-case? Same idea:

$$V_{n+1}(s) = \max_{a \in A} \left\{ R(s,a) + \gamma \sup_{P(s,a) \in \mathcal{P}(s,a)} \left\{ \sum_{s' \in S} P(s,a)(s') V_n(s') \right\} \right\}$$

Where $\sup_{P(s,a) \in \mathcal{P}(s,a)} \left\{ \sum_{s' \in S} P(s,a)(s') V_n(s') \right\}$ is again a convex optimization problem.

Resulting value and policy will be optimistic towards nature's choice.

Optimism in the face of uncertainty!

Special sub-classes of robust MDPs

There are two special sub-classes of robust MDPs that are interesting because they are easy to learn from data and their inner problem can be solved efficiently.

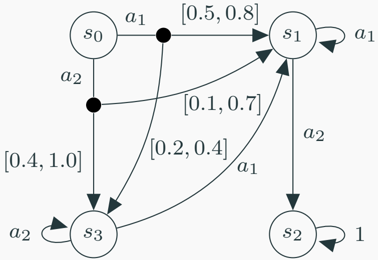
- Interval MDPs (IMDPs): each transition has a probability interval,
- L_1 MDPs: each state-action pair has an uncertainty set around an empirical distribution.

Interval MDPs & Robust Learning

Interval MDPs

Definition (IMDP)
An interval MDP (IMDP) is a tuple $(S, A, \underline{P}, \overline{P}, R, \gamma)$ where

- S, A, R and γ are as for (robust) MDPs,



Interval MDPs

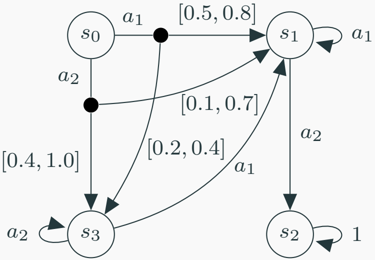
$S \xrightarrow{a}$ [0.1, ...]

[0.5, ...]

[0.2, ...]

Definition (IMDP)
An interval MDP (IMDP) is a tuple $(S, A, \underline{P}, \overline{P}, R, \gamma)$ where

- S, A, R and γ are as for (robust) MDPs,
- $\underline{P}$ assigns a lower bound to each transition:
  $\underline{P}\colon S \times A \times S \to [0,1]$ with $\sum_{s'} \underline{P}(s,a,s') \leq 1$,



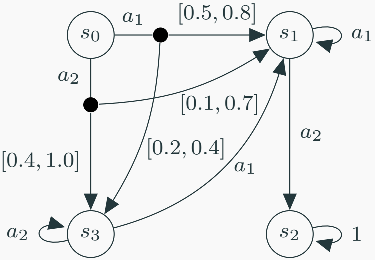
Interval MDPs

$[0.1, 0.2]$

$[0.2, 0.3]$

Definition (IMDP)
An interval MDP (IMDP) is a tuple $(S, A, \underline{P}, \overline{P}, R, \gamma)$ where

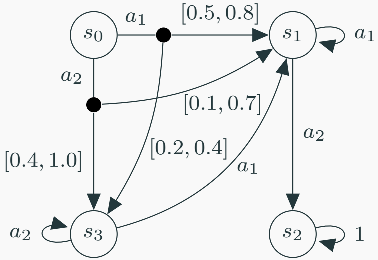
- S, A, R and γ are as for (robust) MDPs,
- $\underline{P}$ assigns a lower bound to each transition: $\underline{P} \colon S \times A \times S \to [0, 1]$ with $\sum_{s'} \underline{P}(s, a, s') \leq 1$,
- $\overline{P}$ assigns an upper bound to each transition: $\overline{P} \colon S \times A \times S \to [0, 1]$ with $\sum_{s'} \overline{P}(s, a, s') \geq 1$,



Interval MDPs

Definition (IMDP)
An interval MDP (IMDP) is a tuple $(S, A, \underline{P}, \overline{P}, R, \gamma)$ where

- S, A, R and γ are as for (robust) MDPs,
- $\underline{P}$ assigns a lower bound to each transition: $\underline{P}\colon S \times A \times S \to [0,1]$ with $\sum_{s'} \underline{P}(s,a,s') \leq 1$,
- $\overline{P}$ assigns an upper bound to each transition: $\overline{P}\colon S \times A \times S \to [0,1]$ with $\sum_{s'} \overline{P}(s,a,s') \geq 1$,
- Each transition is assigned a valid interval: $\forall (s,a,s').\ 0 \leq \underline{P}(s,a,s') \leq \overline{P}(s,a,s') \leq 1.$

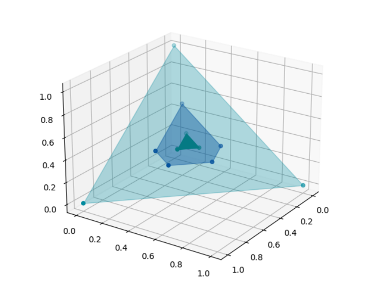


The uncertainty set of IMDPs

An IMDP is an (s, a)-rectangular robust MDP with uncertain transition function $\mathcal{P}$ defined as the set of valid probability distributions in the intervals:

$$\mathcal{P}(s, a) = \left\{ P \in \mathcal{D}(S) \mid \forall s'. P(s') \in \left[\underline{P}(s, a)(s'), \overline{P}(s, a)(s') \right] \right\}.$$

This set is a convex polytope.



Robust value iteration on IMDPs

A convex polytope is bounded subset of $\mathbb{R}^n$ defined by a set of linear inequalities.

Hence, the inner minimization problem can be solved by linear programming in polynomial time.

Yet, more efficient algorithms exist (not part of this lecture).

Solving the inner problem efficiently (IMDPs)

Recall the robust Bellman equation:

$$V_{n+1}(s) = \max_{a \in A} \left\{ R(s,a) + \gamma \inf_{P(s,a) \in \mathcal{P}(s,a)} \left\{ \sum_{s' \in S} P(s,a)(s') V_n(s') \right\} \right\}$$

The minimizing distribution $P(s,a)$ can be found efficiently as follows:

Solving the inner problem efficiently (IMDPs)

Recall the robust Bellman equation:

$$V_{n+1}(s) = \max_{a \in A} \left\{ R(s,a) + \gamma \inf_{P(s,a) \in \mathcal{P}(s,a)} \left\{ \sum_{s' \in S} P(s,a)(s') V_n(s') \right\} \right\}$$

The minimizing distribution $P(s,a)$ can be found efficiently as follows:

- Order the states $s_1, s_2, \ldots, s_m$ according to the current value V_n such that $V_n(s_1) \leq V_n(s_2) \leq \cdots \leq V_n(s_m)$.

Solving the inner problem efficiently (IMDPs)

Recall the robust Bellman equation:

$$V_{n+1}(s) = \max_{a \in A} \left\{ R(s,a) + \gamma \inf_{P(s,a) \in \mathcal{P}(s,a)} \left\{ \sum_{s' \in S} P(s,a)(s') V_n(s') \right\} \right\}$$

The minimizing distribution $P(s,a)$ can be found efficiently as follows:

- Order the states $s_1, s_2, \ldots, s_m$ according to the current value V_n such that $V_n(s_1) \leq V_n(s_2) \leq \cdots \leq V_n(s_m)$.
- Then find index j such that
  - All states indexed $< s_j$ get the upper bound as transition value,
  - All states indexed $> s_j$ get the lower bound as transition value,
  - State s_j gets a value in $[\underline{P}(s_j), \overline{P}(s_j)]$ such that we have a valid distribution.

Solving the inner problem efficiently (IMDPs) - concrete algorithm

1. Order $s_1, \ldots, s_m$ such that $V_n(s_1) \leq \cdots \leq V_n(s_m)$, and let $i = 1$,

Solving the inner problem efficiently (IMDPs) - concrete algorithm

1. Order $s_1, \ldots, s_m$ such that $V_n(s_1) \leq \cdots \leq V_n(s_m)$, and let $i = 1$,
2. $limit = \sum_{s'} \underline{P}(s')$,

Solving the inner problem efficiently (IMDPs) - concrete algorithm

1. Order $s_1, \ldots, s_m$ such that $V_n(s_1) \leq \cdots \leq V_n(s_m)$, and let $i = 1$,
2. $limit = \sum_{s'} \underline{P}(s')$,
3. While $limit - \underline{P}(s_i) + \overline{P}(s_i) < 1$:
   - $limit = limit - \underline{P}(s_i) + \overline{P}(s_i)$,
   - $P(s_i) = \overline{P}(s_i)$,
   - $i = i + 1$,

Solving the inner problem efficiently (IMDPs) - concrete algorithm

1. Order $s_1, \dots, s_m$ such that $V_n(s_1) \leq \dots \leq V_n(s_m)$, and let $i = 1$,
2. $limit = \sum_{s'} \underline{P}(s')$,
3. While $limit - \underline{P}(s_i) + \overline{P}(s_i) < 1$:
   - $limit = limit - \underline{P}(s_i) + \overline{P}(s_i)$,
   - $P(s_i) = \overline{P}(s_i)$,
   - $i = i + 1$,
4. $j = i$,
5. $P(s_j) = 1 - (limit - \underline{P}(s_j))$,

Solving the inner problem efficiently (IMDPs) - concrete algorithm

1. Order $s_1, \ldots, s_m$ such that $V_n(s_1) \leq \cdots \leq V_n(s_m)$, and let $i = 1$,
2. $limit = \sum_{s'} \underline{P}(s')$,
3. While $limit - \underline{P}(s_i) + \overline{P}(s_i) < 1$:
   - $limit = limit - \underline{P}(s_i) + \overline{P}(s_i)$,
   - $P(s_i) = \overline{P}(s_i)$,
   - $i = i + 1$,
4. $j = i$,
5. $P(s_j) = 1 - (limit - \underline{P}(s_j))$,
6. for $k \in \{j + 1, \ldots, m\}$:
   - $P(s_k) = \underline{P}(s_k)$,

Solving the inner problem efficiently (IMDPs) - concrete algorithm

1. Order $s_1, \ldots, s_m$ such that $V_n(s_1) \leq \cdots \leq V_n(s_m)$, and let $i = 1$,
2. $limit = \sum_{s'} \underline{P}(s')$,
3. While $limit - \underline{P}(s_i) + \overline{P}(s_i) < 1$:
    - $limit = limit - \underline{P}(s_i) + \overline{P}(s_i)$,
    - $P(s_i) = \overline{P}(s_i)$,
    - $i = i + 1$,
4. $j = i$,
5. $P(s_j) = 1 - (limit - \underline{P}(s_j))$,
6. for $k \in \{j + 1, \ldots, m\}$:
    - $P(s_k) = \underline{P}(s_k)$,
7. Return P.

Robust learning

We use IMDPs to overcome statistical errors in learning.

Instead of learning point estimates as in frequentist or Bayesian learning, we learn probability intervals.

The resulting model is an IMDP, and a worst-case value and policy will account for those errors.

$s, a \to s_1$

$s, a \to s_1$

$s, a \to s_1$

$s, a \to s_2$

$s \xrightarrow{a} \xrightarrow{3/4} s_1$

$s \xrightarrow{a} \xrightarrow{1/4} s_2$

Robust learning

We use IMDPs to overcome statistical errors in learning.

Instead of learning point estimates as in frequentist or Bayesian learning, we learn probability intervals.

The resulting model is an IMDP, and a worst-case value and policy will account for those errors.

probably approximately correct $|\xleftarrow{\varepsilon} \cdot \xrightarrow{\varepsilon}|$

We consider two ways of learning intervals:

1. PAC learning: gives a formal correctness guarantee on the result,
2. Linearly updating intervals: no formal guarantees, but fast and flexible.

PAC Learning

Probably approximately correct (PAC) learning: formal guarantee on the result.

We construct an IMDP with the following intervals:

PAC Learning

Probably approximately correct (PAC) learning: formal guarantee on the result.

We construct an IMDP with the following intervals:

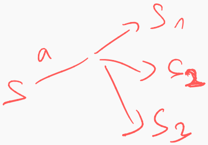
1. Compute point estimates via frequentist or Bayesian learning for every transition (s, a, s'),

PAC Learning

Probably approximately correct (PAC) learning: formal guarantee on the result.

We construct an IMDP with the following intervals:

1. Compute point estimates via frequentist or Bayesian learning for every transition (s, a, s'),
2. Choose an error rate $\epsilon \in (0, 1)$, and compute the error rate for the whole model: $\epsilon_M = \epsilon / \sum_{s,a} |Post(s,a)_{>1}|$, where $|Post_{>1}(s, a)|$ is the number of successor states of (s, a) with probabilities in $(0, 1)$. Then use ϵ_M to compute $\delta_M = \sqrt{\log(2/\epsilon_M)/2N}$.



PAC Learning

Probably approximately correct (PAC) learning: formal guarantee on the result.

We construct an IMDP with the following intervals:

1. Compute point estimates via frequentist or Bayesian learning for every transition (s, a, s'),
2. Choose an error rate $\epsilon \in (0, 1)$, and compute the error rate for the whole model: $\epsilon_M = \epsilon / \sum_{s,a} |Post(s,a)_{>1}|$, where $|Post_{>1}(s, a)|$ is the number of successor states of (s, a) with probabilities in $(0, 1)$. Then use ϵ_M to compute $\delta_M = \sqrt{\log(2/\epsilon_M)/2N}$.
3. For each transition, construct the interval $\tilde{P}(s, a, s') \pm \delta_M$: $\underline{P}(s, a, s') = P(s, a, s') - \delta_M$, $\overline{P}(s, a, s') = P(s, a, s') + \delta_M$.

PAC Learning

Probably approximately correct (PAC) learning: formal guarantee on the result.

We construct an IMDP with the following intervals:

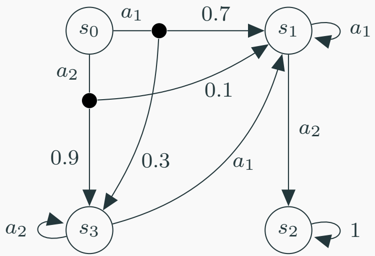
1. Compute point estimates via frequentist or Bayesian learning for every transition (s, a, s'),
2. Choose an error rate $\epsilon \in (0, 1)$, and compute the error rate for the whole model: $\epsilon_M = \epsilon / \sum_{s,a} |Post(s,a)_{>1}|$, where $|Post_{>1}(s, a)|$ is the number of successor states of (s, a) with probabilities in $(0, 1)$. Then use ϵ_M to compute $\delta_M = \sqrt{\log(2/\epsilon_M)/2N}$.
3. For each transition, construct the interval $\tilde{P}(s, a, s') \pm \delta_M$: $\underline{P}(s, a, s') = P(s, a, s') - \delta_M$, $\overline{P}(s, a, s') = P(s, a, s') + \delta_M$.

Then with probability of at least $1 - \epsilon$ the true MDP M is contained in the IMDP $\mathcal{M}$:

$$\Pr(M \in \mathcal{M}) \geq 1 - \epsilon.$$

Example

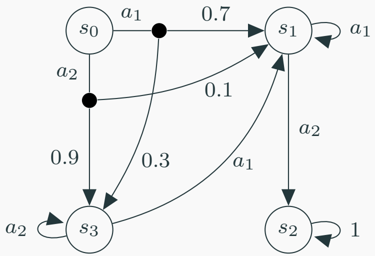
Suppose we want to learn (s_0, a_1) in the MDP:



Suppose we have $N = 20$, $\tilde{P}(s_0, a_1, s_1) = 0.65$, $\tilde{P}(s_0, a_1, s_3) = 0.35$, and set $\epsilon = 0.01$.

Example

Suppose we want to learn (s_0, a_1) in the MDP:

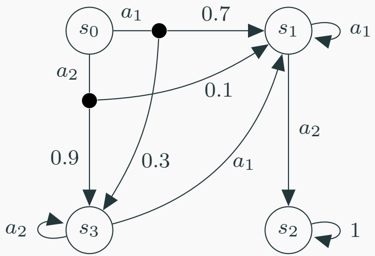


Suppose we have $N = 20$, $\tilde{P}(s_0, a_1, s_1) = 0.65$, $\tilde{P}(s_0, a_1, s_3) = 0.35$, and set $\epsilon = 0.01$.

- $\sum_{s,a} |Post_{>1}(s, a)| = 2 + 2 = 4$,

Example

Suppose we want to learn (s_0, a_1) in the MDP:

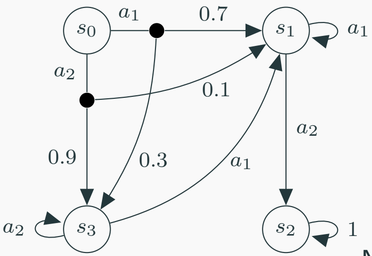


Suppose we have $N = 20$, $\tilde{P}(s_0, a_1, s_1) = 0.65$, $\tilde{P}(s_0, a_1, s_3) = 0.35$, and set $\epsilon = 0.01$.

- $\sum_{s,a} |Post_{>1}(s,a)| = 2 + 2 = 4$,
- $\epsilon_M = 0.0025$, $\delta_M = \sqrt{\frac{\log(2/\epsilon_M)}{2N}} = 0.409$,

Example

Suppose we want to learn (s_0, a_1) in the MDP:



Suppose we have $N = 20$, $\tilde{P}(s_0, a_1, s_1) = 0.65$, $\tilde{P}(s_0, a_1, s_3) = 0.35$, and set $\epsilon = 0.01$.

- $\sum_{s,a} |Post_{>1}(s,a)| = 2 + 2 = 4$,
- $\epsilon_M = 0.0025$, $\delta_M = \sqrt{\frac{\log(2/\epsilon_M)}{2N}} = 0.409$,
- $\underline{P}(s_0, a_1, s_1) = 0.65 - 0.409 = 0.241$,
- $\overline{P}(s_0, a_1, s_1) = 0.65 + 0.409 = 1.059 \equiv 1.0$,
- $\underline{P}(s_0, a_1, s_3) = 0.35 - 0.409 = -0.059 \equiv 0.0$,
- $\overline{P}(s_0, a_1, s_3) = 0.35 + 0.409 = 0.759$.

Note that values are forced into the $[0, 1]$ interval.

Key problems in PAC learning

1. The amount of data required for useful guarantees is enormous,
2. PAC learning assumes the underlying distribution(s) are fixed.

Linearly Updating Intervals

Linearly updating intervals (LUI): no formal guarantees, but fast and flexible when underlying distributions change.

Linearly Updating Intervals

Linearly updating intervals (LUI): no formal guarantees, but fast and flexible when underlying distributions change.

We assume two intervals for each transition:

1. An interval of prior transition probabilities $[\underline{P}(s,a,s'), \overline{P}(s,a,s')]$,
2. A strength interval $[\underline{n}(s,a,s'), \overline{n}(s,a,s')]$.

(1) Serves as prior that will be updated,
(2) Controls how much data we need.

LUI Computation

Assume we want to update transitions $(s, a, s_1), \ldots, (s, a, s_m)$.

LUI Computation

Assume we want to update transitions $(s, a, s_1), \ldots, (s, a, s_m)$.

1. Collect data, and let $N = \#(s, a)$ and $k_i = \#(s, a, s_i)$

$s \xrightarrow{a} s_i$

LUI Computation

Assume we want to update transitions $(s, a, s_1), \ldots, (s, a, s_m)$.

1. Collect data, and let $N = \#(s, a)$ and $k_i = \#(s, a, s_i)$
2. Update lower bound:

$$\underline{P}(s, a, s_i)' = \begin{cases} \frac{\overline{n}(s,a,s_i)\underline{P}(s,a,s_i)+k_i}{\overline{n}(s,a,s_i)+N} & \text{if } \forall j. \frac{k_j}{N} \geq \underline{P}(s, a, s_j) \text{ (prior-data agreement)}, \\ \frac{\underline{n}(s,a,s_i)\underline{P}(s,a,s_i)+k_i}{\underline{n}(s,a,s_i)+N} & \text{if } \exists j. \frac{k_j}{N} < \underline{P}(s, a, s_j) \text{ (prior-data conflict)}. \end{cases}$$

LUI Computation

Assume we want to update transitions $(s, a, s_1), \ldots, (s, a, s_m)$.

1. Collect data, and let $N = \#(s, a)$ and $k_i = \#(s, a, s_i)$
2. Update lower bound:

$$\underline{P}(s, a, s_i)' = \begin{cases} \frac{\overline{n}(s,a,s_i)\underline{P}(s,a,s_i)+k_i}{\overline{n}(s,a,s_i)+N} & \text{if } \forall j. \frac{k_j}{N} \geq \underline{P}(s, a, s_j) \text{ (prior-data agreement),} \\ \frac{\underline{n}(s,a,s_i)\underline{P}(s,a,s_i)+k_i}{\underline{n}(s,a,s_i)+N} & \text{if } \exists j. \frac{k_j}{N} < \underline{P}(s, a, s_j) \text{ (prior-data conflict).} \end{cases}$$

3. Update upper bound:

$$\overline{P}(s, a, s_i)' = \begin{cases} \frac{\overline{n}(s,a,s_i)\overline{P}(s,a,s_i)+k_i}{\overline{n}(s,a,s_i)+N} & \text{if } \forall j. \frac{k_j}{N} \leq \overline{P}(s, a, s_j) \text{ (prior-data agreement),} \\ \frac{\underline{n}(s,a,s_i)\overline{P}(s,a,s_i)+k_i}{\underline{n}(s,a,s_i)+N} & \text{if } \exists j. \frac{k_j}{N} > \overline{P}(s, a, s_j) \text{ (prior-data conflict).} \end{cases}$$

LUI Computation

Assume we want to update transitions $(s, a, s_1), \ldots, (s, a, s_m)$.

1. Collect data, and let $N = \#(s, a)$ and $k_i = \#(s, a, s_i)$
2. Update lower bound:

$$\underline{P}(s, a, s_i)' = \begin{cases} \frac{\overline{n}(s,a,s_i)\underline{P}(s,a,s_i)+k_i}{\overline{n}(s,a,s_i)+N} & \text{if } \forall j. \frac{k_j}{N} \geq \underline{P}(s, a, s_j) \text{ (prior-data agreement)}, \\ \frac{\underline{n}(s,a,s_i)\underline{P}(s,a,s_i)+k_i}{\underline{n}(s,a,s_i)+N} & \text{if } \exists j. \frac{k_j}{N} < \underline{P}(s, a, s_j) \text{ (prior-data conflict)}. \end{cases}$$

3. Update upper bound:

$$\overline{P}(s, a, s_i)' = \begin{cases} \frac{\overline{n}(s,a,s_i)\overline{P}(s,a,s_i)+k_i}{\overline{n}(s,a,s_i)+N} & \text{if } \forall j. \frac{k_j}{N} \leq \overline{P}(s, a, s_j) \text{ (prior-data agreement)}, \\ \frac{\underline{n}(s,a,s_i)\overline{P}(s,a,s_i)+k_i}{\underline{n}(s,a,s_i)+N} & \text{if } \exists j. \frac{k_j}{N} > \overline{P}(s, a, s_j) \text{ (prior-data conflict)}. \end{cases}$$

4. Return updated transitions $[\underline{P}(s, a, \cdot)', \overline{P}(s, a, \cdot)']$ and strengths $[\underline{n}(s, a, \cdot) + N, \overline{n}(s, a, \cdot) + N]$.

Example (single interval)

Prior	strength	estimate	posterior	strength
$[0.0, 1.0]$	$[0, 10]$	$\frac{1}{2}$	$[0.083, 0.917]$	$[2, 12]$
$[0.0, 1.0]$	$[0, 10]$	$\frac{50}{100}$	$[0.45, 0.55]$	$[100, 110]$
$[0.0, 1.0]$	$[0, 1000]$	$\frac{50}{100}$	$[0.045, 0.95]$	$[100, 1100]$
$[0.4, 0.6]$	$[0, 10]$	$\frac{1}{1}$	$[0.45, 1.0]$	$[1, 11]$
$[0.4, 0.6]$	$[10, 100]$	$\frac{1}{1}$	$[0.406, 0.636]$	$[11, 101]$

Robust learning

PAC and LUI learning can be included in an RL-like scheme where we:

1. Collect data,
2. Learn an IMDP,
3. Compute a robust value and policy,
4. Repeat until convergence.

That way, at any time, we have a policy that is robust against the uncertainty from statistical errors and insufficient data.

Summary so far

What to remember:

- Robust MDPs, robust value iteration, especially IMDPs,
- Learning probabilities (frequentist & Bayesian),
- Learning intervals (PAC and LUI),

L_1 MDPs & Reinforcement Learning

L_1 MDPs

The L_1-distance between two distributions is $\| P - Q \|_1 = \sum_s |P(s) - Q(s)|$.

L_1 MDPs

The L_1-distance between two distributions is $\parallel P - Q \parallel_1 = \sum_s |P(s) - Q(s)|$.

Definition (L_1 MDP)
An L_1 MDP is a tuple $(S, A, \tilde{P}, d, R, \gamma)$ where

- S, A, R and γ are as in (robust) MDPs,
- $\tilde{P} \colon S \times A \to \mathcal{D}(S)$ is an estimated transition function,
- $d \colon S \times A \to \mathbb{R}_{\geq 0}$ is a distance bound for each state-action pair.

L_1 MDPs

The L_1-distance between two distributions is $\| P - Q \|_1 = \sum_s |P(s) - Q(s)|$.

stop here

Definition (L_1 MDP)
An L_1 MDP is a tuple $(S, A, \tilde{P}, d, R, \gamma)$ where

Hi Eline

- S, A, R and γ are as in (robust) MDPs,
- $\tilde{P} \colon S \times A \to \mathcal{D}(S)$ is an estimated transition function,
- $d \colon S \times A \to \mathbb{R}_{\geq 0}$ is a distance bound for each state-action pair.

An L_1 MDP is a robust MDP where the uncertainty set $\mathcal{P}$ is the set of all distributions with L_1-distance closer than d to $\tilde{P}$:

$$\mathcal{P}(s,a) = \left\{ P(s,a) \in \mathcal{D}(S) \mid \| P(s,a) - \tilde{P}(s,a) \|_1 \leq d(s,a) \right\}.$$

This is again a convex polytope.

L_1 MDPs - application

L_1 MDPs are commonly used in reinforcement learning algorithms.

One such algorithm is the UCRL2 algorithm (Jaksch, Ortner, and Auer, 2010).

L_1 MDPs - application

L_1 MDPs are commonly used in reinforcement learning algorithms.

One such algorithm is the UCRL2 algorithm (Jaksch, Ortner, and Auer, 2010).

UCRL2 is a model-based, optimistic, algorithm that uses L_1 MDPs as intermediate models to guide exploration: optimism in the face of uncertainty.

We discuss a simplified version that only learns transition probabilities.

UCRL2 - the general idea

Initialize: set confidence parameter $\delta \in (0, 1)$ and time counter $t = 1$.

UCRL2 - the general idea

Initialize: set confidence parameter $\delta \in (0,1)$ and time counter $t = 1$.

1. Build L_1 MDP with

$$\tilde{P}(s,a,s') = \frac{\#(s,a,s')}{\max\{1, \#(s,a)\}}, \quad d(s,a) = \sqrt{\frac{14|S|\log(|A|t_k/\delta)}{\max\{1, \#(s,a)\}}},$$

2. Compute optimistic policy π (next slide),
3. Sample data using π,
4. Repeat.

Solving the optimistic inner problem efficiently (L_1 MDPs)

For UCRL2 we need to compute the optimistic value and policy:

$$V_{n+1}(s) = \max_{a \in A} \left\{ R(s,a) + \gamma \sup_{P(s,a) \in \mathcal{P}(s,a)} \left\{ \sum_{s' \in S} P(s,a)(s') V_n(s') \right\} \right\}$$

Solving the optimistic inner problem efficiently (L_1 MDPs)

For UCRL2 we need to compute the optimistic value and policy:

$$V_{n+1}(s) = \max_{a \in A} \left\{ R(s,a) + \gamma \sup_{P(s,a) \in \mathcal{P}(s,a)} \left\{ \sum_{s' \in S} P(s,a)(s') V_n(s') \right\} \right\}$$

To do so, we have a similar algorithm as for IMDPs:

1. Order $s_1, \ldots, s_m$ such that $V_n(s_1) \geq \cdots \geq V_n(s_m)$,
2. Set $P(s_1) = \min\{1, \tilde{P}(s_1) + d/2\}$ and for $j > 1$: $P(s_j) = \tilde{P}(s_j)$,
3. $l = m$,
4. While $\sum_j P(s_j) > 1$:
   - $P(s_l) = \max\{0, 1 - \sum_{j \neq l} P(s_j)\}$,
   - $l = l - 1$,
5. Return P.

UCRL2 - full algorithm

Set $\delta \in (0,1)$, $t = 1$, $\#(s,a) = 0$, $\#(s,a,s') = 0$,

For episode $k = 1, 2, \ldots,$ do:

UCRL2 - full algorithm

Set $\delta \in (0,1)$, $t = 1$, $\#(s,a) = 0$, $\#(s,a,s') = 0$,

For episode $k = 1, 2, \ldots,$ do:

1. Build L_1 MDP at episode k:
   1.1 $t_k = t$,
   1.2 $\tilde{P}(s,a,s') = \frac{\#(s,a,s')}{\max\{1,\#(s,a)\}}$, $d(s,a) = \sqrt{\frac{14|S|\log(|A|t_k/\delta)}{\max\{1,\#(s,a)\}}}$
   1.3 Compute optimistic policy π_k in L_1 MDP $(S, A, \tilde{P}, d, R, \gamma)$,
2. Sampling:
   2.1 Set local counters $\forall (s,a,s') : v_k(s,a) = 0, v_k(s,a,s') = 0$,
   2.2 While $v_k(s, \pi_k(s)) < \max\{1, \#(s, \pi_k(s))\}$:
      - Execute action $a = \pi_k(s)$, update counter $v_k(s,a) = v_k(s,a) + 1$
      - Observe successor state s', update counter $v_k(s,a,s') = v_k(s,a,s') + 1$,
      - Set s' as the current state: $s = s'$, update $t = t + 1$,

   2.3 End episode k, update global counters $\#(s,a) {+}{=} v_k(s,a)$, $\#(s,a,s') {+}{=} v_k(s,a,s')$

Comparison of different learning methods

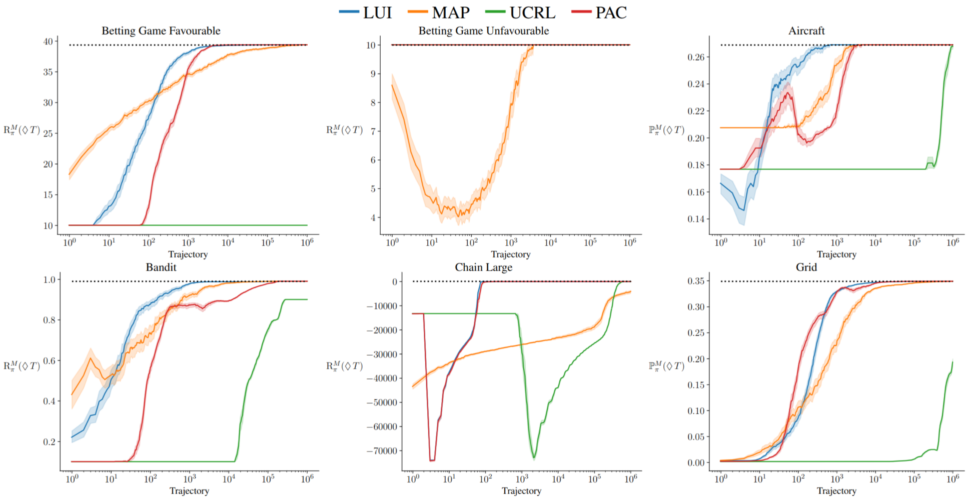
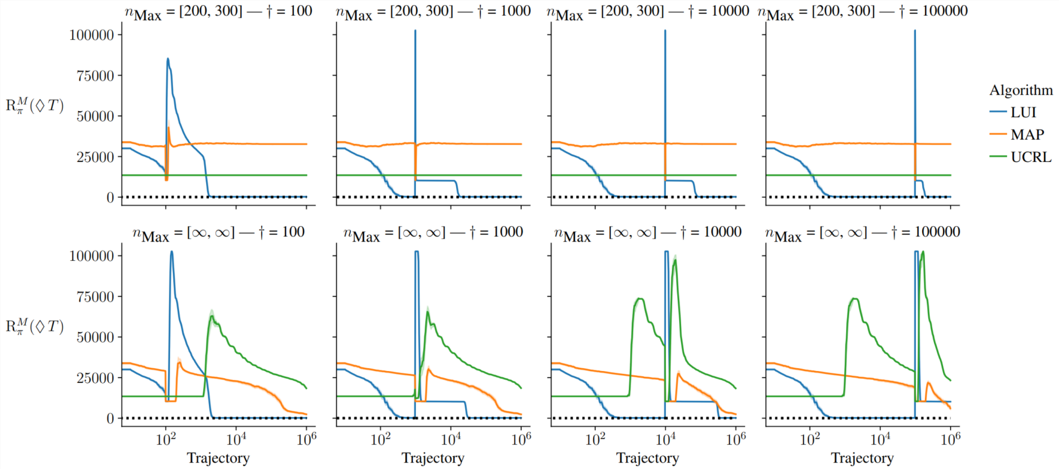


Figure 3: Comparison of the performance of robust policies on different environments against the number of trajectories processed (on log-scale). The dashed line indicates the optimal performance.

Robustness in changing environments



Summary

What to remember:

- Robust MDPs, robust value iteration, especially IMDPs and L_1 MDPs,
- Learning probabilities (frequentist & Bayesian),
- Learning intervals (PAC and LUI),
- Reinforcement learning: UCRL2.

What if the state of the MDP is not fully observable?

(Optional) Reading material

- Marnix Suilen, Thom S. Badings, Eline M. Bovy, David Parker, Nils Jansen. **Robust Markov Decision Processes: A Place Where AI and Formal Methods Meet**. Principles of Verification (3) 2025.
- Iyengar, G. Robust Dynamic Programming. Mathematics of Operations Research. 2005.
- Wiesemann, W., Kuhn, D., & Rustem, B. Robust Markov Decision Processes. Mathematics of Operations Research. 2013.
- Suilen, M., Simão, T. D., Parker, D., & Jansen, N. Robust Anytime Learning of Markov Decision Processes. Advances in Neural Information Processing Systems (NeurIPS). 2022.
- Jaksch, T., Ortner, R., & Auer, P. Near-optimal Regret Bounds for Reinforcement Learning. Journal of Machine Learning Research. 2010.