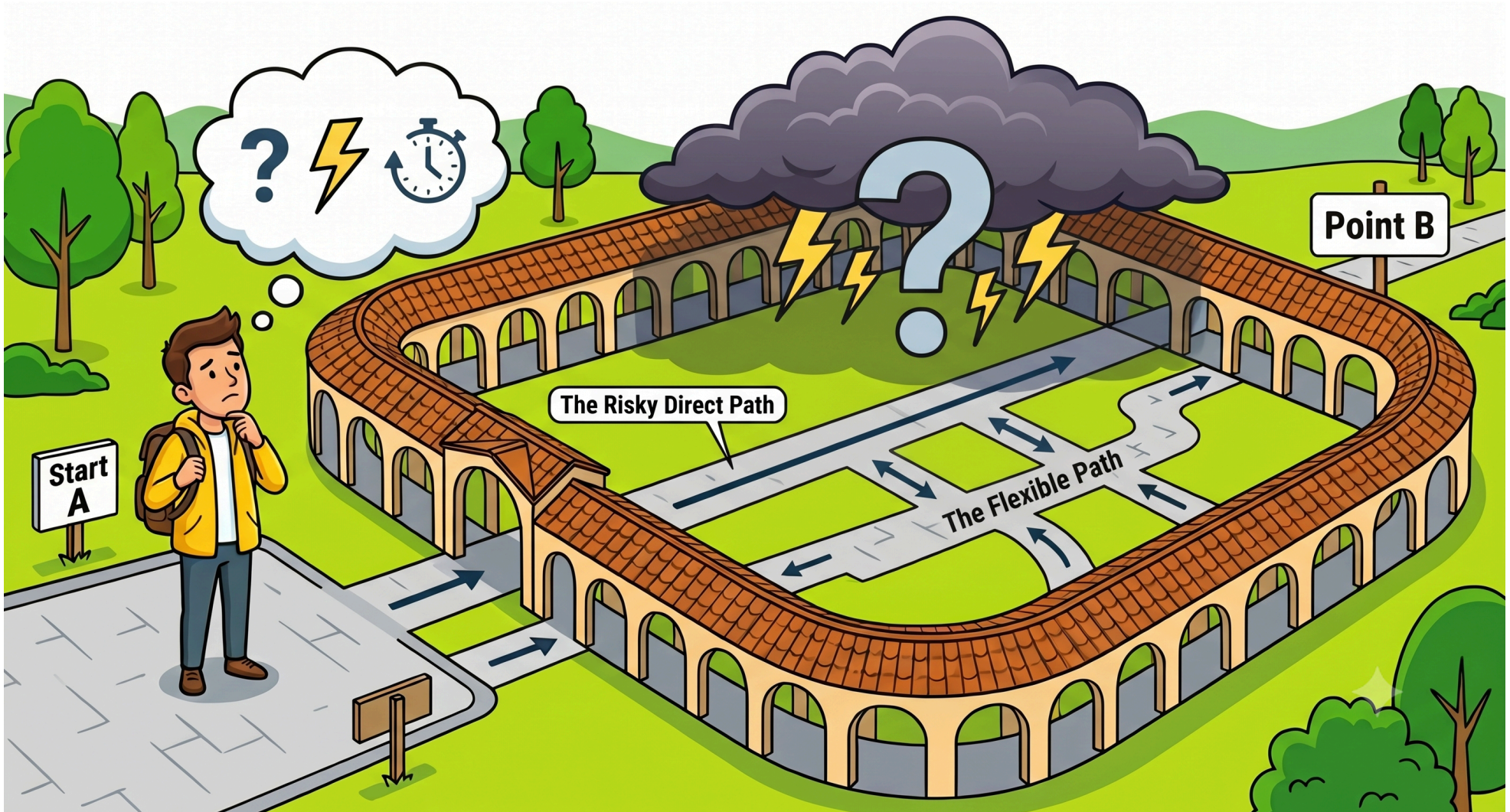
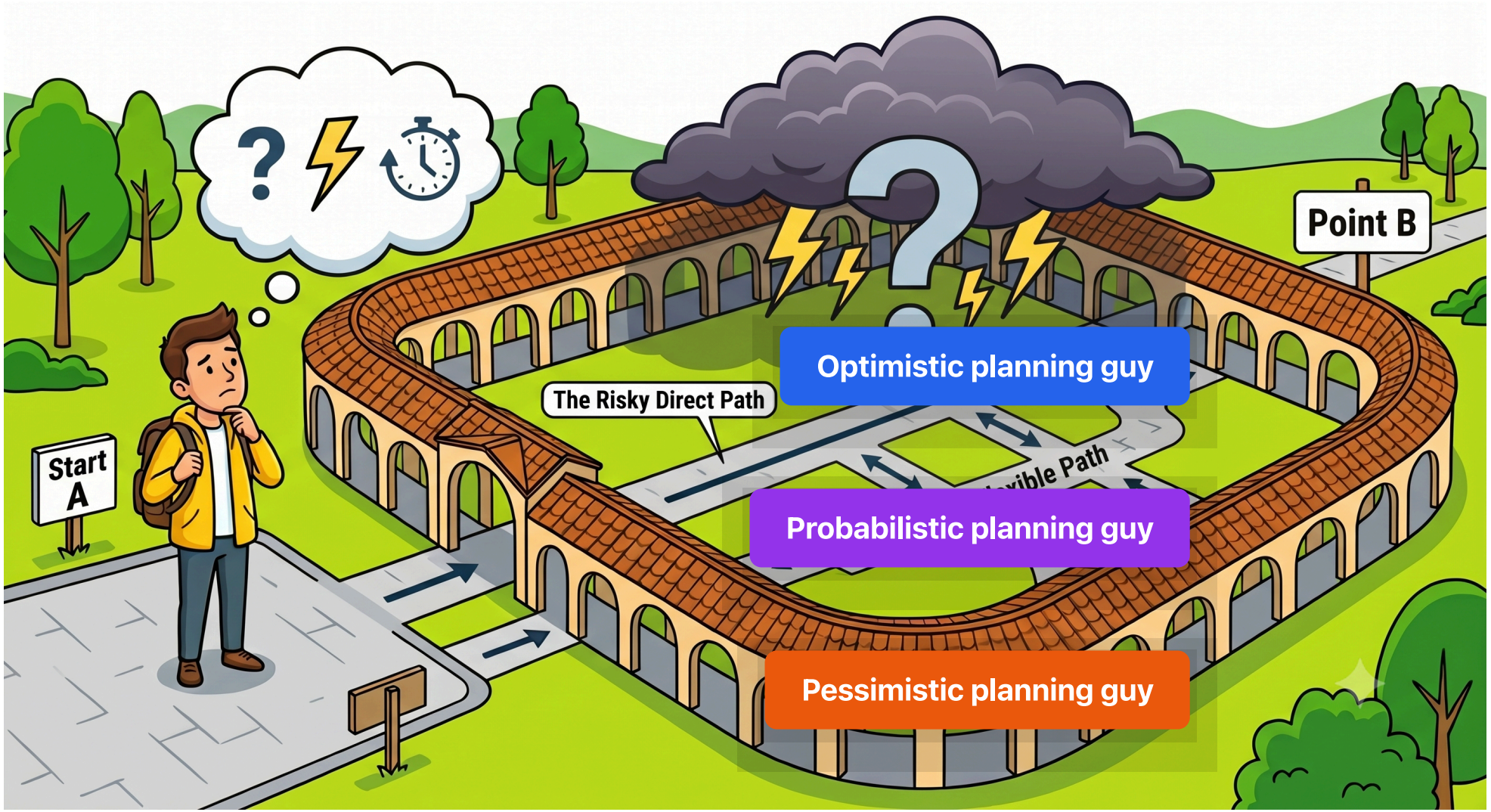
Probabilistic Planning

Models, Algorithms, and Applications

ICAPS 2026 Summer School

Florent Teichteil-Koenigsbuch, AIRBUS AI Research


Point B
The Risky Direct Path
The Flexible Path
Start
A


Point B
Optimistic planning guy
The Risky Direct Path
Probabilistic planning guy
Start
A
Pessimistic planning guy

Course Roadmap

Part 1: Foundations of Probabilistic Planning

From deterministic plans to optimal policies under uncertainty

What You Already Know

Classical AI planning in a nutshell

- **Deterministic transitions**: action a in state s yields exactly one successor s'
- **Full observability**: the agent always knows the current state
- **Solution**: an action sequence $\langle a_0, a_1, \ldots, a_n \rangle$ reaching goal $\mathcal{G}$
- **Algorithms**: A*, greedy best-first search, heuristics (h^{ff}, $h^{\max}$, landmarks)

> **Key property**
> Determinism makes planning equivalent to graph search — one fixed action sequence handles everything.

Ghallab, Nau & Traverso (2004). Automated Planning: Theory and Practice.

Live Demo: A* on a Maze

scikit-decide's Maze domain with live visualization

Deterministic vs. Stochastic Grid World

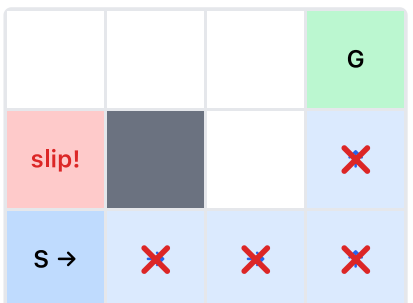
What happens when the world does not cooperate?

Deterministic model:

A fixed **action sequence** suffices:

$$\langle \text{Right}, \text{Right}, \text{Right}, \text{Up}, \text{Up} \rangle$$

Stochastic model (noisy actuators):



Outcome of `Right`	Prob
Intended direction	0.8
Slip perpendicular	0.1 each

⚠ **Key insight**

What Happens When A*'s Plan Meets Uncertainty?

Executing A*'s deterministic plan on the stochastic maze

Sources of Uncertainty in Planning

Where does randomness come from?

- **Stochastic action effects** — actions have multiple outcomes with known probabilities (e.g. robot movement, grasping)
- **Exogenous events** — the environment changes independently of the agent (e.g. weather, other agents)
- **Partial observability** — the agent cannot fully observe the current state; requires belief-state reasoning (POMDPs)

> ⚠ **This course**
> We focus on **stochastic effects** under **full observability** — the agent always knows which state it is in, but not which state it will transition to. We will briefly introduce POMDPs later.

The Planning Spectrum

From certainty to full uncertainty

Framework	Actions	Observability	Solution
Classical Planning	Deterministic	Full	Action sequence
Conformant Planning	Deterministic	None	Action sequence
Probabilistic Planning	Stochastic	Full	Policy
Contingent Planning	Non-det.	Partial	Conditional plan
POMDP	Stochastic	Partial	Belief-space policy

This course

This course covers **probabilistic planning** (MDPs, stochastic actions + full observability) and extends to **POMDPs** (partial observability, belief-space policies).

From Plans to Policies

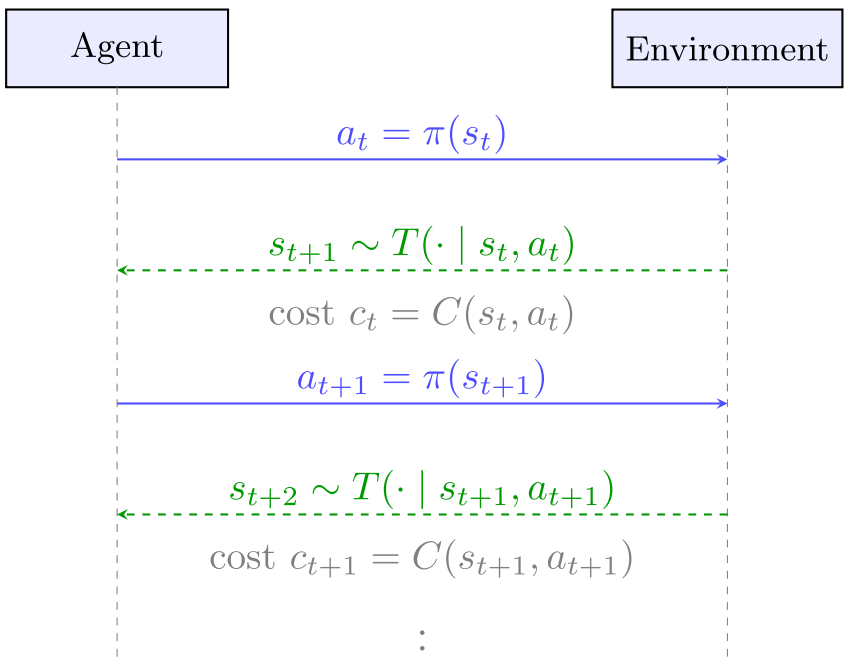
The fundamental shift in probabilistic planning

Markov Decision Processes

The mathematical foundation of probabilistic planning

Agent-Environment Interaction

The MDP loop



At each time step t:

1. Agent observes current state s_t
2. Agent selects action $a_t = \pi(s_t)$
3. Environment transitions to $s_{t+1} \sim T(\cdot \mid s_t, a_t)$
4. Agent incurs cost $C(s_t, a_t)$
5. Repeat: observe s_{t+1}, select $a_{t+1} = \pi(s_{t+1})$
6. Environment transitions again... and so on until goal

MDP: Formal Definition

Definition (Markov Decision Process)

An MDP is a tuple $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, T, C, s_0, \mathcal{G} \rangle$ where:

- $\mathcal{S}$ — finite set of **states**
- $\mathcal{A}$ — finite set of **actions**; $A(s) \subseteq \mathcal{A}$ denotes actions applicable in state s
- $T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ — **transition function**

$$T(s' \mid s, a) = \Pr(s_{t+1} = s' \mid s_t = s, a_t = a)$$

- $C : \mathcal{S} \times \mathcal{A} \to \mathbb{R}^+$ — **cost function** (non-negative)
- $s_0 \in \mathcal{S}$ — **initial state**
- $\mathcal{G} \subseteq \mathcal{S}$ — **goal states**

For all $s \in \mathcal{S}$ and $a \in A(s)$: $\sum_{s' \in \mathcal{S}} T(s' \mid s, a) = 1$

The Classic Grid World

Russell and Norvig's 4x3 grid

The Markov Property

Memorylessness as a structural assumption

Markov Property

The probability of transitioning to state s_{t+1} depends **only** on the current state s_t and action a_t, not on the history:

$$\Pr(s_{t+1} \mid s_t, a_t, s_{t-1}, a_{t-1}, \ldots, s_0, a_0) = \Pr(s_{t+1} \mid s_t, a_t) = T(s_{t+1} \mid s_t, a_t)$$

Why is this powerful?

- The current state is a **sufficient statistic** for predicting the future
- Policies can be **stationary**: $\pi(s)$ does not need to depend on time t
- Enables **dynamic programming**: the principle of optimality applies

Warning

Policy: Formal Definition

The solution concept for MDPs

Definition (Deterministic stationary policy)

A policy is a function $\pi : \mathcal{S} \to \mathcal{A}$ such that $\pi(s) \in A(s)\ \forall s \in \mathcal{S}$

Example policy for the grid world:

	Col 1	Col 2	Col 3	Col 4
Row 3	Right	Right	Right	**Goal**
Row 2	Up	Wall	Up	**Trap**
Row 1	Up	Left	Up	Left

Example

Evaluating a Policy

How good is a given policy?

Given policy π, the agent generates a random trajectory:

$$s_0 \xrightarrow{\pi(s_0)} s_1 \xrightarrow{\pi(s_1)} s_2 \xrightarrow{\pi(s_2)} \cdots$$

The **total cost** of one trajectory is: $\sum_{t=0}^{\infty} C(s_t, \pi(s_t))$

But trajectories are random! We care about the **expected** total cost.

Definition (Value function)

The value of state s under policy π is:

$$V^{\pi}(s) = \mathbb{E}^{\pi}\left[\sum_{t=0}^{\infty} C(s_t, \pi(s_t)) \quad s_0 = s\right]$$

with the convention $V^{\pi}(s) = 0$ for all $s \in \mathcal{G}$.

Bellman Expectation Equation

Recursive structure of the value function

✦ Bellman Expectation Equation

For any policy π and non-goal state $s \notin \mathcal{G}$:

$$\underbrace{V^\pi(s)}_{\text{total}} = \underbrace{C(s, \pi(s))}_{\text{immediate}} + \underbrace{\sum_{s'} T(s'|s, \pi(s)) \cdot V^\pi(s')}_{\text{expected future cost}}$$

This gives a **system of linear equations** — one per state — that can be solved to find V^π.

Optimal Value Function

The best achievable expected cost

Definition (Optimal value function)

The optimal value of state s is:

$$V^*(s) = \min_{\pi} V^{\pi}(s)$$

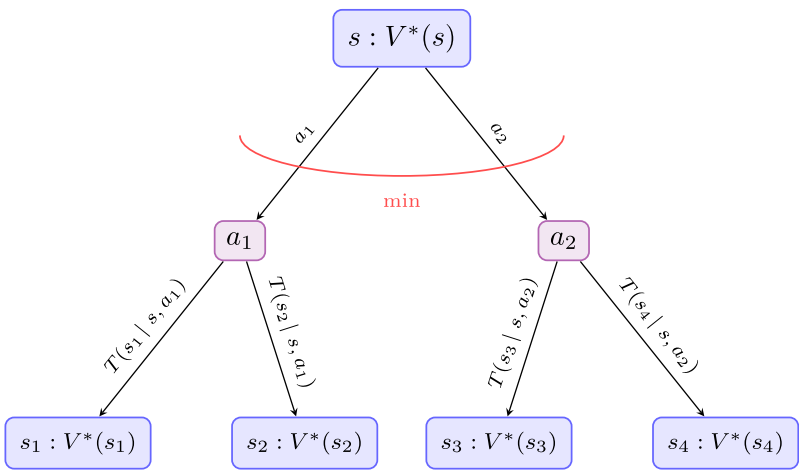
The optimal policy π^* achieves $V^*(s)$ simultaneously for **all** states s.

Bellman Optimality Equation

For all $s \notin \mathcal{G}$:

$$V^*(s) = \min_{a \in A(s)} \left[C(s,a) + \sum_{s'} T(s'|s,a) \cdot V^*(s') \right]$$

with $V^*(s) = 0$ for $s \in \mathcal{G}$.



$$V^* = \min\left(\mathrm{Q}(s,a_1),\ \mathrm{Q}(s,a_2)\right)$$

Key diff. from expectation equation: instead of following a fixed policy, we **minimize** over actions.

Q-Values and Policy Extraction

From values to actions

Definition (Optimal Q-function)

The Q-value of taking action a in state s and then acting optimally:

$$Q^*(s,a) = C(s,a) + \sum_{s' \in \mathcal{S}} T(s' \mid s,a) \cdot V^*(s')$$

The relationship between V^* and Q^*: $V^*(s) = \min_{a \in A(s)} Q^*(s,a)$

Greedy policy extraction

Given V^*, the optimal policy is:

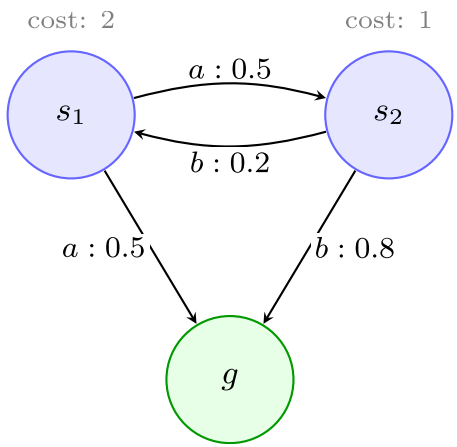
$$\pi^*(s) = \arg\min_{a \in A(s)} Q^*(s,a) = \arg\min_{a \in A(s)} \left[C(s,a) + \sum_{s' \in \mathcal{S}} T(s' \mid s,a) \cdot V^*(s') \right]$$

Worked Example: A Tiny MDP

Computing V^* step by step

3-state MDP

$\mathcal{S} = \{s_1, s_2, g\}$, $\mathcal{G} = \{g\}$, one action per state.



Bellman equations:

$$V^*(s_1) = 2 + 0.5 \cdot V^*(s_2) + 0.5 \cdot 0$$

$$V^*(s_2) = 1 + 0.8 \cdot 0 + 0.2 \cdot V^*(s_1)$$

Solving: $V^*(s_1) = \frac{25}{9} \approx 2.78$, $V^*(s_2) = \frac{59}{45} \approx 1.56$

Discounted MDPs

An alternative convergence mechanism

Discounted Bellman Optimality

For $\gamma \in [0, 1)$:

$$V^*(s) = \min_a \left[C(s,a) + \gamma \sum_{s'} T(s'|s,a) V^*(s') \right]$$

Why discount?

- Guarantees convergence without goal states ($\gamma^t \to 0$)
- Models "time value" of costs
- Bounds V^*: $V^*(s) \leq C_{\max}/(1-\gamma)$

For goal-oriented planning

Discounting can be unnatural — it says reaching the goal in 100 steps is "almost free" if γ is small. We often prefer the **undiscounted** ($\gamma = 1$) setting with goal-based termination instead.

	Discounted ($\gamma < 1$)	**Undiscounted** ($\gamma = 1$)
Convergence	Always (geometric decay)	Needs proper policies
Goal states	Optional	Required
Natural for	RL, continuing tasks	Goal-oriented planning

$\Rightarrow$ We use **undiscounted** ($\gamma = 1$) for the rest of this course.

Stochastic Shortest Path Problems

The right MDP framework for goal-oriented planning

SSP: Formal Definition

The right framework for goal-oriented probabilistic planning

With $\gamma = 1$, costs may diverge and some policies never reach the goal. SSPs add structural assumptions that resolve both issues.

Definition (Stochastic Shortest Path)

An SSP is an MDP $\langle \mathcal{S}, \mathcal{A}, T, C, s_0, \mathcal{G} \rangle$ satisfying:

1. **Goal states are absorbing and cost-free:**

$$\forall g \in \mathcal{G}, \forall a \in A(g) : \; T(g \mid g, a) = 1 \text{ and } C(g, a) = 0$$

2. **Positive costs outside the goal:**

$$\forall s \notin \mathcal{G}, \forall a \in A(s) : \; C(s, a) > 0$$

3. **Proper policy existence:**

$$\exists \text{ policy } \pi \text{ s.t. } \Pr(\exists t \geq 0 : s_t \in \mathcal{G} \mid \pi, s_0 = s) = 1 \;\; \forall s \in \mathcal{S}$$

SSP generalizes classical planning

Classical planning is the special case where $T(s' \mid s, a) \in \{0, 1\}$.

Bertsekas & Tsitsiklis (1991). An Analysis of Stochastic Shortest Path Problems.

Proper vs. Improper Policies

A critical distinction

Definition

Proper policy: π is proper if $\Pr(\text{reach } \mathcal{G} \mid \pi, s_0 = s) = 1$ for all $s \in \mathcal{S}$.

Improper policy: there exists some state s from which π fails to reach $\mathcal{G}$ with probability 1.

Proper policy example:

- Always move toward the goal
- Might take many steps due to stochastic slips
- But **eventually** reaches the goal (prob 1)
- $V^\pi(s) < \infty$ for all s

Improper policy example:

- Gets stuck in a cycle with positive probability
- Example: always go Left on the left wall
- $V^\pi(s) = \infty$ for some s
- Accumulates cost forever

Theorem

Proper vs. Improper: Maze Example

Same maze, two different policies — stochastic transitions: 0.8 intended, 0.1 each perpendicular

SSP: Fundamental Theorem

Existence and uniqueness of V^*

✦ **Theorem (Bertsekas & Tsitsiklis, 1991)**

Under SSP assumptions:

1. The optimal value function V^* is the **unique** solution to the Bellman equation:

$$V^*(s) = \min_{a \in A(s)} \left[C(s,a) + \sum_{s' \in \mathcal{S}} T(s' \mid s,a) \cdot V^*(s') \right]$$

2. There exists a **deterministic stationary optimal policy** π^*.

3. The optimal policy is **proper**: it reaches $\mathcal{G}$ with probability 1.

4. **Value Iteration** and **Policy Iteration** both converge to V^*.

This theorem is the theoretical foundation for all the algorithms we will study in Part 2.

When SSP Assumptions Break

Dead-ends and the connection to classical planning

What if no proper policy exists?

⚠ Unavoidable dead-ends

- States from which **no policy** can reach the goal (e.g. a robot falls into a pit)
- SSP assumption violated: $V^*(s) = \infty$ for dead-end states
- Standard VI/PI may **not converge** to V^*

📖 Generalizing classical dead-ends

- **Classical planning**: dead-ends = states with **no applicable action sequence** to the goal
- **SSPs**: dead-ends = states where **no policy can reach the goal with probability 1**
- Even if actions exist, stochastic transitions may make the goal unreachable

💡 We will address this in Part 3

- Dead-end detection
- MAXPROB objective (maximize probability of reaching the goal)
- FRET framework

Partially Observable MDPs

When the agent cannot observe the full state

POMDP

A **POMDP** extends an MDP with:

- Ω — finite set of **observations**
- $O(o \mid s', a)$ — **observation function**: probability of observing o after reaching state s' via action a

Belief state $b \in \Delta(\mathcal{S})$: probability distribution over states

$$b'(s') = \eta \cdot O(o \mid s', a) \sum_{s} T(s' \mid s, a) \cdot b(s)$$

Key insight

A POMDP is a **belief-space MDP**: the state is the belief b, actions update beliefs via Bayes' rule. The belief space is continuous even when $\mathcal{S}$ is finite — this is what makes POMDPs computationally hard (PSPACE-complete).

POMDP Example: Noisy Sensor Grid

The Probabilistic Planning Landscape

A roadmap for the rest of the course

Part 1: Key Takeaways

What to remember going forward

1. Uncertainty demands policies, not plans.

Policies $\pi : \mathcal{S} \to \mathcal{A}$ replace fixed action sequences.

2. MDPs are the mathematical foundation.

$\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, T, C, s_0, \mathcal{G} \rangle$; Markov property enables DP.

3. The Bellman equation is the central tool.

$$V^*(s) = \min_a \left[C(s,a) + \sum_{s'} T(s'|s,a) \cdot V^*(s') \right]$$

4. SSP is the right framework for planning.

Undiscounted, proper policies, positive costs, unique V^*.

5. POMDPs extend MDPs to partial observability.

Belief-space planning; practical solvers exist (POMCP, Goal-HSVI).

Next up

Part 2 — Algorithms: Value Iteration, Policy Iteration, LAO*, LRTDP, UCT.

Part 2: Algorithms for Probabilistic Planning

Dynamic Programming, Heuristic Search, and Scalable Solvers

From Equations to Algorithms

We established in Part 1 the **Bellman optimality equation**:

$$V^*(s) = \min_{a \in A(s)} \left[C(s,a) + \sum_{s' \in \mathcal{S}} T(s'|s,a)\, V^*(s') \right]$$

This is a system of $|\mathcal{S}|$ simultaneous equations — generally **nonlinear** (due to the $\min$).

How do we solve it?

Two classical families of approaches:

1. **Value Iteration** — iteratively refine value estimates
2. **Policy Iteration** — iteratively improve policies

Both are forms of **dynamic programming** (Bellman, 1957).

The Bellman Backup Operator

Definition

The **Bellman backup operator** $\mathcal{B}$ maps a value function V to a new value function $\mathcal{B}[V]$:

$$\mathcal{B}[V](s) = \min_{a \in A(s)} \left[C(s,a) + \sum_{s' \in \mathcal{S}} T(s'|s,a)\, V(s') \right]$$

Key properties:

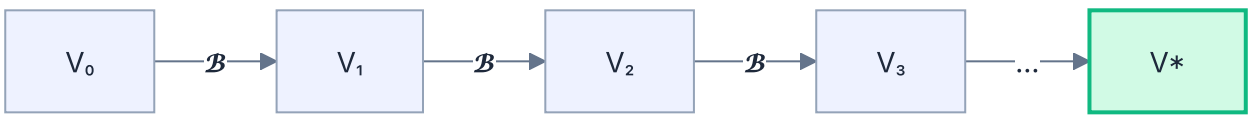
- V^* is a **fixed point**: $\mathcal{B}[V^*] = V^*$
- $\mathcal{B}$ is a **contraction** (for discounted MDPs): $\|\mathcal{B}[V] - \mathcal{B}[U]\|_\infty \leq \gamma \|V - U\|_\infty$
- For SSPs under proper policy assumptions, similar contraction properties hold

By Banach's fixed point theorem, repeatedly applying $\mathcal{B}$ converges to V^*.

Value Iteration: The Idea

Core idea: Start with an initial guess V_0 and repeatedly apply the Bellman backup:

$$V_{n+1}(s) = \mathcal{B}[V_n](s) = \min_{a \in A(s)} \left[C(s,a) + \sum_{s'} T(s'|s,a)\, V_n(s') \right]$$



- Initialize $V_0(s) = 0$ for all s (or use an admissible heuristic)
- Set $V(s) = 0$ for all goal states $s \in G$ (SSP setting)
- Stop when $\max_s |V_{n+1}(s) - V_n(s)| < \varepsilon$

Value Iteration: Algorithm & Convergence

{} Algorithm: Value Iteration (VI)

Input: MDP $\langle \mathcal{S}, \mathcal{A}, T, C, G \rangle$, tolerance $\varepsilon > 0$

1. **Initialize:** $V_0(s) \leftarrow 0$ for all $s \in \mathcal{S}$; $n \leftarrow 0$
2. **Repeat:**
   - For each $s \in \mathcal{S} \setminus G$:
     - $V_{n+1}(s) \leftarrow \min_{a \in A(s)} \left[C(s,a) + \sum_{s'} T(s'|s,a)\, V_n(s') \right]$
   - $n \leftarrow n + 1$
3. **Until** $\max_s |V_n(s) - V_{n-1}(s)| < \varepsilon$
4. **Return** greedy policy: $\pi(s) = \arg\min_{a \in A(s)} \left[C(s,a) + \sum_{s'} T(s'|s,a)\, V_n(s') \right]$

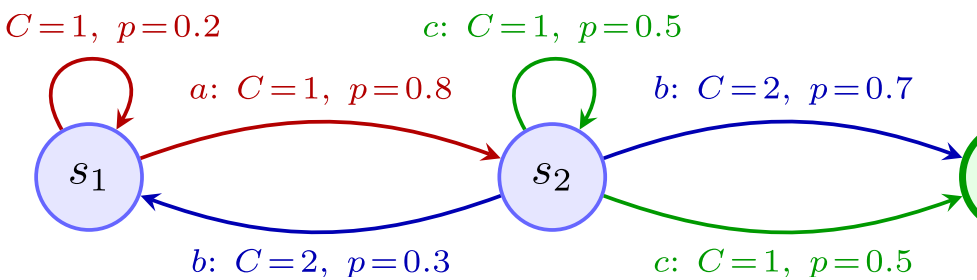
✦ Convergence (Contraction Theorem)

Live Demo: Value Iteration

Watch VI sweep through all states and converge to V^*

VI: Worked Example

Consider a 3-state SSP:



Iter	$V(s_1)$	$V(s_2)$	$V(G)$
n=0	0	0	0
1	1.00	1.00	0
n=2	1.80	1.50	0
n=3	2.20	1.75	0
...	...	...	0
V^*	**3.25**	**2.00**	0

Example

Solving $V^* = \mathcal{B}[V^*]$ at s_2:

Action c gives $V^*(s_2) = 1 + 0.5 \cdot 0 + 0.5 \cdot V^*(s_2)$, so $V^*(s_2) = 2$.

Action b gives $Q = 2 + 0.3 \cdot V^*(s_1) \geq 2$, so c is optimal.

Value Iteration: Complexity & Practical Variants

Per-iteration cost:

- For each state s: evaluate all actions, each requiring a sum over successor states
- Cost per state: $O(|\mathcal{A}| \cdot |\mathcal{S}|)$ | Total per iteration: $O(|\mathcal{S}|^2 \cdot |\mathcal{A}|)$
- Total: $O\left(\frac{|\mathcal{S}|^2 \cdot |\mathcal{A}| \cdot \log(1/\varepsilon)}{\log(1/\gamma)}\right)$

> ⚠ **The bottleneck**
> VI updates **all states** at every iteration, even states that are irrelevant to the agent's initial state.

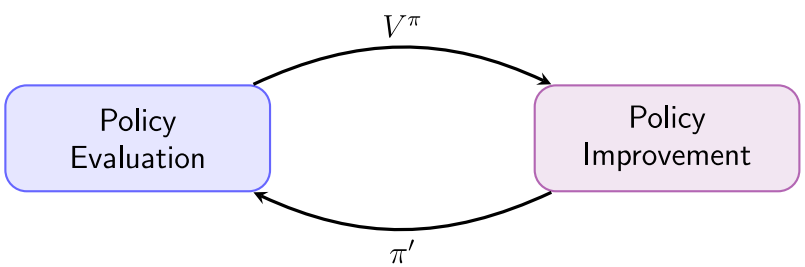
Practical variants:

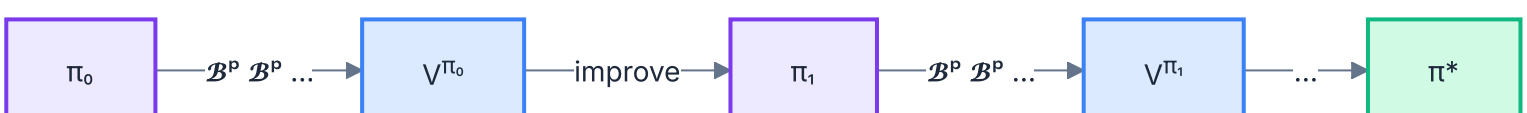
- **Gauss-Seidel (asynchronous) updates:** use updated values immediately within the same iteration — often converges faster
- **Prioritized sweeping:** update states with the largest **Bellman residual** first
- **Topological VI:** decompose the MDP into strongly connected components, solve in reverse topological order

Policy Iteration: Overview

Different philosophy: instead of iterating on values, iterate on **policies**.

Two alternating steps:





1. **Policy Evaluation:** Given policy π, compute V^π exactly
2. **Policy Improvement:** Given V^π, find a better policy π'

Repeat until the policy no longer changes.

Policy Evaluation & Policy Improvement

Policy Evaluation: Given a fixed policy π, the value function V^π satisfies a **linear system** (no $\min$):

$$V^\pi(s) = C(s, \pi(s)) + \sum_{s' \in \mathcal{S}} T(s'|s, \pi(s))\, V^\pi(s') \quad \iff \quad (I - T_\pi) \cdot V^\pi = C_\pi$$

Solve via Gaussian elimination in $O(|\mathcal{S}|^3)$, or iteratively.

Policy Improvement: Given V^π, construct an improved policy:

$$\pi'(s) = \arg\min_{a \in A(s)} \left[C(s, a) + \sum_{s'} T(s'|s, a)\, V^\pi(s') \right]$$

✦ **Policy Improvement Theorem**

If $\pi' \neq \pi$, then $V^{\pi'}(s) \leq V^\pi(s)$ for all $s \in \mathcal{S}$, with strict inequality for at least one state. Each iteration produces a **strictly better** policy (or we've converged).

Policy Iteration: Algorithm & Convergence

{} Algorithm: Policy Iteration (PI)

Input: MDP $\langle \mathcal{S}, \mathcal{A}, T, C, G \rangle$

1. **Initialize:** π_0 arbitrarily (e.g., random proper policy); $n \leftarrow 0$
2. **Repeat:**
   - **Policy Evaluation:** Solve for V^{π_n} via $(I - T_{\pi_n})\, V^{\pi_n} = C_{\pi_n}$
   - **Policy Improvement:** For each $s \in \mathcal{S} \setminus G$:
     - $\pi_{n+1}(s) \leftarrow \arg\min_{a \in A(s)} \left[C(s,a) + \sum_{s'} T(s'|s,a)\, V^{\pi_n}(s') \right]$
   - $n \leftarrow n + 1$
3. **Until** $\pi_n = \pi_{n-1}$
4. **Return** π_n, V^{π_n}

✦ Convergence

PI converges in at most $|\mathcal{A}|^{|\mathcal{S}|}$ iterations (finitely many deterministic policies, each visited at most once). In practice: typically **fewer than 10 iterations** regardless of $|\mathcal{S}|$. Each iteration costs $O(|\mathcal{S}|^3 + |\mathcal{S}|^2 \cdot |\mathcal{A}|)$.

Live Demo: Policy Iteration

Watch PI alternate between evaluation and improvement

Value Iteration vs Policy Iteration

	Value Iteration	Policy Iteration
Per iteration	$O(\|S\|^2 \cdot \|A\|)$	$O(\|S\|^3 + \|S\|^2 \cdot \|A\|)$
# iterations	Many (geometric rate)	Few (≤ 10 typical)
Convergence	ε-optimal	Exact (policy match)
State space	Can handle larger $\|S\|$	Limited by linear system
Sweet spot	Large $\|S\|$, simple T	Small-to-medium $\|S\|$

Modified Policy Iteration

A practical hybrid — perform policy evaluation approximately (a few VI sweeps instead of exact solve), then improve. Interpolates between VI and PI.

LP and MILP Formulations for MDPs

A third approach: mathematical programming

LP formulation (dual)

The optimal value function V^* can be computed via a **linear program** on occupation measures:

$$\min \sum_{s,a} C(s,a) \cdot x(s,a) \quad \text{s.t.} \quad \sum_{a} x(s,a) - \sum_{s',a'} T(s|s',a') \cdot x(s',a') = \alpha(s)$$

where $x(s,a) \geq 0$ are occupation measures and α is the initial state distribution.

Why LP/MILP matters:

- Provides an alternative to VI/PI with **different computational trade-offs**
- Dual variables give the optimal value function directly
- **Extends naturally** to constrained MDPs (add linear constraints on occupation measures)
- MILP extensions handle integer constraints, indicator variables, and combinatorial structure

Connection to iDual

The **iDual** algorithm (covered later) uses iterative LP relaxations as admissible heuristics for heuristic search — combining LP duality with the search-based approach.

Puterman (1994). Markov Decision Processes. Wiley. Chapter 6.

The Curse of Dimensionality

Both VI and PI enumerate **all states** — but state spaces grow **exponentially**.

$$|\mathcal{S}| = |D_1| \times |D_2| \times \ldots \times |D_n|$$

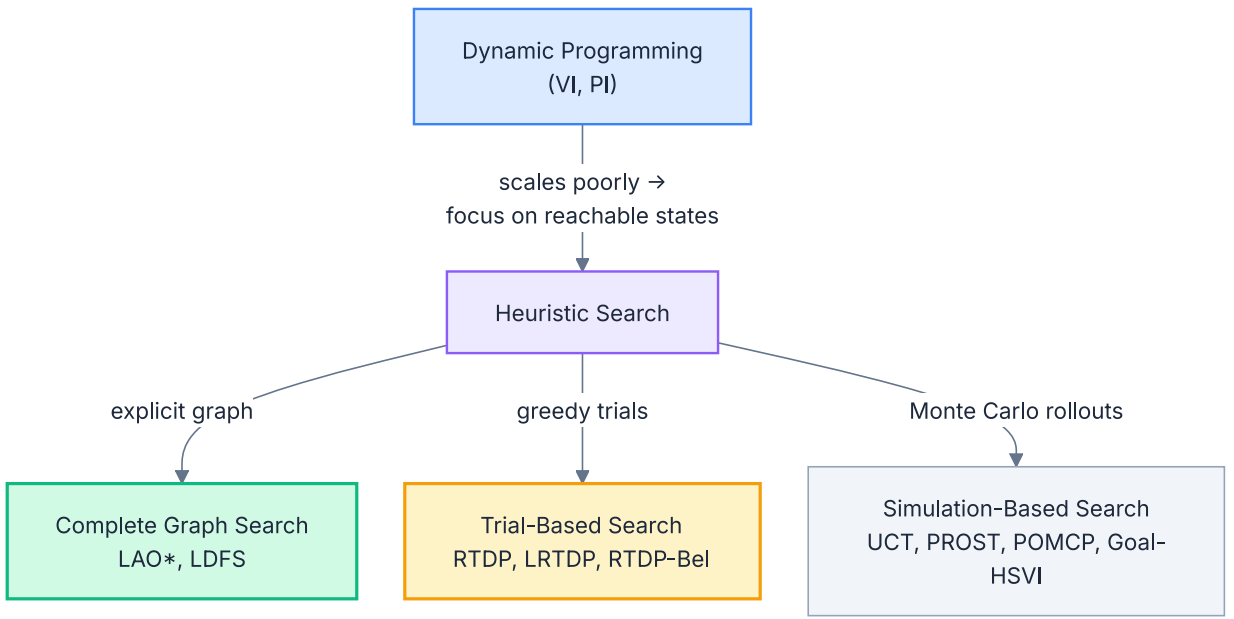
Example

A logistics domain with 10 packages, each in one of 5 locations: $|\mathcal{S}| = 5^{10} \approx 10^7$ states. Add 5 trucks with 3 states each: $|\mathcal{S}| = 5^{10} \times 3^5 \approx 2.4 \times 10^9$.

Key observation

We only need to solve the MDP for states **reachable from** s_0 under an optimal policy. Most states are **unreachable** from the initial state!

Two Families of Heuristic Search



Three categories of heuristic search

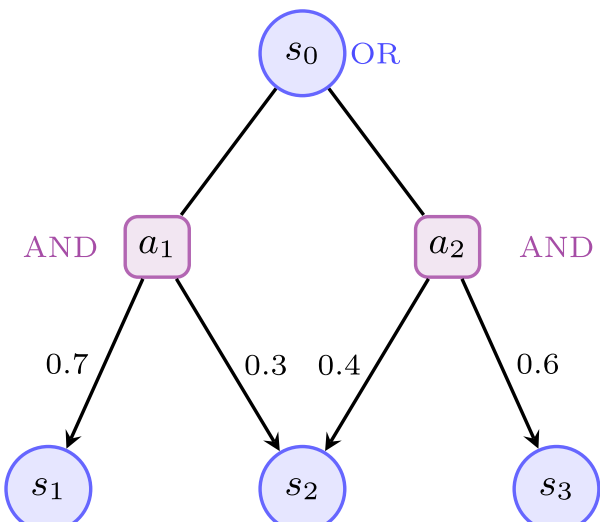
- **Complete graph search** (LAO∗, LDFS): build an explicit graph of reachable states, solve the sub-MDP. Offline, optimal, but must store the graph.
- **Trial-based search** (RTDP, LRTDP, RTDP-Bel): simulate greedy trials from s_0, performing Bellman backups along the way. Offline, optimal, memory-efficient.
- **Simulation-based search** (UCT, PROST, POMCP, Goal-HSVI): Monte Carlo rollouts or point-based updates from the current state. Online, anytime, scales to huge/continuous spaces and POMDPs.

Family 1: Complete Graph Search

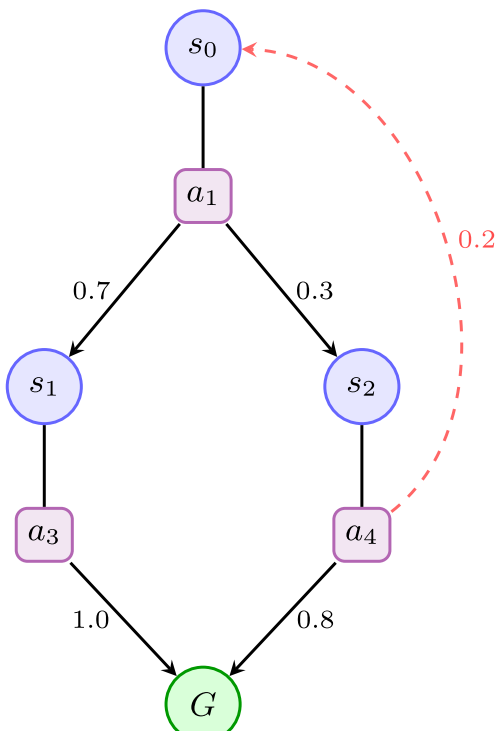
LAO* and LDFS — building and solving the reachable sub-MDP

MDPs as AND/OR Graphs

MDPs have a natural representation as **AND/OR graphs**: **OR nodes** = states (agent *chooses*), **AND nodes** = actions (nature *chooses* outcome).



A **solution graph** (= policy) selects **one** action at each OR node and includes **all** outcomes at each AND node. Solution graphs may contain **cycles**.



LAO*: Handling Cyclic Solutions

LAO∗ extends AO* to handle **cyclic** AND/OR graphs.

From AO* to LAO∗

- **AO∗** (Nilsson, 1980): searches AND/OR graphs, requires **acyclic** solutions, propagates costs bottom-up
- **LAO∗**: extends AO* to **cyclic** solutions by replacing bottom-up propagation with **VI on the envelope** (current sub-MDP)

LAO*: Algorithm

LAO*: Properties

✦ Theorem

LAO* with an admissible heuristic h converges to the **optimal** solution V^* for all states in the solution graph.

Key properties:

- Explores only states **reachable** from s_0 under the evolving best policy
- Handles **cyclic** solution graphs correctly via VI on the envelope
- With a good heuristic, explores far fewer states than full VI

Variants:

- **ILAO*:** run a single VI pass (not to convergence) in the update step — faster per iteration, more iterations total
- **RLAO*:** reverse the expansion order for some problems

⚠ Cost of VI on envelope

As the envelope grows, VI becomes expensive. The envelope can eventually include many states.

Live Demo: iLAO* (Complete Graph Search)

Watch LAO* build a solution graph

LDFS: Labeled Depth-First Search

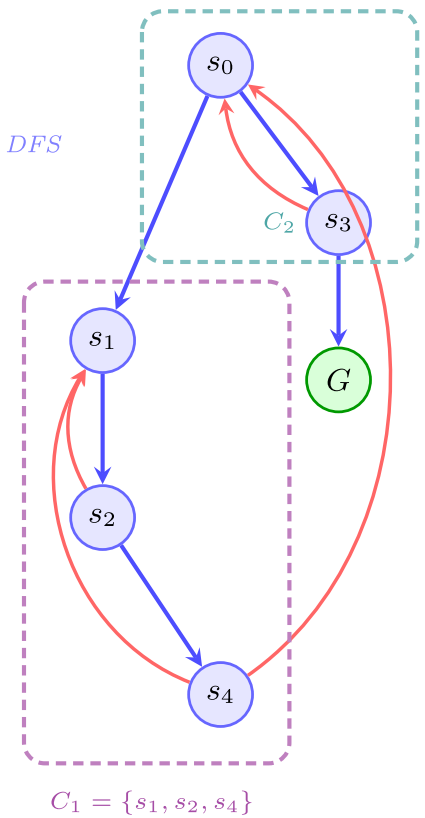
Complete graph search with DFS exploration

{} Algorithm: LDFS (for MDPs)

- **Repeat** $\mathrm{LDFS}(s_0)$ until s_0 is solved:
  - If s solved/terminal: return *true*. If s active (on DFS stack): return *false*
  - Mark s **active**; push with $s.\mathrm{idx} = s.\mathrm{low} = \mathrm{index{+}{+}}$
  - Pick greedy a with $Q_V(a, s) - V(s) \leq \epsilon$; recurse on $s' \in F(a, s)$, update $s.\mathrm{low} \leftarrow \min(s.\mathrm{low}, s'.\mathrm{low})$
  - If inconsistency found: backup $V(s) \leftarrow \min_a Q_V(a, s)$
  - If $s.\mathrm{low} = s.\mathrm{idx}$ and all consistent: **label entire SCC solved**

✦ SCC-based labeling

MDP policies can have **cycles**, so per-state labeling fails — **Tarjan's SCC algorithm** labels entire components as solved when all states have residual $\leq \epsilon$. Unlike LRTDP's random trials, LDFS explores **systematically** via DFS — more efficient on deep policy branches, but sensitive to real-valued costs.



Live Demo: LDFS

Watch LDFS systematically explore via depth-first search

Trial-Based Search: The Bridge

RTDP and LRTDP — simulating greedy trials with convergence detection

Real-Time Dynamic Programming (RTDP)

RTDP takes a completely different approach: **trial-based** simulation.

Labeled RTDP (LRTDP)

LRTDP = RTDP + **convergence labeling**

{} Algorithm: LRTDP

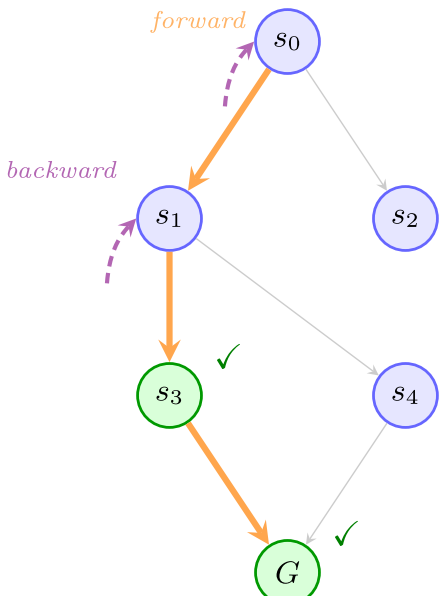
- **Init:** $V(s) \leftarrow h(s)$; mark only goals as solved
- **Repeat** from $s \leftarrow s_0$ **until** s_0 solved:
  - **Forward:** push s, backup $V(s)$, move to $s' \sim T(\cdot \mid s, a^*)$; skip solved/goal
  - **Backward:** pop visited; $\mathrm{CheckSolved}(s, \varepsilon)$ — stop on first failure

✦ CheckSolved(s, ε)

DFS over the **greedy envelope** of s. If all reachable states have residual $\leq \varepsilon$: label them **all solved** at once. Otherwise: Bellman-update visited states. Each call either labels states or increases some value by $> \varepsilon$, guaranteeing finite convergence.

Properties

Converges to V^* with admissible h. Solved states are **never revisited** — trials shorten over time. Simpler than LAO* (no explicit graph). One of the most widely used SSP algorithms.



Live Demo: (L)RTDP

RTDP-Bel: RTDP for POMDPs

Extending trial-based search to belief spaces

{} RTDP-Bel

Extends RTDP to belief-space MDPs (POMDPs): - State = belief vector b in Delta(S) - Actions update beliefs via Bayes' rule after observing o - Bellman backups on belief points - Trials simulate belief trajectories: b_0 → b_1 → ...

Key challenges:

- Continuous belief space – uses a **set of belief points** as representatives
- Value function represented as collection of alpha-vectors
- Interpolates value between nearest belief points

✦ Bridge to POMDPs

RTDP-Bel shows that trial-based MDP methods extend naturally to partial observability. It bridges the gap between MDP algorithms and the POMDP solvers (POMCP, Goal-HSVI) we'll see next.

Bonet & Geffner (2009). Solving POMDPs: RTDP-Bel vs. Point-based Algorithms.

Live Demo: RTDP-Bel on a POMDP Maze

Heuristics for Probabilistic Planning

The quality of the heuristic h is crucial for heuristic search performance.

Admissible heuristic

$h(s) \leq V^*(s)$ for all $s \in \mathcal{S}$. Required by LAO$*$, RTDP, and LRTDP for optimality guarantees.

The determinization approach (= min-min heuristic, Bonet & Geffner 2005)**:**

1. **All-outcomes determinization**: create one deterministic action per stochastic outcome
2. Solve with a classical planner (e.g., FF, LAMA) → plan cost = $h^{det}(s)$

Theorem

$h^{det}(s)$ is **admissible** — the deterministic problem is a relaxation of the MDP. But it searches in the original state space, which is expensive.

Short-Sighted SSPs

Solving SSPs by decomposition into bounded subproblems

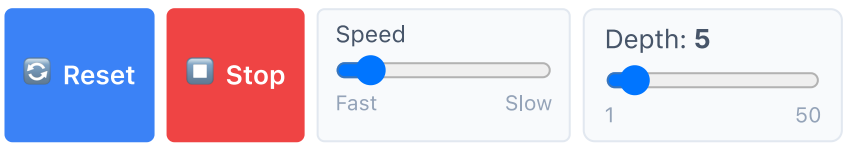
Live Demo: Short-Sighted SSP

Watch SSiPP solve via bounded-depth subproblems

```python
# algorithm: ssipp
# speed: 300
# depth: 5
# SSiPP on a stochastic maze (ice & traps)
# Solves via bounded-depth sub-SSPs
# Blue = explored, arrows = policy
from skdecide.hub.solver.ssipp import SSiPP
from stochastic_maze import StochasticMaze
from skdecide.core import Value

maze = StochasticMaze()
with SSiPP(
    domain_factory=lambda: StochasticMaze(),
    heuristic=lambda d, s: Value(
      cost=abs(d._goal.x - s.x)
        + abs(d._goal.y - s.y)),
    depth=5,  # short-sighted depth
    discount=1.0,
    epsilon=0.01,
) as solver:
    solver.solve()
```

❌ Backend not running. Start: cd backend && uv run



Ready — click ▶ Run on the code

The iDual Approach

LP-based heuristic search for SSPs

Live Demo: iDual

Watch iDual iteratively expand the state envelope

Family 2: Simulation-Based Search

UCT, PROST, POMCP, Goal-HSVI — online planning via simulation and point-based methods

Online vs. Offline Planning

Offline (VI, PI, LAO*, LRTDP):

- Compute policy for **all reachable states** before execution
- Policy lookup at runtime: O(1)
- Must enumerate state space (or reachable subset)

Online (UCT, POMCP, Goal-HSVI):

- Plan only for the **current state** at each step
- Interleave planning and execution
- No need to enumerate the full state space

When to go online?

- State space too large for full enumeration
- Simulator available but no explicit model
- Planning budget per step is sufficient
- POMDPs (belief space is continuous)

Monte Carlo Tree Search & UCT

Online planning via simulation

Live Demo: UCT on Stochastic Maze

PROST: UCT for RDDL Domains

State-of-the-art simulation-based solver

📖 **PROST (Keller & Eyerich, 2012)**

Extends UCT with domain-specific enhancements for RDDL:

- **IPC (Iterative Deepening)**: adaptive depth limits
- **Reward lock detection**: prune subtrees where reward is fixed
- **Outcome caching**: reuse simulation results for identical state-action pairs

IPPC track record:

Year	Result
2011	Winner
2014	Strong competitor

✦ **Key lesson**

Raw UCT is domain-agnostic. PROST shows that domain-specific enhancements (exploiting RDDL structure) dramatically improve performance. This is the advantage of having a declarative model.

Keller & Eyerich (2012). PROST: Probabilistic Planning Based on UCT.

POMCP: UCT for POMDPs

Particle-based belief + Monte Carlo tree search

{} POMCP

Represent beliefs as **particle sets** (samples from b), run UCT in belief-action-observation space:

1. Sample state $s \sim B$ (draw from particle belief)
2. Simulate UCT from s through action-observation tree
3. After action a, observation o: keep particles consistent with o
4. Return best action from root statistics

Why particles?

- No explicit belief vector needed – scales to large $|S|$
- Belief update = particle filtering (discard inconsistent particles)
- Converges to optimal POMDP policy as $N \to \infty$

Scale

Demonstrated on PocMan (1.56 × 10^56 states) – far beyond exact POMDP methods. The key: particles avoid enumerating the belief simplex.

Silver & Veness (2010). Monte-Carlo Planning in Large POMDPs.

Live Demo: POMCP on a POMDP Maze

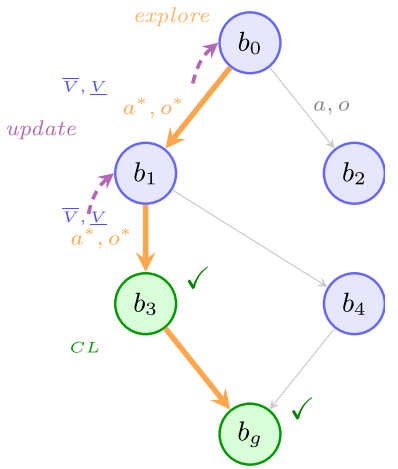
Goal-HSVI: Point-Based Value Iteration for Goal-POMDPs

{} Algorithm: Goal-HSVI

- **Init:** $\overline{V}$ (uniform), $\underline{V}$ (fast-informed), $CL \leftarrow \emptyset$
- **While** gap $> \varepsilon$: call **Explore**$(b_0, \varepsilon, 0, \emptyset, \emptyset)$
- **Explore**$(b, \varepsilon, t, \bar{a}, \bar{o})$:
  - If gap $\leq \eta$ or depth $\geq T$: add history to CL, return
  - Select greedy a^*, update bounds $\overline{V}, \underline{V}$ at b
  - Pick unexplored o^* with max weighted gap, recurse to $b^{o^*}_{a^*}$
  - Update bounds again at b

✦ First convergence guarantee

Goal-HSVI is the **first** algorithm with **provable convergence** for Goal-POMDPs. Provides anytime upper and lower bounds on $V^*(b_0)$.



🔧 Fixes over vanilla HSVI

1. **Uniform policy** init (not blind)
2. **Bounded depth** T (terminates)
3. **Closed list** CL (no revisits)

Horák, Bošanský & Chatterjee (2018). Goal-HSVI: Heuristic Search Value Iteration for Goal-POMDPs. IJCAI.

Live Demo: Goal-HSVI on a POMDP Maze

Algorithm Summary

Algorithm	Family	Observability	Explores	Best for
VI / PI	DP	Full	All states	Small MDPs
LAO*	Graph	Full	Reachable	Goal-oriented MDPs
LDFS	Graph	Full	Reachable	SSPs with deep goals
RTDP / LRTDP	Trial	Full	Sampled paths	Large SSPs
RTDP-Bel	Trial	Partial	Belief points	Small POMDPs
UCT	Simulation	Full	Sampled	Large MDPs, bounded horizon
PROST	Simulation	Full	Sampled	RDDL domains
POMCP	Simulation	Partial	Sampled	Large POMDPs