Decision-Making Under Uncertainty (Meets Neurosymbolic AI)

Prof. Dr. Nils Jansen

MOVEP 2024

RUHR
UNIVERSITÄT
BOCHUM

Today's Agenda

Decision-Making under Uncertainty

Markov Decision Processes

Learning Probabilities from Data

Robust Markov Decision Processes

Partially Observable Markov Decision Processes

Who Are We?

- **Studies and PhD at RWTH Aachen, Germany**
- **Postdoctoral Researcher at UT Austin, TX, USA**
- **Assistant/Associate Professor Radboud University Nijmegen**
- **Chair of Artificial Intelligence and Formal Methods at RUB**
- **Group of Safe and Dependable Artificial Intelligence at Radboud University Nijmegen, NL**
- **European Research Council (ERC) Starting Grant: Data-Driven Verification and Learning Under Uncertainty (DEUCE)**

Our Mission: Increase the trustworthiness of artificial intelligence (AI).

RWTH AACHEN UNIVERSITY

TEXAS The University of Texas at Austin

Radboud University

RUHR UNIVERSITÄT BOCHUM RUB

DEUCE. erc
Data-Driven Verification
and Learning under Uncertainty.

Artificial Intelligence

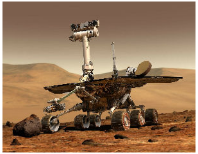
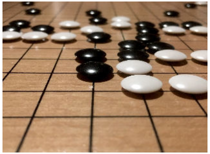
Formal Methods

Motivation: Artificial Intelligence (AI) Systems

…have great potential for our society…

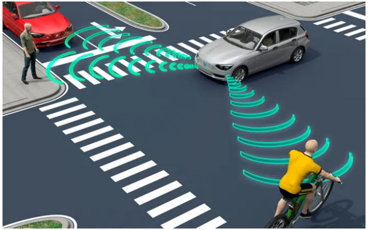




Motivation: Safety in AI

Delivery Drone
Amazon

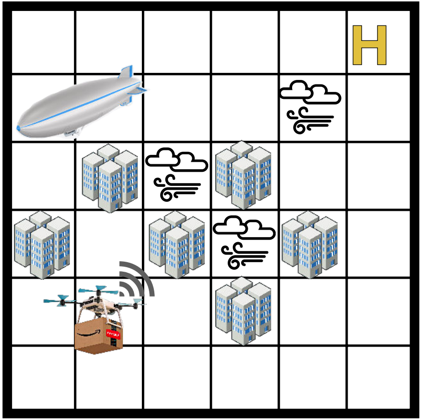


Self-driving Car
The Guardian

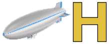
Spacecraft Operations
Airbus

“AI systems need to be resilient and secure. They need to be safe, ensuring a fall back plan in case something goes wrong, as well as being accurate, reliable and reproducible.”
European Commission. Ethics Guidelines for Trustworthy Artificial Intelligence. 2019.

Intelligent Decision-Making Under Uncertainty



Uncertainty caused by sensor imprecision, wind gusts, and limited view



Complex **task specification**

“Almost surely, always return to the halting pad after a delivery, and a crash can only occur with probability at most 0.001%.”

What are the challenges if we aim to provide guarantees on the behavior of an agent?

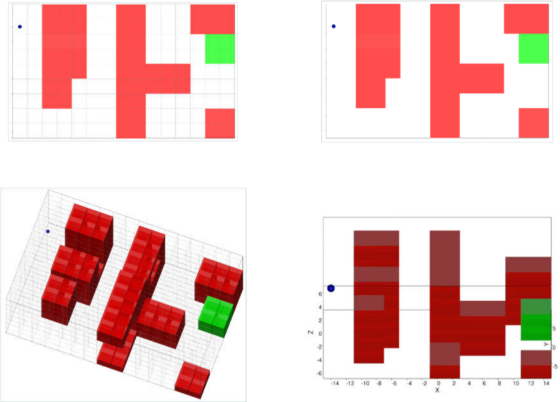
Formal specification in probabilistic temporal logic:

$$Pr_{=1}(\Box\,(\textbf{delivery} \rightarrow \Diamond \textbf{H})) \;\wedge\; Pr_{\leq 0.001}(\Diamond \textbf{crash})$$

Scientific Challenges

Major scientific challenges remain in decision-making under uncertainty.

- **scalability** for realistic applications with high-dimensional feature spaces
- **continuous state** and **action spaces**
- **uncertainty** and **partial information**
- **guarantees** for **data-driven problems**



Reinforcement Learning

A reinforcement learning (RL) agent

- Explores its environment by taking **actions** and **observing** feedback signals

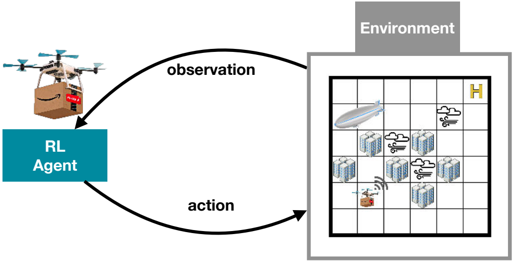
- Episodically determines the optimal way to make decisions within the environment

Limitations

- Exploration is **safety-critical**

- RL is **data-hungry**

- Rewards cannot capture **sophisticated task specifications**



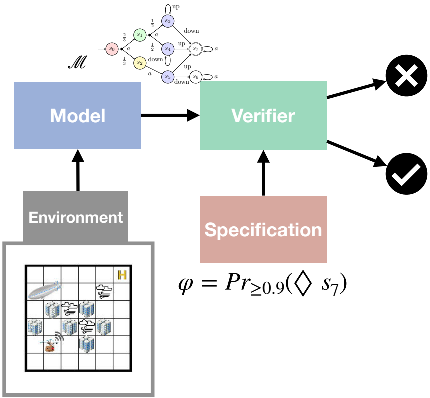
Find a policy π that maximizes $\mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R_t\right]$

with the discount factor $0 \leq \gamma^t \leq 1$ and reward R_t at time t.

Formal Verification

Model Checking

- Given a formal **model** of an **environment**, **prove** its correctness regarding a formal **specification**
- Rigorous numerical techniques for **uncertainty models**
- **Markov decision process (MDP)** $\mathscr{M}$, extensions: **Uncertainty**, **partial observability**, **adversaries**



$$\varphi = Pr_{\geq 0.9}(\Diamond\ s_7)$$

Limitations

- **Hardness** of underlying problems, **scalability** to real-world scenarios
- Availability of **models**
- Integration with **data-driven** problems

$$\forall s \in S \backslash T \,.\, \forall P \in \mathscr{P} \,.\quad p_s \leq \sum_{a \in Act} \sigma(s,a) \cdot \sum_{s' \in S} P(s,\alpha,s') \cdot p_{s'}$$

(**nonconvex** and **semi-infinite** optimization problems)

For the model $\mathscr{M}$, compute a policy π such that $\mathscr{M}^{\pi} \models \varphi$ or prove that no such policy exists.

A Multidisciplinary Approach

Innovation: Tightly integrated verification and reinforcement learning methods that are robust against uncertainty under real-world conditions

Rigorous Model-Based Verification

Formal Methods

Control Theory

Control for Continuous Spaces

Decision-Making under Uncertainty

Artificial Intelligence

Robotics

Case Studies

A Multidisciplinary Approach

Innovation: Tightly integrated verification and reinforcement learning methods that are robust against uncertainty under real-world conditions

Rigorous Model-Based Verification

Formal Methods

Uncertainty

Control Theory

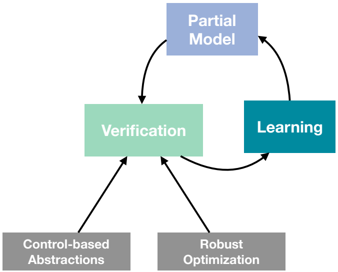
Control for Continuous Spaces

Decision-Making under Uncertainty

Artificial Intelligence

Robotics

Case Studies

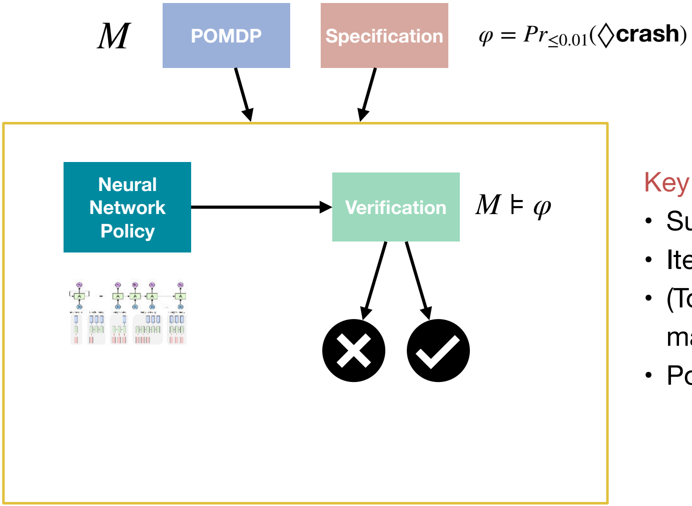


Reduce uncertainty or be robust against uncertainty.

"Uncertainty is largely related to the lack of predictability of some major events or stakes, or a lack of data"

Argote, L. (1982). Input uncertainty and organizational coordination in hospital emergency units. Administrative science quarterly, 420-434.

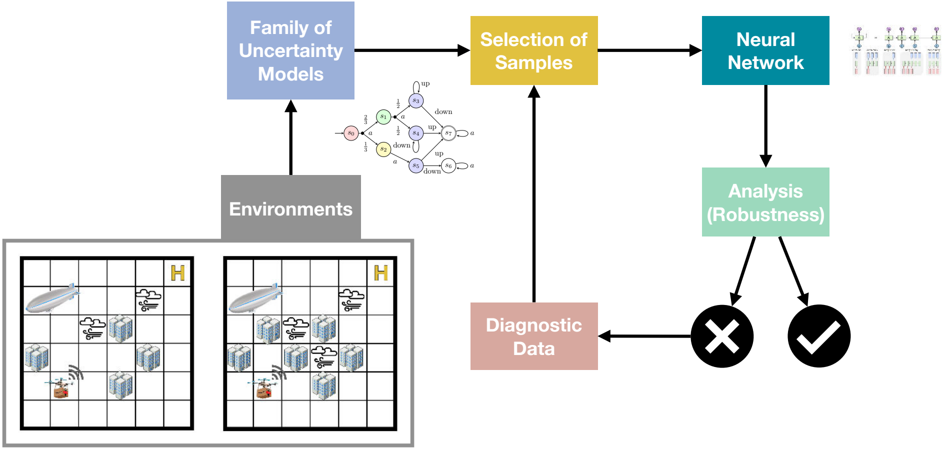
Data-Driven Verification



Key Requirements

- Suitable **neural network architecture**
- Iteratively improve the level of training
- (Towards) understandable decision-making
- Policy should be **easy-to-verify**

Challenge: Robust Neural Network Controllers

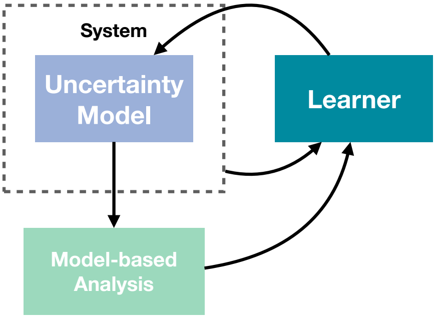


Challenge: Uncertainty Models

“How to combine data-driven learning and model-based reasoning?”
Dutch Research Council (NWO). Artificial Intelligence research agenda for the Netherlands. 2019.

Problem: Learn and analyze a model

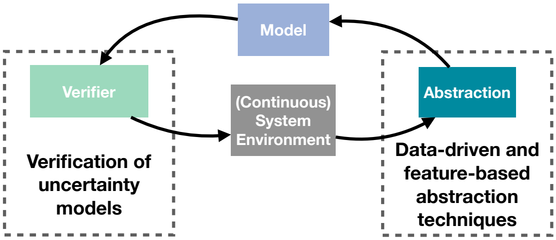
- from **data** or **samples** obtained by actively probing a system
- that **robustly captures uncertainty and probabilities**
- that is **amenable to efficient model-based analysis**

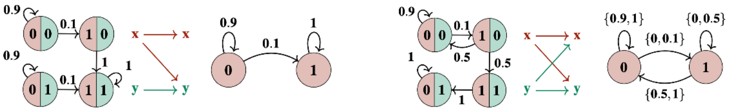


Challenge: Data-Driven Verification and Abstraction

Data-driven abstraction integrated with verification

- **Construct** candidate abstraction that is good-to-verify
- Reduced features spaces or suitable discretization
- Provide **exact** or **probably-approximately-correct (PAC)** guarantees





feature-based abstraction for MDPs M_1 and M_2 with features x and y and dynamic Bayesian networks defining variable dependencies

Challenge: Safety and Correctness in Reinforcement Learning

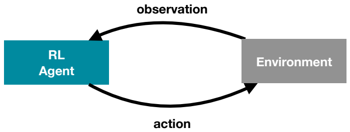
- Ensure **safe and correct** behavior or **exploration** of RL via a **shield** that blocks **unsafe, incorrect,** or **irrelevant** actions

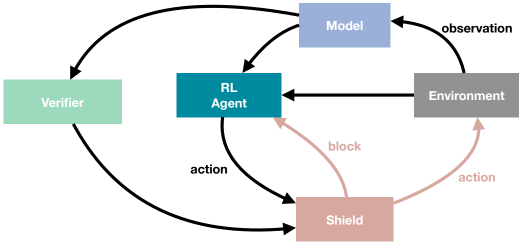
- Improve the **convergence rate** of RL

- A shield injects **domain knowledge** to reduce the search space for RL

- Integrate a **verifier** with RL that constructs and updates a **shield** according to data

- Tradeoff between **correctness** and RL **exploration**





Constrained RL

Maximize $\mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R_t\right]$ subject to $\mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t C_t\right] < \lambda$

with additional cost function C and threshold λ

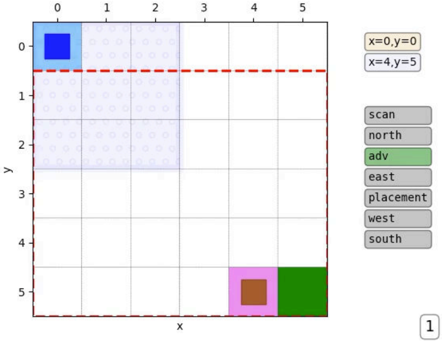
Shielded RL

Maximize $\mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R_t\right]$ subject to $\mathcal{M}^\pi \models \varphi$

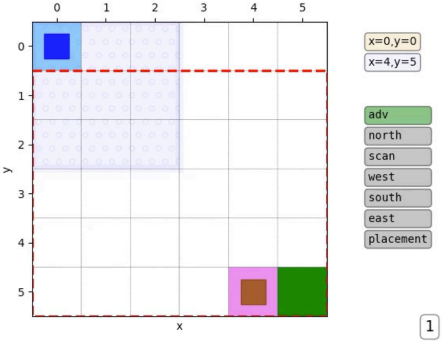
with temporal logic specification φ

Shielded and Unshielded RL

- (Tuned) RL with REINFORCE
- Simple shield construction using the Storm model checker and mask() function of tensorflow



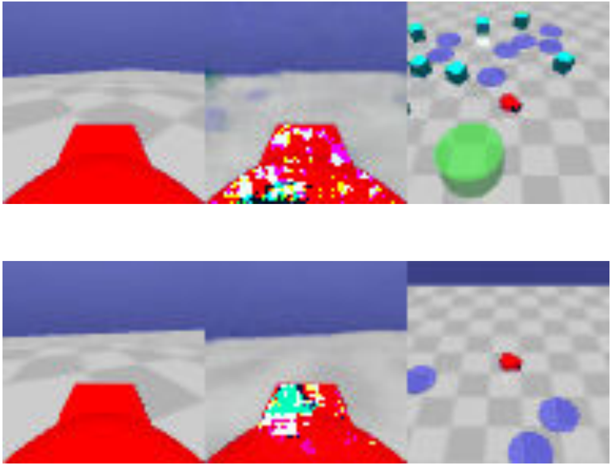
Unshielded



Shielded

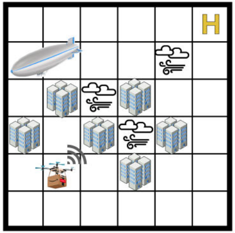
Learning Safely From Pixels

- Stochastic Latent Actor-Critic Model
- High-dimensional observations driven by low-dimensional underlying latent process
- Great performance with high sample efficiency
- Learn a safety critic to train policy

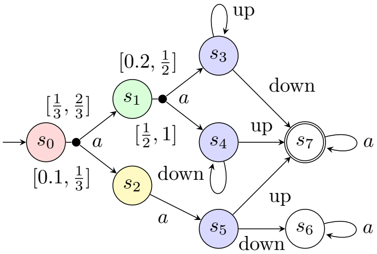


Demonstrator: Drone - Task and Motion Planning

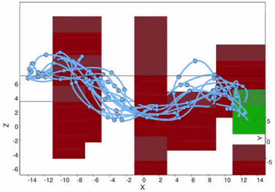
Delivery Drone
in Urban Environments
AMAZON



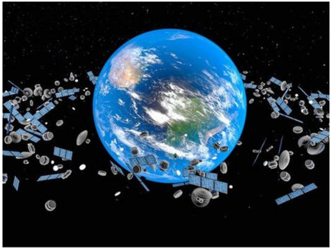
Path Planning Around Buildings
Under Uncertainty
AAAI 2022



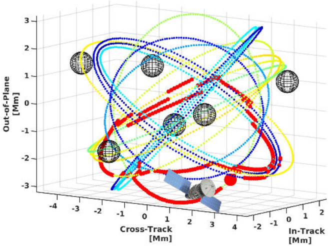
Uncertain MDP



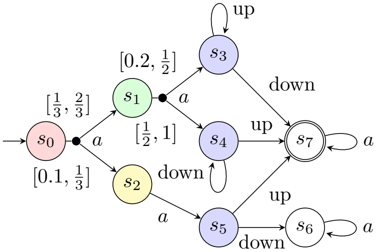
Demonstrator: Satellite - Planning and Collision Avoidance



Spacecraft Operations in Crowded Environments
NASA

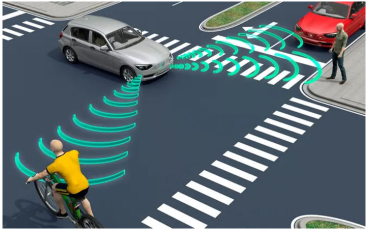


Orbit Switches and Collision Avoidance
AAAI 2021

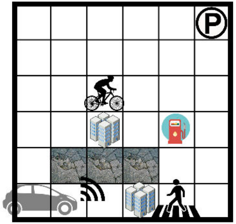


Uncertain Partially Observable MDP

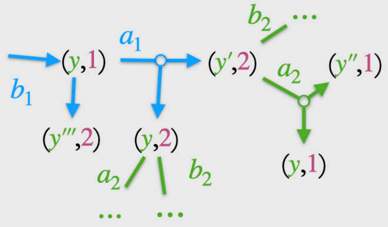
Demonstrator: Autonomous Car - Safe Decision Making



Self-driving Car With Imprecise Sensors
The Guardian

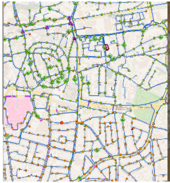


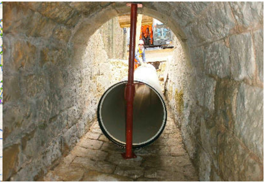
Safe Decision-Making Under Partial Information
RSS 2021

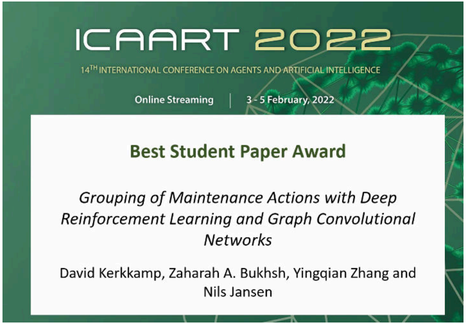


Partially Observable Stochastic Game

Relevance for Industry?







Selected Theses Projects

- **Convex optimization for uncertain Markov decision processes**
  Bachelor 2018, **IJCAI 2019**

- **Human-in-the-loop strategy synthesis: PAC-MAN verified**
  Bachelor 2019

- **Routing Algorithms for Autonomous Agricultural Vehicles**
  Bachelor 2019

- **Robust Convex Optimization for Uncertain Partially Observable Markov Decision Processes**
  Master 2019, **IJCAI 2020**

- **Entropy-guided decision making in multiple-environment Markov decision processes**
  Master 2020

- **Approximating Black-Box Deep Neural Networks using Active Learning as a Proxy Measurement for Robustness**
  Master 2020

- **Grouping of Maintenance Actions on Sewer Pipes: Using Deep Reinforcement Learning and Graph Neural Networks**
  Master 2021, **ICAART 2022**

- **Safe Reinforcement Learning From Pixels Using a Stochastic Latent Representation**
  Master 2022, **ICLR 2023**

Formal Verification

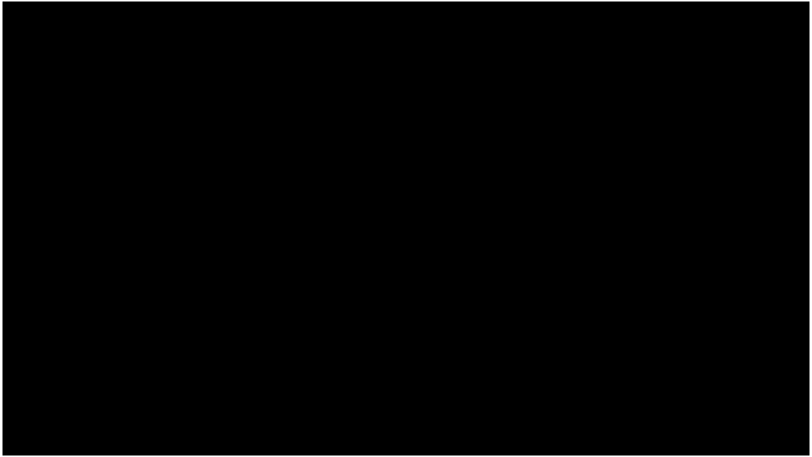
Machine Learning and AI

Industrial Applications

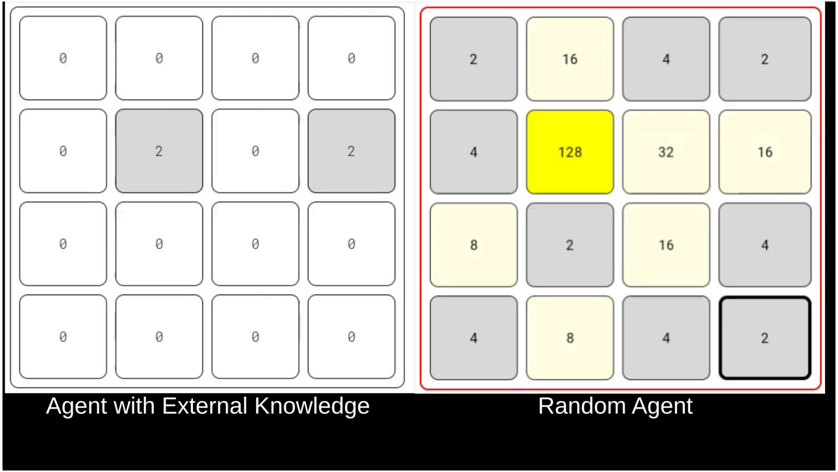
Convex Optimization

Games

Learning From Human Data



Side Information for RL Agents



Literature

Thom S. Badings, Thiago D. Simão, Marnix Suilen, Nils Jansen:
Decision-making under uncertainty: beyond probabilities. Int. J. Softw. Tools Technol. Transf. 25(3): 375-391 (2023)

Today's Agenda

Decision-Making under Uncertainty

Markov Decision Processes

Learning Probabilities from Data

Robust Markov Decision Processes

Partially Observable Markov Decision Processes

Today's Agenda

Decision-Making under Uncertainty

Markov Decision Processes

Learning Probabilities from Data

Robust Markov Decision Processes

Partially Observable Markov Decision Processes

Today's Agenda

Decision-Making under Uncertainty

Markov Decision Processes

Learning Probabilities from Data

Robust Markov Decision Processes

Partially Observable Markov Decision Processes

Today’s Agenda

Decision-Making under Uncertainty

Markov Decision Processes

Learning Probabilities from Data

Robust Markov Decision Processes

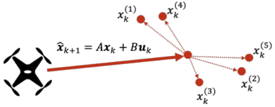
Partially Observable Markov Decision Processes

Challenges for AI in Robotics

Challenge 1: How to obtain a model for an AI system under (epistemic) uncertainty?

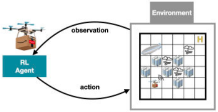
Challenge 2: How to explore an uncertain environment safely?

Challenge 3: How to actively exploit an autonomous system's sensing capabilities?



Challenge 4: How to plan for an autonomous system if only limited data is available?

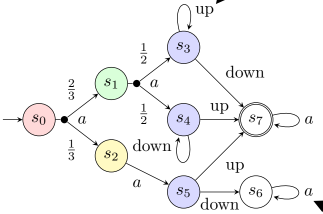
Challenge 5: How to provide safety guarantees if we are dealing with realistic continuous spaces?



Challenge 6: Neurosymbolic AI: How to learn and verify explainable controllers?

Coming: Programmatic RL

RL Agent



Formal Model

```
s : [0..6];

// reading
[] s=0 -> 1/3: (s'=0) + 1/3: (s'=1) + 1/3: (s'=2);
[] s=1 -> 1/4: (s'=2) + 1/4: (s'=3) + 1/2: (s'=5);
[] s=2 -> 1: (s'=4);
[] s=3 -> 1/2: (s'=3) + 1/2: (s'=5);
[] s=4 -> 1/5: (s'=5) + 4/5: (s'=6);
[] s=5 -> 1: (s'=5);
[] s=6 -> 1: (s'=6);
```

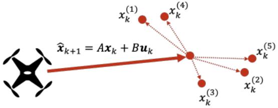
Probabilistic Programming Language

The key objective of the DEUCE project is to elevate the state-of-the-art in safe and correct decision-making for AI systems.

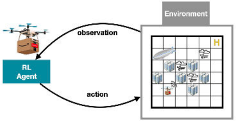
DEUCE.

Data-Driven Verification and Learning under Uncertainty.

Uncertainty and Partial Information



Continuous Spaces



(Safe) Reinforcement Learning

I WANT YOU FOR NEUROSYMBOLIC AI!

(a.k.a. we're hiring!)

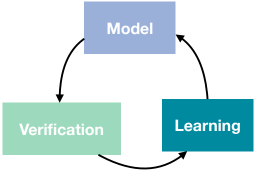
Thanks for the slides!

Marnix Suilen

https://www.marnixsuilen.nl/

Summary and Vision

- Tightly integrated and novel learning and verification methods that are dedicated to AI systems
- Fundamental scientific research that **elevates the state-of-the-art in formal verification and safe reinforcement learning**
- **Our vision of the future is to help developing data-driven systems whose decisions are known to be correct.**



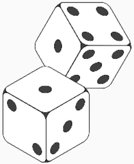
Nils Jansen
http://nilsjansen.org
n.jansen@rub.de

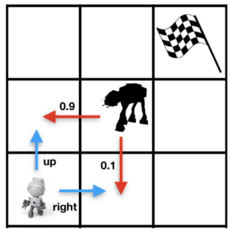
DEUCE.
Data-Driven Verification and Learning under Uncertainty.

erc

Markov decision processes

What is this lecture about? Probabilities!

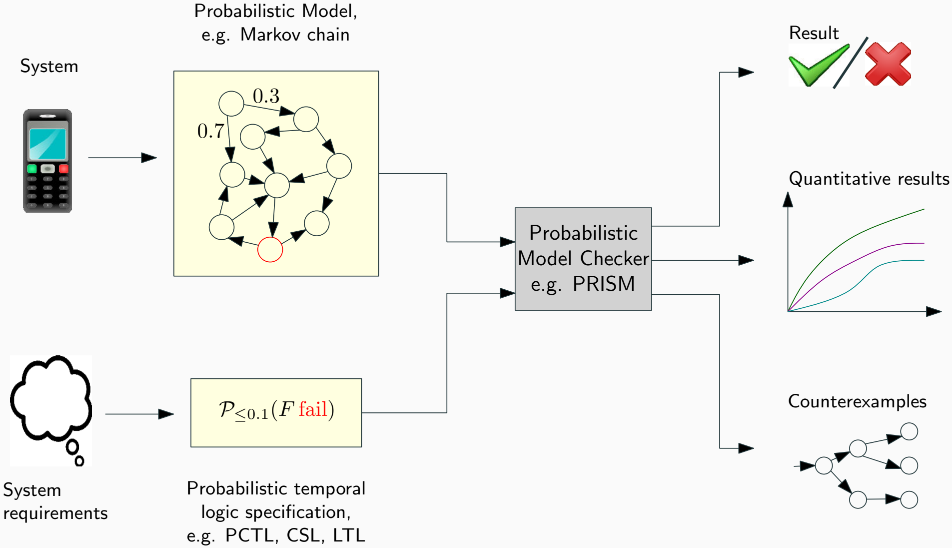




The probabilistic model space

DTMC	=	Discrete-time Markov chain
DTMRM	=	Discrete-time Markov reward model
DTMDP	=	Discrete-time Markov decision process
DTMRDP	=	Discrete-time Markow reward decision process
CTMC	=	Continuous-time Markov chain
CTMRM	=	Continuous-time Markov reward model
CTMDP	=	Continuous-time Markov decision process
CTMRDP	=	Continuous-time Markow reward decision process

The Model Checking Flow



Probabilistic model checking involves . . .

- Construction of models

  from a description in a high-level language

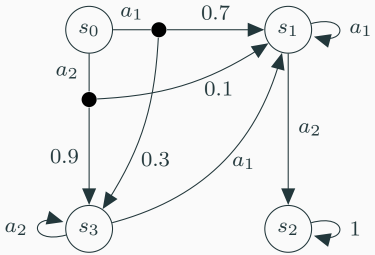
- Probabilistic model checking algorithms
  - graph-theoretical algorithms
    - for reachability, identifying strongly connected components, . . .
  - numerical computation
    - linear equation systems, linear optimization problems
    - iterative methods, direct methods
  - automata for regular languages
  - sampling-based methods for approximate analysis

- Efficient implementation techniques
  - essential for scalability to real-life applications
  - symbolic data structures based on BDDs
  - algorithms for model minimization, abstraction, . . .

Markov Decision Processes

Markov decision process (MDP) is a tuple (S, A, P):

- S finite set of states,
- A finite set of actions,
- $P\colon S \times A \to \mathcal{D}(S)$ transition function.



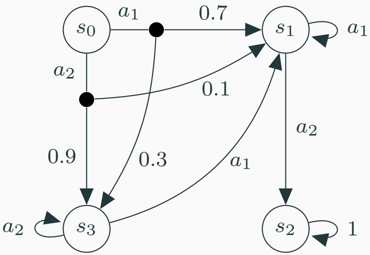
Markov Decision Processes

Markov decision process (MDP) is a tuple (S, A, P):

- S finite set of states,
- A finite set of actions,
- $P\colon S \times A \to \mathcal{D}(S)$ transition function.

MDP with discounted reward is a tuple (S, A, P, R, γ):

- $R\colon S \times A \to \mathbb{R}$ reward function,
- $\gamma \in (0, 1)$ discount factor.



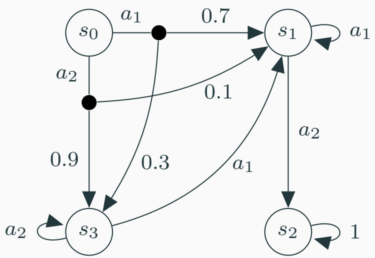
Markov Decision Processes

Markov decision process (MDP) is a tuple (S, A, P):

- S finite set of states,
- A finite set of actions,
- $P\colon S \times A \to \mathcal{D}(S)$ transition function.

MDP with discounted reward is a tuple (S, A, P, R, γ):

- $R\colon S \times A \to \mathbb{R}$ reward function,
- $\gamma \in (0, 1)$ discount factor.



For simplicity, we often write $P(s, a, s')$ for the probability $P(s, a)(s')$.

Solving MDPs

Several approaches:

1. Value iteration
    - approximate with iterative solution method
    - corresponds to a fixed point computation
    - preferable in practice, implemented in PRISM
2. Reduction to a linear programming (LP) problem
    - solve with linear optimization techniques (Simplex algorithm)
    - exact solution using well-known methods
    - better (theoretical) complexity, good for small examples
3. Policy iteration
    - iteration over policies.

Solving MDPs—Value Iteration

For an MDP (S, A, P, R, γ), the goal is to compute a policy $\pi \colon S \to A$ that maximizes the expected discounted reward

$$\mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t\right]$$

where r_t is the reward collected at time t.

Solving MDPs—Value Iteration

For an MDP (S, A, P, R, γ), the goal is to compute a policy $\pi \colon S \to A$ that maximizes the expected discounted reward

$$\mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t\right]$$

where r_t is the reward collected at time t.

Just as for reachability, memoryless deterministic policies are sufficient for optimizing discounted reward.

Discounted value iteration

1. Initialize $V_0(s) = 0$ for all $s \in S$, set a precision ϵ, $error = 1$.

Discounted value iteration

1. Initialize $V_0(s) = 0$ for all $s \in S$, set a precision ϵ, $error = 1$.
2. Repeat until convergence (While $error > \epsilon$):

Discounted value iteration

1. Initialize $V_0(s) = 0$ for all $s \in S$, set a precision ϵ, $error = 1$.
2. Repeat until convergence (While $error > \epsilon$):
   - Update value function for each $s \in S$:

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

Discounted value iteration

1. Initialize $V_0(s) = 0$ for all $s \in S$, set a precision ϵ, $error = 1$.
2. Repeat until convergence (While $error > \epsilon$):
   - Update value function for each $s \in S$:

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

   - Update error: $error = \max_{s \in S} \{ |V_{n+1}(s) - V_n(s)| \}$,

Discounted value iteration

1. Initialize $V_0(s) = 0$ for all $s \in S$, set a precision ϵ, $error = 1$.
2. Repeat until convergence (While $error > \epsilon$):
   - Update value function for each $s \in S$:

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

   - Update error: $error = \max_{s \in S} \{|V_{n+1}(s) - V_n(s)|\}$,
3. After convergence V^* is the optimal value function, and the associated optimal policy π^* can be found by

$$\pi^*(s) = \arg\max_{a \in A} \left\{ R(s,a) + \gamma \sum_{s' \in S} P(s,a,s') V^*(s') \right\}.$$

Theory on value iteration

Value iteration is a fixed point operation of applying the Bellman operator the value function:

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

because of the discount factor $\gamma \in (0,1)$ this equation is a contraction mapping, with a unique fixed point

$$V^*(s) = \max_{a \in A} \left\{ R(s,a) + \gamma \sum_{s' \in S} P(s,a,s') V^*(s') \right\}.$$

Why rewards

Rewards allow for more complicated task specifications beyond reachability.

Optimal policies for (discounted) reward objectives exist, are memoryless deterministic, and computable in polynomial time (via linear programming).

In contrast, LTL objectives are more expressive, but require (finite) memory policies and are computationally more expensive.

Optimal policies for rewards are also learnable in a reinforcement learning setting.

Summary so far

What to remember:

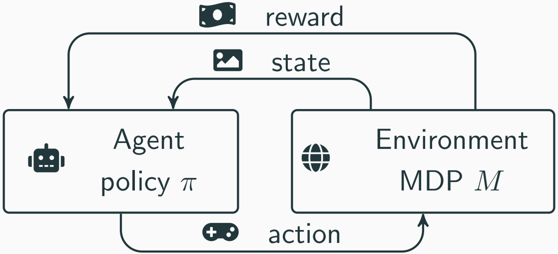
- Definition of MDPs
- Solving MDPs

Where do the probabilities come from?

Learning probabilities from Data

Reinforcement learning (RL)

Reinforcement learning (RL) is a general technique to find a policy in an MDP where the transition function is unknown.



What will we learn?

This lecture considers models and algorithms for:

- How to learn probabilities from data,
- Robust MDPs: a more general MDP model where the transition function is uncertain and only known to be in some set,
- Robust learning: using robust MDPs in an RL setting to account for statistical errors and changing environments,
- UCRL2: an RL algorithm that uses optimism in the face of uncertainty to achieve efficient data collection.

Frequentist Learning

Frequentist learning = counting!

Frequentist Learning

Frequentist learning $=$ counting!

Suppose we have a state-action (s, a) pair with m successor states, and want to learn the probabilities

$$P(s, a, s_1), \ldots, P(s, a, s_m).$$

Frequentist Learning

Frequentist learning = counting!

Suppose we have a state-action (s, a) pair with m successor states, and want to learn the probabilities

$$P(s, a, s_1), \ldots, P(s, a, s_m).$$

Definition (Frequentist learning)

1. Take N samples of (s, a),
2. Count how many times we see successor state s_i, call this $\#(s, a, s_i)$,
3. We estimate $\tilde{P}(s, a, s_i) = \frac{\#(s,a,s_i)}{N}$.

Frequentist learning - why does this work?

An MDP is Markovian: transition probability $P(s, a, s_i)$ is independent of all other transition probabilities.

Frequentist learning - why does this work?

An MDP is Markovian: transition probability $P(s, a, s_i)$ is independent of all other transition probabilities.

$\tilde{P}(s, a, \cdot)$ forms a valid probability distribution:

$$N = \sum_j \#(s, a, s_j) \implies \sum_i \tilde{P}(s, a, s_i) = \sum_i \frac{\#(s, a, s_i)}{\sum_j \#(s, a, s_j)} = 1.$$

Key problem in frequentist learning

Frequentist learning is sensitive to observations.

Key problem in frequentist learning

Frequentist learning is sensitive to observations.

If we do not observe a transition, we have $\#(s, a, s_i) = 0$,
and then we learn $\tilde{P}(s, a, s_i) = 0$.

What to do if we know that this transition exists, i.e., $P(s, a, s_i) > 0$?

Bayesian Learning

Bayesian learning allows us to incorporate prior knowledge.

General idea:

$$Posterior \propto Prior \cdot Likelihood.$$

Bayesian Learning

Bayesian learning allows us to incorporate prior knowledge.

General idea:

$$Posterior \propto Prior \cdot Likelihood.$$

Conjugate prior: for certain families of priors and likelihoods, the posterior distribution is already known.

The Dirichlet distribution is conjugate to the multinomial likelihood (the probability of counts):

$$Dirichlet \propto Dirichlet \cdot Multinomial.$$

Bayesian Learning

Bayesian learning starts again with counting in a data set.

Bayesian Learning

Bayesian learning starts again with counting in a data set.

Suppose we have a state-action (s, a) pair with m successor states, and want to learn the probabilities

$$P(s, a, s_1), \ldots, P(s, a, s_m).$$

Again we take $N = \#(s, a)$ samples and count how many times we see s_i: $k_i = \#(s, a, s_i)$.

Updating distributions

These counts have a multinomial likelihood

$$Mn(k_1, \ldots, k_m \mid P(s, a, \cdot)) \propto \prod_{i=1}^{m} P(s, a, s_i)^{k_i}.$$

Updating distributions

These counts have a multinomial likelihood

$$Mn(k_1, \ldots, k_m \mid P(s, a, \cdot)) \propto \prod_{i=1}^{m} P(s, a, s_i)^{k_i}.$$

The Dirichlet distribution is a conjugate prior to the multinomial likelihood:

$$Dir(P(s, a, \cdot) \mid \alpha_1, \ldots, \alpha_m) \propto \prod_{i=1}^{m} P(s, a, s_i)^{\alpha_i - 1}.$$

Updating distributions

These counts have a multinomial likelihood

$$Mn(k_1, \dots, k_m \mid P(s, a, \cdot)) \propto \prod_{i=1}^{m} P(s, a, s_i)^{k_i}.$$

The Dirichlet distribution is a conjugate prior to the multinomial likelihood:

$$Dir(P(s, a, \cdot) \mid \alpha_1, \dots, \alpha_m) \propto \prod_{i=1}^{m} P(s, a, s_i)^{\alpha_i - 1}.$$

Given a prior Dirichlet distribution and a multinomial likelihood, we can update the prior to a posterior Dirichlet distribution with

$$Dir(P(s, a, \cdot) \mid \alpha_1 + k_1, \dots, \alpha_m + k_m).$$

MAP estimation

After computing the posterior distribution $Dir(P(s, a, \cdot) \mid \alpha_1, \ldots, \alpha_m)$, we derive point estimates via the mode:

$$\tilde{P}(s, a, s_i) = \frac{\alpha_i - 1}{(\sum_{j=1}^{m} \alpha_j) - m}.$$

Key problem in Bayesian learning

Bayesian learning (MAP estimation) can be heavily biased to the prior.

Hence, a challenge is choosing a good prior as starting point.

Key problem in Bayesian learning

Bayesian learning (MAP estimation) can be heavily biased to the prior.

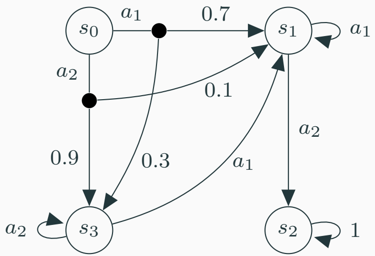
Hence, a challenge is choosing a good prior as starting point.

A Dirichlet distribution with $\alpha_i = \alpha_j$ for all i, j yields a uniform distribution.
The higher the values for α_i, the more data you need to shift away from the prior.

Depending on the specific situation, better choices may exist!

Example

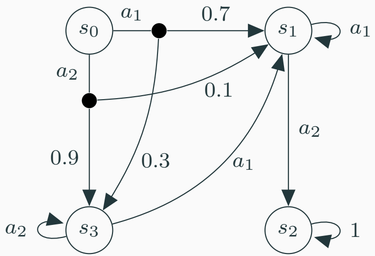
Suppose we want to learn (s_0, a_1) in the MDP:



Suppose $N = 20$, $\#(s_0, a_1, s_1) = 13$, $\#(s_0, a_1, s_3) = 7$.

Example

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

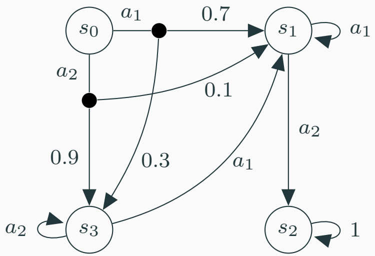


Suppose $N = 20$, $\#(s_0, a_1, s_1) = 13$, $\#(s_0, a_1, s_3) = 7$.

- Frequentist: $\tilde{P}(s_0, a_1, s_1) = \frac{13}{20} = 0.65$, $\tilde{P}(s_0, a_1, s_3) = \frac{7}{20} = 0.35$.

Example

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



Suppose $N = 20$, $\#(s_0, a_1, s_1) = 13$, $\#(s_0, a_1, s_3) = 7$.

- Frequentist: $\tilde{P}(s_0, a_1, s_1) = \frac{13}{20} = 0.65$, $\tilde{P}(s_0, a_1, s_3) = \frac{7}{20} = 0.35$.
- Bayesian: Assume prior Dirichlet distribution with $\alpha_1 = \alpha_3 = 10$.
  Posterior: $\alpha_1 = 10 + 13$, $\alpha_3 = 10 + 7$.
  MAP-estimation:
  $\tilde{P}(s_0, a_1, s_1) = \frac{22}{38} = 0.579$,
  $\tilde{P}(s_0, a_1, s_3) = \frac{16}{38} = 0.421$.

Summary so far

What to remember:

- Solving MDPs
- Learning probabilities (frequentist & Bayesian),

Question: What about numerical imprecision and statistical errors?

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

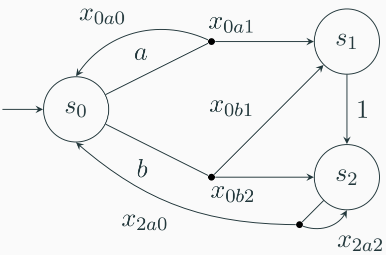
The uncertain transition function $\mathcal{P}$ is a set of standard transition functions $P \colon S \times A \to \mathcal{D}(S)$. We also write $P \in \mathcal{P}$.

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

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:

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

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

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!

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.

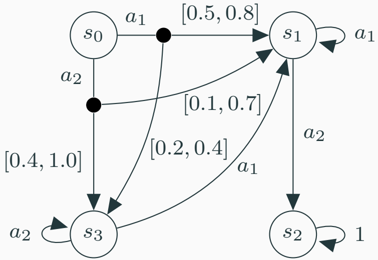
- Interval MDPs (IMDPs): each transition has a probability interval,
- L_1 MDPs: each state-action pair has an uncertainty set around an empirical distribution.

Interval MDPs & Robust Learning

Interval MDPs

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

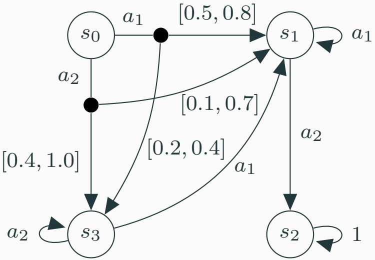
- S, A, R and γ are as for (robust) MDPs,



Interval MDPs

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

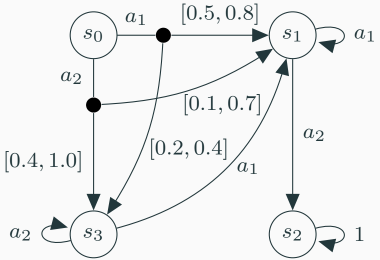
- S, A, R and γ are as for (robust) MDPs,
- $\underline{P}$ assigns a lower bound to each transition: $\underline{P}\colon S \times A \times S \to [0,1]$ with $\sum_{s'} \underline{P}(s,a,s') \leq 1$,



Interval MDPs

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

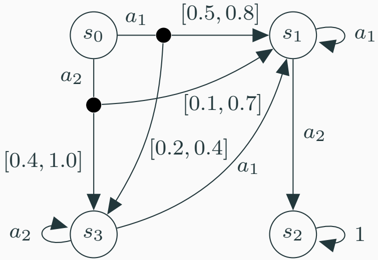
- S, A, R and γ are as for (robust) MDPs,
- $\underline{P}$ assigns a lower bound to each transition: $\underline{P}\colon S \times A \times S \to [0,1]$ with $\sum_{s'} \underline{P}(s,a,s') \leq 1$,
- $\overline{P}$ assigns an upper bound to each transition: $\overline{P}\colon S \times A \times S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

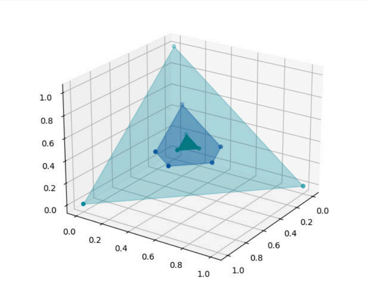


The uncertainty set of IMDPs

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).

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.

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.

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:

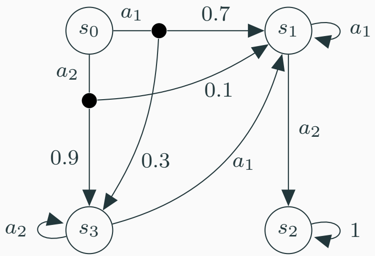
1. Compute point estimates via frequentist or Bayesian learning for every transition (s, a, s'),
2. Choose an error rate $\epsilon \in (0, 1)$, and compute the error rate for the whole model: $\epsilon_M = {}^{\epsilon}/_{\sum_{s,a} |Post(s,a)_{>1}|}$, where $|Post_{>1}(s, a)|$ is the number of successor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

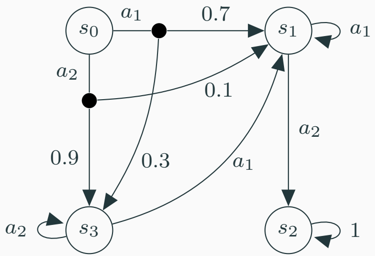
Suppose we want to learn (s_0, a_1) in the MDP:



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:

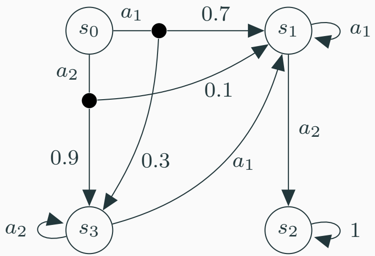


Suppose we have $N = 20$, $\tilde{P}(s_0, a_1, s_1) = 0.65$, $\tilde{P}(s_0, a_1, s_3) = 0.35$, and set $\epsilon = 0.01$.

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

Example

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

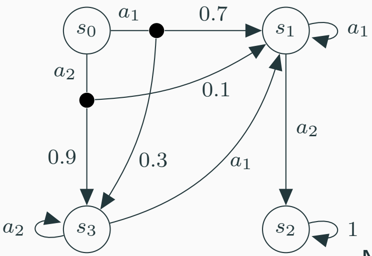


Suppose we have $N = 20$, $\tilde{P}(s_0, a_1, s_1) = 0.65$, $\tilde{P}(s_0, a_1, s_3) = 0.35$, and set $\epsilon = 0.01$.

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

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 $k_i = \#(s, a, s_i)$

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 $(s, a, s_1), \dots, (s, a, s_m)$.

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 $\| P - Q \|_1 = \sum_s |P(s) - Q(s)|$.

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 $\| P - Q \|_1 = \sum_s |P(s) - Q(s)|$.

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.

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 $\#(s,a) \mathrel{+}= v_k(s,a)$, $\#(s,a,s') \mathrel{+}= v_k(s,a,s')$

Comparison of different learning methods

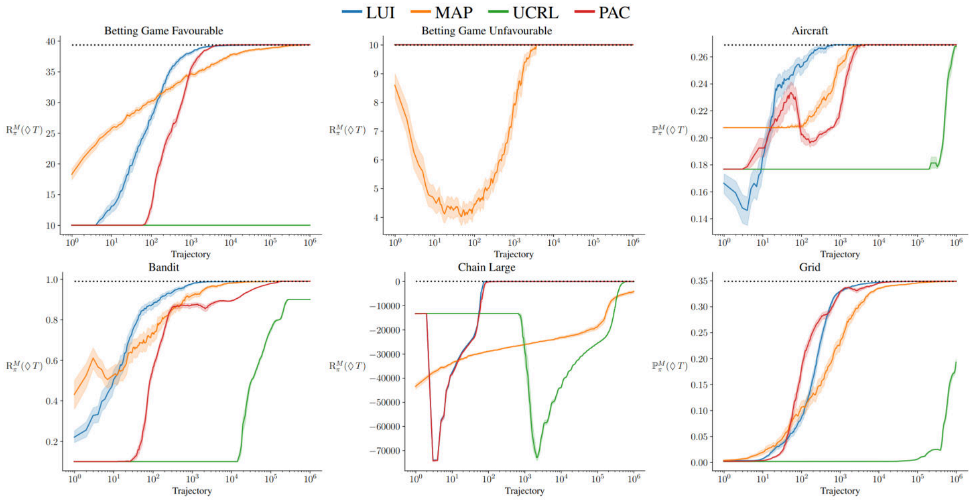
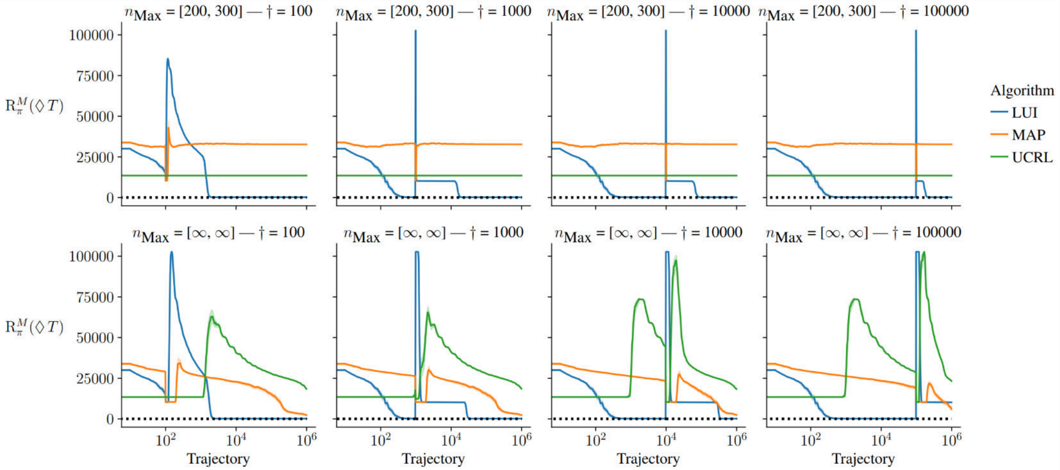


Figure 3: Comparison of the performance of robust policies on different environments against the number of trajectories processed (on log-scale). The dashed line indicates the optimal performance.

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

-
- 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.

Partially Observable MDPs

POMDPs

Partially observable MDPs (POMDPs) are an extension of MDPs with an observation function.

POMDPs

Partially observable MDPs (POMDPs) are an extension of MDPs with an observation function.

Definition (POMDP)
A POMDP is a tuple $(S, A, P, s_0, R, Z, O) = (M, Z, O)$ where

- $M = (S, A, P, s_0)$ is an MDP,
- Z is a finite set of observations,
- $O\colon S \times A \to \mathcal{D}(Z)$ is the probabilistic observation function.

POMDPs

Partially observable MDPs (POMDPs) are an extension of MDPs with an observation function.

Definition (POMDP)
A POMDP is a tuple $(S, A, P, s_0, R, Z, O) = (M, Z, O)$ where

- $M = (S, A, P, s_0)$ is an MDP,
- Z is a finite set of observations,
- $O \colon S \times A \to \mathcal{D}(Z)$ is the probabilistic observation function.

Often we restrict to POMDPs with deterministic observations: $O \colon S \times A \to Z$.

POMDPs

Partially observable MDPs (POMDPs) are an extension of MDPs with an observation function.

Definition (POMDP)
A POMDP is a tuple $(S, A, P, s_0, R, Z, O) = (M, Z, O)$ where

- $M = (S, A, P, s_0)$ is an MDP,
- Z is a finite set of observations,
- $O \colon S \times A \to \mathcal{D}(Z)$ is the probabilistic observation function.

Often we restrict to POMDPs with deterministic observations: $O \colon S \times A \to Z$.

Every POMDP with randomized observations can be transformed into a (larger) POMDP with deterministic observations.

Why POMDPs?

Rich framework with many realistic applications: robotics, healthcare, aircraft collision avoidance, ...

Partial observability is everywhere: sensors have imprecisions, vision is limited, ...

"We cannot avoid POMDPs, however, because the real world is one."

— from Artificial Intelligence: A Modern Approach by Peter Norvig and Stuart Russel

Paths through POMDPs

In an MDP, a path is a sequence of states and actions: $(s_0, a_0, s_1, a_1, \ldots) \in (S \times A)^*$.

Paths through POMDPs

In an MDP, a path is a sequence of states and actions: $(s_0, a_0, s_1, a_1, \ldots) \in (S \times A)^*$.

In a POMDP, the states cannot be observed, instead we have observations: $(z_0, a_0, z_1, a_1, \ldots) \in (Z \times A)^*$. This is called an observation sequence.

Paths through POMDPs

In an MDP, a path is a sequence of states and actions: $(s_0, a_0, s_1, a_1, \dots) \in (S \times A)^*$.

In a POMDP, the states cannot be observed, instead we have observations: $(z_0, a_0, z_1, a_1, \dots) \in (Z \times A)^*$. This is called an observation sequence.

Each observation may have multiple underlying states, and each state may have multiple observations (chosen probabilistically).

Solving (PO)MDP

The problems we are interested in for POMDPs are essentially the same as for MDPs:

Given a (PO)MDP M and a temporal logic or expected reward specification φ, compute a policy π such that $M^\pi \models \varphi$.

Solving (PO)MDP

The problems we are interested in for POMDPs are essentially the same as for MDPs:

Given a (PO)MDP M and a temporal logic or expected reward specification φ, compute a policy π such that $M^\pi \models \varphi$.

Theorem (Optimal policies for MDPs)
For MDPs with reachability or expected reward specifications, there exists an optimal deterministic memoryless policy $\pi : S \to A$.

Solving (PO)MDP

The problems we are interested in for POMDPs are essentially the same as for MDPs:

Given a (PO)MDP M and a temporal logic or expected reward specification φ, compute a policy π such that $M^\pi \models \varphi$.

Theorem (Optimal policies for MDPs)
For MDPs with reachability or expected reward specifications, there exists an optimal deterministic memoryless policy $\pi : S \to A$.

In POMDPs, however, a policy needs to be observation based, as we cannot see the states.

Optimal policies in POMDPs

For POMDPs, memoryless deterministic policies do not suffice.

We need (finite) memory observation-based policies: $\pi\colon (Z \times A)^* \times Z \to A$.

Key problem: trade-off between states with similar observations.

How much memory do we need?

Complexity of POMDPs

How much memory do we need?

Complexity of POMDPs

How much memory do we need? Possibly infinite.

Complexity of POMDPs

How much memory do we need? Possibly infinite.

Theorem (Complexity of POMDPs)
Computing an optimal policy (and the optimal value) for a quantitative specification in a POMDP is undecidable.

Complexity of POMDPs

Making the problem simpler helps (a little):

Theorem (Finite-horizon complexity of POMDPs)
Computing an optimal policy (and the optimal value) for a finite-horizon specification (i.e. maximize the reachability of T within k steps) in a POMDP is PSPACE-complete.

Theorem (Almost-sure complexity of POMDPs)
Computing an optimal policy for almost-sure reachability specifications ($\mathbb{P}_{=1}(\Diamond T)$) is EXPTIME-complete.

Definition (Belief state & belief update)
A belief state (or just belief) b is a distribution over states: $b \in \mathcal{D}(S)$.

Upon taking an action a and receiving an observation z, the agent updates their belief b to a new belief b' via the belief update $BU\colon \mathcal{D}(S) \times A \times Z \to \mathcal{D}(S)$:

$$\mathrm{BU}(b, a, z)(s') = \frac{O(s', a)(z) \cdot \sum_{s \in S} P(s, a)(s') \cdot b(s)}{\sum_{s'' \in S} O(s'', a)(z) \cdot \sum_{s \in S} P(s, a)(s'') \cdot b(s)}$$

Using belief states, a POMDP can be mapped to a continuous-state fully observable belief MDP.

Belief MDPs

Definition (Belief MDP)
For a POMDP (S, A, P, R, Z, O) we define the belief MDP as a tuple $(\mathcal{B}, A, \tau, \rho)$, where

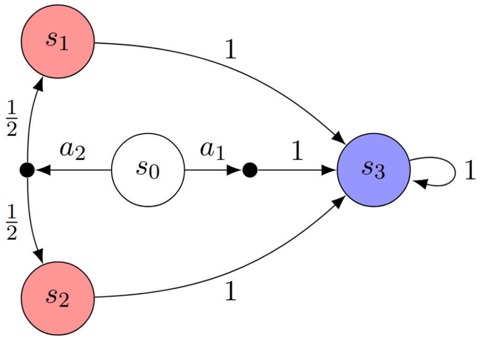
- $\mathcal{B}$ is the set of belief states: $\mathcal{B} = \mathcal{D}(S)$,
- A is the set of actions,
- τ is the transition function defined as $\tau(b, a)(b') =$

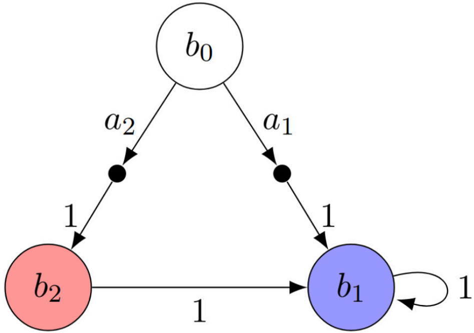
$$\sum_{z \in Z} \mathbf{1}[BU(b, a, z) = b'] \cdot \Big(\sum_{s'' \in S} O(s'', a)(z) \cdot \sum_{s \in S} P(s, a)(s'') \cdot b(s) \Big)$$

- $\rho \colon \mathcal{B} \times A \to \mathbb{R}_{\geq 0}$ is the reward function defined by

$$\rho(b, a) = \sum_{s \in S} b(s) \cdot R(s, a).$$

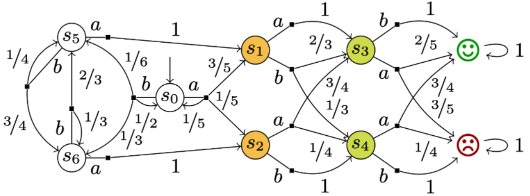
Belief MDP example (small)

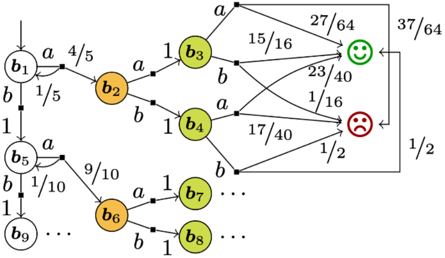




Where $b_0 = \{s_0 \mapsto 1\}$, $b_1 = \{s_3 \mapsto 1\}$, $b_2 = \{s_1 \mapsto 0.5, s_2 \mapsto 0.5\}$.

Belief MDP example (big)





b_1: $\{s_0 \mapsto 1\}$

b_2: $\{s_1 \mapsto 3/4, s_2 \mapsto 1/4\}$

b_3: $\{s_3 \mapsto 15/16, s_4 \mapsto 1/16\}$

b_4: $\{s_3 \mapsto 1/2, s_4 \mapsto 1/2\}$

b_5: $\{s_0 \mapsto 1/2, s_5 \mapsto 1/6, s_6 \mapsto 1/3\}$

b_6: $\{s_1 \mapsto 14/27, s_2 \mapsto 13/27\}$

b_7: $\{s_3 \mapsto 95/108, s_4 \mapsto 13/108\}$

b_8: $\{s_3 \mapsto 28/81, s_4 \mapsto 53/81\}$

b_9: $\{s_0 \mapsto 1/4, s_5 \mapsto 25/72, s_6 \mapsto 29/72\}$

Thanks to Alexander Bork, Sebastian Junges, Joost-Pieter Katoen, and Tim Quatmann

Belief-support MDP

Sometimes a simpler notion of belief is sufficient.

Belief-support MDP

Sometimes a simpler notion of belief is sufficient.

For almost-sure reachability (with probability 1), we do not care about the exact probabilities, only the graph.

The support of a belief b is the set of states $s \in S$ with $b(s) > 0$.

The belief-support MDP of a POMDP is the belief MDP, except with all possible supports as states instead of belief states.

Solving POMDPs

A POMDP is a continuous state (belief) MDP.

A belief is a sufficient statistic for the entire history (observation sequence) that has been generated so-far.

A belief-based policy $\pi\colon \mathcal{D}(S) \to A$ computed on the belief MDP is thus also a policy that maps observation sequences to actions $\pi\colon (Z \times A)^* \to A$.

Hence, computing a policy on the belief MDP also gives a policy for the POMDP.

Other classes of policies for POMDPs

Besides belief-based policies, we can use other classes of finite-memory policies that are easier to compute but may be sub-optimal.

Other classes of policies for POMDPs

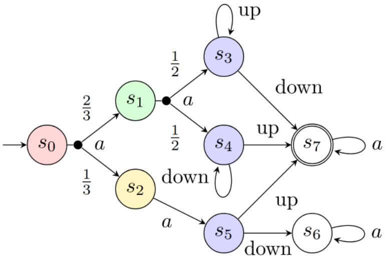
Besides belief-based policies, we can use other classes of finite-memory policies that are easier to compute but may be sub-optimal.

- Randomized finite-memory: $\pi\colon (Z \times A)^{k-1} \times Z \to \mathcal{D}(A)$,
- Deterministic finite-memory: $\pi\colon (Z \times A)^{k-1} \times Z \to A$,
- Randomized memoryless: $\pi\colon Z \to \mathcal{D}(A)$,
- Deterministic memoryless: $\pi\colon Z \to A$.

Even computing the simplest classes (memoryless) is still NP-hard.

POMDP policies: example

Find observation-based policies for $\mathbb{P}_{\mathsf{Max}}(\Diamond s_7)$ that are:

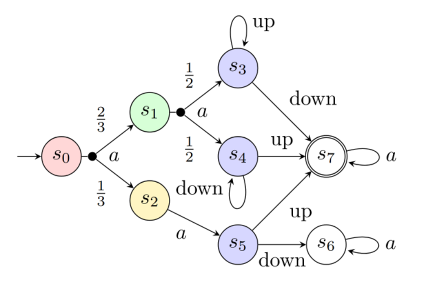


- Deterministic memoryless:
- Randomized memoryless:
  - Deterministic finite-memory:
  - Randomized finite-memory:

POMDP policies: example

Find observation-based policies for $\mathbb{P}_{\mathsf{Max}}(\Diamond s_7)$ that are:

- Deterministic memoryless: always choose up. Result: $\frac{2}{3}$.
- Randomized memoryless:

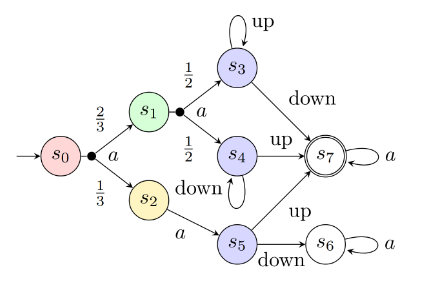


- Deterministic finite-memory:
- Randomized finite-memory:

POMDP policies: example

Find observation-based policies for $\mathbb{P}_{\mathsf{Max}}(\Diamond s_7)$ that are:

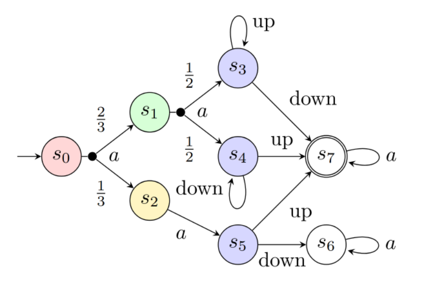
- Deterministic memoryless: always choose up. Result: $\frac{2}{3}$.
- Randomized memoryless: choose up with $1 - \epsilon$ and down with ϵ. Result: $1 - \epsilon$.
  - Deterministic finite-memory:
  - Randomized finite-memory:



POMDP policies: example

Find observation-based policies for $\mathbb{P}_{\mathsf{Max}}(\Diamond s_7)$ that are:

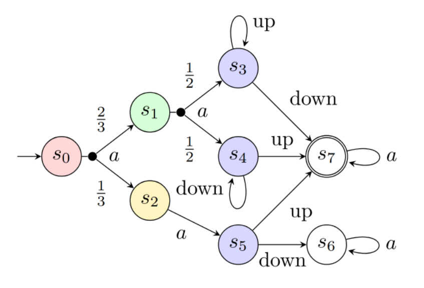
- Deterministic memoryless: always choose up. Result: $\frac{2}{3}$.
- Randomized memoryless: choose up with $1 - \epsilon$ and down with ϵ. Result: $1 - \epsilon$.



- Deterministic finite-memory: if we see yellow then blue choose up, otherwise if #blue is even choose up and down if uneven. Result: 1.
- Randomized finite-memory:

POMDP policies: example

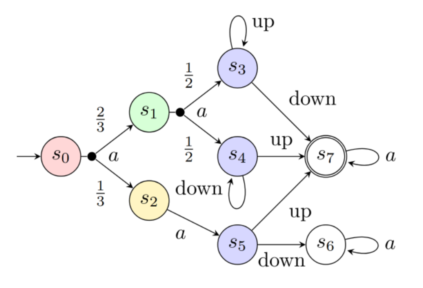
Find observation-based policies for $\mathbb{P}_{\mathsf{Max}}(\Diamond s_7)$ that are:



- Deterministic memoryless: always choose up. Result: $\frac{2}{3}$.
- Randomized memoryless: choose up with $1 - \epsilon$ and down with ϵ. Result: $1 - \epsilon$.
- Deterministic finite-memory: if we see yellow then blue choose up, otherwise if #blue is even choose up and down if uneven. Result: 1.
- Randomized finite-memory: if we see yellow then blue choose up, otherwise randomize up and down with 0.5. Result: 1.

POMDP policies: example

Find observation-based policies for $\mathbb{P}_{\mathsf{Max}}(\Diamond s_7)$ that are:



- Deterministic memoryless: always choose up. Result: $\frac{2}{3}$.
- Randomized memoryless: choose up with $1 - \epsilon$ and down with ϵ. Result: $1 - \epsilon$.
  - Deterministic finite-memory: if we see yellow then blue choose up, otherwise if #blue is even choose up and down if uneven. Result: 1.
  - Randomized finite-memory: if we see yellow then blue choose up, otherwise randomize up and down with 0.5. Result: 1.

In general: randomization can reduce the amount of memory needed!

Solving POMDPs: methods

Problem: most methods for computing policies (value iteration, point-based methods) operate on the belief MDP and do not scale.

We will now look into two approaches:

1. QMDP,
2. Parametric Markov chains.

QMDP

QMDP is an algorithm to find sub-optimal belief-based policies for a POMDP.

Key advantage: the algorithm is very simple.

QMDP

QMDP is an algorithm to find sub-optimal belief-based policies for a POMDP.

Key advantage: the algorithm is very simple.

Definition (QMDP Algorithm)

1. Find an optimal policy $\pi^*\colon S \to A$ for the underlying MDP of the POMDP.
2. For each belief $b \in \mathcal{D}(S)$, weigh the actions of π^* according to the belief:

$$\pi(b) = \sum_s b(s) \cdot \pi(s).$$

This yields a randomized belief-based policy π.

Small example: MDP policy $\pi^*\colon \{s_1 \mapsto a_1, s_2 \mapsto a_2\}$. Belief $b\colon \{s_1 \mapsto 0.8, s_2 \mapsto 0.2\}$, then the belief-based POMDP policy π is given by:

$$\pi(b) = 0.8 \cdot \pi(s_1) + 0.2 \cdot \pi(s_2) = \{a_1 \mapsto 0.8, a_2 \mapsto 0.2\}$$

POMDPs and Parametric Markov Chains

Finite-state controllers

We can encode finite-memory into the state-space of a POMDP.

To do that, use a finite-state controller.

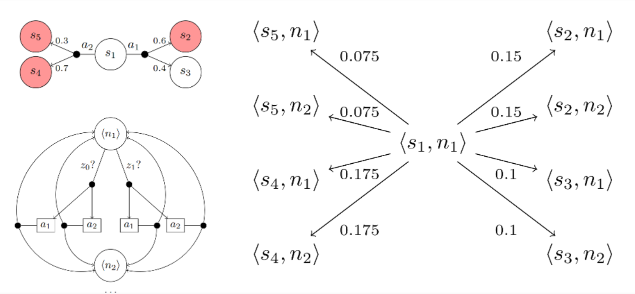
Definition (Finite-state controller (FSC))
A finite-state controller (FSC) is a tuple (N, n_I, γ, δ), where

- N is a set of memory nodes. $|N| = k$ means we have finite-memory of size k,
- $n_i \in N$ the initial memory node,
- $\gamma \colon N \times Z \to \mathcal{D}(A)$ is the action mapping,
- $\delta \colon N \times Z \times A \to \mathcal{D}(N)$ is the memory update function.

Encoding memory into a POMDP

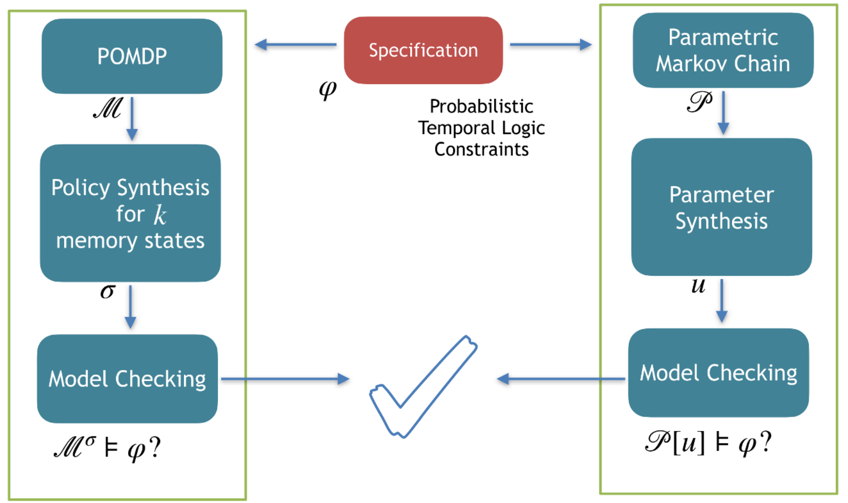
Given a POMDP and an FSC, compute the product POMDP.

A memoryless policy on this product then corresponds to a *k*-finite-memory policy for the original POMDP.



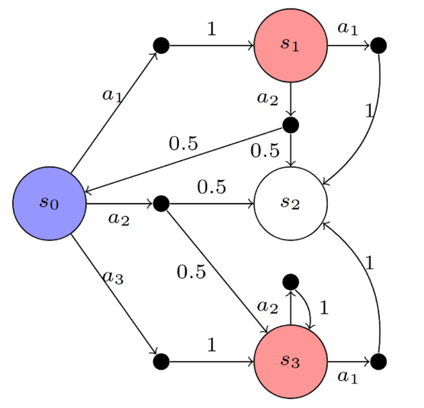
(Informal idea of product construction).

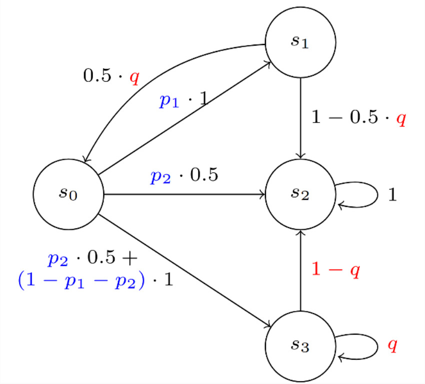
POMDPs $\Longleftrightarrow$ pMCs



POMDPs $\iff$ pMCs

Replace actions by parameters, states with the same observation get the same parameter.





POMDPs $\iff$ pMCs

Given the pMC, use parameter synthesis to find a valuation for the parameters.

POMDPs $\iff$ pMCs

Given the pMC, use parameter synthesis to find a valuation for the parameters.

Map the valuation back to the actions and you have a memoryless randomized policy!

Efficient convex optimization-based techniques for parameter synthesis make this approach fast and scalable.

Downside: need to specify the amount of memory beforehand.

Next Summary So far

POMDPs are very general models that capture a lot of scenarios and applications.

Hard to compute policies, but feasible, non-trivial, techniques exist.