Decision-theoretic approaches to planning, coordination and communication in multiagent systems

Matthijs Spaan[1] Frans Oliehoek[2] Stefan Witwicki[3]

[1]Delft University of Technology
[2]U. of Liverpool & U. of Amsterdam
[3]EPFL

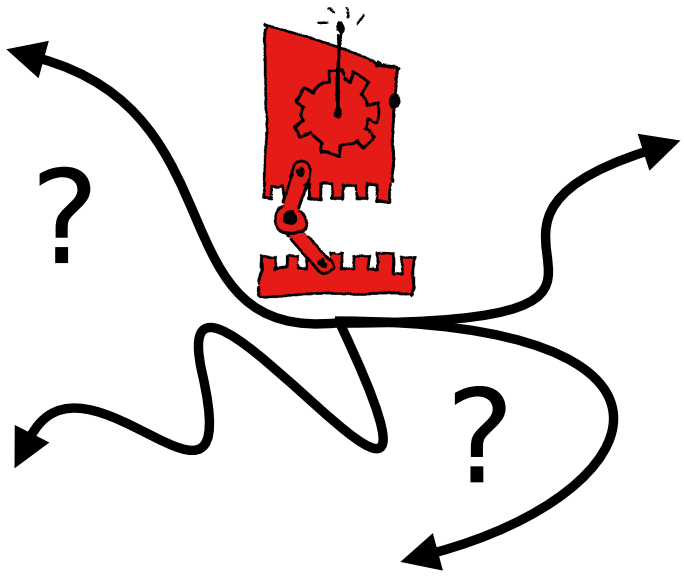
ESSENCE Autumn School
Ischia, Italy

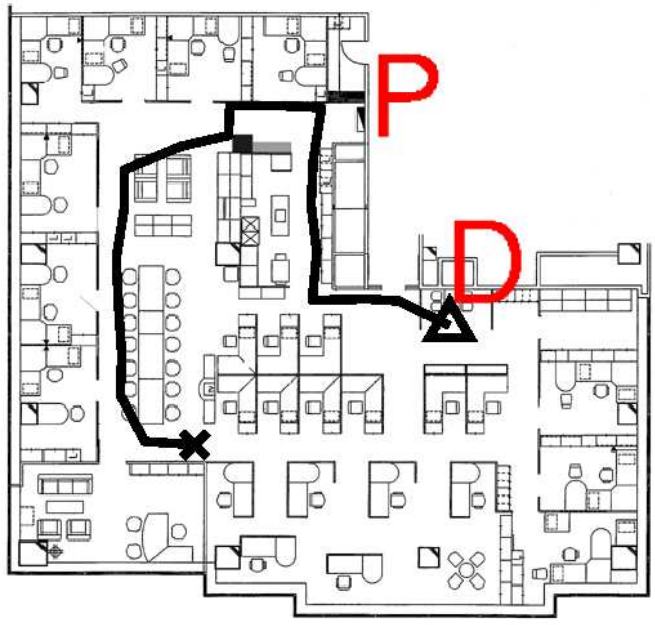
October 29, 2014

Introduction

Introduction

- Goal in Artificial Intelligence: to build intelligent agents.
- Our definition of “intelligent”: perform an assigned task as well as possible.
- Problem: how to act?
- We will explicitly model uncertainty.





Motivation

- Intelligent distributed systems are becoming ubiquitous:
  - Smart energy grid infrastructure
  - Surveillance camera networks
  - Autonomous guided vehicles, vehicular networks
  - Internet or smart phone applications
- Devices can sense, compute, act and interact.

Motivation

- Intelligent distributed systems are becoming ubiquitous:
  - Smart energy grid infrastructure
  - Surveillance camera networks
  - Autonomous guided vehicles, vehicular networks
  - Internet or smart phone applications
- Devices can sense, compute, act and interact.
- Growing need for scalable and flexible multiagent planning.
- Key issue: uncertainty

Outline

Before the break:

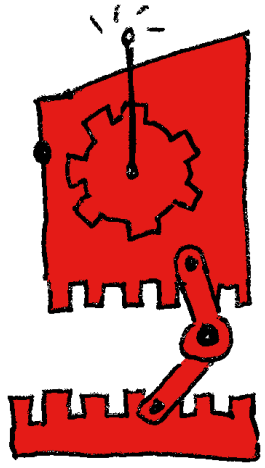
1. Introduction to decision making under uncertainty
2. Planning under action uncertainty (MDPs)
3. Planning under sensing uncertainty (POMDPs)

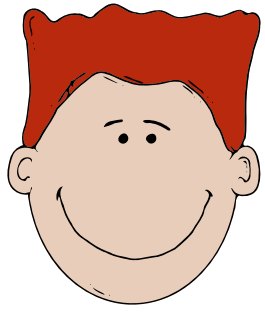
After the break:

1. Multiagent planning (Dec-POMDPs)
2. Communication

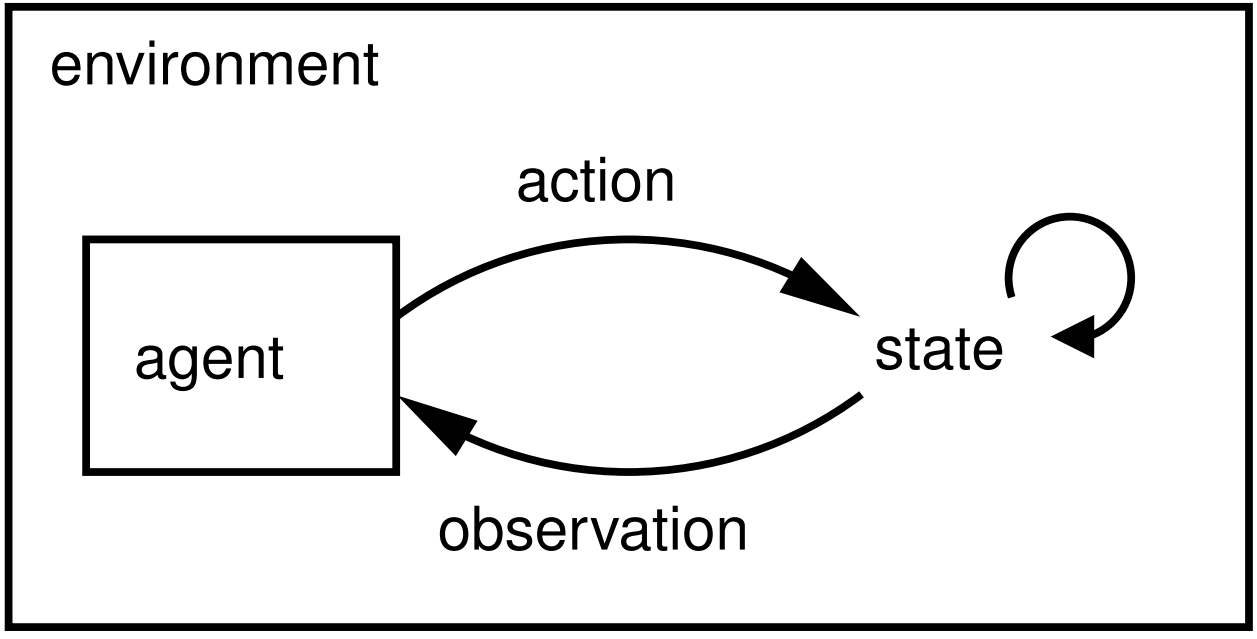
Agents

- An agent is a (rational) decision maker who is able to perceive its external (physical) environment and act autonomously upon it (Russell and Norvig, 2003).
- Rationality means reaching the optimum of a performance measure.
- Examples: humans, robots, some software programs.





Agents



- It is useful to think of agents as being involved in a perception-action loop with their environment.
- But how do we make the right decisions?

Planning

Planning:

- A plan tells an agent how to act.
- For instance
  - A sequence of actions to reach a goal.
  - What to do in a particular situation.
- We need to model:
  - the agent's actions
  - its environment
  - its task

We will model planning as a sequence of decisions.

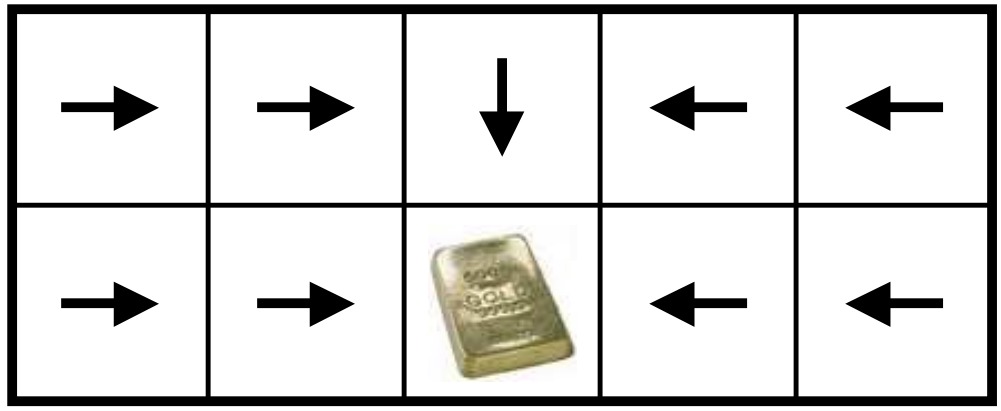
Classic planning

- Classic planning: sequence of actions from start to goal.
- Task: robot should get to gold as quickly as possible.
- Actions: $\rightarrow \downarrow \leftarrow \uparrow$
- Limitations:
  - New plan for each start state.
  - Environment is deterministic.

Classic planning

- Classic planning: sequence of actions from start to goal.
- Task: robot should get to gold as quickly as possible.
- Actions: $\rightarrow \downarrow \leftarrow \uparrow$
- Limitations:
  - New plan for each start state.
  - Environment is deterministic.
- Three optimal plans: $\rightarrow \rightarrow \downarrow$, $\rightarrow \downarrow \rightarrow$, $\downarrow \rightarrow \rightarrow$.

Conditional planning



- Assume our robot has noisy actions (wheel slip, overshoot).
- We need conditional plans.
- Map situations to actions.

Decision-theoretic planning

−0.1	−0.1	−0.1	−0.1	−0.1
−0.1	−0.1	10	−0.1	−0.1

- Positive reward when reaching goal, small penalty for all other actions.
- Agent's plan maximizes **value**: the sum of future rewards.
- Decision-theoretic planning successfully handles noise in acting and sensing.

Decision-theoretic planning

Plan #1:

→	→	↓		
				

Reward:

−0.1	−0.1	−0.1	−0.1	−0.1
−0.1	−0.1	10	−0.1	−0.1

Decision-theoretic planning

Values of this plan:

?	?	?		
		10		

Reward:

−0.1	−0.1	−0.1	−0.1	−0.1
−0.1	−0.1	10	−0.1	−0.1

Decision-theoretic planning

Values of this plan:

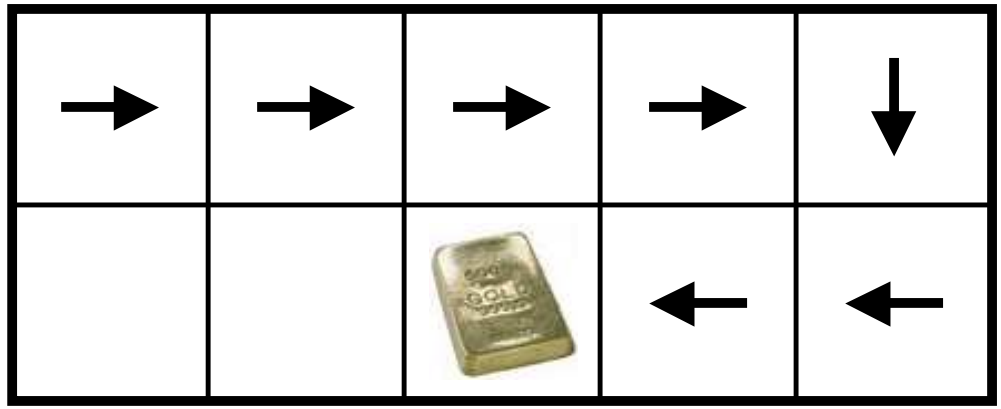
9.7	9.8	9.9		
		10		

Reward:

−0.1	−0.1	−0.1	−0.1	−0.1
−0.1	−0.1	10	−0.1	−0.1

Decision-theoretic planning

Plan #2:



Reward:

−0.1	−0.1	−0.1	−0.1	−0.1
−0.1	−0.1	10	−0.1	−0.1

Decision-theoretic planning

Values of this plan:

?	?	?	?	?
		10	?	?

Reward:

−0.1	−0.1	−0.1	−0.1	−0.1
−0.1	−0.1	10	−0.1	−0.1

Decision-theoretic planning

Values of this plan:

9.3	9.4	9.5	9.6	9.7
		10	9.9	9.8

Reward:

−0.1	−0.1	−0.1	−0.1	−0.1
−0.1	−0.1	10	−0.1	−0.1

Decision-theoretic planning

Optimal values (encode optimal plan):

9.7	9.8	9.9	9.8	9.7
9.8	9.9	10	9.9	9.8

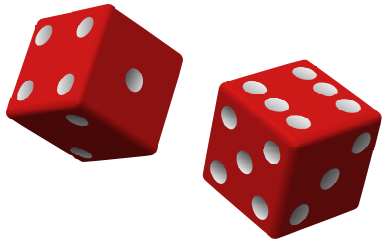
Reward:

−0.1	−0.1	−0.1	−0.1	−0.1
−0.1	−0.1	10	−0.1	−0.1

Markov Decision Processes

Sequential decision making under uncertainty

- Uncertainty is abundant in **real-world planning** domains.
- **Bayesian** approach $\Rightarrow$ probabilistic models.



Main assumptions:

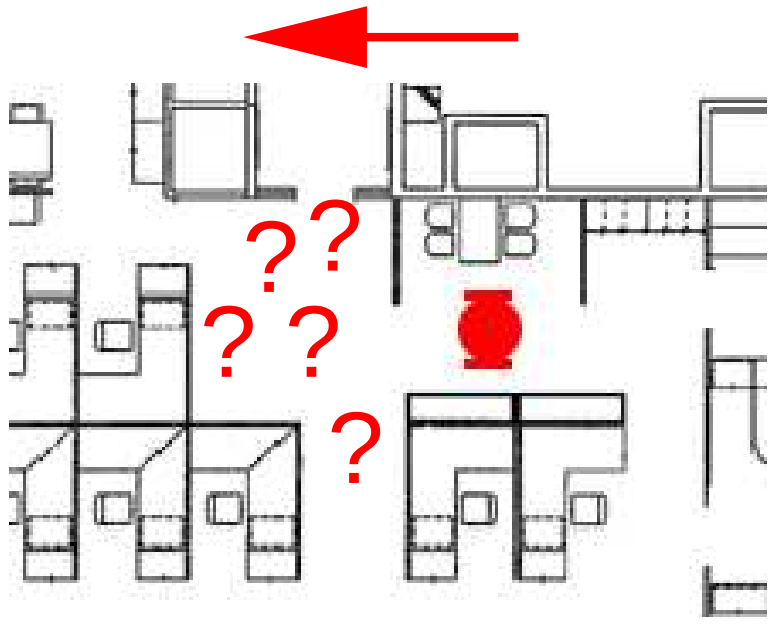
Sequential decisions: problems are formulated as a sequence of "independent" decisions;

Markovian environment: the state at time t depends only on the events at time $t-1$;

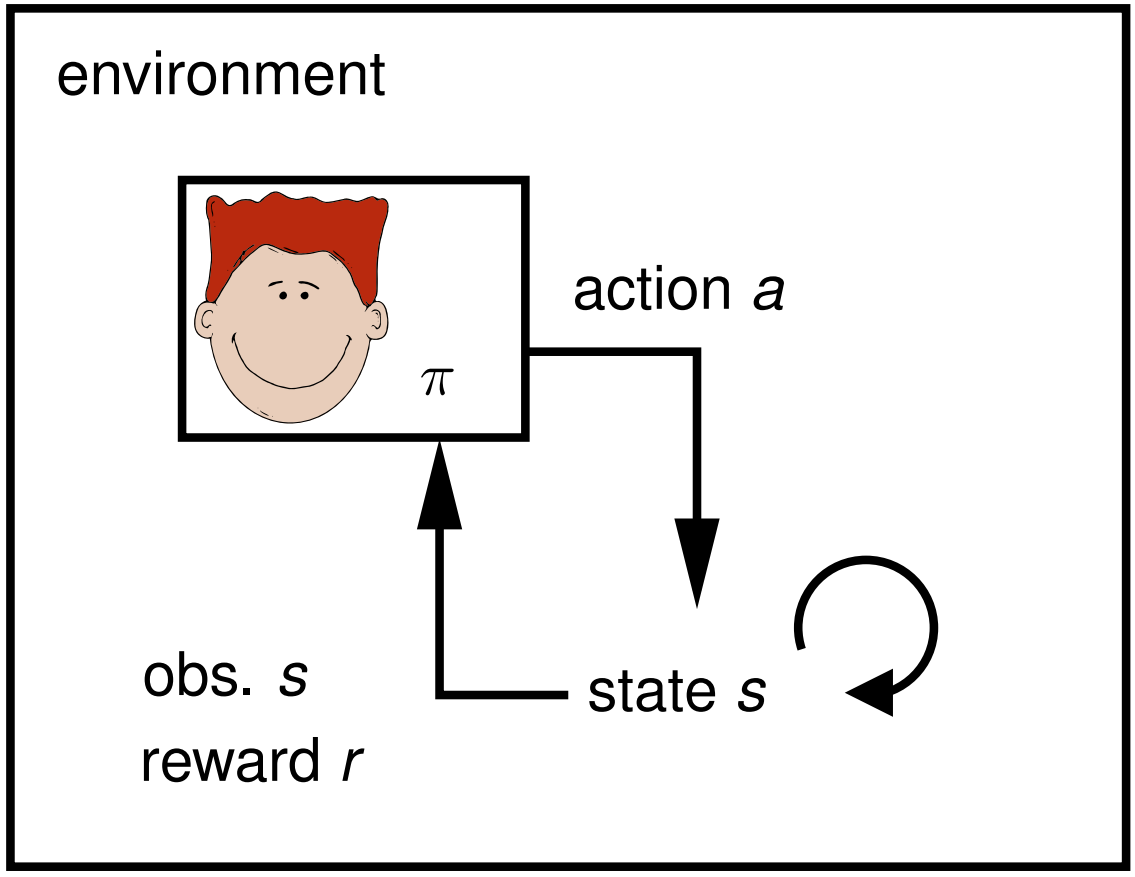
Evaluative feedback: use of a reinforcement signal as performance measure (reinforcement learning);

Transition model

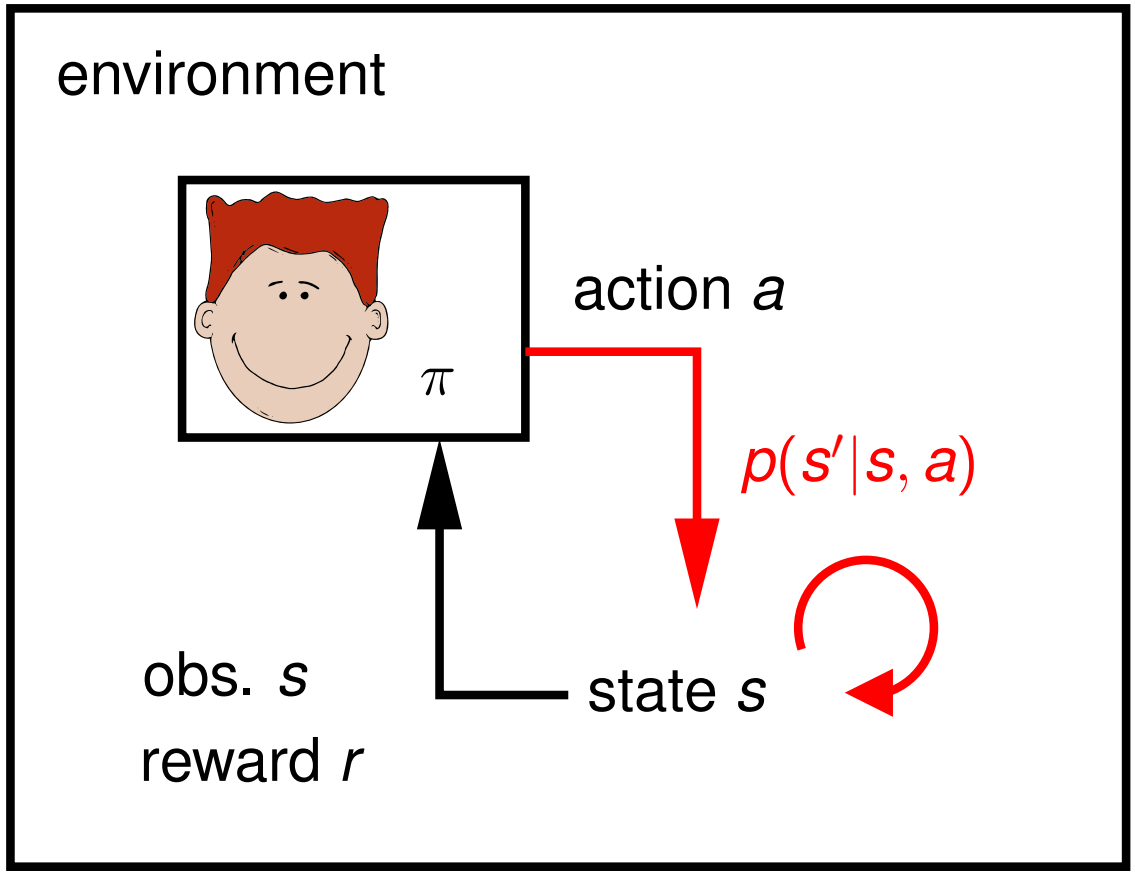
- For instance, robot motion is inaccurate.
- Transitions between states are **stochastic**.
- $p(s'|s, a)$ is the probability to jump from state s to state s' after taking action a.



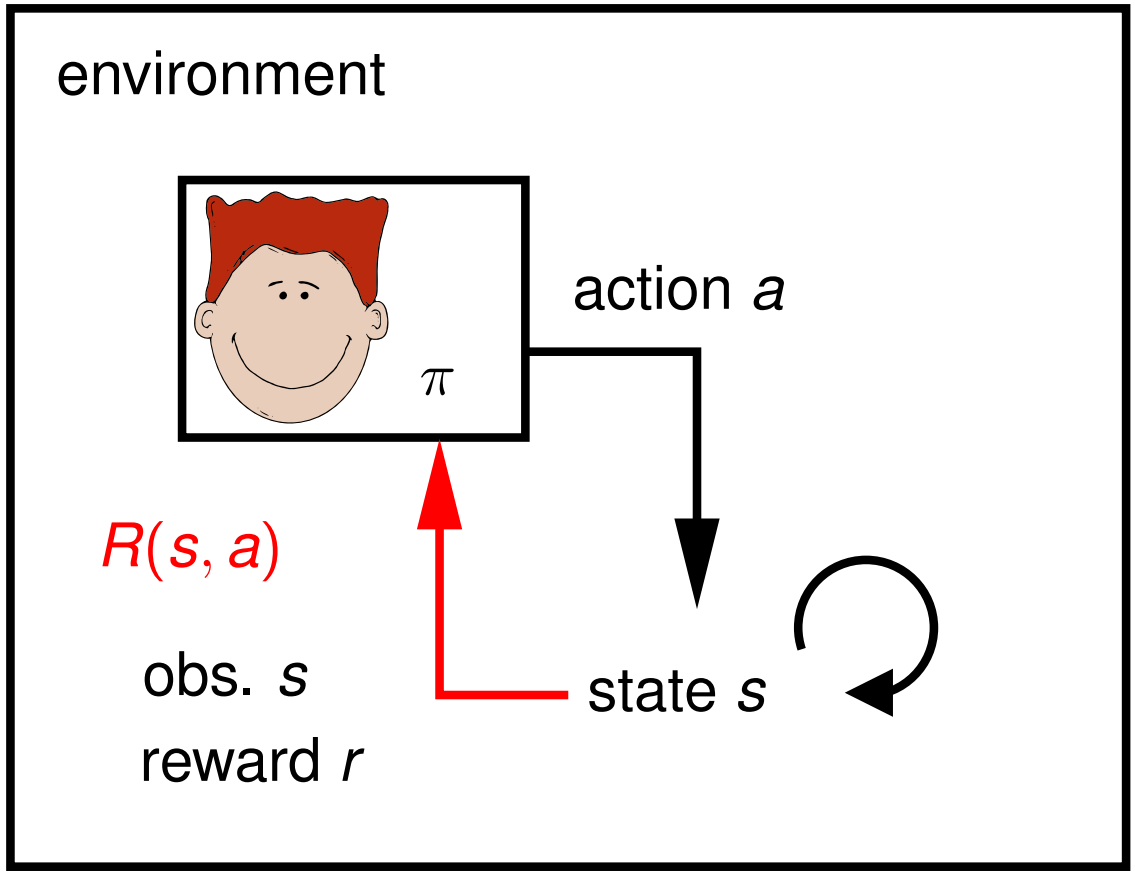
MDP Agent



MDP Agent



MDP Agent



Optimality criterion

For instance, agent should maximize the value

$$E\Big[\sum_{t=0}^{h} \gamma^t R_t\Big], \tag{1}$$

where

- h is the planning horizon, can be finite or ∞
- γ is a discount rate, $0 \leq \gamma < 1$

Reward hypothesis (Sutton and Barto, 1998):
All goals and purposes can be formulated as the maximization of the cumulative sum of a received scalar signal (reward).

Discrete MDP model

Discrete Markov Decision Process model (Puterman, 1994; Bertsekas, 2000):

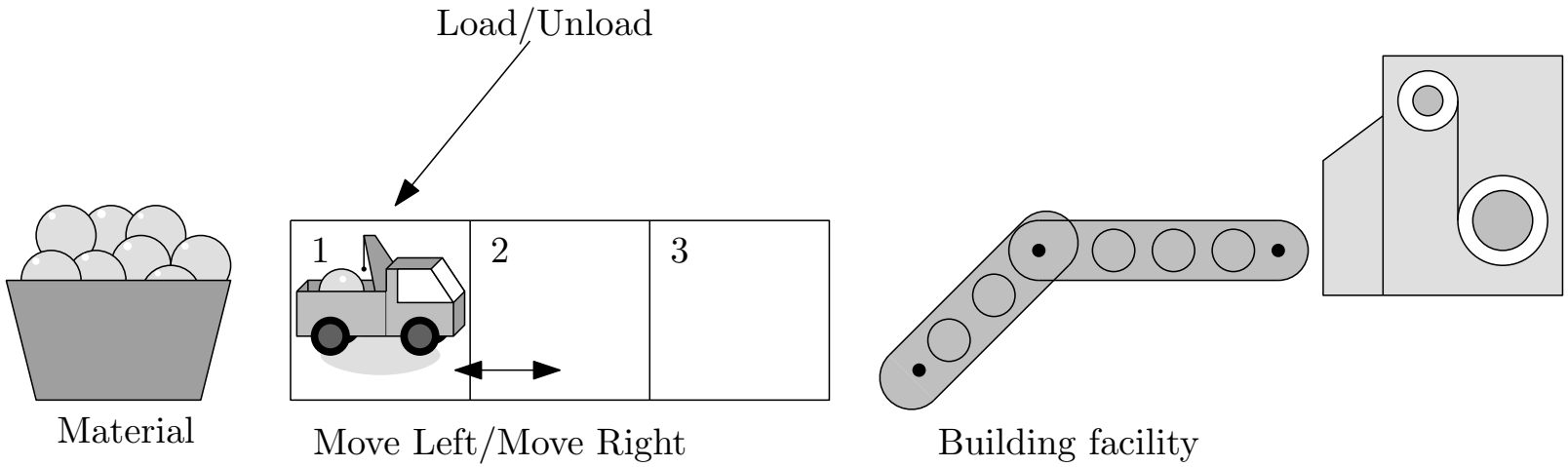
- Time t is discrete.
- State space S.
- Set of actions A.
- Reward function $R : S \times A \mapsto \mathbb{R}$.
- Transition model $p(s'|s, a)$, $T_a : S \times A \mapsto \Delta(S)$.
- Initial state s_0 is drawn from $\Delta(S)$.

The Markov property entails that the next state s_{t+1} only depends on the previous state s_t and action a_t:

$$p(s_{t+1}|s_t, s_{t-1}, \ldots, s_0, a_t, a_{t-1}, \ldots, a_0) = p(s_{t+1}|s_t, a_t). \quad (2)$$

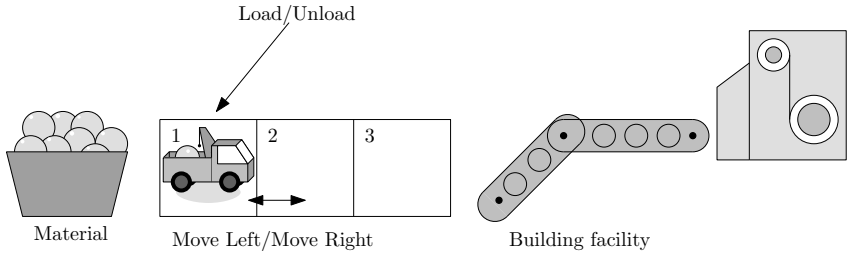
A simple problem

Problem:
An autonomous robot must learn how to transport material from a deposit to a building facility.



(thanks to F. Melo)

Load/Unload as an MDP



- States: $S = \{1_U, 2_U, 3_U, 1_L, 2_L, 3_L\}$;
  - 1_U Robot in position 1 (unloaded);
  - 2_U Robot in position 2 (unloaded);
  - 3_U Robot in position 3 (unloaded);
  - 1_L Robot in position 1 (loaded);
  - 2_L Robot in position 2 (loaded);
  - 3_L Robot in position 3 (loaded)
- Actions: $A = \{$Left, Right, Load, Unload$\}$;

Load/Unload as an MDP (1)

- Transition probabilities: “Left”/“Right” move the robot in the corresponding direction; “Load” loads material (only in position 1); “Unload” unloads material (only in position 3). Ex:

$$\begin{aligned}(2_L, \text{Right}) &\to 3_L;\\ (3_L, \text{Unload}) &\to 3_U;\\ (1_L, \text{Unload}) &\to 1_L.\end{aligned}$$

- Reward: We assign a reward of $+10$ for every unloaded package (payment for the service).

Load/Unload as an MDP (2)

- For each action $a \in A$, T_a is a matrix.
  Ex:

$$T_{\text{Right}} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

- Recall: $S = \{1_U, 2_U, 3_U, 1_L, 2_L, 3_L\}$.

Load/Unload as an MDP (3)

- The reward $R(s, a)$ can also be represented as a matrix
  Ex:

$$R = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & +10 \end{bmatrix}$$

$S = \{1_U, 2_U, 3_U, 1_L, 2_L, 3_L\}$, $A = \{\text{Left, Right, Load, Unload}\}$

Policies and value

- Policy π: tells the agent how to act.
- A deterministic policy $\pi : S \mapsto A$ is a mapping from states to actions.
- Value: how much reward $E[\sum_{t=0}^{h} \gamma^t R_t]$ does the agent expect to gather.
- Value denoted as $Q^{\pi}(s, a)$: start in s, do a and follow π afterwards.

Policies and value (1)

- Extracting a policy π from a value function Q is easy:

$$\pi(s) = \arg\max_{a \in A} Q(s, a). \tag{3}$$

- Optimal policy π^*: one that maximizes $E[\sum_{t=0}^{h} \gamma^t R_t]$ (for every state).
- In an infinite-horizon MDP there is always an optimal deterministic stationary (time-independent) policy π^*.
- There can be many optimal policies π^*, but they all share the same optimal value function Q^*.

Dynamic programming

Since S and A are finite, $Q^*(s, a)$ is a matrix.
Iterations of dynamic programming ($\gamma = 0.95$):

$$Q_0 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \qquad Q_1 = \begin{bmatrix} ? & ? & ? & ? \\ ? & ? & ? & ? \\ ? & ? & ? & ? \\ ? & ? & ? & ? \\ ? & ? & ? & ? \\ ? & ? & ? & ? \end{bmatrix}$$

$S = \{1_U, 2_U, 3_U, 1_L, 2_L, 3_L\}$, $A = \{\text{Left, Right, Load, Unload}\}$

Dynamic programming

Since S and A are finite, $Q^*(s, a)$ is a matrix.
Iterations of dynamic programming ($\gamma = 0.95$):

$$Q_0 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \qquad Q_1 = \begin{bmatrix} ? & ? & ? & ? \\ ? & ? & ? & ? \\ ? & ? & ? & ? \\ ? & ? & ? & ? \\ ? & ? & ? & ? \\ ? & ? & ? & ? \end{bmatrix}$$

$S = \{1_U, 2_U, 3_U, 1_L, 2_L, 3_L\}$, $A = \{\text{Left, Right, Load, Unload}\}$

Dynamic programming

Since S and A are finite, $Q^*(s, a)$ is a matrix.
Iterations of dynamic programming ($\gamma = 0.95$):

$$Q_0 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \qquad Q_1 = \begin{bmatrix} 0 & ? & ? & ? \\ 0 & ? & ? & ? \\ 0 & ? & ? & ? \\ 0 & ? & ? & ? \\ 0 & ? & ? & ? \\ 0 & ? & ? & ? \end{bmatrix}$$

$S = \{1_U, 2_U, 3_U, 1_L, 2_L, 3_L\}$, $A = \{\text{Left, Right, Load, Unload}\}$

Dynamic programming

Since S and A are finite, $Q^*(s, a)$ is a matrix.
Iterations of dynamic programming ($\gamma = 0.95$):

$$Q_0 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \qquad Q_1 = \begin{bmatrix} 0 & \mathbf{?} & ? & ? \\ 0 & \mathbf{?} & ? & ? \\ 0 & \mathbf{?} & ? & ? \\ 0 & \mathbf{?} & ? & ? \\ 0 & \mathbf{?} & ? & ? \\ 0 & \mathbf{?} & ? & ? \end{bmatrix}$$

$S = \{1_U, 2_U, 3_U, 1_L, 2_L, 3_L\}$, $A = \{\text{Left, Right, Load, Unload}\}$

Dynamic programming

Since S and A are finite, $Q^*(s, a)$ is a matrix.
Iterations of dynamic programming ($\gamma = 0.95$):

$$Q_0 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \qquad Q_1 = \begin{bmatrix} 0 & 0 & ? & ? \\ 0 & 0 & ? & ? \\ 0 & 0 & ? & ? \\ 0 & 0 & ? & ? \\ 0 & 0 & ? & ? \\ 0 & 0 & ? & ? \end{bmatrix}$$

$S = \{1_U, 2_U, 3_U, 1_L, 2_L, 3_L\}$, $A = \{\text{Left, Right, Load, Unload}\}$

Dynamic programming

Since S and A are finite, $Q^*(s, a)$ is a matrix.
Iterations of dynamic programming ($\gamma = 0.95$):

$$Q_0 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \qquad Q_1 = \begin{bmatrix} 0 & 0 & \mathbf{?} & ? \\ 0 & 0 & \mathbf{?} & ? \\ 0 & 0 & \mathbf{?} & ? \\ 0 & 0 & \mathbf{?} & ? \\ 0 & 0 & \mathbf{?} & ? \\ 0 & 0 & \mathbf{?} & ? \end{bmatrix}$$

$S = \{1_U, 2_U, 3_U, 1_L, 2_L, 3_L\}$, $A = \{\text{Left, Right, Load, Unload}\}$

Dynamic programming

Since S and A are finite, $Q^*(s, a)$ is a matrix.
Iterations of dynamic programming ($\gamma = 0.95$):

$$Q_0 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \qquad Q_1 = \begin{bmatrix} 0 & 0 & 0 & ? \\ 0 & 0 & 0 & ? \\ 0 & 0 & 0 & ? \\ 0 & 0 & 0 & ? \\ 0 & 0 & 0 & ? \\ 0 & 0 & 0 & ? \end{bmatrix}$$

$S = \{1_U, 2_U, 3_U, 1_L, 2_L, 3_L\}$, $A = \{\text{Left, Right, Load, Unload}\}$

Dynamic programming

Since S and A are finite, $Q^*(s, a)$ is a matrix.
Iterations of dynamic programming ($\gamma = 0.95$):

$$Q_0 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \qquad Q_1 = \begin{bmatrix} 0 & 0 & 0 & ? \\ 0 & 0 & 0 & ? \\ 0 & 0 & 0 & ? \\ 0 & 0 & 0 & ? \\ 0 & 0 & 0 & ? \\ 0 & 0 & 0 & ? \end{bmatrix}$$

$S = \{1_U, 2_U, 3_U, 1_L, 2_L, 3_L\}$, $A = \{\text{Left, Right, Load, Unload}\}$

Dynamic programming

Since S and A are finite, $Q^*(s, a)$ is a matrix.
Iterations of dynamic programming ($\gamma = 0.95$):

$$Q_0 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \qquad Q_1 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 10 \end{bmatrix}$$

$S = \{1_U, 2_U, 3_U, 1_L, 2_L, 3_L\}$, $A = \{\text{Left, Right, Load, Unload}\}$

Dynamic programming

Since S and A are finite, $Q^*(s, a)$ is a matrix.
Iterations of dynamic programming ($\gamma = 0.95$):

$$Q_1 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 10 \end{bmatrix} \qquad Q_2 = \begin{bmatrix} ? & ? & ? & ? \\ ? & ? & ? & ? \\ ? & ? & ? & ? \\ ? & ? & ? & ? \\ ? & ? & ? & ? \\ ? & ? & ? & ? \end{bmatrix}$$

$S = \{1_U, 2_U, 3_U, 1_L, 2_L, 3_L\}$, $A = \{\text{Left, Right, Load, Unload}\}$

Dynamic programming

Since S and A are finite, $Q^*(s, a)$ is a matrix.
Iterations of dynamic programming ($\gamma = 0.95$):

$$Q_1 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 10 \end{bmatrix} \qquad Q_2 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & ? & 0 & 0 \\ 0 & ? & ? & 10 \end{bmatrix}$$

$S = \{1_U, 2_U, 3_U, 1_L, 2_L, 3_L\}$, $A = \{\text{Left, Right, Load, Unload}\}$

Dynamic programming

Since S and A are finite, $Q^*(s, a)$ is a matrix.
Iterations of dynamic programming ($\gamma = 0.95$):

$$Q_1 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 10 \end{bmatrix} \qquad Q_2 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 9.5 & 0 & 0 \\ 0 & 9.5 & 9.5 & 10 \end{bmatrix}$$

$S = \{1_U, 2_U, 3_U, 1_L, 2_L, 3_L\}$, $A = \{\text{Left, Right, Load, Unload}\}$

Dynamic programming

Iterations of dynamic programming ($\gamma = 0.95$):

$$Q_5 = \begin{bmatrix} 0 & 0 & 8.57 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 8.57 & 9.03 & 8.57 & 8.57 \\ 8.57 & 9.5 & 9.03 & 9.03 \\ 9.03 & 9.5 & 9.5 & 10 \end{bmatrix}$$

$S = \{1_U, 2_U, 3_U, 1_L, 2_L, 3_L\}$, $A = \{\text{Left, Right, Load, Unload}\}$

Dynamic programming

Iterations of DP:

$$Q_{20} = \begin{bmatrix} 18.53 & 17.61 & 19.51 & 18.54 \\ 18.53 & 16.73 & 17.61 & 17.61 \\ 17.61 & 16.73 & 16.73 & 16.73 \\ 19.51 & 20.54 & 19.51 & 19.51 \\ 19.51 & 21.62 & 20.54 & 20.54 \\ 20.54 & 21.62 & 21.62 & 26.73 \end{bmatrix}$$

$S = \{1_U, 2_U, 3_U, 1_L, 2_L, 3_L\}$, $A = \{\text{Left, Right, Load, Unload}\}$

Dynamic programming

Final Q^* and policy:

$$Q^* = \begin{bmatrix} 30.75 & 29.21 & 32.37 & 30.75 \\ 30.75 & 27.75 & 29.21 & 29.21 \\ 29.21 & 27.75 & 27.75 & 27.75 \\ 32.37 & 34.07 & 32.37 & 32.37 \\ 32.37 & 35.86 & 34.07 & 34.07 \\ 34.07 & 35.86 & 35.86 & 37.75 \end{bmatrix} \qquad \pi^* = \begin{bmatrix} \text{Load} \\ \text{Left} \\ \text{Left} \\ \text{Right} \\ \text{Right} \\ \text{Unload} \end{bmatrix}$$

Value iteration

- Value iteration: successive approximation technique.
- Start with all values set to 0.
- In order to consider one step deeper into the future, i.e., to compute V^*_{n+1} from V^*_n:

$$Q^*_{n+1}(s,a) := R(s,a) + \gamma \sum_{s' \in S} p(s'|s,a) \max_{a' \in A} Q^*_n(s',a'), \quad (4)$$

  which is known as the dynamic programming update or Bellman backup.
- Bellman (1957) equation:

$$Q^*(s,a) = R(s,a) + \gamma \sum_{s' \in S} p(s'|s,a) \max_{a' \in A} Q^*(s',a'). \quad (5)$$

Value iteration

Value iteration discussion:

- As $n \to \infty$, value iteration converges.
- Value iteration has converged when the largest update δ in an iteration is below a certain threshold ϵ.
- Exhaustive sweeps are not required for convergence, provided that in the limit all states are visited infinitely often.
- This can be exploited by backing up the most promising states first, known as prioritized sweeping.

Q-learning

- Reinforcement-learning techniques learn from experience, no knowledge of the model is required.
- Q-learning update:

$$Q(s,a) = (1-\beta)\ Q(s,a) + \beta\left[r + \gamma \max_{a' \in A} Q(s',a')\right], \quad (6)$$

where $0 < \beta \leq 1$ is a learning rate.

Q-learning

Q-learning discussion:

- Q-learning is guaranteed to converge to the optimal Q-values if all $Q(s, a)$ values are updated infinitely often.
- In order to make sure all actions will eventually be tried in all states exploration is necessary.
- A common exploration method is to execute a random action with small probability ϵ, which is known as ϵ-greedy exploration.

Solution methods: MDPs

Model based

- Basic: dynamic programming (Bellman, 1957), value iteration, policy iteration.
- Advanced: prioritized sweeping, function approximators.

Model free, reinforcement learning (Sutton and Barto, 1998)

- Basic: Q-learning, TD(λ), SARSA, actor-critic.
- Advanced: generalization in infinite state spaces, exploration/exploitation issues.

POMDPs

Beyond MDPs

- MDPs have been very successful, but requires to have an observable Markovian state.
- Many domains this is impossible (or expensive) to obtain:
- Diagnosis (medical, maintenance)

Beyond MDPs

- MDPs have been very successful, but requires to have an observable Markovian state.
- Many domains this is impossible (or expensive) to obtain:

- Diagnosis (medical, maintenance)
- Robot navigation

Beyond MDPs

- MDPs have been very successful, but requires to have an observable Markovian state.
- Many domains this is impossible (or expensive) to obtain:

- Diagnosis (medical, maintenance)
- Robot navigation
- Tutoring

Beyond MDPs

- MDPs have been very successful, but requires to have an observable Markovian state.
- Many domains this is impossible (or expensive) to obtain:

- Diagnosis (medical, maintenance)
- Robot navigation
- Tutoring
- Dialog systems

Beyond MDPs

- MDPs have been very successful, but requires to have an observable Markovian state.
- Many domains this is impossible (or expensive) to obtain:

- Diagnosis (medical, maintenance)
- Robot navigation
- Tutoring
- Dialog systems
- Vision-based robotics

Beyond MDPs

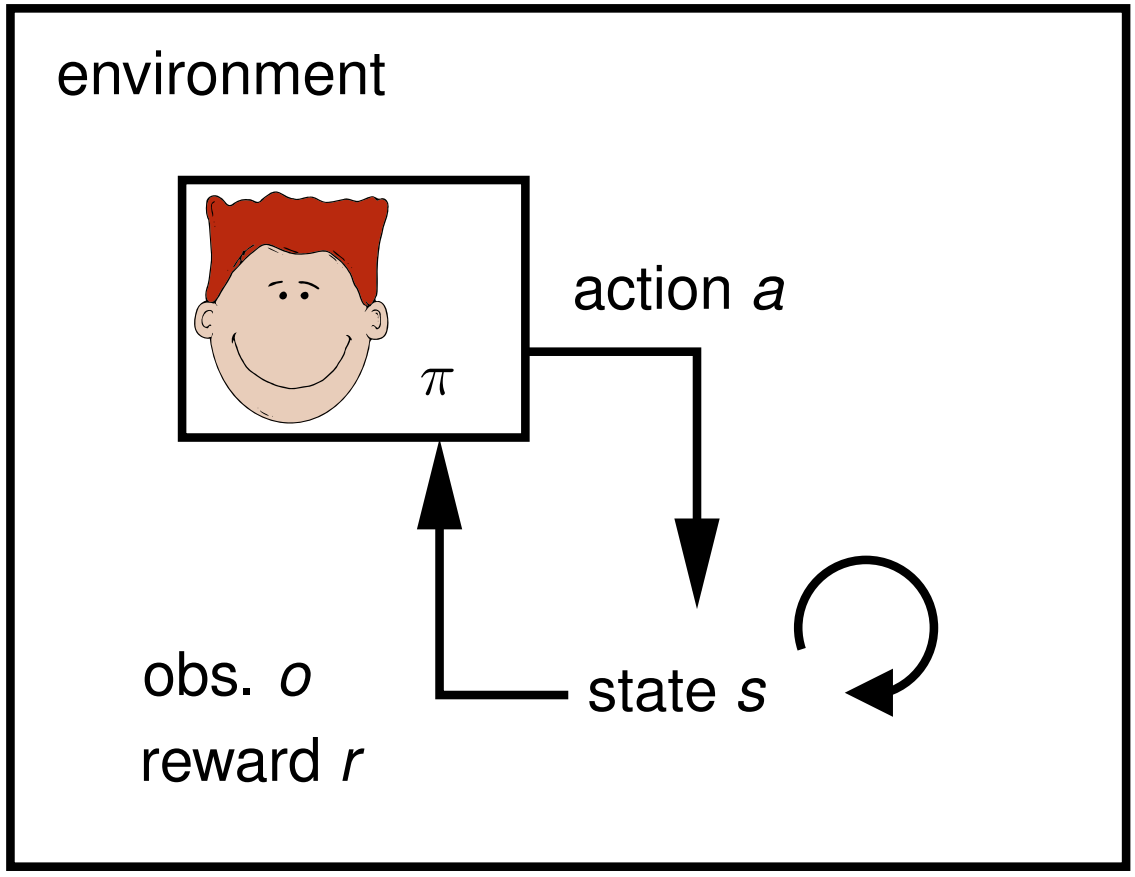
- MDPs have been very successful, but requires to have an observable Markovian state.
- Many domains this is impossible (or expensive) to obtain:

- Diagnosis (medical, maintenance)
- Robot navigation
- Tutoring
- Dialog systems
- Vision-based robotics
- Fault recovery

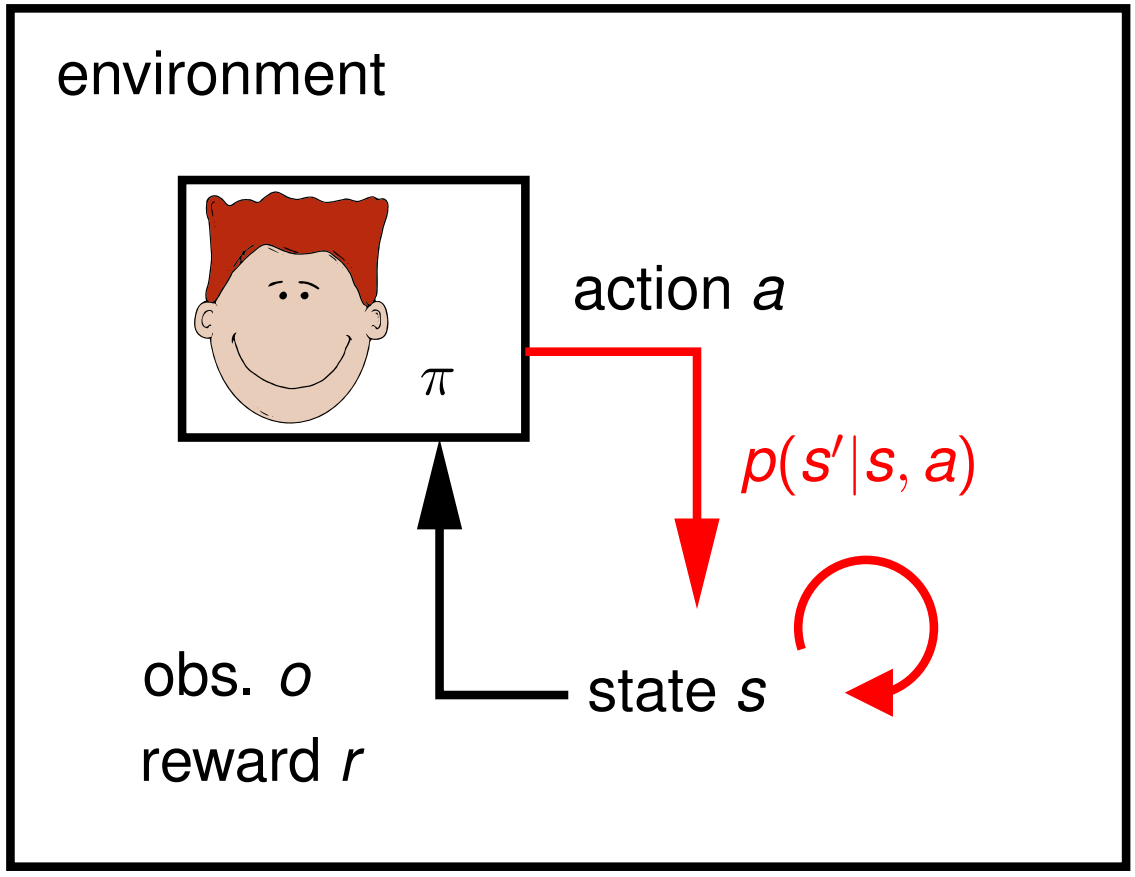
Observation model

- Imperfect sensors.
- Partially observable environment:
  - Sensors are **noisy**.
  - Sensors have a **limited view**.
- $p(o|s', a)$ is the probability the agent receives observation o in state s' after taking action a.

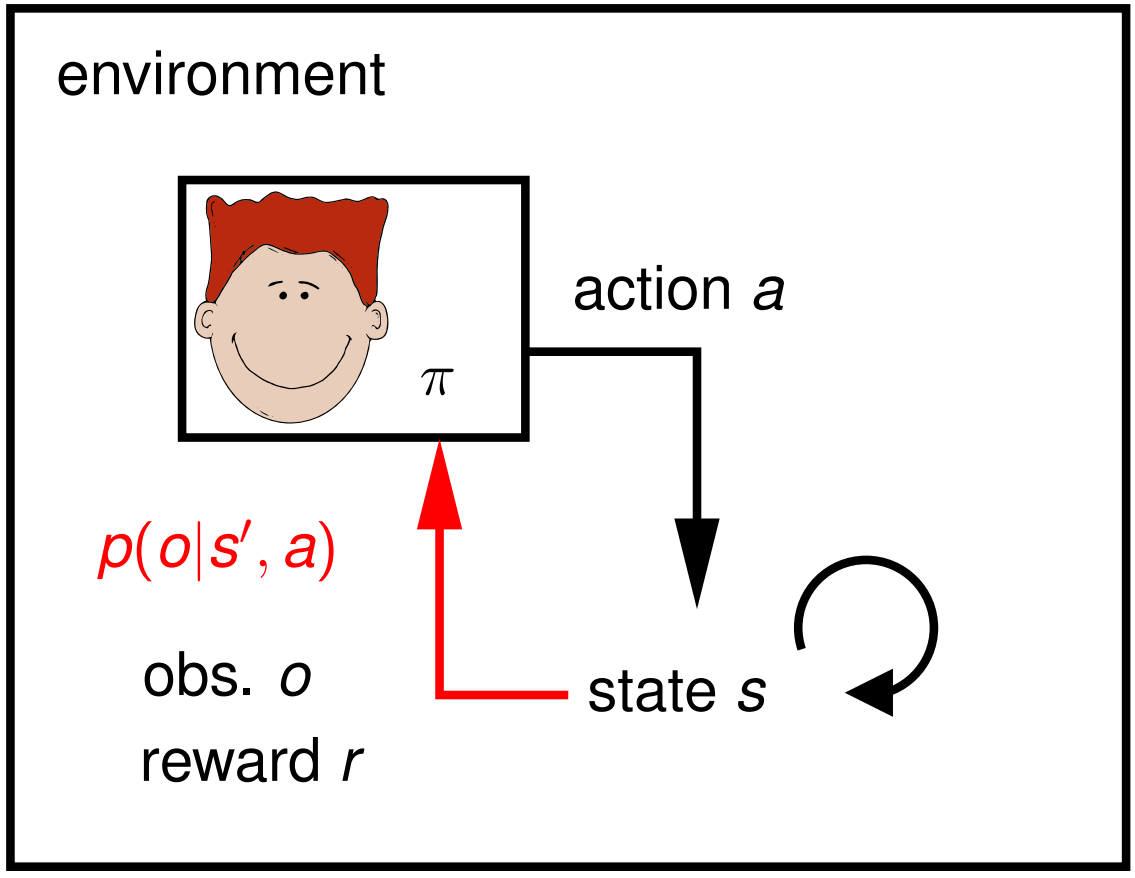
POMDP Agent



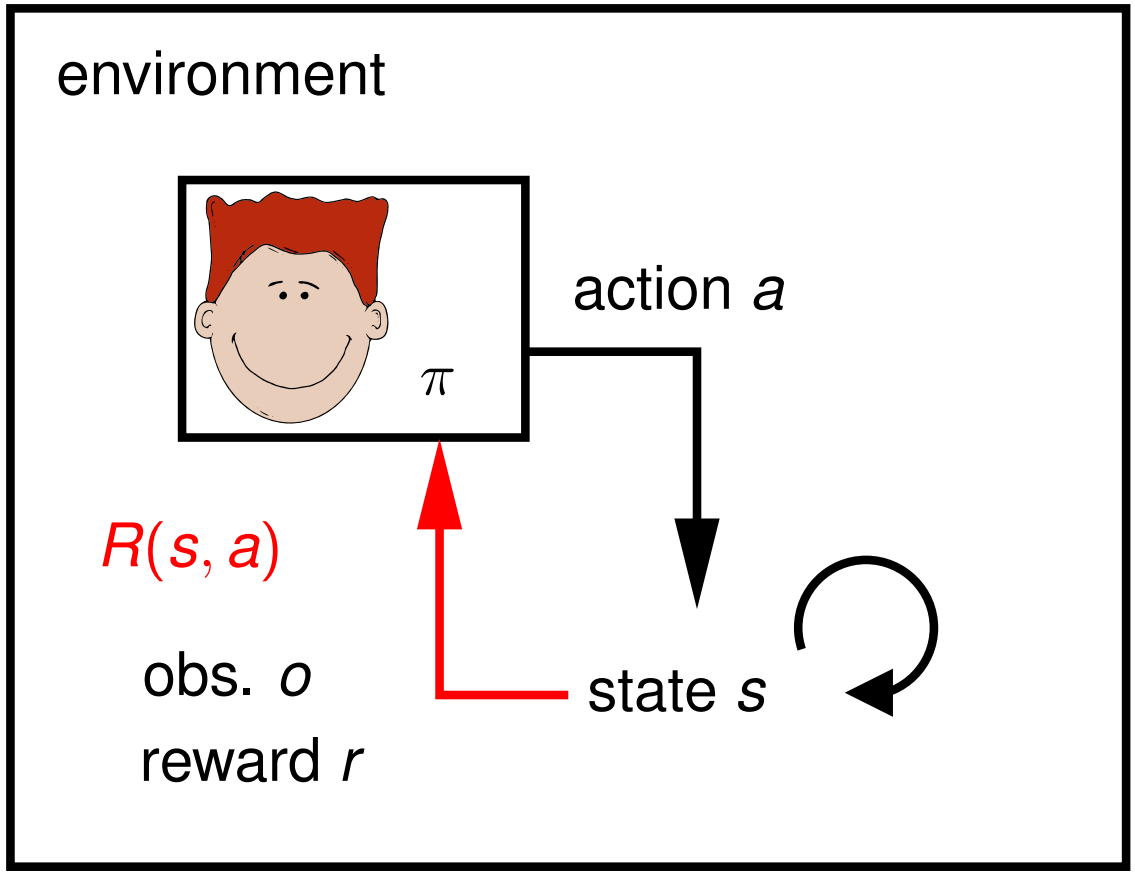
POMDP Agent



POMDP Agent



POMDP Agent



POMDPs

Partially observable Markov decision processes (POMDPs) (Kaelbling et al., 1998):

- Framework for agent planning under uncertainty.

POMDPs

Partially observable Markov decision processes (POMDPs) (Kaelbling et al., 1998):

- Framework for agent planning under uncertainty.
- Typically assumes discrete sets of states S, actions A and observations O.

POMDPs

Partially observable Markov decision processes (POMDPs) (Kaelbling et al., 1998):

- Framework for agent planning under uncertainty.
- Typically assumes discrete sets of states S, actions A and observations O.
- Transition model $p(s'|s, a)$: models the effect of **actions**.

POMDPs

Partially observable Markov decision processes (POMDPs) (Kaelbling et al., 1998):

- Framework for agent planning under uncertainty.
- Typically assumes discrete sets of states S, actions A and observations O.
- Transition model $p(s'|s, a)$: models the effect of **actions**.
- Observation model $p(o|s', a)$: relates **observations** to states.

POMDPs

Partially observable Markov decision processes (POMDPs) (Kaelbling et al., 1998):

- Framework for agent planning under uncertainty.
- Typically assumes discrete sets of states S, actions A and observations O.
- Transition model $p(s'|s, a)$: models the effect of **actions**.
- Observation model $p(o|s', a)$: relates **observations** to states.
- Task is defined by a **reward** model $R(s, a)$.

POMDPs

Partially observable Markov decision processes (POMDPs) (Kaelbling et al., 1998):

- Framework for agent planning under uncertainty.
- Typically assumes discrete sets of states S, actions A and observations O.
- Transition model $p(s'|s, a)$: models the effect of **actions**.
- Observation model $p(o|s', a)$: relates **observations** to states.
- Task is defined by a **reward** model $R(s, a)$.
- A planning horizon h (finite or ∞).

POMDPs

Partially observable Markov decision processes (POMDPs) (Kaelbling et al., 1998):

- Framework for agent planning under uncertainty.
- Typically assumes discrete sets of states S, actions A and observations O.
- Transition model $p(s'|s, a)$: models the effect of **actions**.
- Observation model $p(o|s', a)$: relates **observations** to states.
- Task is defined by a **reward** model $R(s, a)$.
- A planning horizon h (finite or ∞).
- A discount rate $0 \leq \gamma < 1$.

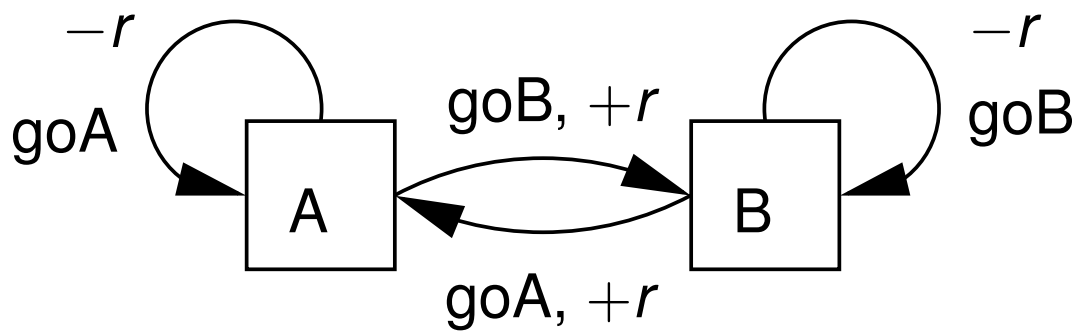
POMDPs

Partially observable Markov decision processes (POMDPs) (Kaelbling et al., 1998):

- Framework for agent planning under uncertainty.
- Typically assumes discrete sets of states S, actions A and observations O.
- Transition model $p(s'|s, a)$: models the effect of **actions**.
- Observation model $p(o|s', a)$: relates **observations** to states.
- Task is defined by a **reward** model $R(s, a)$.
- A planning horizon h (finite or ∞).
- A discount rate $0 \leq \gamma < 1$.
- Goal is to compute plan, or **policy** π, that maximizes long-term reward.

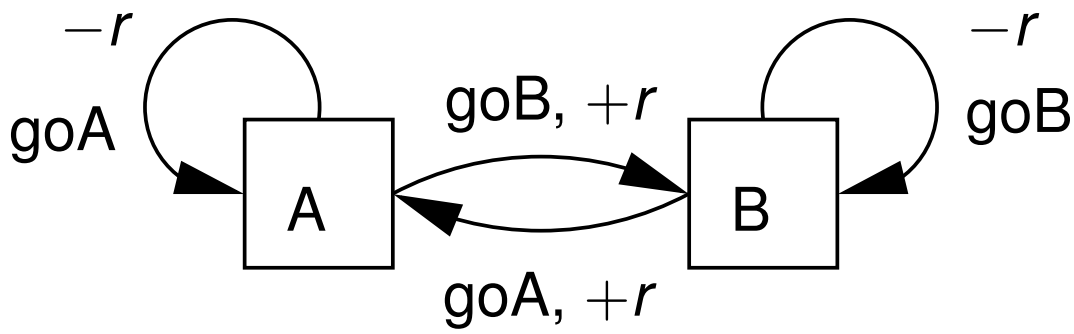
Memory

- In POMDPs memory is required for optimal decision making.
- In this non-observable example (Singh et al., 1994):



Memory

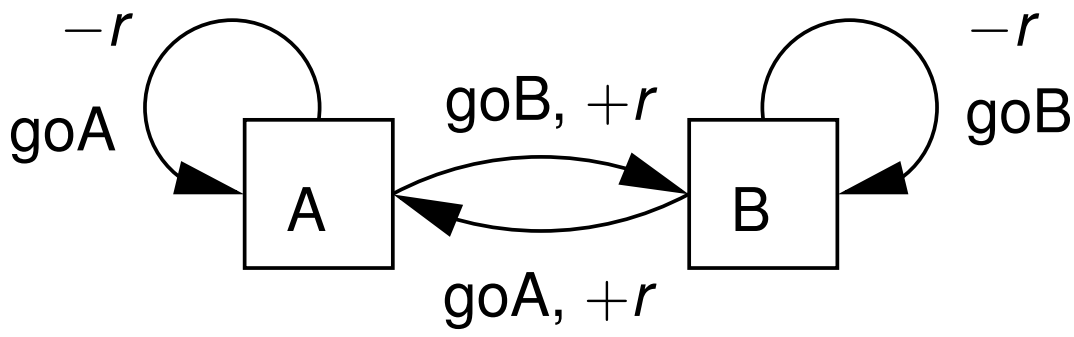
- In POMDPs memory is required for optimal decision making.
- In this non-observable example (Singh et al., 1994):



Policy	Value
MDP: optimal policy	
POMDP: memoryless deterministic	
POMDP: memoryless stochastic	
POMDP: memory-based (optimal)	

Memory

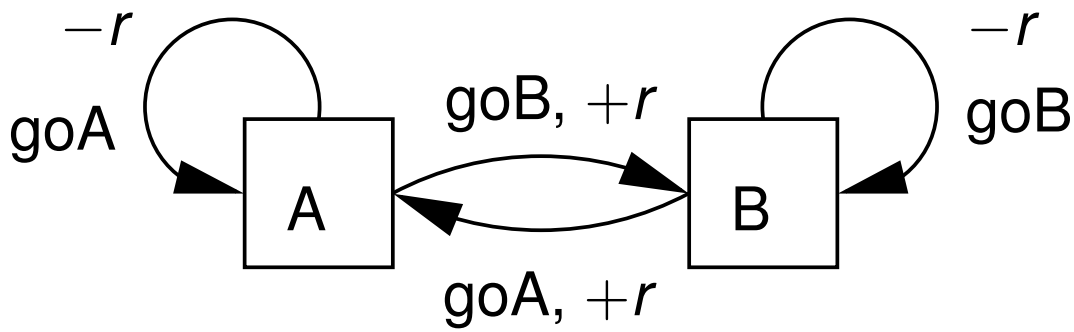
- In POMDPs memory is required for optimal decision making.
- In this non-observable example (Singh et al., 1994):



Policy	Value
MDP: optimal policy	$V = \sum_{t=0}^{\infty} \gamma^t r = \frac{r}{1-\gamma}$
POMDP: memoryless deterministic	
POMDP: memoryless stochastic	
POMDP: memory-based (optimal)	

Memory

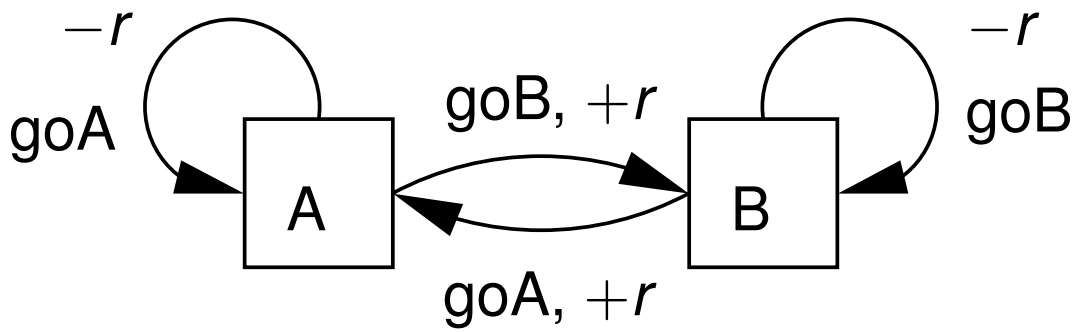
- In POMDPs memory is required for optimal decision making.
- In this non-observable example (Singh et al., 1994):



Policy	Value
MDP: optimal policy	$V = \sum_{t=0}^{\infty} \gamma^t r = \frac{r}{1-\gamma}$
POMDP: memoryless deterministic	$V_{\max} = r - \frac{\gamma r}{1-\gamma}$
POMDP: memoryless stochastic	
POMDP: memory-based (optimal)	

Memory

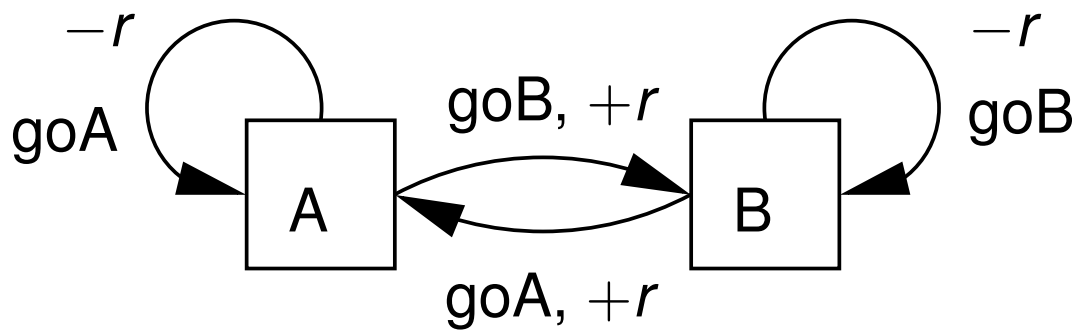
- In POMDPs memory is required for optimal decision making.
- In this non-observable example (Singh et al., 1994):



Policy	Value
MDP: optimal policy	$V = \sum_{t=0}^{\infty} \gamma^t r = \frac{r}{1-\gamma}$
POMDP: memoryless deterministic	$V_{\max} = r - \frac{\gamma r}{1-\gamma}$
POMDP: memoryless stochastic	$V = 0$
POMDP: memory-based (optimal)	

Memory

- In POMDPs memory is required for optimal decision making.
- In this non-observable example (Singh et al., 1994):



Policy	Value
MDP: optimal policy	$V = \sum_{t=0}^{\infty} \gamma^t r = \frac{r}{1-\gamma}$
POMDP: memoryless deterministic	$V_{\max} = r - \frac{\gamma r}{1-\gamma}$
POMDP: memoryless stochastic	$V = 0$
POMDP: memory-based (optimal)	$V_{\min} = \frac{\gamma r}{1-\gamma} - r$

Beliefs

Beliefs:

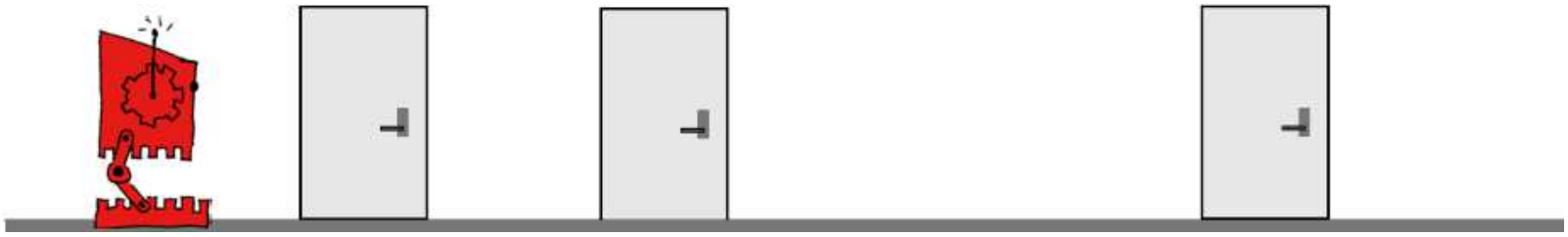
- The agent maintains a **belief** $b(s)$ of being at state s.
- After action $a \in A$ and observation $o \in O$ the belief $b(s)$ can be updated using Bayes' rule:

$$b'(s') \propto p(o|s') \sum_s p(s'|s,a) b(s)$$

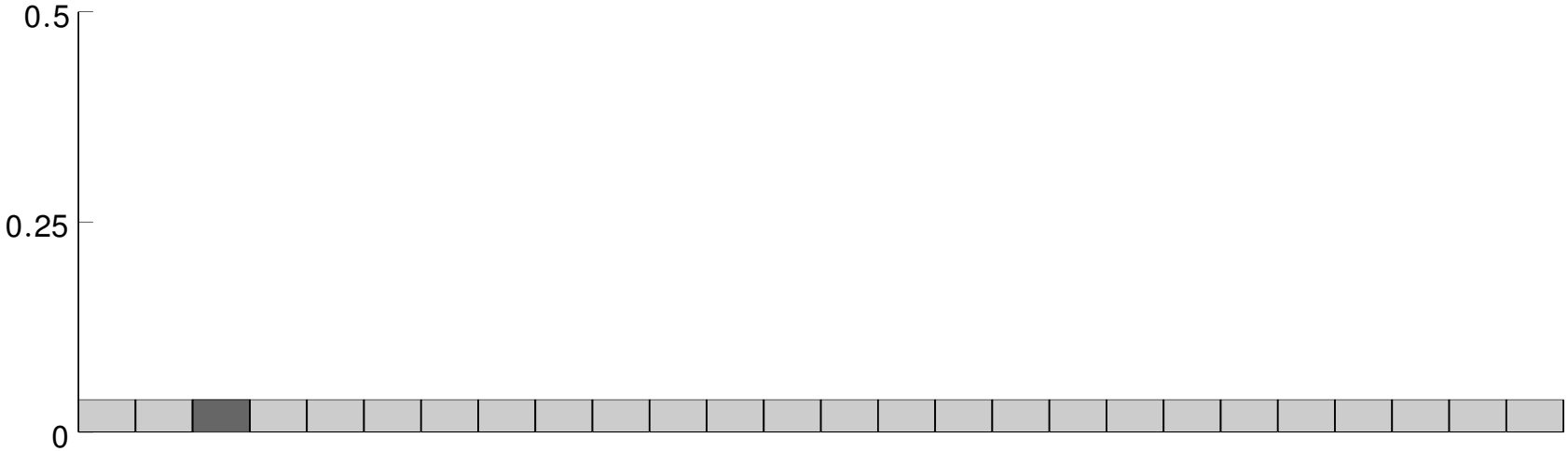
- The belief vector is a **Markov** signal for the planning task.

Belief update example

True situation:



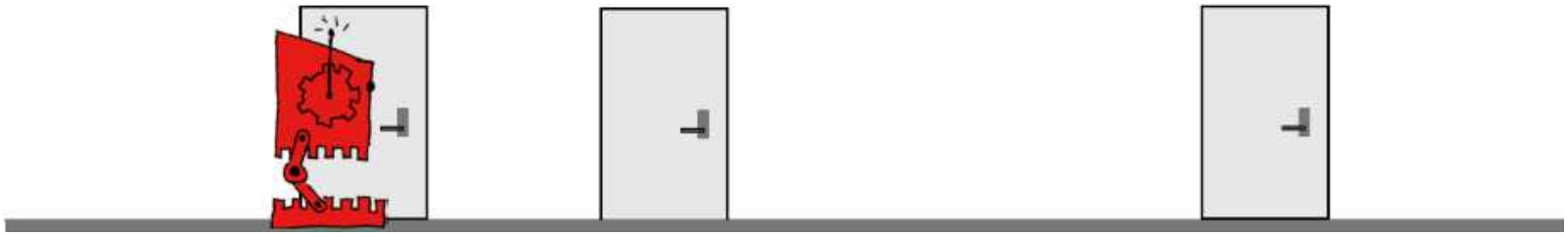
Robot's belief:



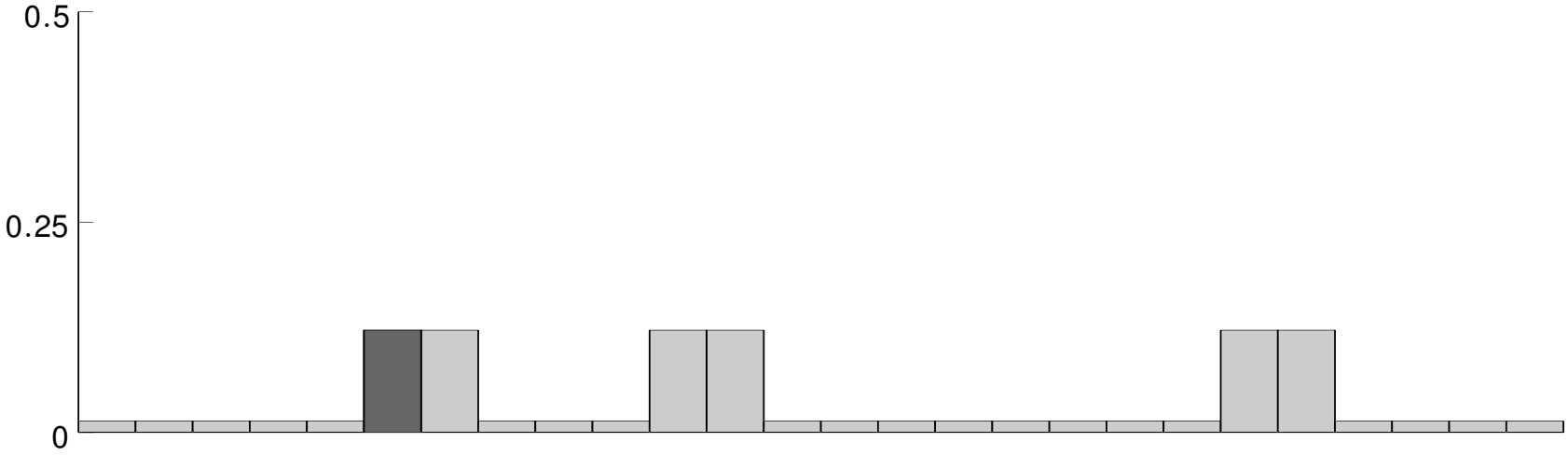
- Observations: *door* or *corridor*, 10% noise.
- Action: moves 3 (20%), 4 (60%), or 5 (20%) states.

Belief update example

True situation:



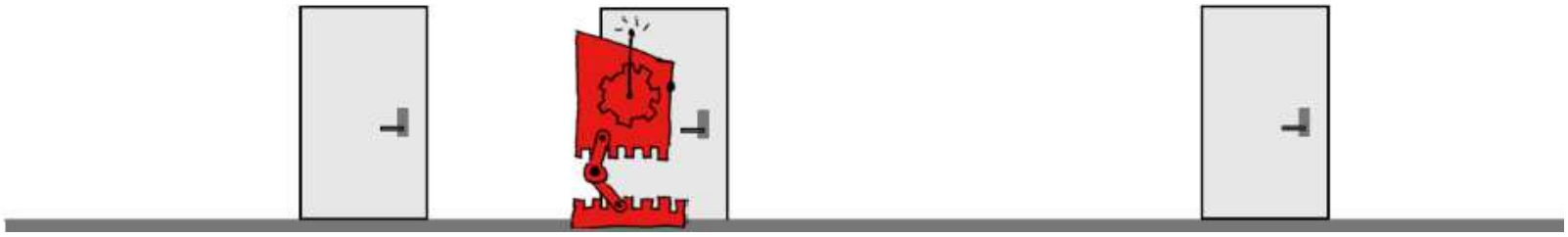
Robot's belief:



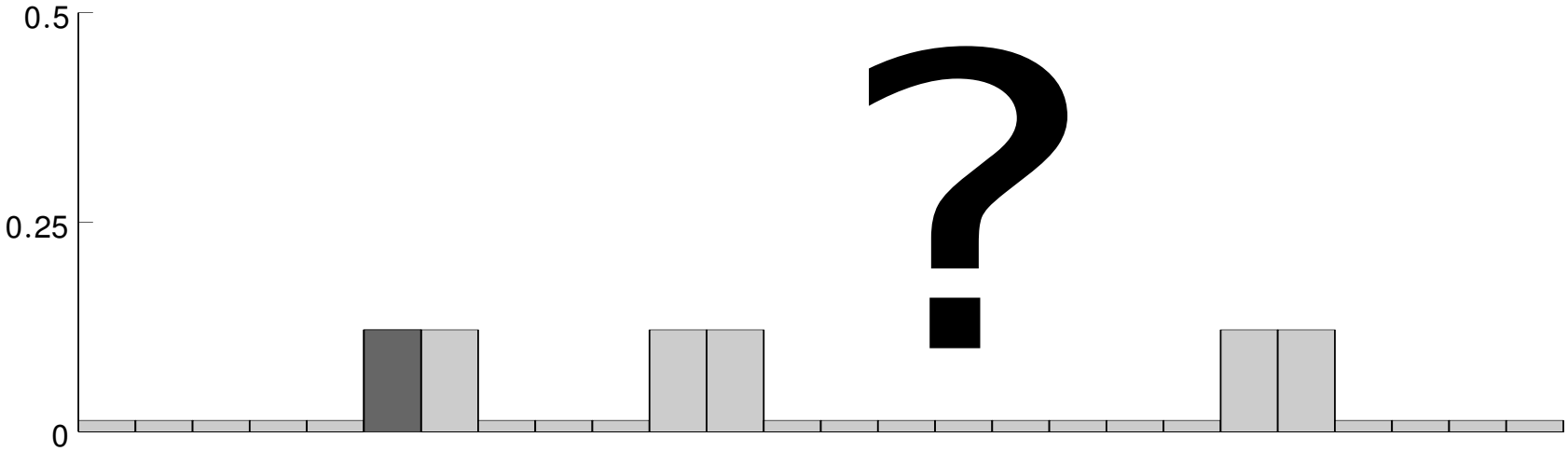
- Observations: **door** or *corridor*, 10% noise.
- Action: moves 3 (20%), 4 (60%), or 5 (20%) states.

Belief update example

True situation:



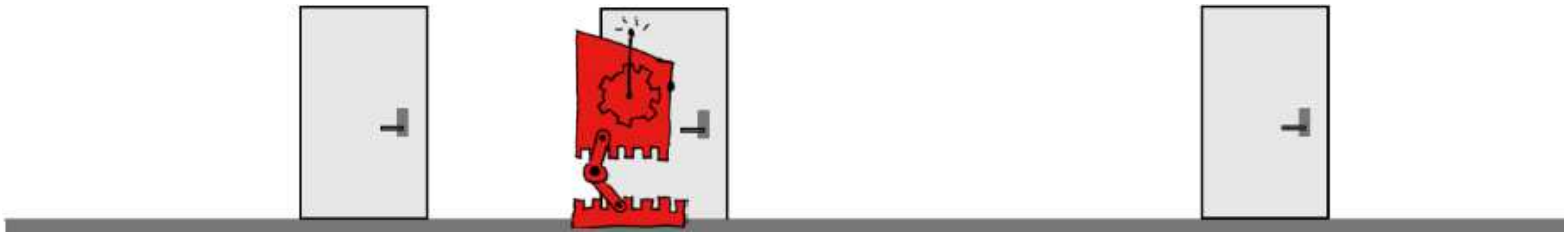
Robot's belief:



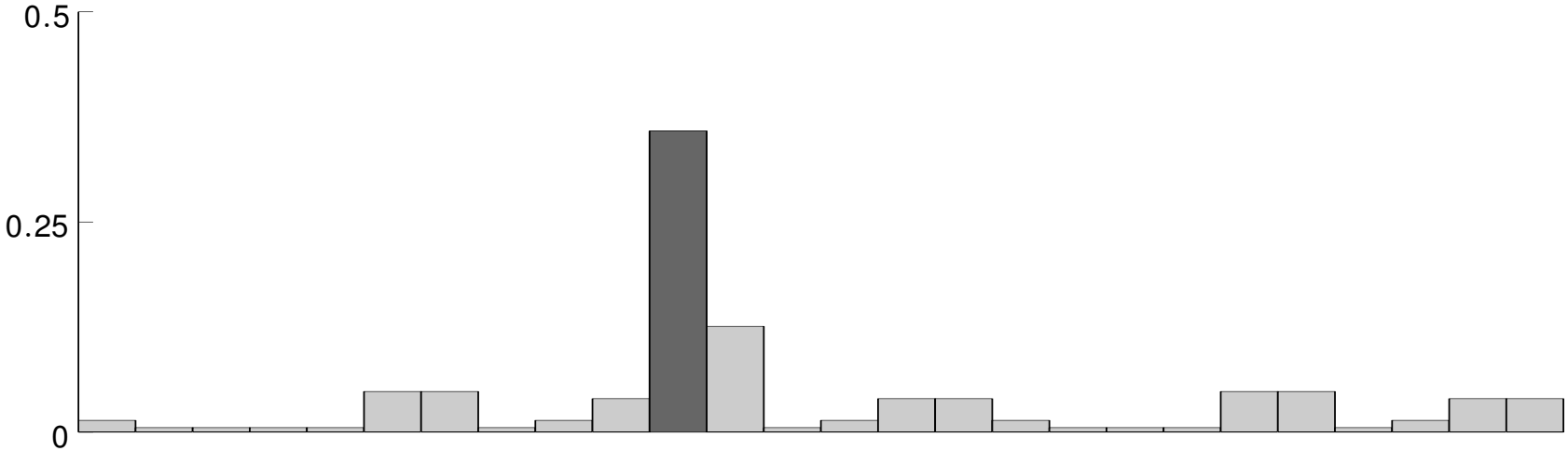
- Observations: **door** or *corridor*, 10% noise.
- Action: moves 3 (20%), 4 (60%), or 5 (20%) states.

Belief update example

True situation:



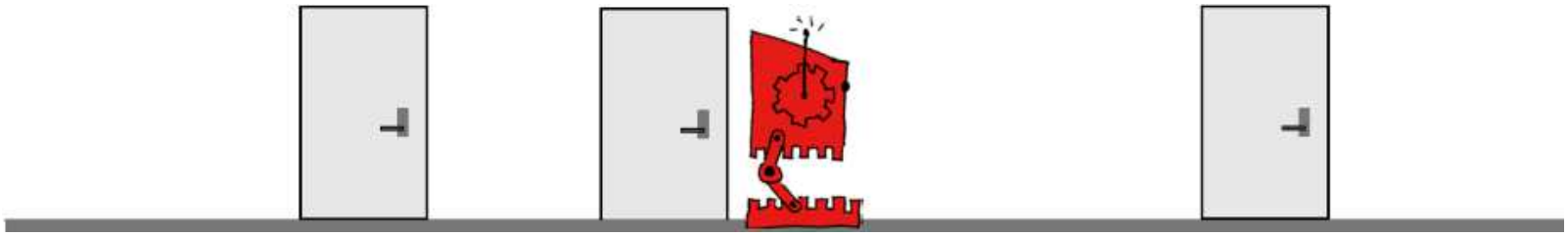
Robot's belief:



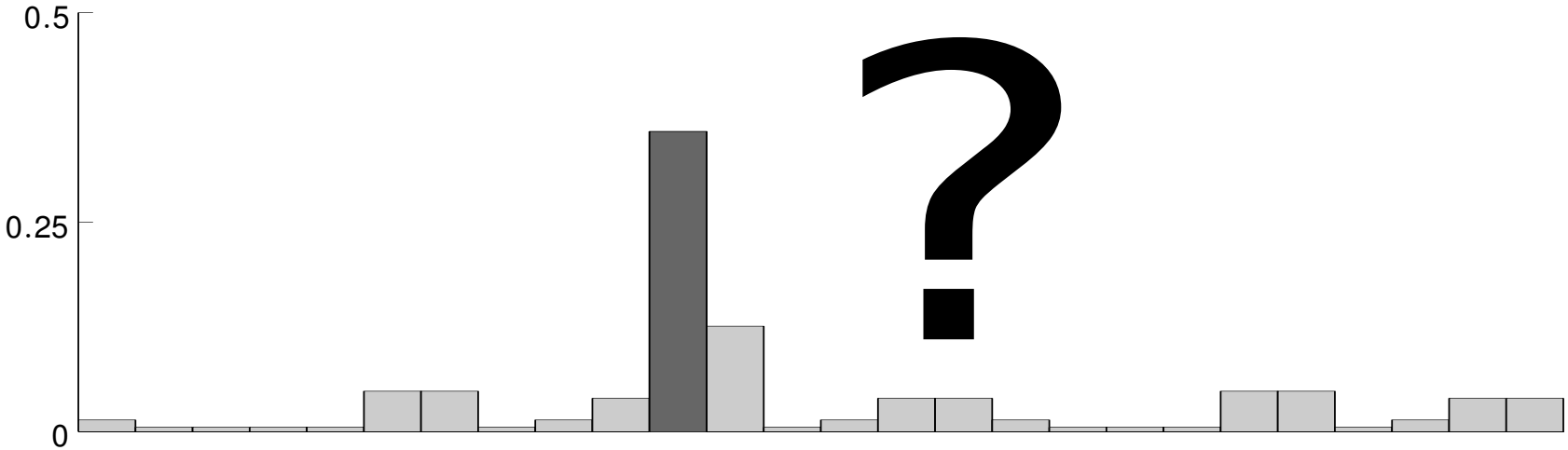
- Observations: **door** or *corridor*, 10% noise.
- Action: moves 3 (20%), 4 (60%), or 5 (20%) states.

Belief update example

True situation:



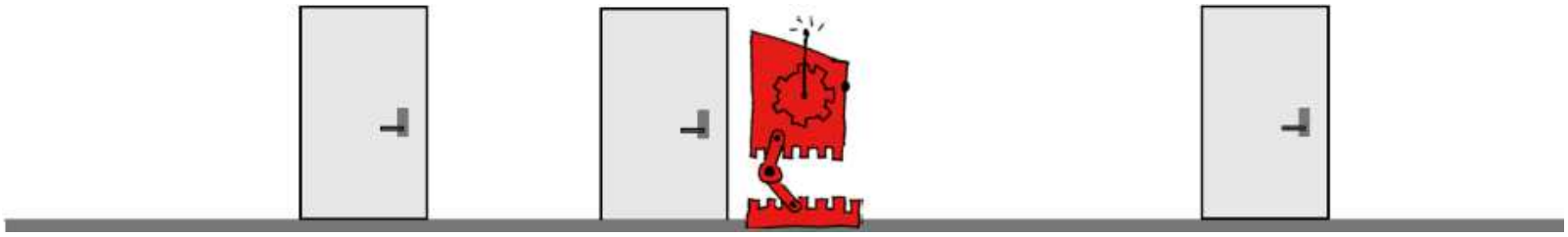
Robot's belief:



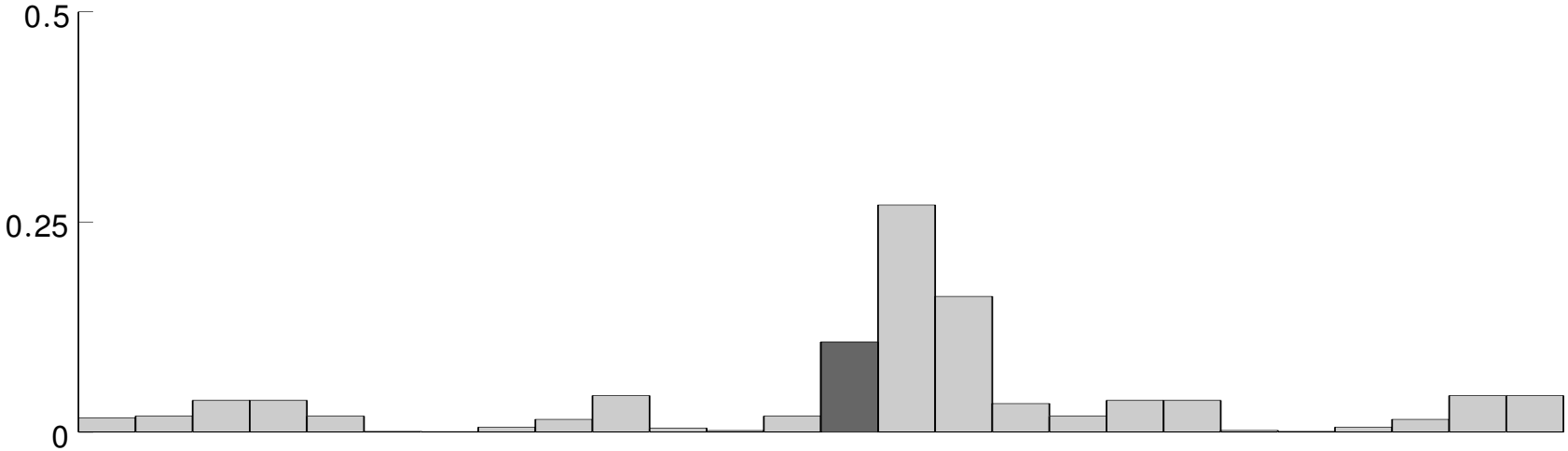
- Observations: *door* or **corridor**, 10% noise.
- Action: moves 3 (20%), 4 (60%), or 5 (20%) states.

Belief update example

True situation:



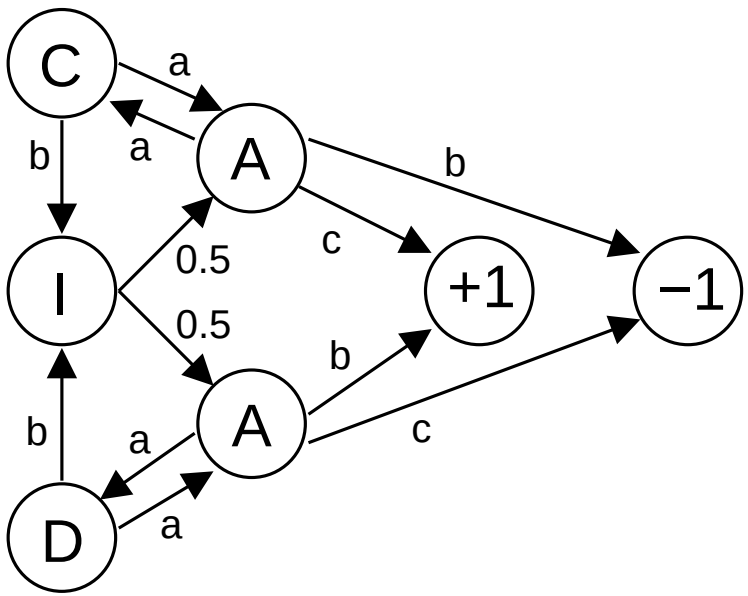
Robot's belief:



- Observations: *door* or **corridor**, 10% noise.
- Action: moves 3 (20%), 4 (60%), or 5 (20%) states.

MDP-based algorithms

- Exploit belief state, and use the MDP solution as a heuristic.
- Most likely state (Cassandra et al., 1996): $\pi_{MLS}(b) = \pi^*(\arg\max_s b(s))$.
- Q_{MDP} (Littman et al., 1995): $\pi_{Q_{\text{MDP}}}(b) = \arg\max_a \sum_s b(s)Q^*(s,a)$.



(Parr and Russell, 1995)

POMDPs as continuous-state MDPs

A belief-state POMDP can be treated as a continuous-state MDP:

- Continuous state space Δ: a simplex in $[0, 1]^{|S|-1}$.

POMDPs as continuous-state MDPs

A belief-state POMDP can be treated as a continuous-state MDP:

- Continuous state space Δ: a simplex in $[0, 1]^{|S|-1}$.
- Stochastic Markovian transition model $p(b_a^o|b, a) = p(o|b, a)$. This is the normalizer of Bayes' rule.

POMDPs as continuous-state MDPs

A belief-state POMDP can be treated as a continuous-state MDP:

- Continuous state space Δ: a simplex in $[0, 1]^{|S|-1}$.
- Stochastic Markovian transition model $p(b_a^o|b, a) = p(o|b, a)$. This is the normalizer of Bayes' rule.
- Reward function $R(b, a) = \sum_s R(s, a)b(s)$. This is the average reward with respect to $b(s)$.

POMDPs as continuous-state MDPs

A belief-state POMDP can be treated as a continuous-state MDP:

- Continuous state space Δ: a simplex in $[0, 1]^{|S|-1}$.
- Stochastic Markovian transition model $p(b_a^o|b, a) = p(o|b, a)$. This is the normalizer of Bayes' rule.
- Reward function $R(b, a) = \sum_s R(s, a)b(s)$. This is the average reward with respect to $b(s)$.
- The robot fully 'observes' the new belief-state b_a^o after executing a and observing o.

Solving POMDPs

- A solution to a POMDP is a **policy**, i.e., a mapping $\pi : \Delta \mapsto A$ from beliefs to actions.

Solving POMDPs

- A solution to a POMDP is a **policy**, i.e., a mapping $\pi : \Delta \mapsto A$ from beliefs to actions.
- The optimal value V^* of a POMDP satisfies the Bellman optimality equation $V^* = HV^*$:

$$V^*(b) = \max_a \left[R(b,a) + \gamma \sum_o p(o|b,a) V^*(b_a^o) \right]$$

Solving POMDPs

- A solution to a POMDP is a **policy**, i.e., a mapping $\pi : \Delta \mapsto A$ from beliefs to actions.
- The optimal value V^* of a POMDP satisfies the Bellman optimality equation $V^* = HV^*$:

$$V^*(b) = \max_a \left[R(b,a) + \gamma \sum_o p(o|b,a) V^*(b_a^o) \right]$$

- Value iteration repeatedly applies $V_{n+1} = HV_n$ starting from an initial V_0.

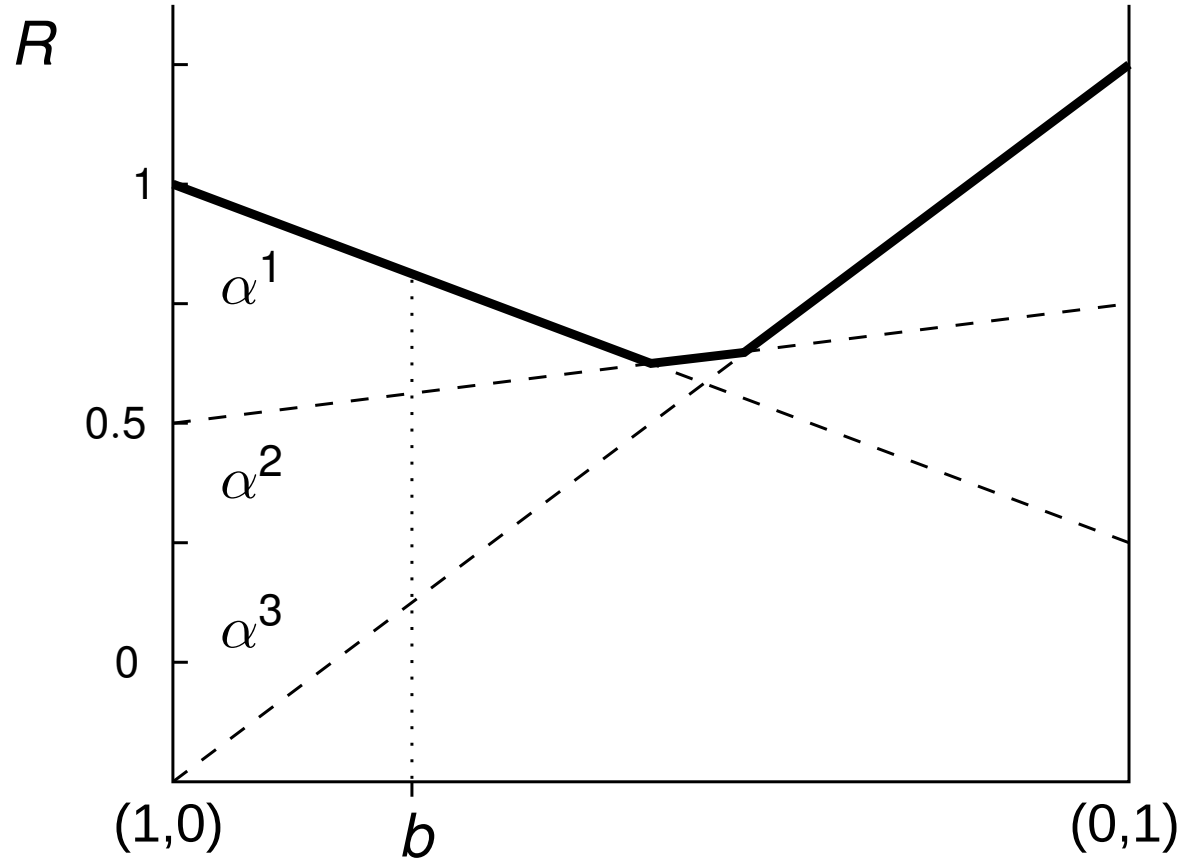
Solving POMDPs

- A solution to a POMDP is a **policy**, i.e., a mapping $\pi : \Delta \mapsto A$ from beliefs to actions.
- The optimal value V^* of a POMDP satisfies the Bellman optimality equation $V^* = HV^*$:

$$V^*(b) = \max_a \left[R(b,a) + \gamma \sum_o p(o|b,a) V^*(b_a^o) \right]$$

- Value iteration repeatedly applies $V_{n+1} = HV_n$ starting from an initial V_0.
- Computing the optimal value function is a hard problem (PSPACE-complete for finite horizon, undecidable for infinite horizon).

Example V_0



$R(s,a)$	a_1	a_2	a_3
s_1	1.00	0.50	-0.25
s_2	0.25	0.75	1.25

PWLC shape of V_n

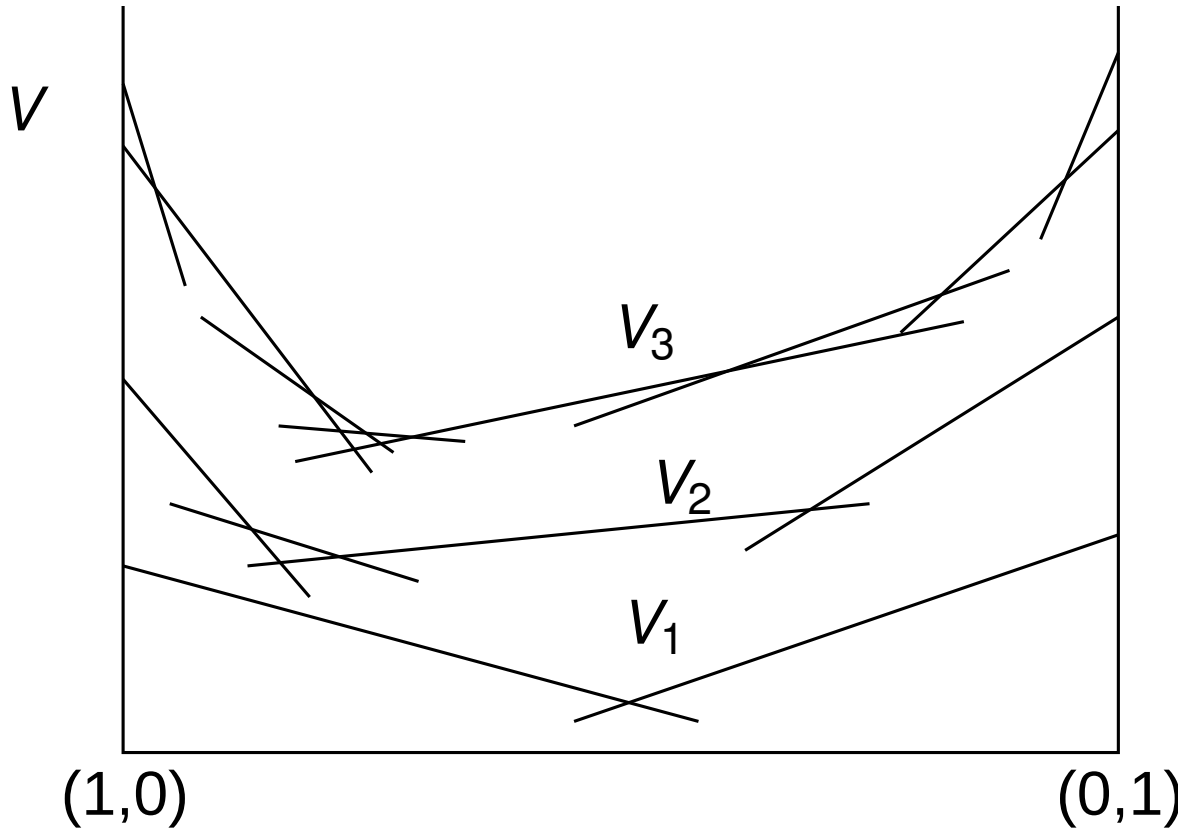
- Like V_0, V_n is as well piecewise linear and convex.
- Rewards $R(b, a) = b \cdot R(s, a)$ are linear functions of b. Note that the value of a point b satisfies:

$$V_{n+1}(b) = \max_a \left[b \cdot R(s, a) + \gamma \sum_o p(o|b, a) V_n(b_a^o) \right]$$

  which involves a maximization over (at least) the vectors $R(s, a)$.
- Intuitively: less uncertainty about the state (low-entropy beliefs) means better decisions (thus higher value).

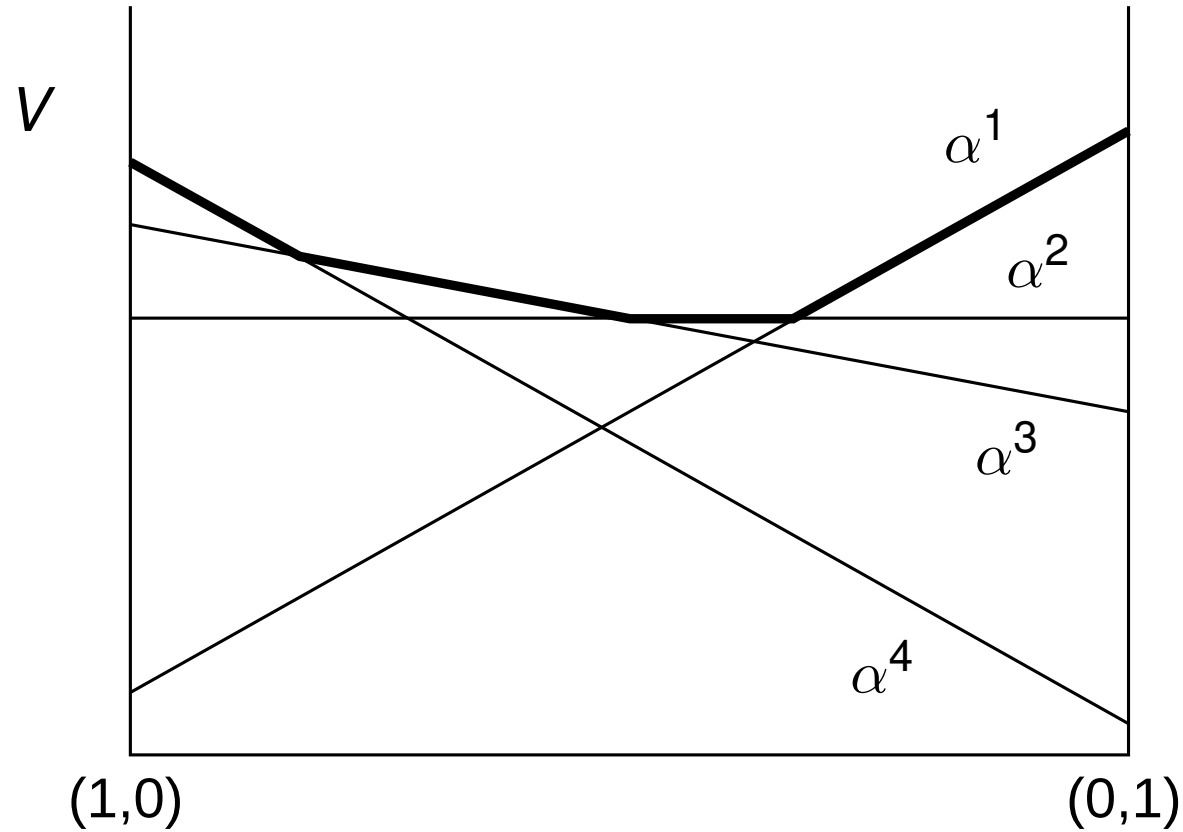
Exact value iteration

Value iteration computes a sequence of value function estimates $V_1, V_2, \ldots, V_n$, using the POMDP backup operator H, $V_{n+1} = HV_n$.

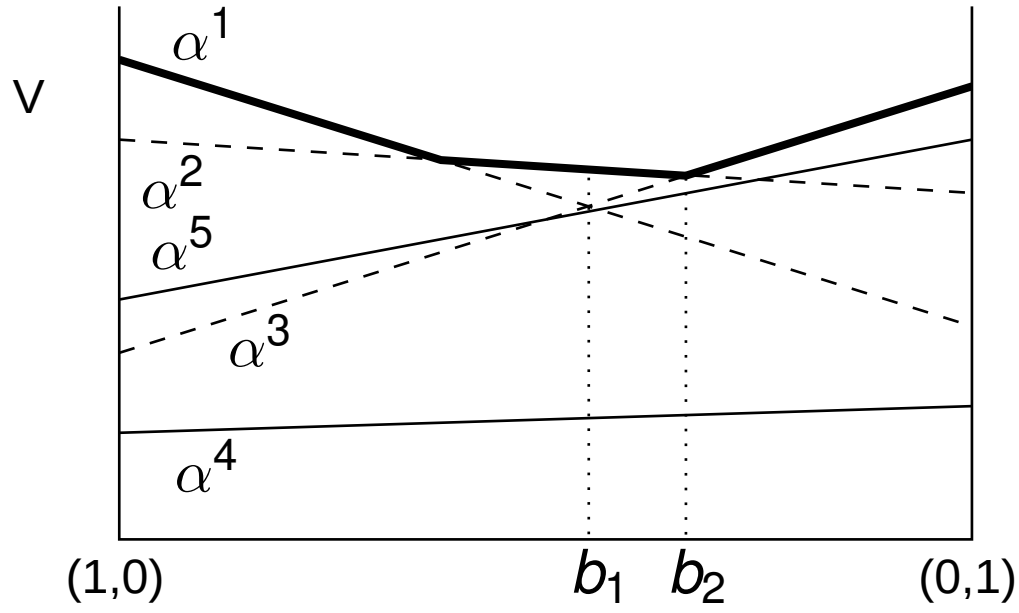


Optimal value functions

The optimal value function of a (finite-horizon) POMDP is piecewise linear and convex: $V(b) = \max_\alpha b \cdot \alpha$.



Vector pruning



Linear program for pruning:

variables: $\forall s \in S, b(s); x$
maximize: x
subject to:

$$b \cdot (\alpha - \alpha') \geq x, \forall \alpha' \in V, \alpha' \neq \alpha$$
$$b \in \Delta(S)$$

Optimal POMDP methods

Enumerate and prune:

- Most straightforward: Monahan (1982)'s enumeration algorithm. Generates a maximum of $|A||V_n|^{|O|}$ vectors at each iteration, hence requires pruning.
- Incremental pruning (Zhang and Liu, 1996; Cassandra et al., 1997).

Search for witness points:

- One Pass (Sondik, 1971; Smallwood and Sondik, 1973).
- Relaxed Region, Linear Support (Cheng, 1988).
- Witness (Cassandra et al., 1994).

Sub-optimal techniques

- Grid-based approximations

  (Drake, 1962; Lovejoy, 1991; Brafman, 1997; Zhou and Hansen, 2001; Bonet, 2002).
- Optimizing finite-state controllers

  (Platzman, 1981; Hansen, 1998b; Poupart and Boutilier, 2004).
- Heuristic search in the belief tree

  (Satia and Lave, 1973; Hansen, 1998a).
- Compression or clustering

  (Roy et al., 2005; Poupart and Boutilier, 2003; Virin et al., 2007).
- Point-based techniques

  (Pineau et al., 2003; Smith and Simmons, 2004; Spaan and Vlassis, 2005; Shani et al., 2007; Kurniawati et al., 2008).
- Monte Carlo tree search

  (Silver and Veness, 2010).

Point-based backup

- For finite horizon V^* is piecewise linear and convex, and for infinite horizons V^* can be approximated arbitrary well by a PWLC value function (Smallwood and Sondik, 1973).

Point-based backup

- For finite horizon V^* is piecewise linear and convex, and for infinite horizons V^* can be approximated arbitrary well by a PWLC value function (Smallwood and Sondik, 1973).
- Given value function V_n and a particular belief point b we can easily compute the vector α_{n+1}^b of HV_n such that

$$\alpha_{n+1}^b = \arg\max_{\{\alpha_{n+1}^k\}_k} b \cdot \alpha_{n+1}^k,$$

where $\{\alpha_{n+1}^k\}_{k=1}^{|HV_n|}$ is the (unknown) set of vectors for HV_n. We will denote this operation $\alpha_{n+1}^b = \texttt{backup}(b)$.

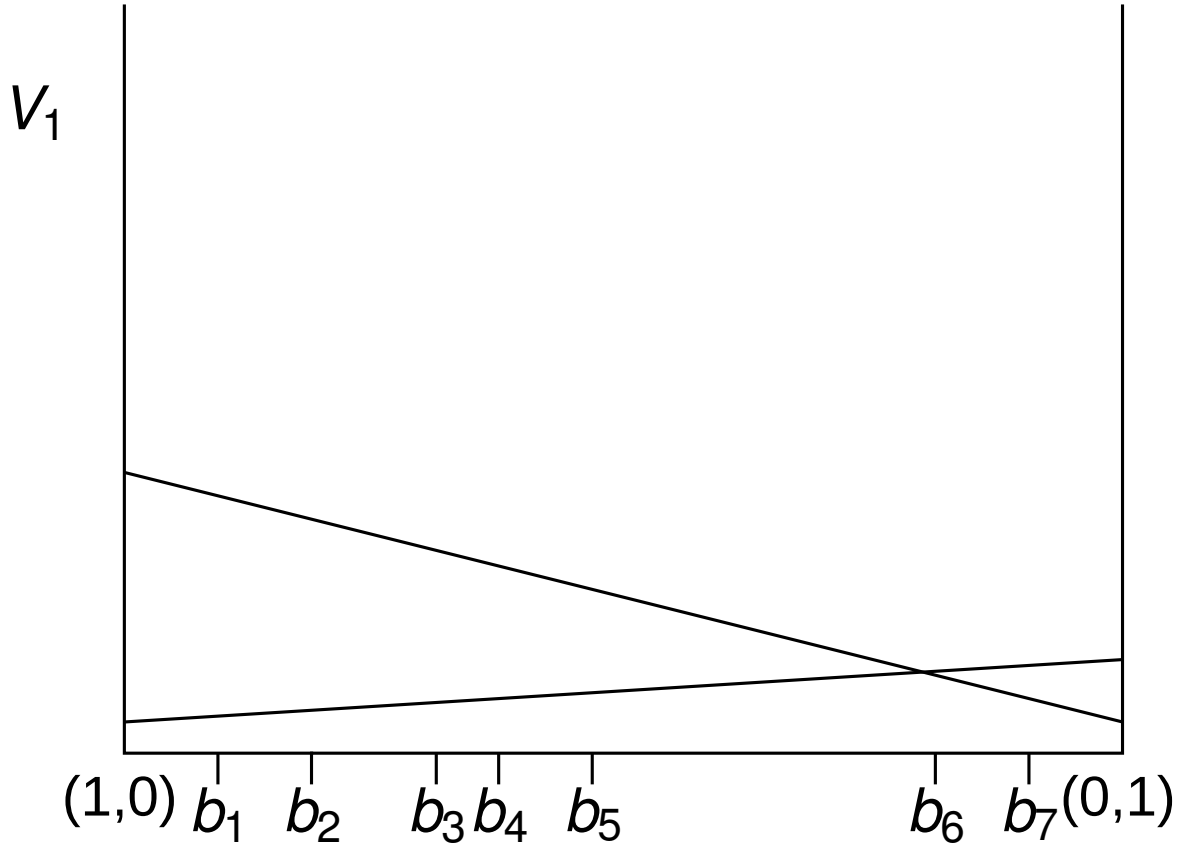
Point-based (approximate) methods

Point-based (approximate) value iteration plans only on a limited set of **reachable** belief points:

1. Let the robot explore the environment.
2. Collect a set B of belief points.
3. Run approximate value iteration on B.

PERSEUS: randomized point-based VI

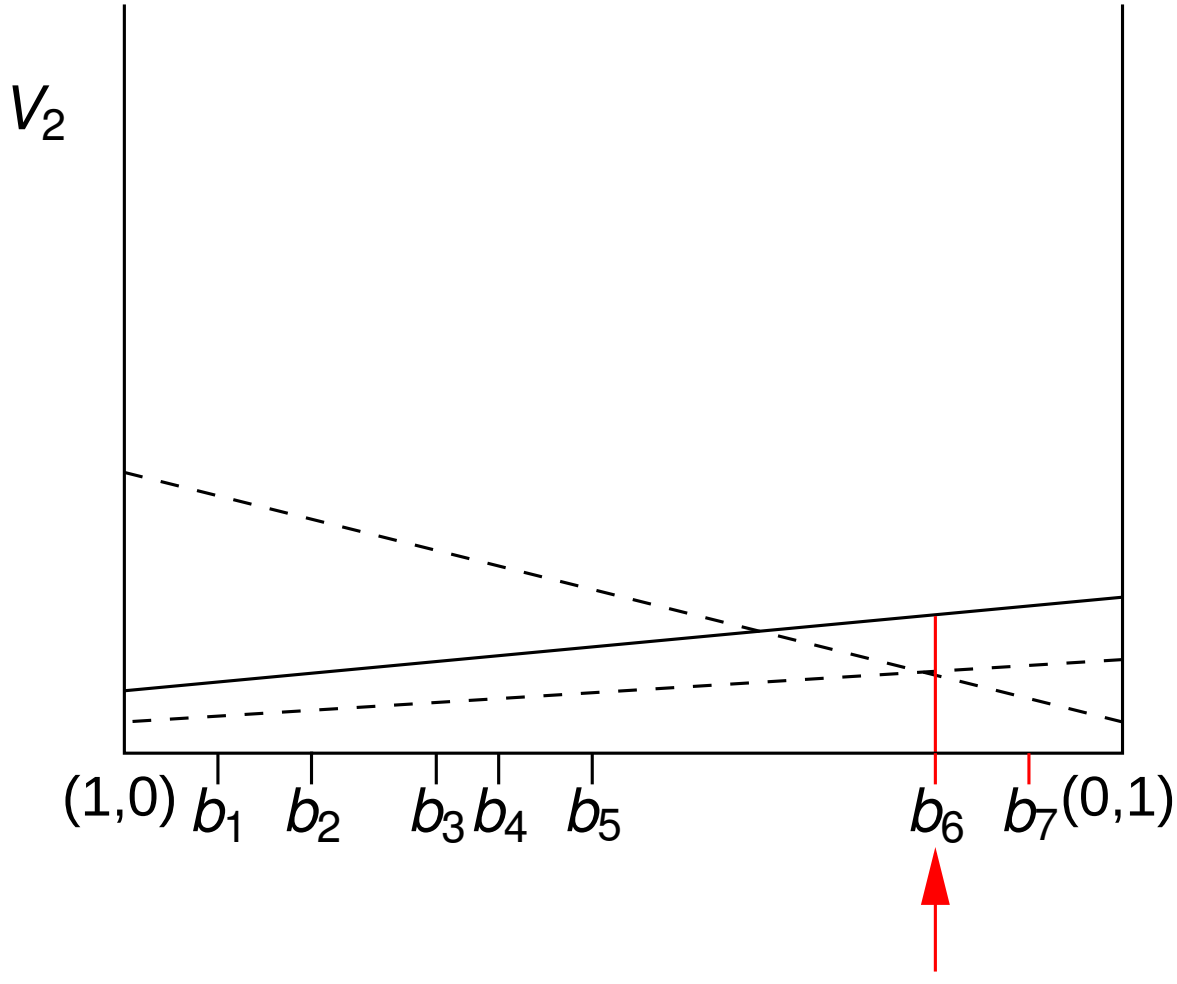
Idea: at every backup stage improve the value of all $b \in B$.



(Spaan and Vlassis, 2005)

PERSEUS: randomized point-based VI

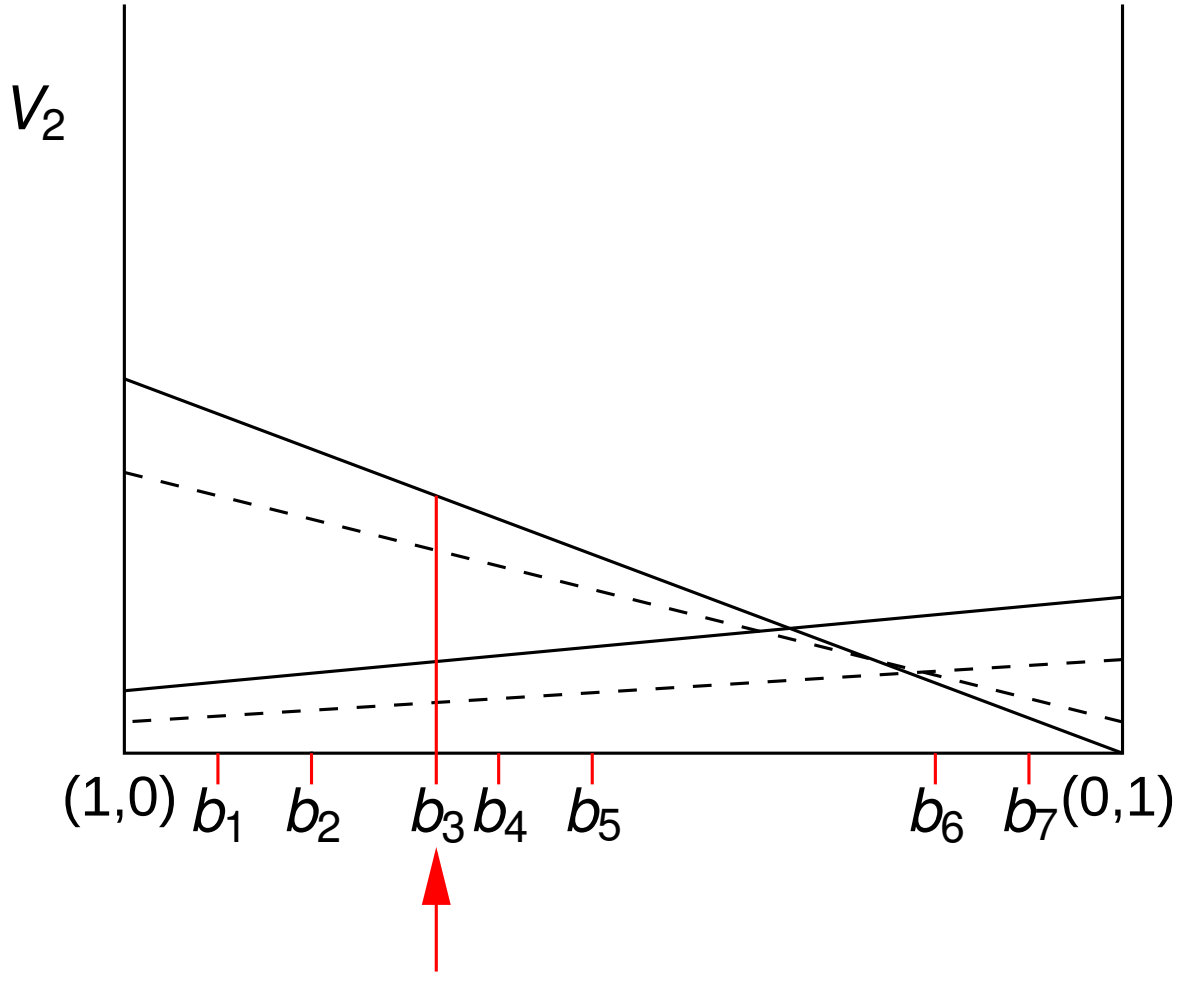
Idea: at every backup stage improve the value of all $b \in B$.



(Spaan and Vlassis, 2005)

PERSEUS: randomized point-based VI

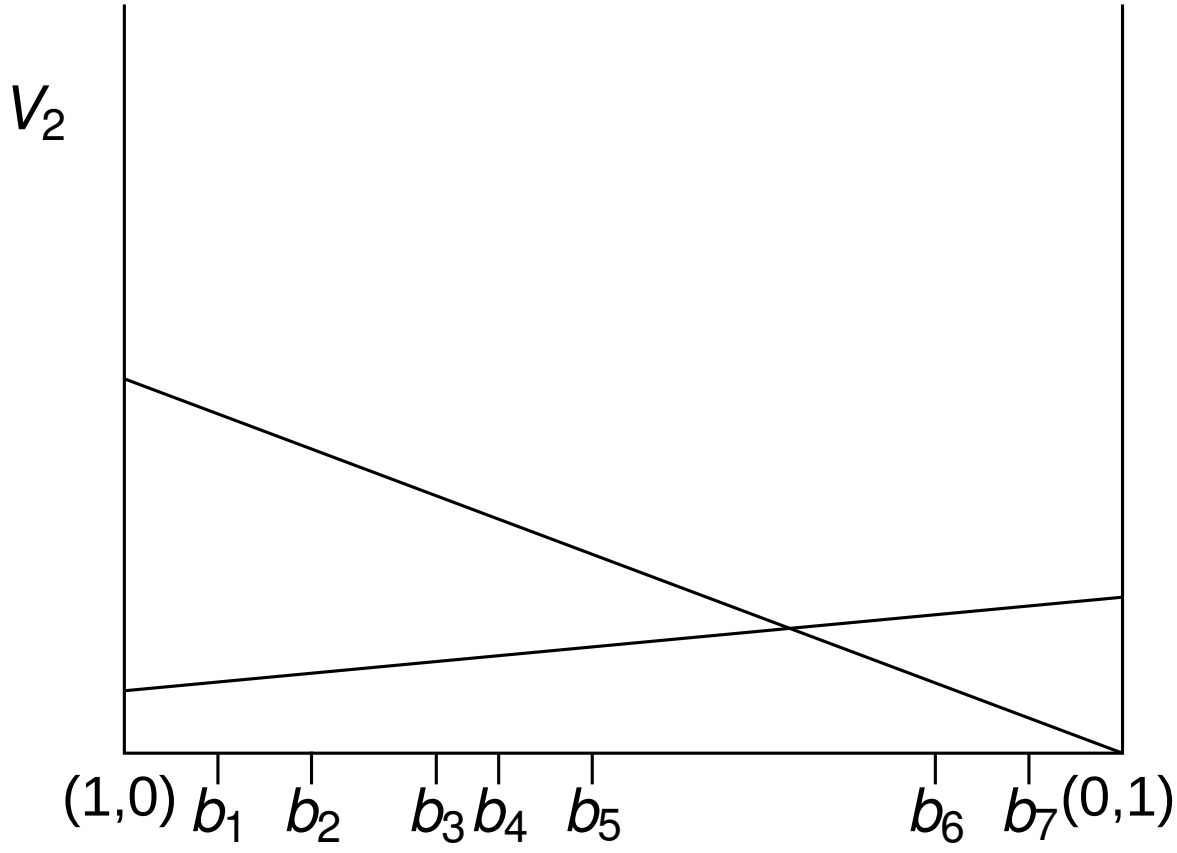
Idea: at every backup stage improve the value of all $b \in B$.



(Spaan and Vlassis, 2005)

PERSEUS: randomized point-based VI

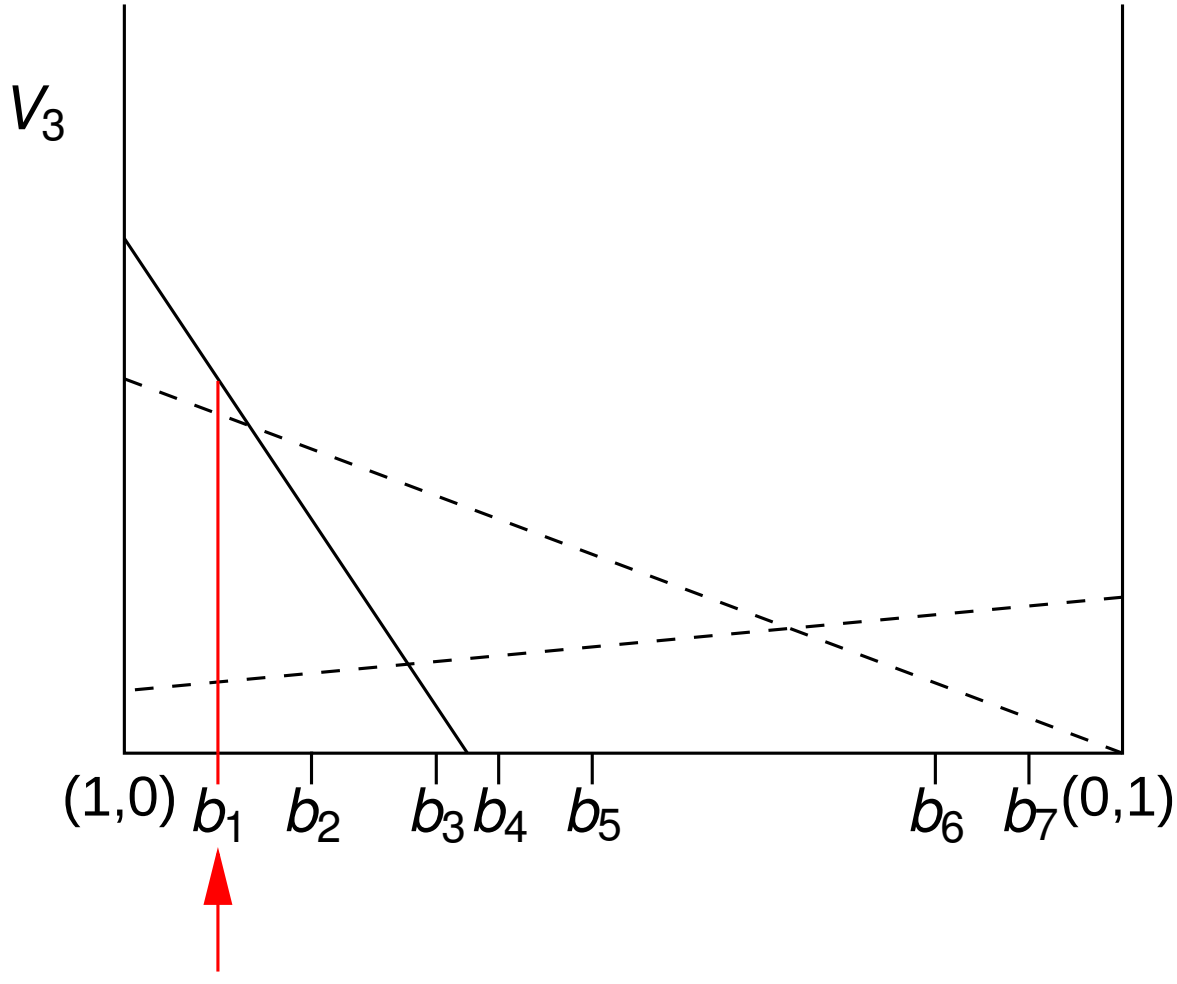
Idea: at every backup stage improve the value of all $b \in B$.



(Spaan and Vlassis, 2005)

PERSEUS: randomized point-based VI

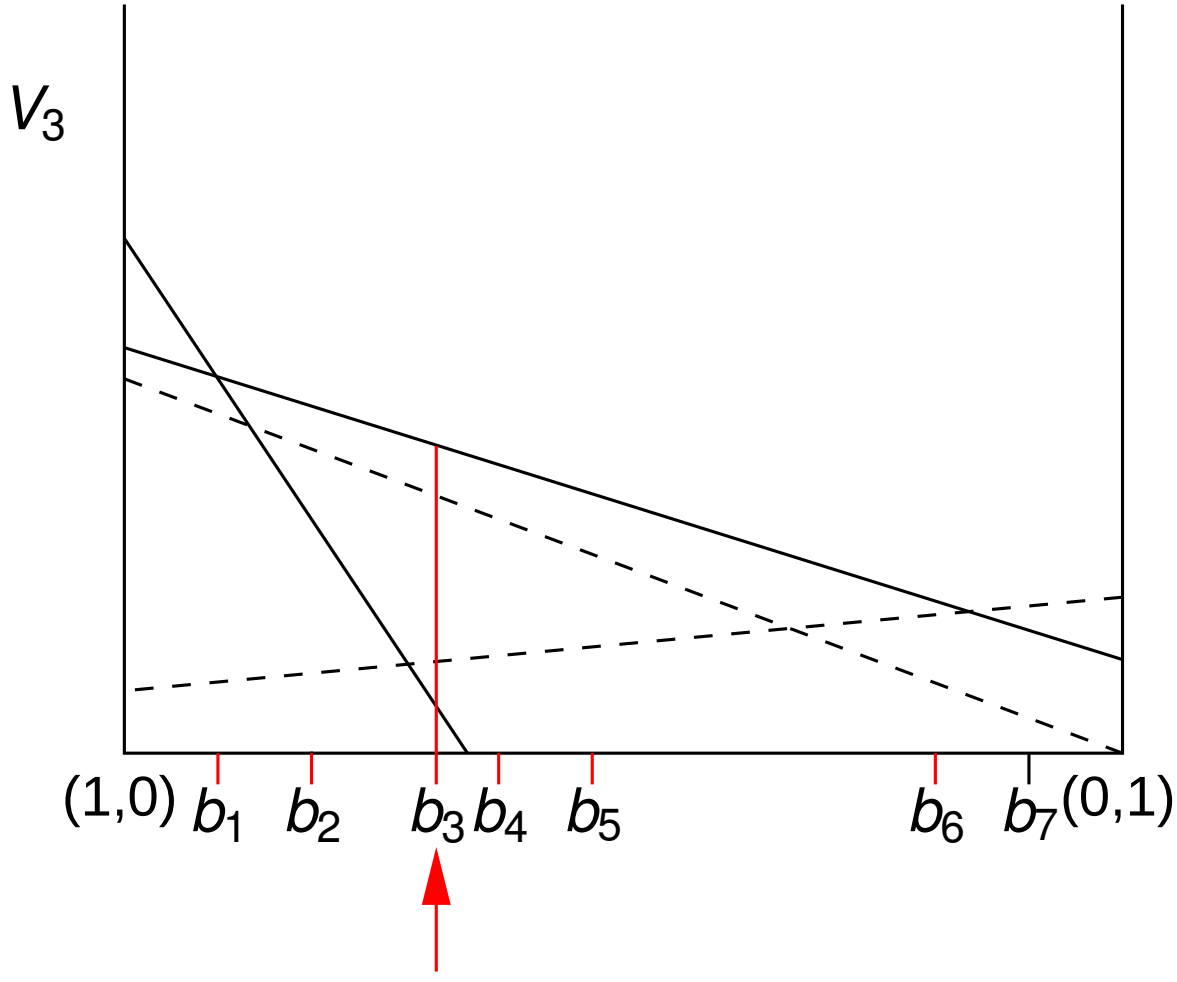
Idea: at every backup stage improve the value of all $b \in B$.



(Spaan and Vlassis, 2005)

PERSEUS: randomized point-based VI

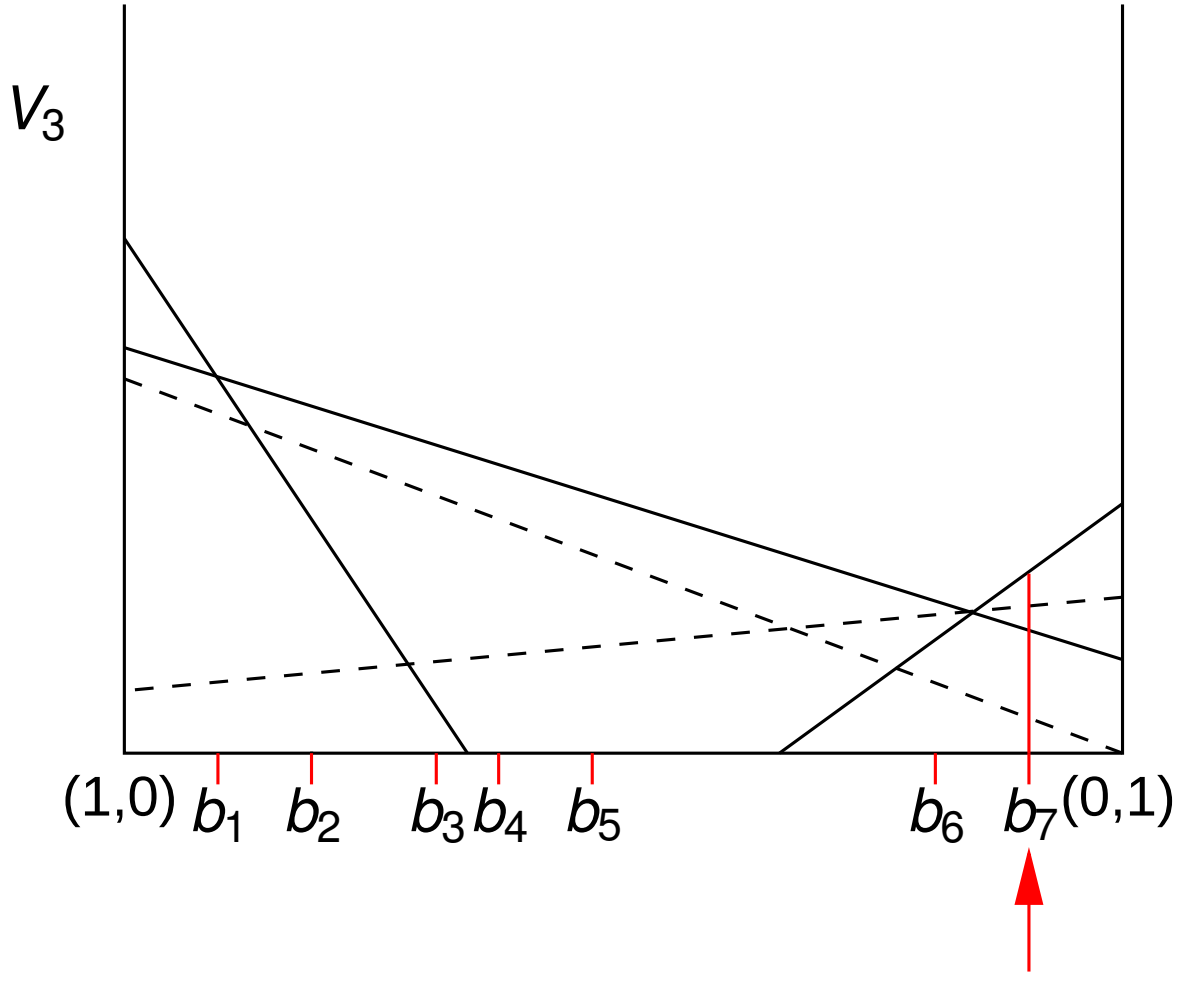
Idea: at every backup stage improve the value of all $b \in B$.



(Spaan and Vlassis, 2005)

PERSEUS: randomized point-based VI

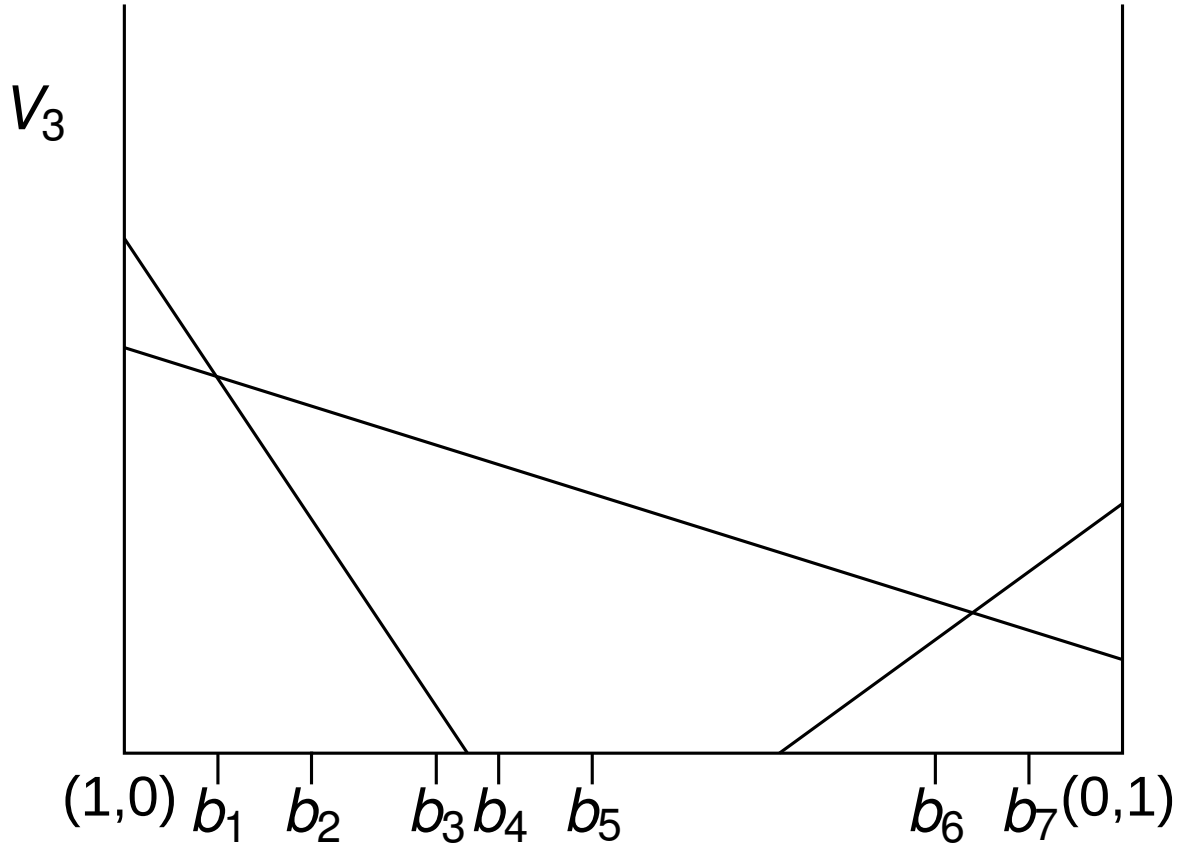
Idea: at every backup stage improve the value of all $b \in B$.



(Spaan and Vlassis, 2005)

PERSEUS: randomized point-based VI

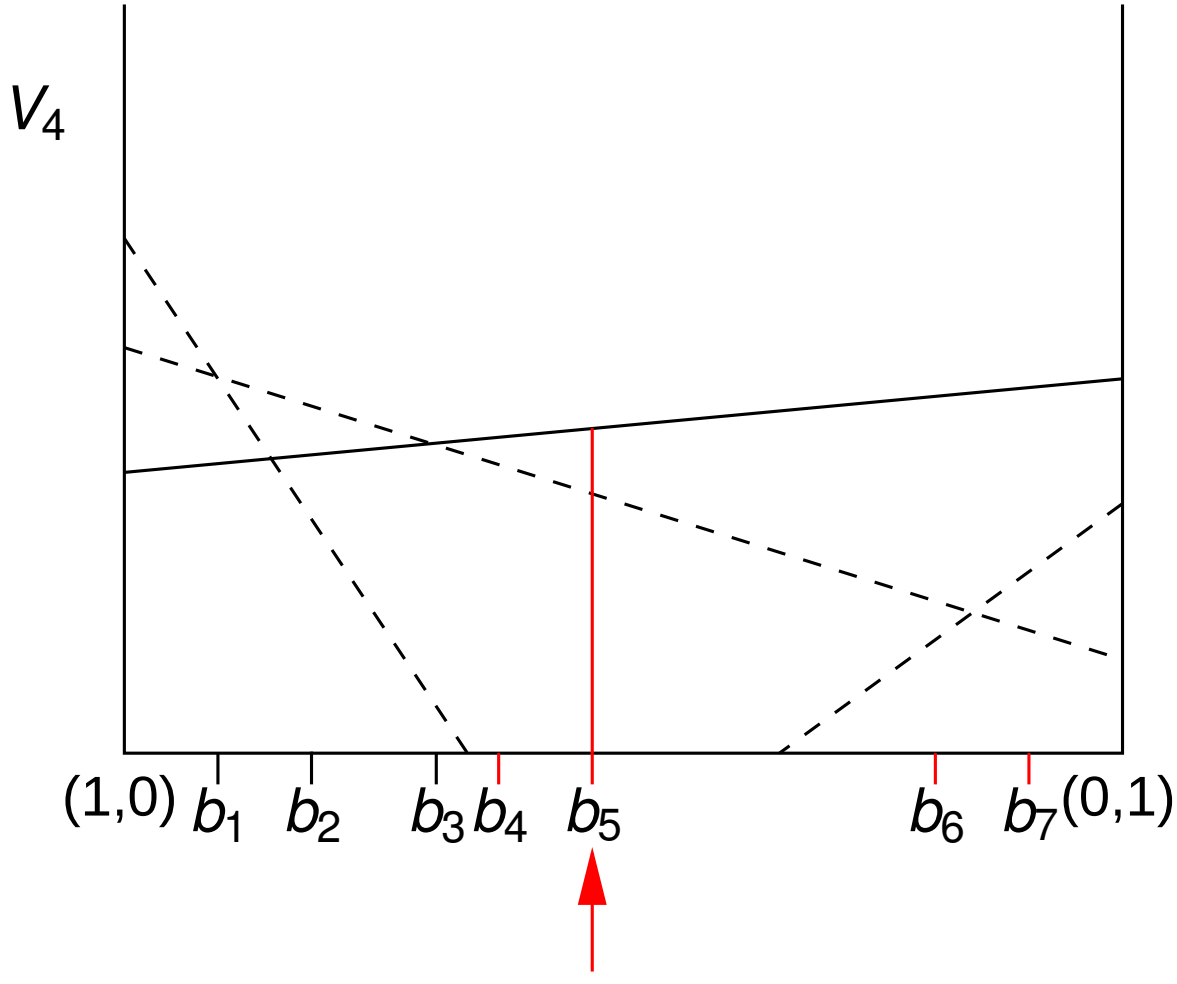
Idea: at every backup stage improve the value of all $b \in B$.



(Spaan and Vlassis, 2005)

PERSEUS: randomized point-based VI

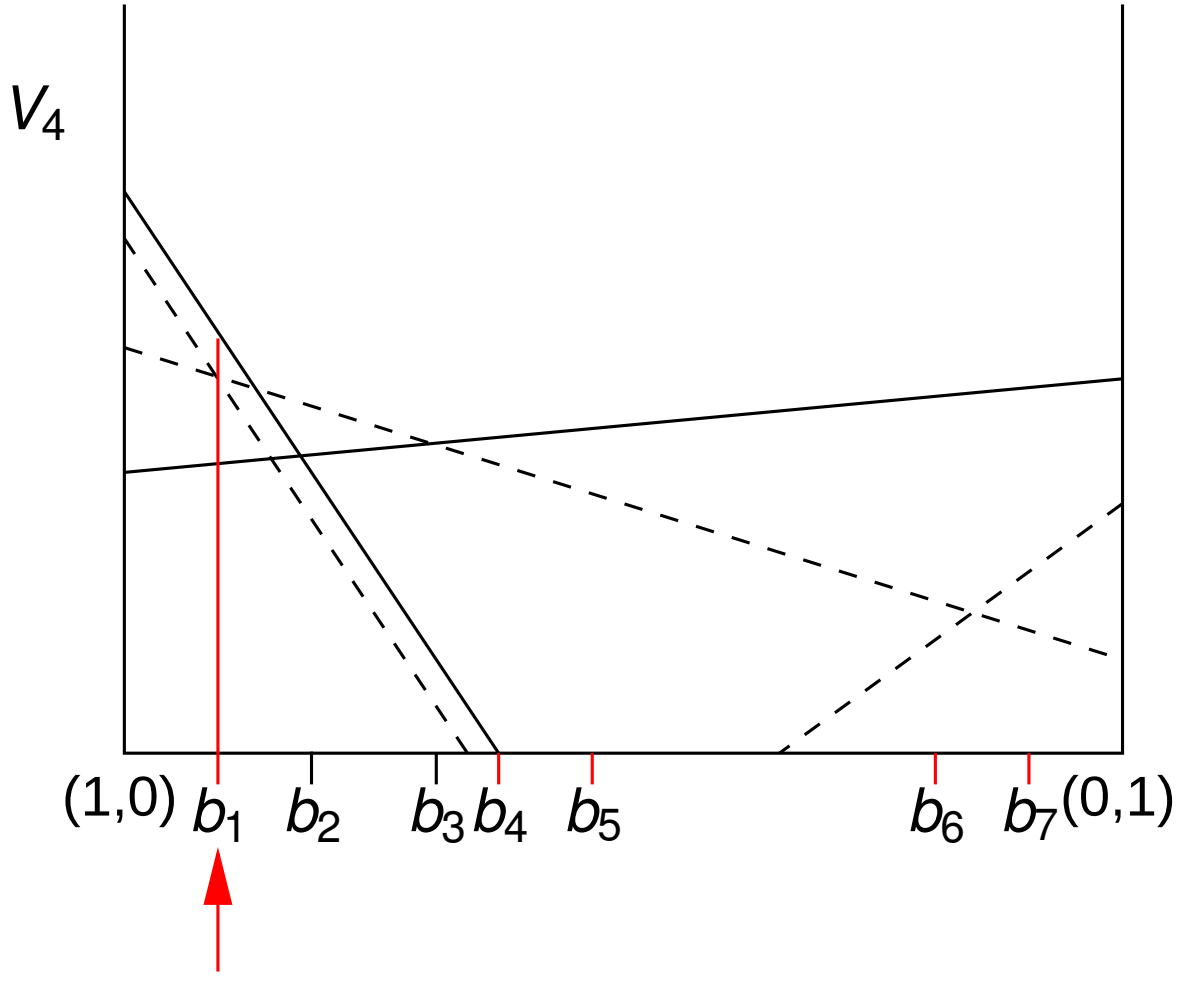
Idea: at every backup stage improve the value of all $b \in B$.



(Spaan and Vlassis, 2005)

PERSEUS: randomized point-based VI

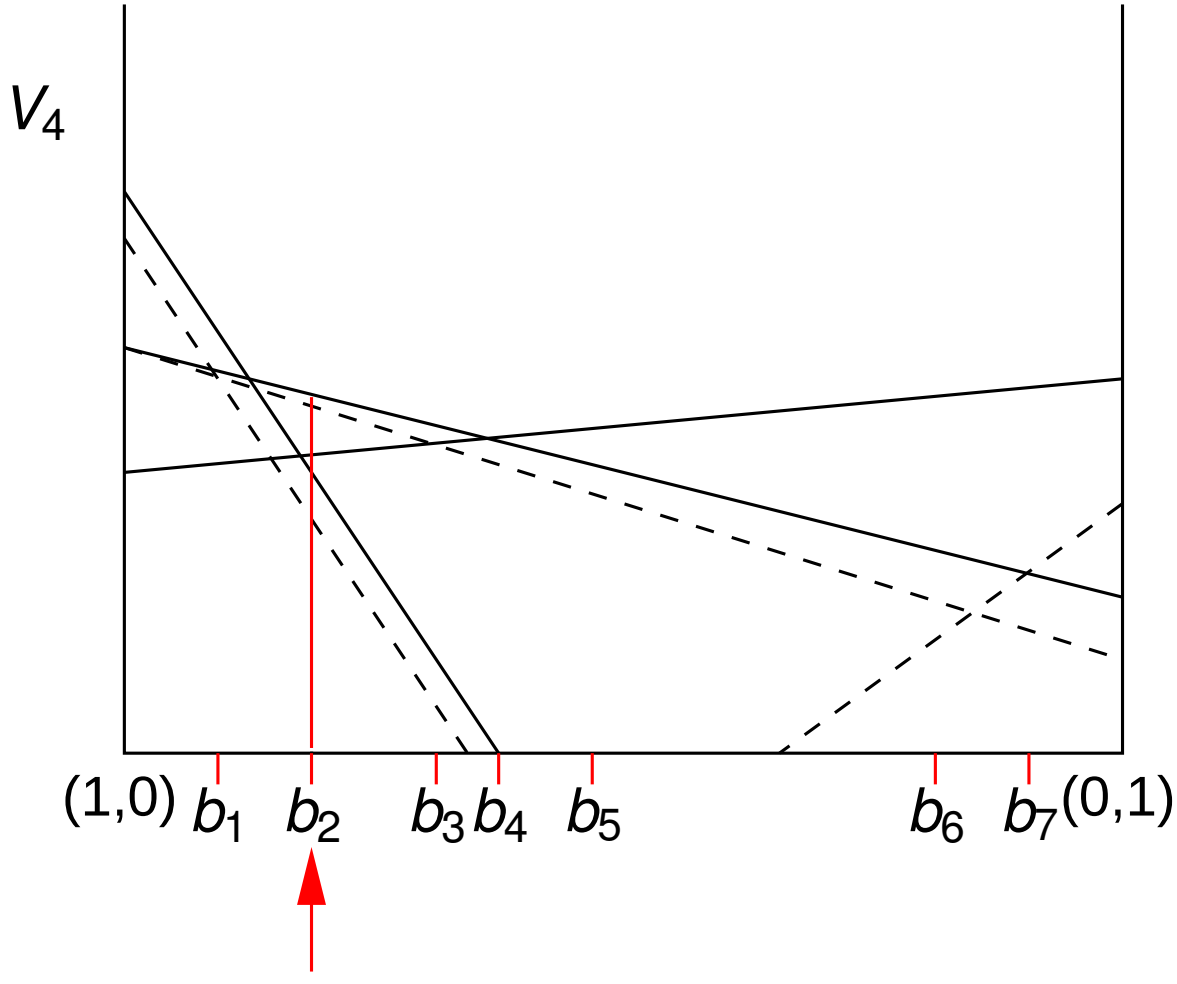
Idea: at every backup stage improve the value of all $b \in B$.



(Spaan and Vlassis, 2005)

PERSEUS: randomized point-based VI

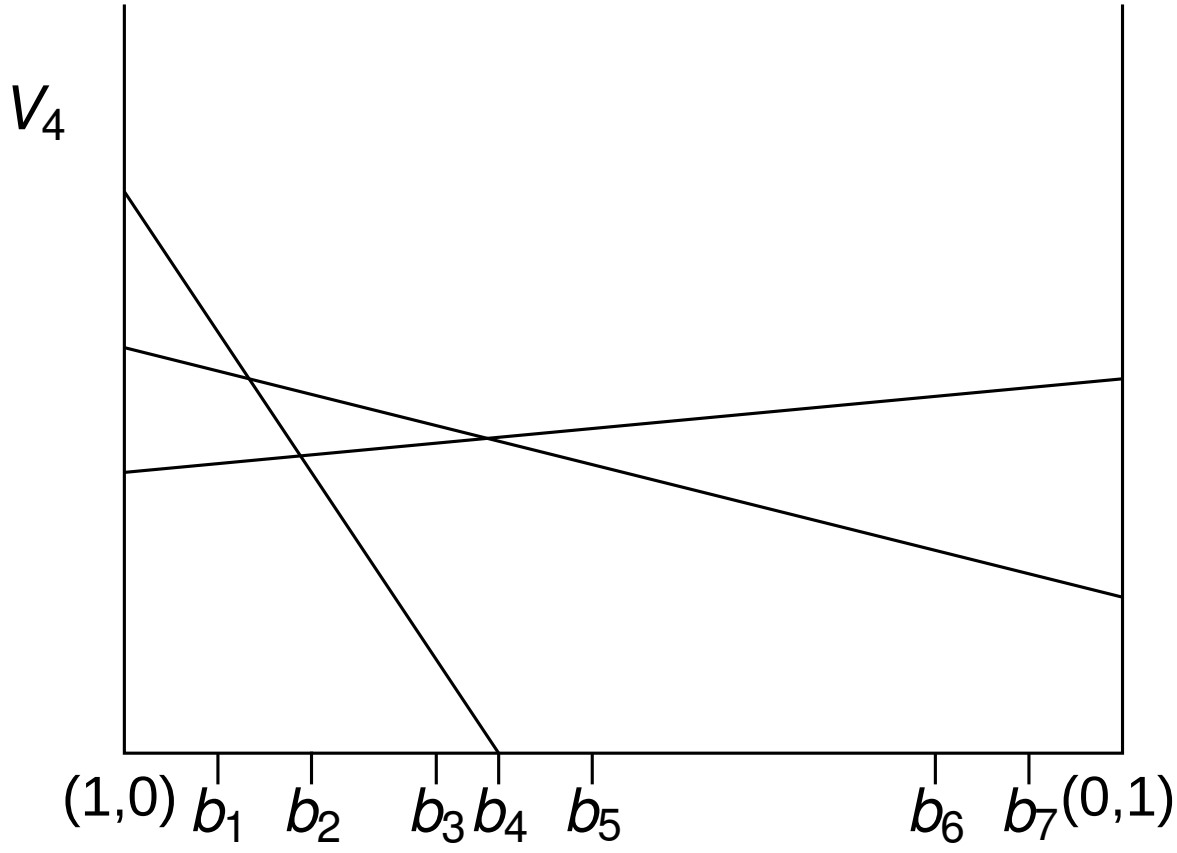
Idea: at every backup stage improve the value of all $b \in B$.



(Spaan and Vlassis, 2005)

PERSEUS: randomized point-based VI

Idea: at every backup stage improve the value of all $b \in B$.



(Spaan and Vlassis, 2005)

Sub-optimal techniques

- Grid-based approximations

  (Drake, 1962; Lovejoy, 1991; Brafman, 1997; Zhou and Hansen, 2001; Bonet, 2002).
- Optimizing finite-state controllers

  (Platzman, 1981; Hansen, 1998b; Poupart and Boutilier, 2004).
- Heuristic search in the belief tree

  (Satia and Lave, 1973; Hansen, 1998a).
- Compression or clustering

  (Roy et al., 2005; Poupart and Boutilier, 2003; Virin et al., 2007).
- Point-based techniques

  (Pineau et al., 2003; Smith and Simmons, 2004; Spaan and Vlassis, 2005; Shani et al., 2007; Kurniawati et al., 2008).
- Monte Carlo tree search

  (Silver and Veness, 2010).

Multiagent Planning

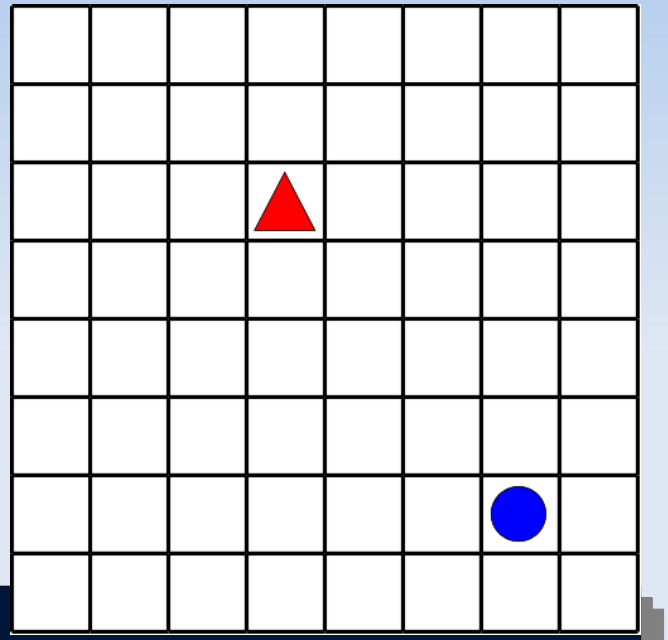
Multiagent Systems (MASs)

Why MASs?

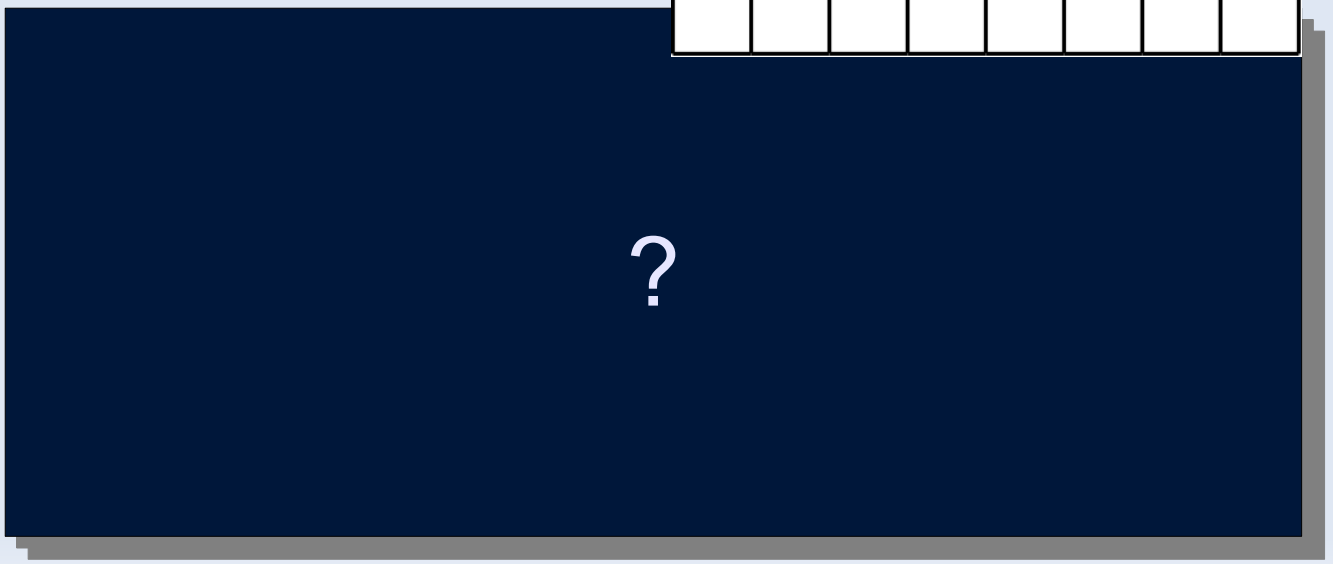
- If we can make intelligent agents, soon there will be many...
- Physically distributed systems: centralized solutions expensive and brittle.
- can potentially provide [Vlassis, 2007,Sycara, 1998]
  - Speedup and efficiency
  - Robustness and reliability (‘graceful degradation’)
  - Scalability and flexibility (adding additional agents)

Example: Predator-Prey Domain

- Predator-Prey domain – still single agent!
  - 1 agent: the predator (blue)
  - prey (red) is part of the environment
  - on a torus ('wrap around world')

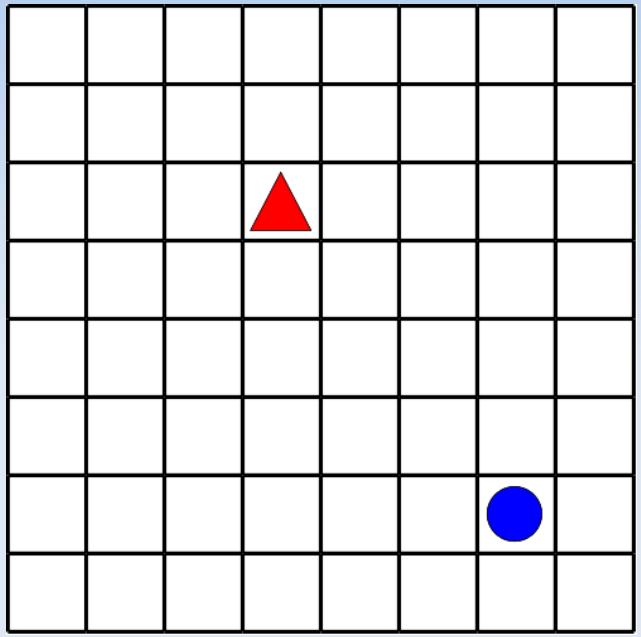


- Formalization:
  - states
  - actions
  - transitions
  - rewards



Example: Predator-Prey Domain

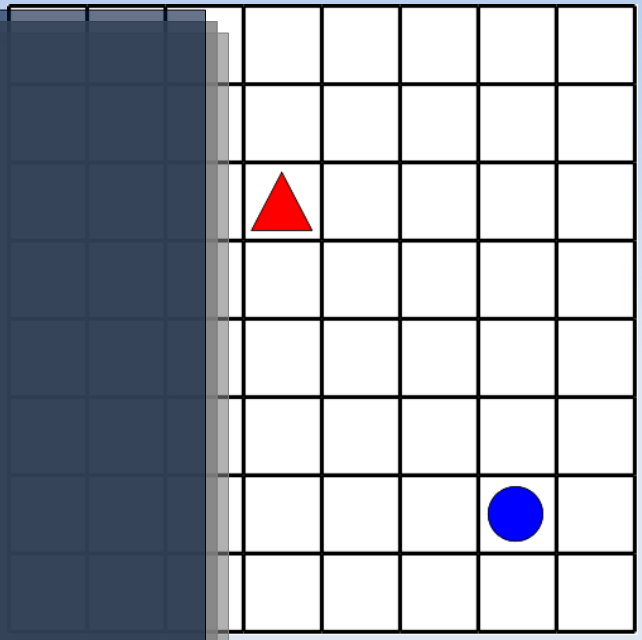
- Predator-Prey domain
  - 1 agent: the predator (blue)
  - prey (red) is part of the environment
  - on a torus ('wrap around world')



- Formalization:
  - states (-3,4)
  - actions N,W,S,E
  - transitions probability of failing to move, prey moves
  - rewards reward for capturing

Example: Predator-Prey Domain

- Predator-Prey domain
  - 1 agent: the predator (blue)
  - prey (red) is part of the environment
  - on a torus ('wrap around world')
  - Formalization:
    - states (-3,4)
    - actions N,W,S,E
    - transitions probability of failing to move, prey moves
    - rewards reward for capturing



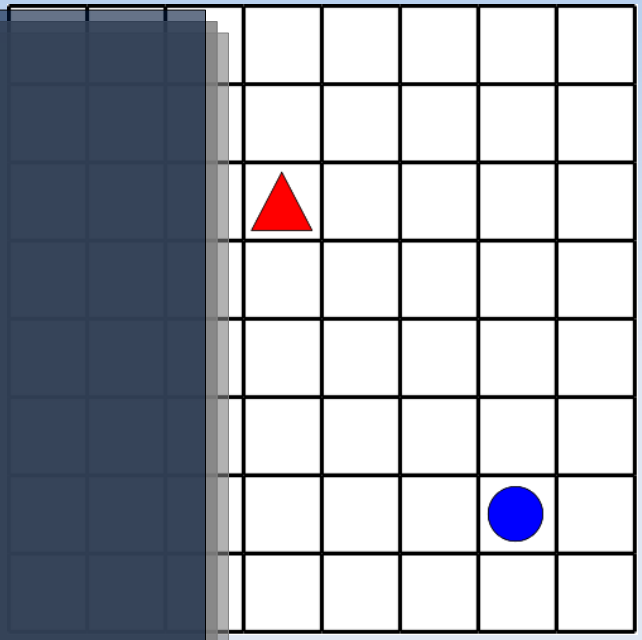
Markov decision process (MDP)

Example: Predator-Prey Domain

- Predator-Prey domain
  - 1 agent: the predator (blue)
  - prey (red) is part of the environment
  - on a torus ('wrap around world')
  - states (-3,4)
  - actions N,W,S,E
  - transitions probability of failing to move, prey moves
  - rewards reward for capturing

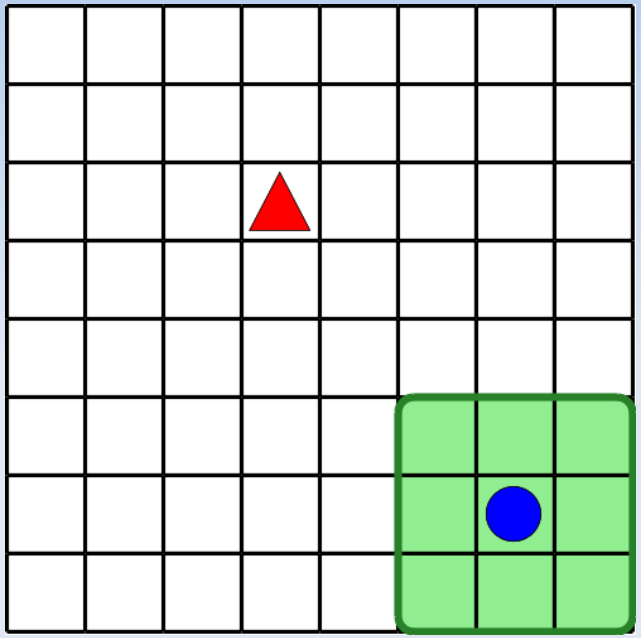
Markov decision process (MDP)

- Markovian state *s*...
- ...which is observed
- policy π maps states → actions
- Value function Q(s,a)
- Value iteration: way to compute it.



Partial Observability

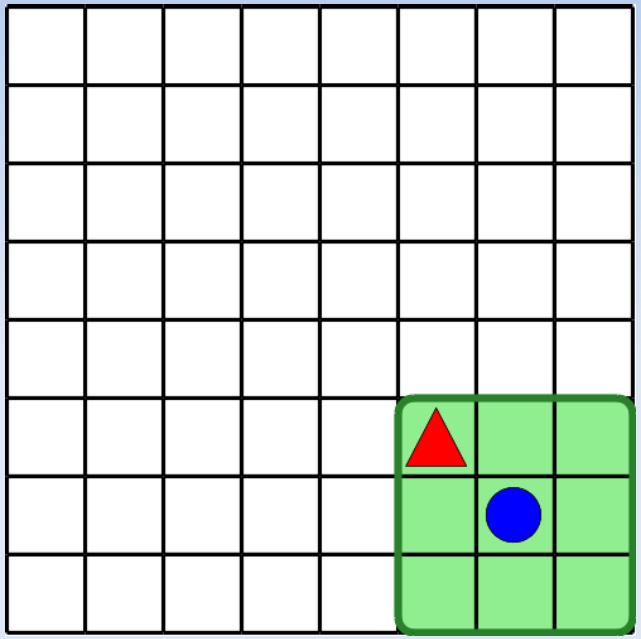
- Now: partial observability
  - E.g., limited range of sight
- MDP + observations
  - explicit observations
  - observation probabilities
    - noisy observations (detection probability)



$o = \text{'nothing'}$

Partial Observability

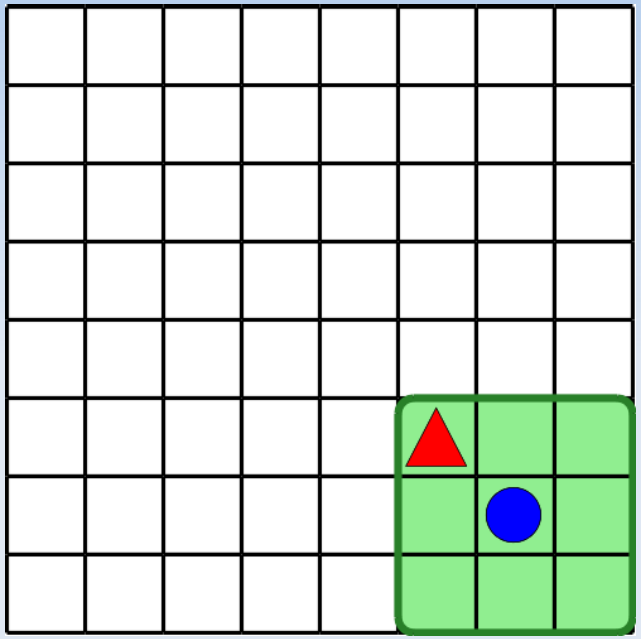
- Now: partial observability
  - E.g., limited range of sight
- MDP + observations
  - explicit observations
  - observation probabilities
    - noisy observations (detection probability)



$o = (-1, 1)$

Partial Observability

- Now: partial observability
  - E.g., limited range of sight
- MDP + observations
  - explicit observations
  - observation probabilities
    - noisy observations (detection probability)

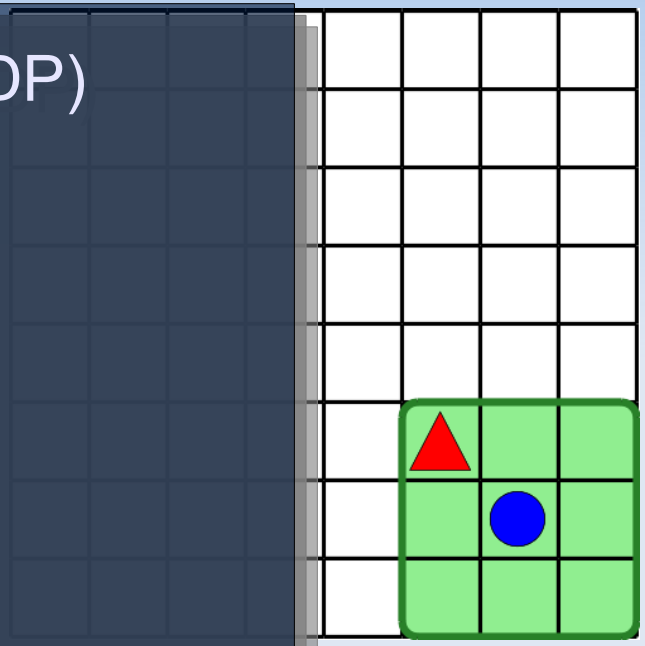


$o=(-1,1)$

Can not observe the state
→ Need to maintain a belief over states *b(s)*
→ Policy maps beliefs to actions $\pi(b)=a$

Partial Observability

- Now: partial observability
  - E.g., Partially Observable MDP (POMDP)
- MDP + observations
  - explicit observations
  - observation probabilities
    - noisy observations (detection probability)



$o=(-1,1)$

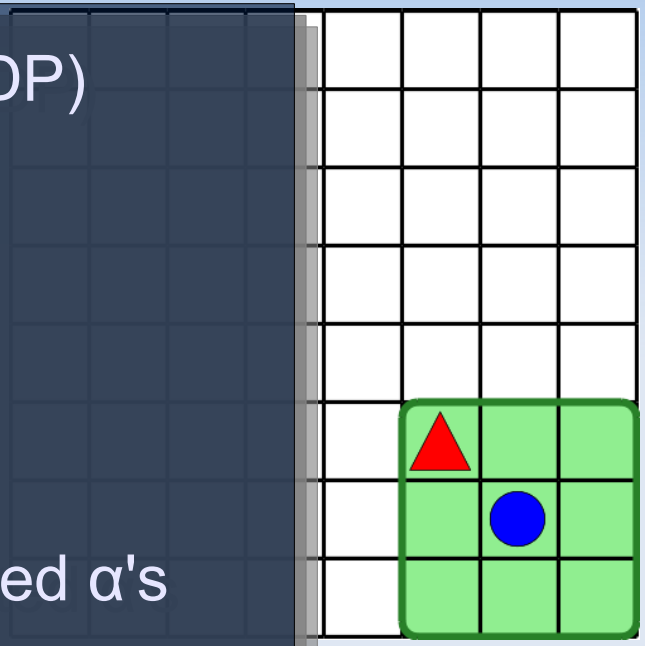
Can not observe the state
→ Need to maintain a belief over states *b(s)*
→ Policy maps beliefs to actions $\pi(b)=a$

Partial Observability

- Now: partial observability

Partially Observable MDP (POMDP)

- reduction → continuous state MDP (in which the belief **is** the state)
- Value iterations:
  - make use of α-vectors (correspond to complete policies)
  - perform pruning: eliminate dominated α's

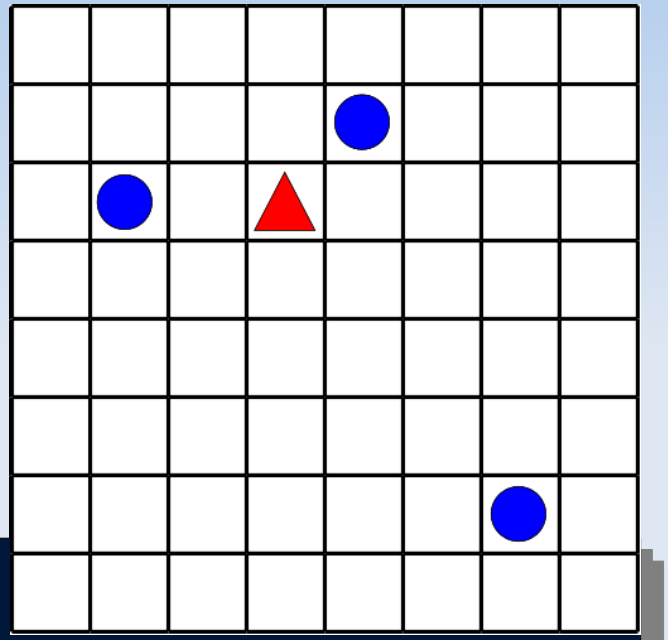


$o=(-1,1)$

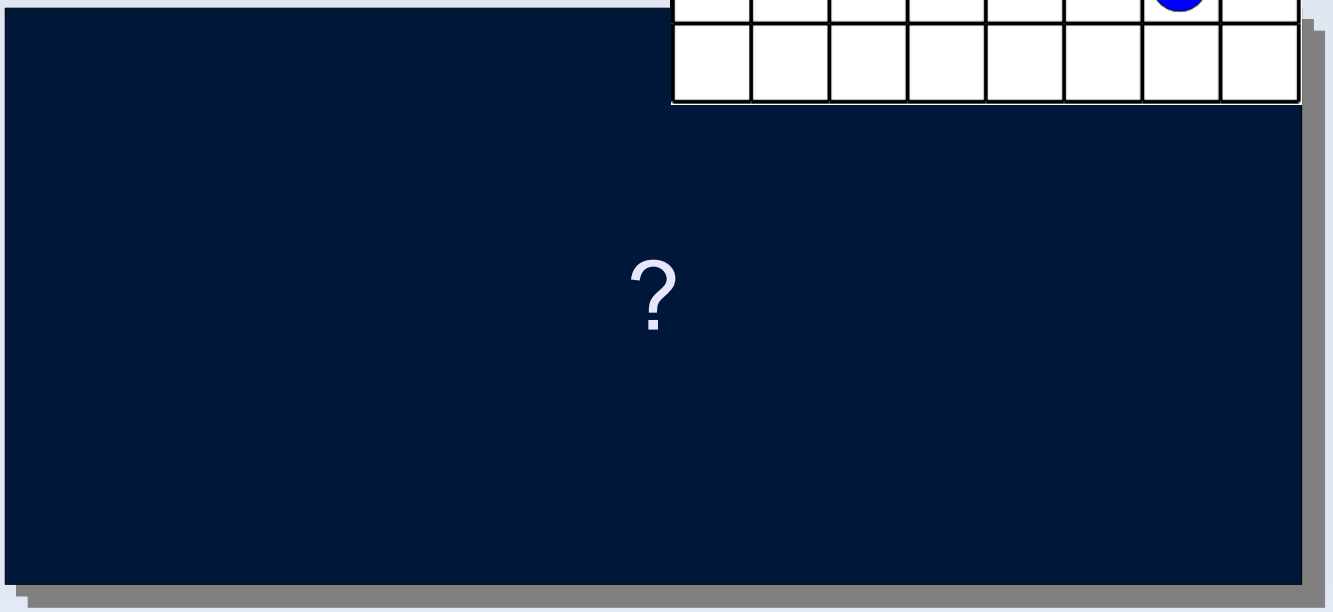
Can not observe the state
→ Need to maintain a belief over states *b(s)*
→ Policy maps beliefs to actions $\pi(b)=a$

Multiple Agents

- Now: multiple agents
  - fully observable

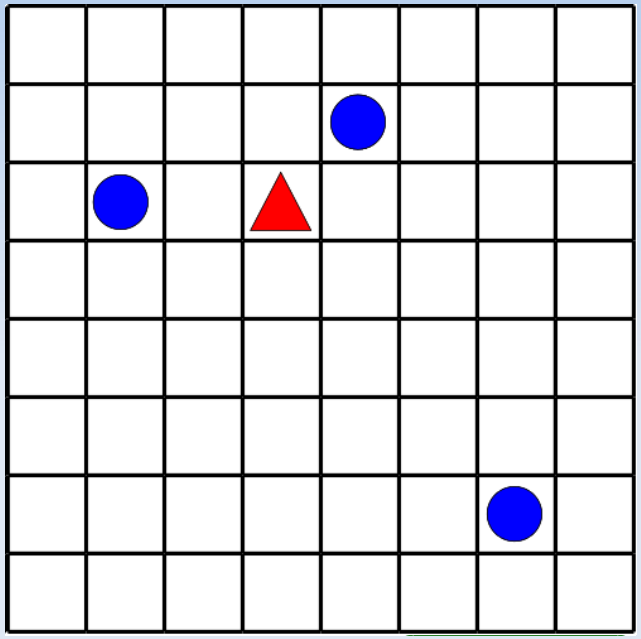


- Formalization:
  - states
  - actions
  - **joint** actions
  - transitions
  - rewards



Multiple Agents

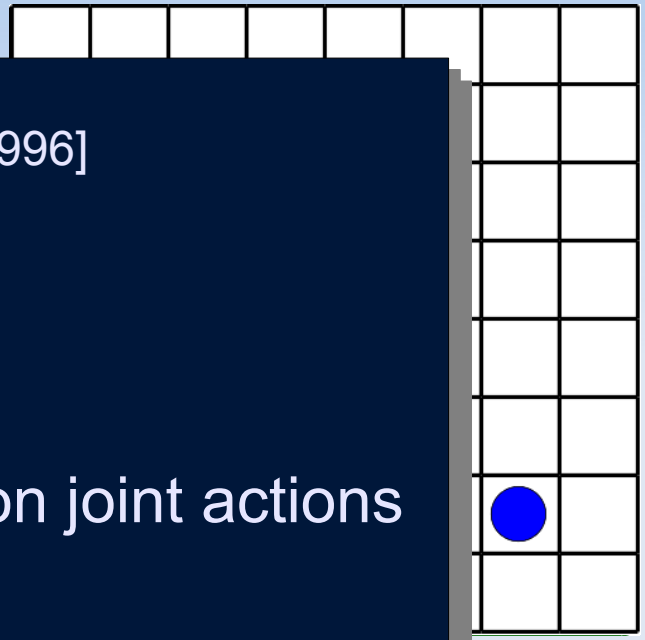
- Now: multiple agents
  - fully observable



- Formalization:
  - states ((3,-4), (1,1), (-2,0))
  - actions {N,W,S,E}
  - **joint** actions {(N,N,N), (N,N,W),...,(E,E,E)}
  - transitions probability of failing to move, prey moves
  - rewards reward for capturing jointly

Multiple Agents

- Now: multiple agents
  - fully observable
- Fo
  -
  -
  -
  -
  - rewards reward for capturing jointly



Multiagent MDP [Boutilier 1996]

- Differences with MDP
  - *n* agents
  - joint actions $a \equiv \langle a_1, a_2, \ldots, a_n \rangle$
  - transitions and rewards depend on joint actions
- Solution:
  - Treat as normal MDP with 1 'puppeteer agent'
  - Optimal policy $\pi(s) \equiv a$
  - Every agent executes its part

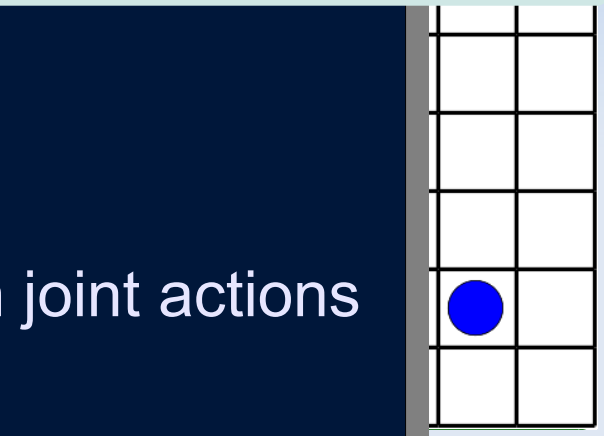
Multiple Agents

- Now: multiple agents
  - fully observable
- Fo
  -
  -
  -
  -
  - rewards reward for capturing jointly

Catch: …?

Multiagent

- Differences with MDP
  - *n* agents
  - joint actions $a \equiv \langle a_1, a_2, \ldots, a_n \rangle$
  - transitions and rewards depend on joint actions
- Solution:
  - Treat as normal MDP with 1 'puppeteer agent'
  - Optimal policy $\pi(s) \equiv a$
  - Every agent executes its part



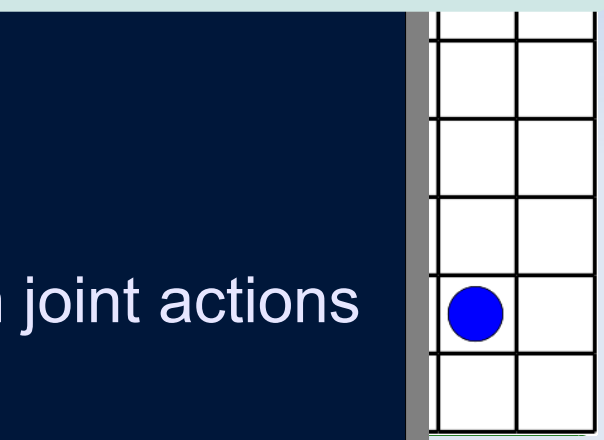
Multiple Agents

- Now: multiple agents
  - fully observable
- Fo
  -
  -
  -
  -
  - rewards reward for capturing jointly

Multiagent MDP

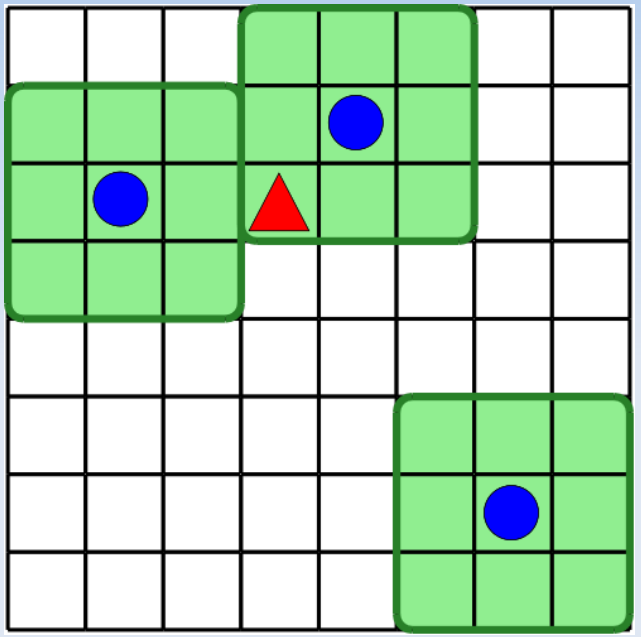
- Differences with MDP
  - *n* agents
  - joint actions $a \equiv \langle a_1, a_2, \ldots, a_n \rangle$
  - transitions and rewards depend on joint actions
- Solution:
  - Treat as normal MDP with 1 'puppeteer agent'
  - Optimal policy $\pi(s) \equiv a$
  - Every agent executes its part

Catch: number of joint actions is **exponential**!
(but other than that, conceptually simple.)



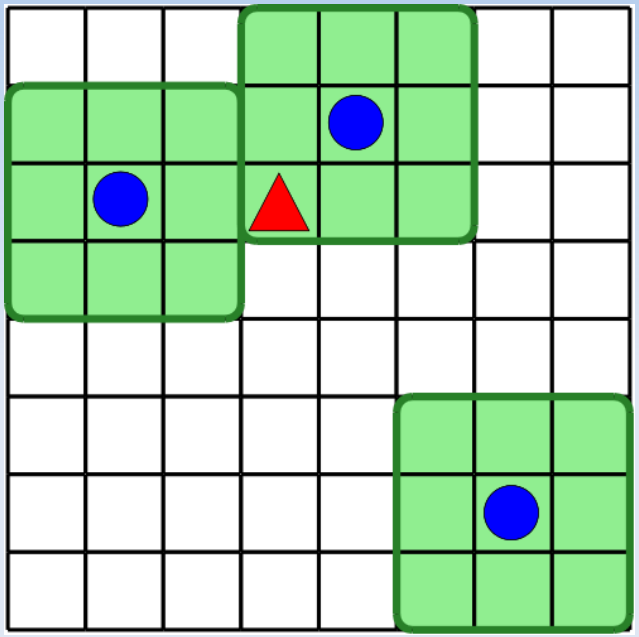
Multiple Agents & Partial Observability

- Now: Both
  - partial observability
  - multiple agents



Multiple Agents & Partial Observability

- Now: Both
  - partial observability
  - multiple agents



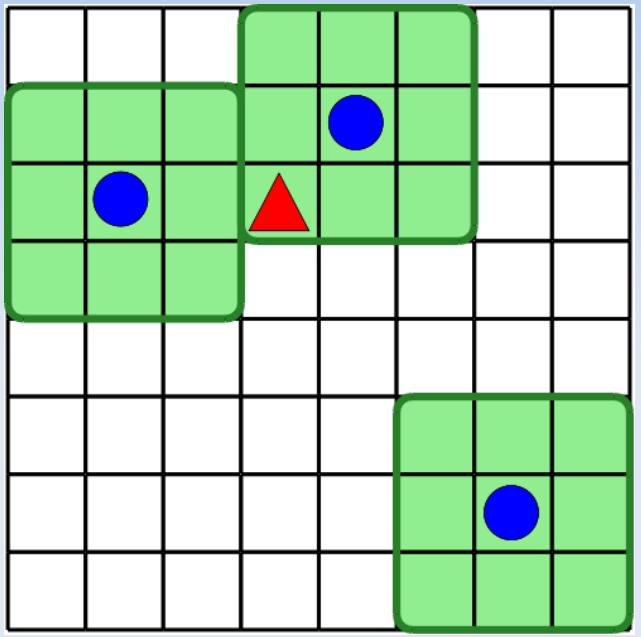
- Decentralized POMDPs (Dec-POMDPs) [Bernstein et al. 2002]

- both
  - joint actions and
  - joint observations

Multiple Agents & Partial Observability

- Again we can make a reduction...

  any idea?

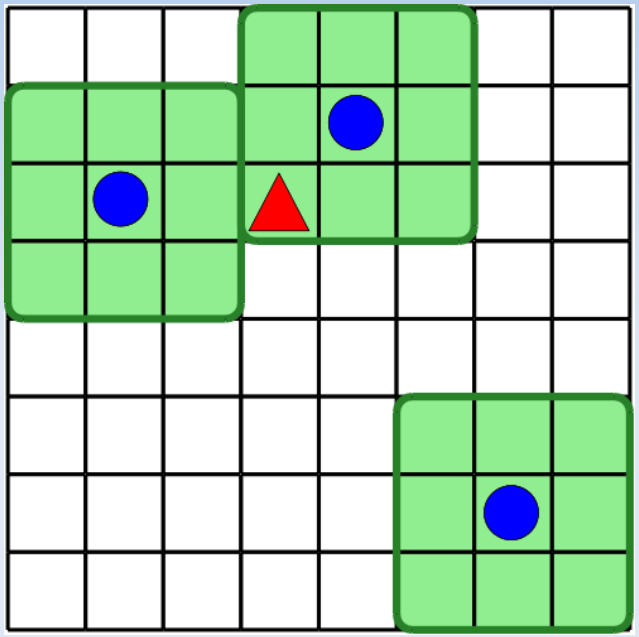


Multiple Agents & Partial Observability

- Again we can make a reduction...

  Dec-POMDPs → MPOMDP

  (multiagent POMDP)

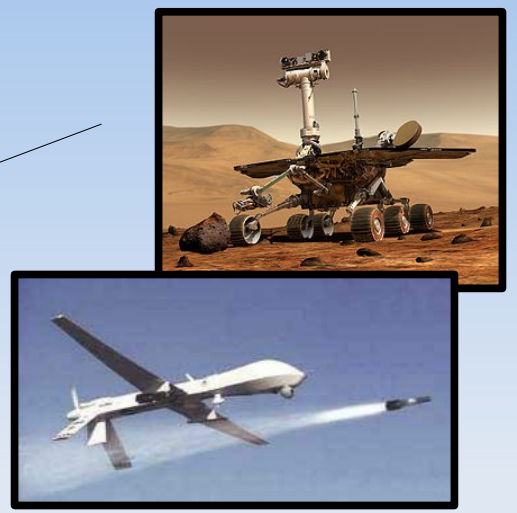


- 'puppeteer' agent that
  - receives joint observations
  - takes joint actions
- requires broadcasting observations!
  - instantaneous, cost-free, noise-free communication → optimal
    [Pynadath and Tambe 2002]
- Without such communication: no easy reduction.

The Dec-POMDP Model

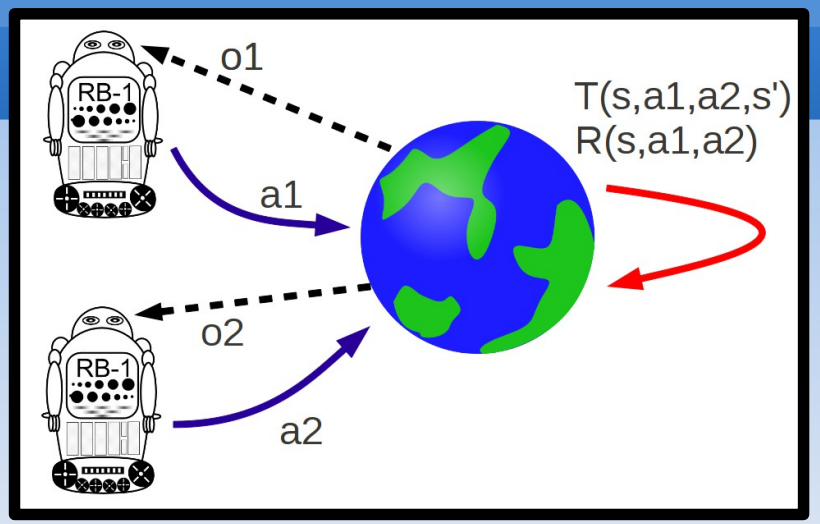
Acting Based On Local Observations

- MPOMDP: Act on global information
- Can be impractical:
  - communication not possible
  - significant cost (e.g battery power)
  - not instantaneous or noise free
  - scales poorly with number of agents!



- Alternative: act based only on local observations
  - Other side of the spectrum: no communication at all
  - (Also other intermediate approaches: delayed communication, stochastic delays)

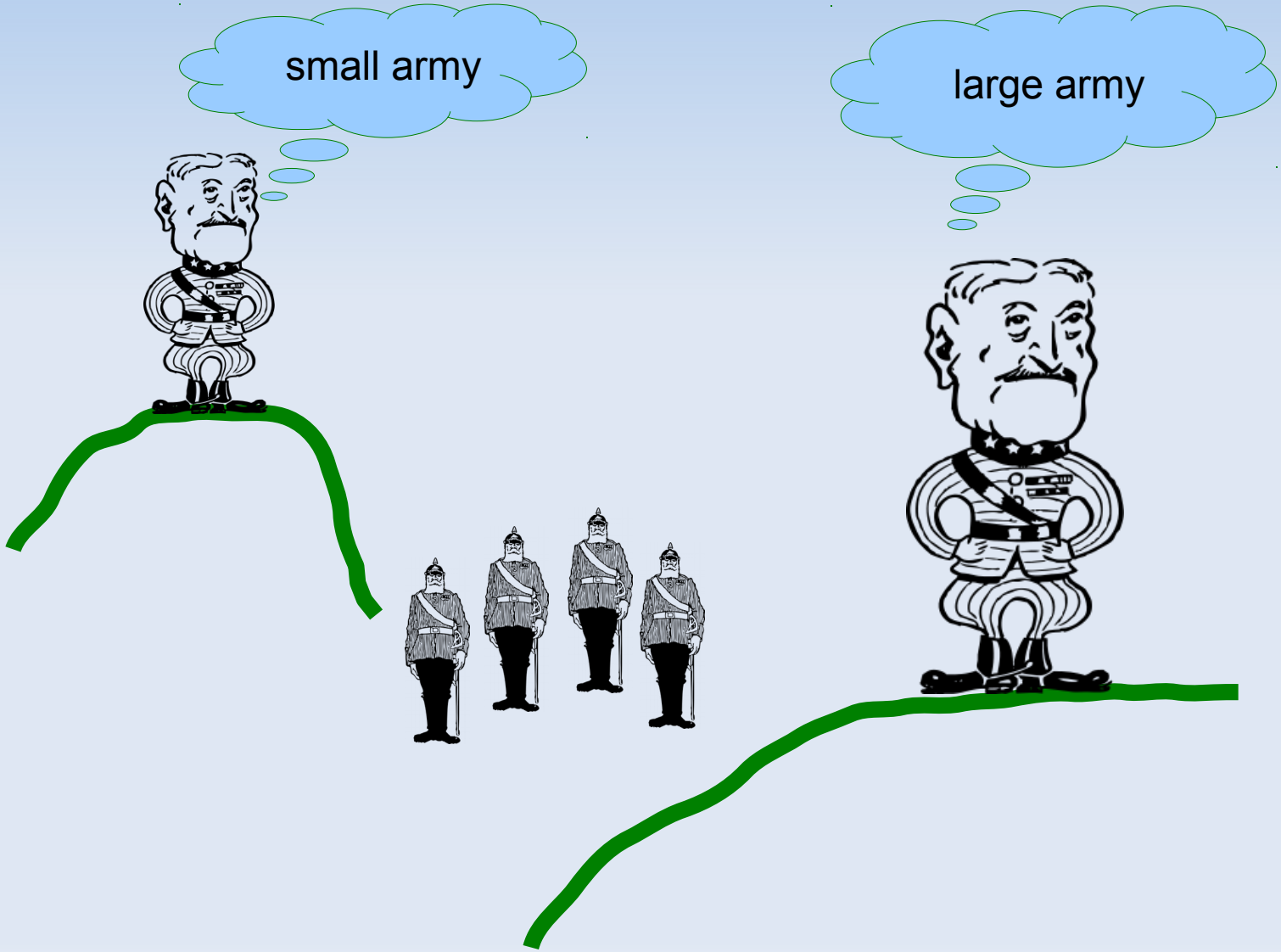
Formal Model



- A Dec-POMDP
  - $\langle S, A, P_T, O, P_O, R, h \rangle$
  - *n* agents
  - *S* – set of states
  - *A* – set of **joint** actions $a = \langle a_1, a_2, \ldots, a_n \rangle$
  - P_T – transition function $P(s'|s,a)$
  - *O* – set of **joint** observations $o = \langle o_1, o_2, \ldots, o_n \rangle$
  - P_O – observation function $P(o|a,s')$
  - *R* – reward function $R(s,a)$
  - *h* – horizon (finite)

Running Example

- 2 generals problem



Running Example

- 2 generals problem

S – { s_L, s_S }

A_i – { (O)bserve, (A)ttack }

O_i – { (L)arge, (S)mall }

Transitions

- Both Observe: no state change
- At least 1 Attack: reset with 50% probability

Observations

- Probability of correct observation: 0.85
- E.g., P(<L, L> | s_L) = 0.85 * 0.85 = 0.7225

Running Example

- 2 generals problem

S – { s_L, s_S }

A_i – { (O)bserve, (A)ttack }

O_i – { (L)arge, (S)mall }

Rewards

- 1 general attacks: he loses the battle
  - R(*,<A,O>) = -10
- Both generals Observe: small cost
  - R(*,<O,O>) = -1
- Both Attack: depends on state
  - R(s_L,<A,A>) = -20
  - R(s_R,<A,A>) = +5

Running Example

- 2 generals problem

S – { s_L, s_S }

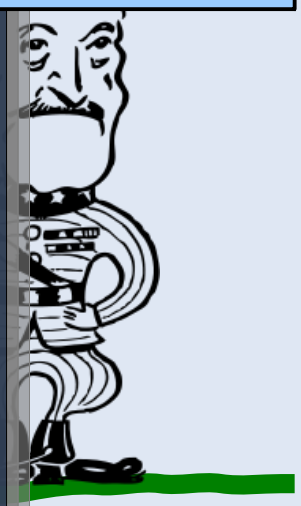
A_i – { (O)bserve, (A)ttack }

O_i – { (L)arge, (S)mall }

Rewards

- 1 general attacks: he loses the battle
  - R(*,<A,O>) = -10
- Both generals Observe: small cost
  - R(*,<O,O>) = -1
- Both Attack: depends on state
  - R(s_L,<A,A>) = -20
  - R(s_R,<A,A>) = +5

suppose h=3,
what do you think is optimal in this problem?



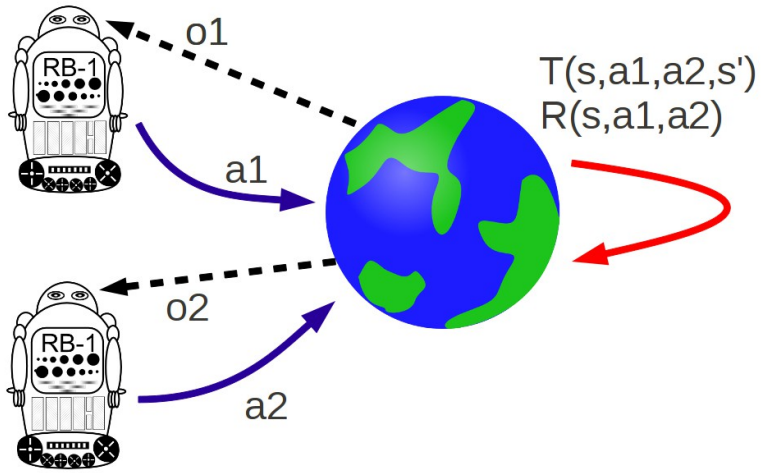
Off-line / On-line phases

- off-line planning, on-line execution is decentralized

Planning Phase

$$\pi = \langle \pi_1, \pi_2 \rangle$$

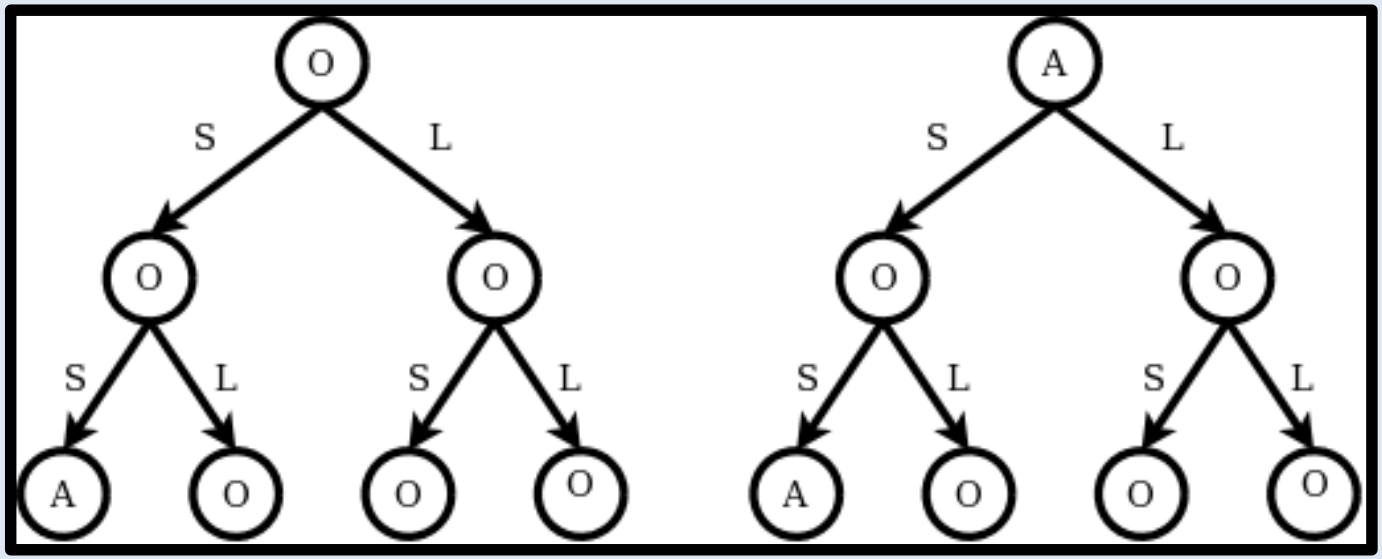
Execution Phase



Policy Domain

- What do policies look like?
  - In general histories → actions
  - before: more compact representations...
- Now, this is difficult: no such representation known!

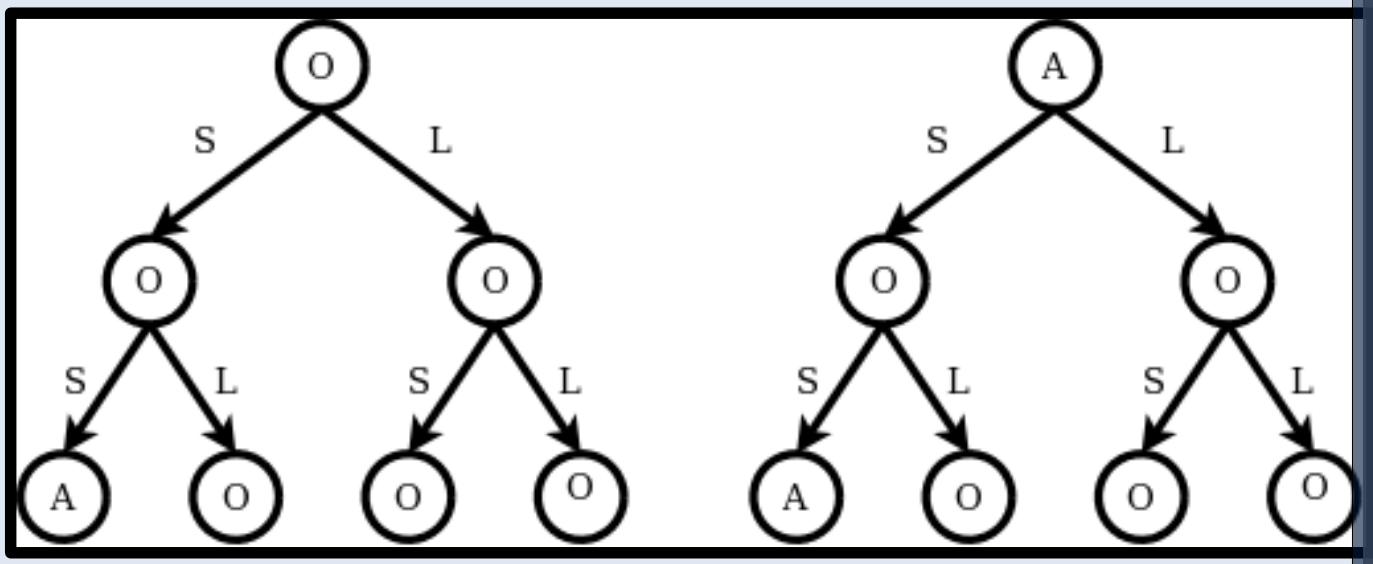
  → So we will be stuck with histories



Policy Domain

- What do policies look like?
  - In general histories → actions
  - before: more compact representations...
- Now, this is difficult: no such representation known!

  → So we will be stuck with histories



Most general, AOHs:

$$\left(a_i^0, o_i^1, a_i^1, \ldots, a_i^{t-1}, o_i^t\right)$$

But: can restrict to deterministic policies → only need OHs:

$$\vec{o}_i = \left(o_i^1, \ldots, o_i^t\right)$$

No Compact Representation?

There are a number of types of beliefs considered

- **Joint Belief**, *b(s)* (as in MPOMDP) [Pynadath and Tambe 2002]
  - compute b(s) using joint actions and observations
  - Problem: ?

No Compact Representation?

There are a number of types of beliefs considered

- **Joint Belief**, *b(s)* (as in MPOMDP) [Pynadath and Tambe 2002]
  - compute b(s) using joint actions and observations
  - Problem: agents do not know those during execution

No Compact Representation?

There are a number of types of beliefs considered

- **Joint Belief**, *b(s)* (as in MPOMDP) [Pynadath and Tambe 2002]
  - compute b(s) using joint actions and observations
  - Problem: agents do not know those during execution
- **Multiagent belief**, $b_i(s,q_{-i})$ [Hansen et al. 2004]
  - belief over (future) policies of other agents
  - Need to be able to predict the other agents!
    - for belief update $P(s'|s,a_i,a_{-i})$, $P(o|a_i,a_{-i},s')$, and prediction of $R(s,a_i,a_{-i})$
  - form of those other policies? most general: $\pi_{j_j} : \vec{o}_{j_j} \rightarrow a_{j_j}$
  - if they use beliefs? → infinite recursion of beliefs!

Goal of Planning

- **Find** the **optimal** joint policy $\pi^* = \langle \pi_1, \pi_2 \rangle$
  - where individual policies map OHs to actions $\pi_i : \vec{O}_i \rightarrow A_i$

- What is the optimal one?
  - Define **value** as the expected sum of rewards:

$$V(\pi) = \boldsymbol{E}\left[\sum_{t=0}^{h-1} R(s,a) \mid \pi, b^0\right]$$

  - optimal joint policy is one with maximal value (can be more that achieve this)

Goal of Planning

- **Find t**
  - whe
- What
  - Def
  - opti
    (ca

Optimal policy for 2 generals, h=3

value=-2.86743

() --> observe
(o_small) --> observe
(o_large) --> observe
(o_small,o_small) --> attack
(o_small,o_large) --> attack
(o_large,o_small) --> attack
(o_large,o_large) --> observe

() --> observe
(o_small) --> observe
(o_large) --> observe
(o_small,o_small) --> attack
(o_small,o_large) --> attack
(o_large,o_small) --> attack
(o_large,o_large) --> observe

Goal of Planning

- **Find t**
  - whe
- What
  - Def
  - opti
    (ca

Optimal policy for 2 generals, h=3

value=-2.86743

() --> observe
(o_small) --> observe
(o_large) --> observe
(o_small,o_small) --> attack
(o_small,o_large) --> attack
(o_large,o_small) --> attack
(o_large,o_large) --> observe

() --> observe
(o_small) --> observe
(o_large) --> observe
(o_small,o_small) --> attack
(o_small,o_large) --> attack
(o_large,o_small) --> attack
(o_large,o_large) --> observe

conceptually:

what should policy optimize to allow for good coordination (thus high value)

?

Coordination vs. Exploitation of Local Information

- Inherent trade-off

  coordination vs. **exploitation of local information**

  - Ignore own observations → 'open loop plan'
    - E.g., “ATTACK on 2nd time step”

      \+ maximally predictable
      \- low quality
  - Ignore coordination
    - E.g., compute an individual belief $b_i(s)$

      $$b_i(s) = \sum_{q_{-i}} b(s, q_{-i})$$

      and execute the MPOMDP policy
      \+ uses local information
      \- likely to result in mis-coordination
  - Optimal policy π^* should balance between these.

Planning Methods

Brute Force Search

- We can compute the value of a joint policy $V(\pi)$
  - using a Bellman-like equation [Oliehoek 2012]
- So the **stupidest algorithm** is:
  - compute $V(\pi)$, for all π
  - select a π with maximum value
- Number of joint policies is huge!
  (doubly exponential in horizon h)
- Clearly intractable...

h	num. joint policies
1	4
2	64
3	16384
4	1.0737e+09
5	4.6117e+18
6	8.5071e+37
7	2.8948e+76
8	3.3520e+153

Brute Force Search

- We can compute the value of a joint policy $V(\pi)$
  - using a Bellman-like equation [Oliehoek 2012]
- So the simplest algorithm is:
  - compute $V(\pi)$ for all π
  - return best joint policy
- Number of joint policies is huge!
  (doubly exponential in horizon h)
- Clearly intractable...

No easy way out...

The problem is
NEXP-complete [Bernstein et al. 2002]

most likely (assuming EXP != NEXP)
doubly exponential time required.

h	num. joint policies
1	4
2	64
3	16384
4	1.0737e+09
5	4.6117e+18
6	8.5071e+37
7	2.8948e+76
8	3.3520e+153

Brute Force Search

- We can compute the value of a joint policy $V(\pi)$
  - using a Bellman-like equation [Oliehoek 2012]
- So the simplest algorithm is:
  - compute $V(\pi)$ for all π
  - Return best joint policy found
- Number of joint policies is huge!
  (doubly exponential in horizon h)
- Clearly intracta

No easy way out...

The problem is
NEXP-complete [Bernstein et al. 2002]

most likely (assuming EXP != NEXP)
doubly exponential time required.

h	num. joint policies
1	4
2	64
3	16384
4	1.0737e+09
5	4.6117e+18
6	8.5071e+37
7	2.8948e+76

- Still, there are better algorithms that work better for at least some problems...
- Useful to understand what optimal really means!
  (trying to compute it helps understanding)

Algorithmic Developments

- Dynamic Programming
  - DP for POSGs/Dec-POMDPs [Hansen et al. 2004]
  - Improvements to exhaustive backup [Amato et al. 2009]
  - Compression of values (LPC) [Boularias & Chaib-draa 2008]
  - (Point-based) Memory bounded DP [Seuken & Zilberstein 2007a]
  - Improvements to PB backup [Seuken & Zilberstein 2007b, Carlin and Zilberstein, 2008; Dibangoye et al, 2009; Amato et al, 2009; Wu et al, 2010, etc.]
- Heuristic Search
  - No backtracking: just most promising path
    [Emery-Montemerlo et al. 2004, Oliehoek et al. 2008]
  - Clustering of histories: reduce number of child nodes
    [Oliehoek et al. 2009]
  - Incremental expansion: avoid expanding all child nodes
    [Spaan et al. 2011, Oliehoek et al., 2013]
- MILP [Aras and Dutech 2010]

State of The Art

To get an impression...

- Optimal solutions
  - Improvements of MAA* lead to significant increases
  - but problem dependent

h	MILP	LPC	GMAA-ICE*
4	72	534.9	0.04
6		-	46.43*

dec-tiger – runtime (s)

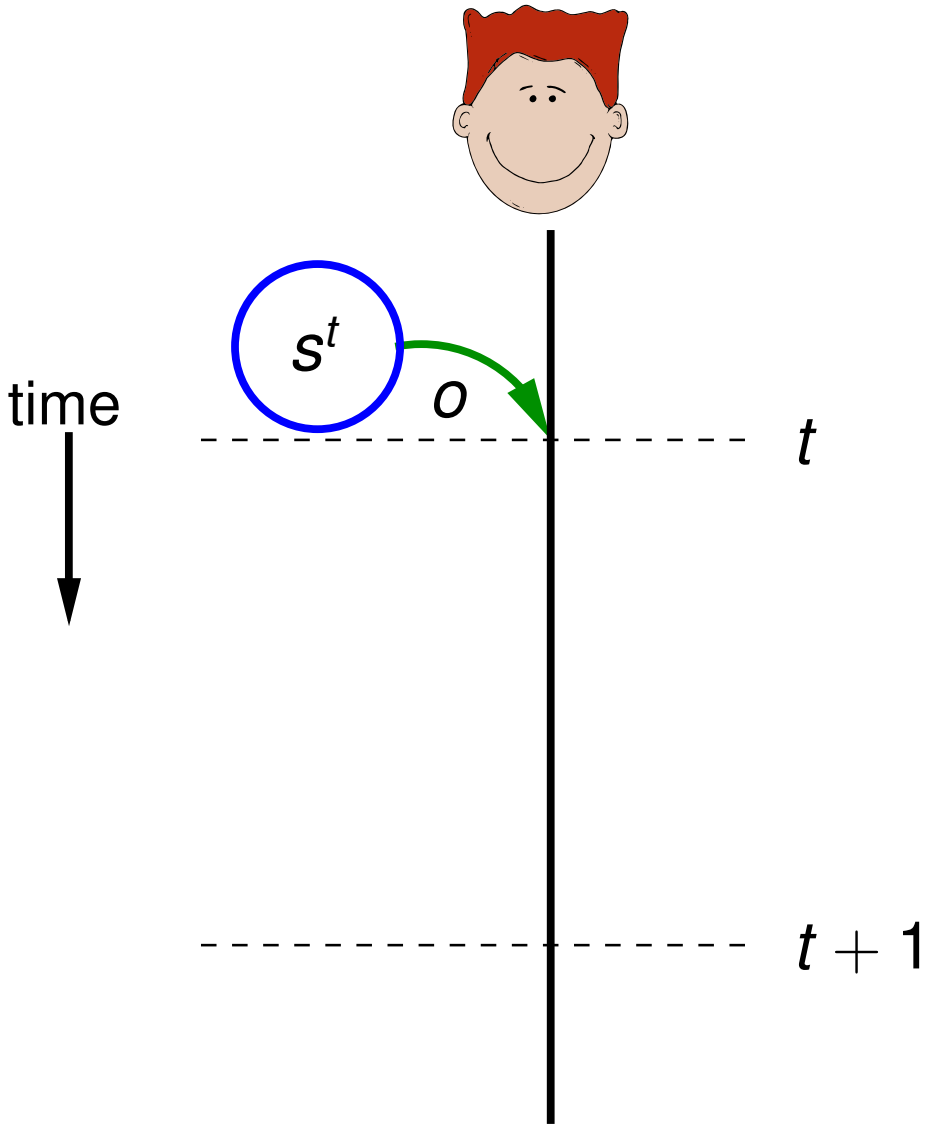
h	MILP	LPC	GMAA-ICE*
5	25	–	<0.01
500	–	–	0.94*

broadcast channel runtime (s)
* excluding heuristic

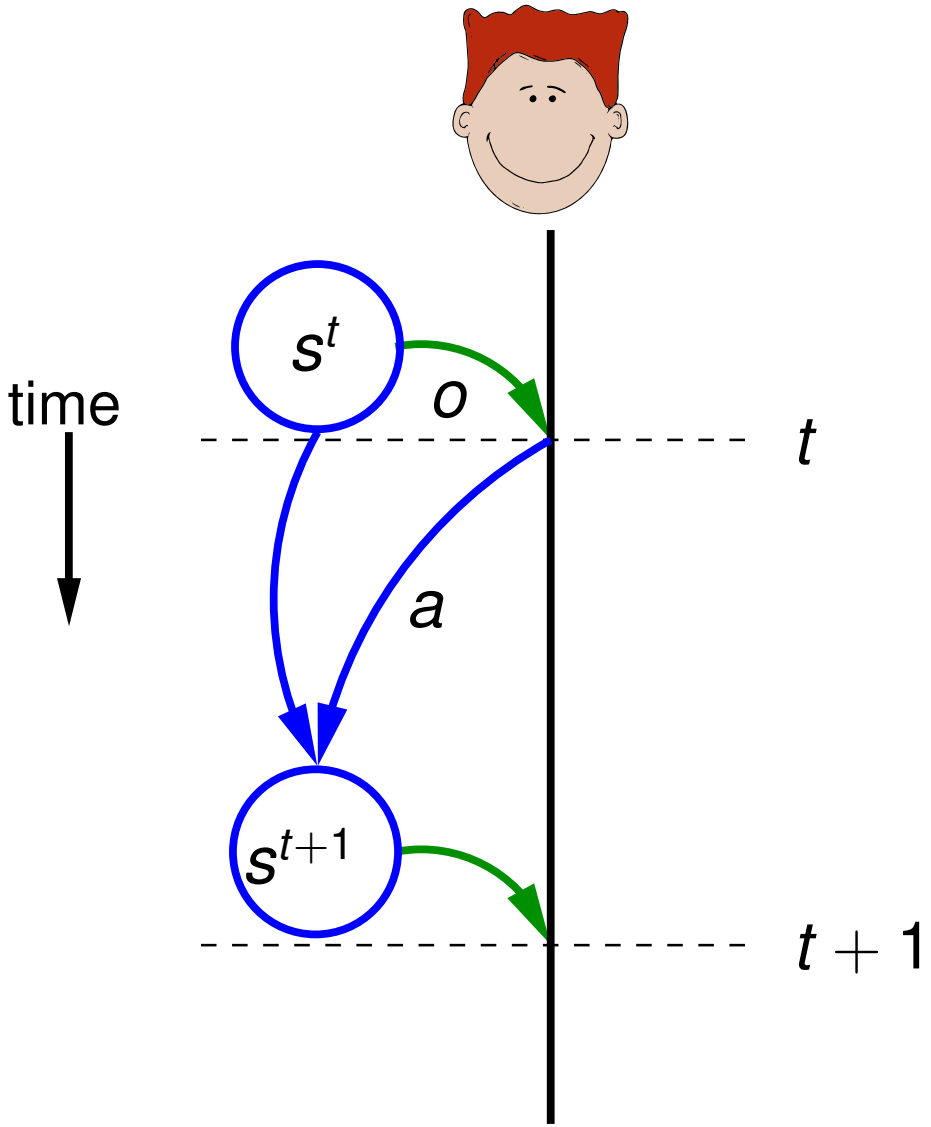
- Approximate (no quality guarantees)
  - MBDP: linear in horizon [Seuken & zilberstein 2007a]
  - Rollout sampling extension: up to 20 agents [Wu et al. 2010b]
  - Transfer planning: use smaller problems to solve large (structured) problems (up to 1000) agents [Oliehoek 2010]

Communication

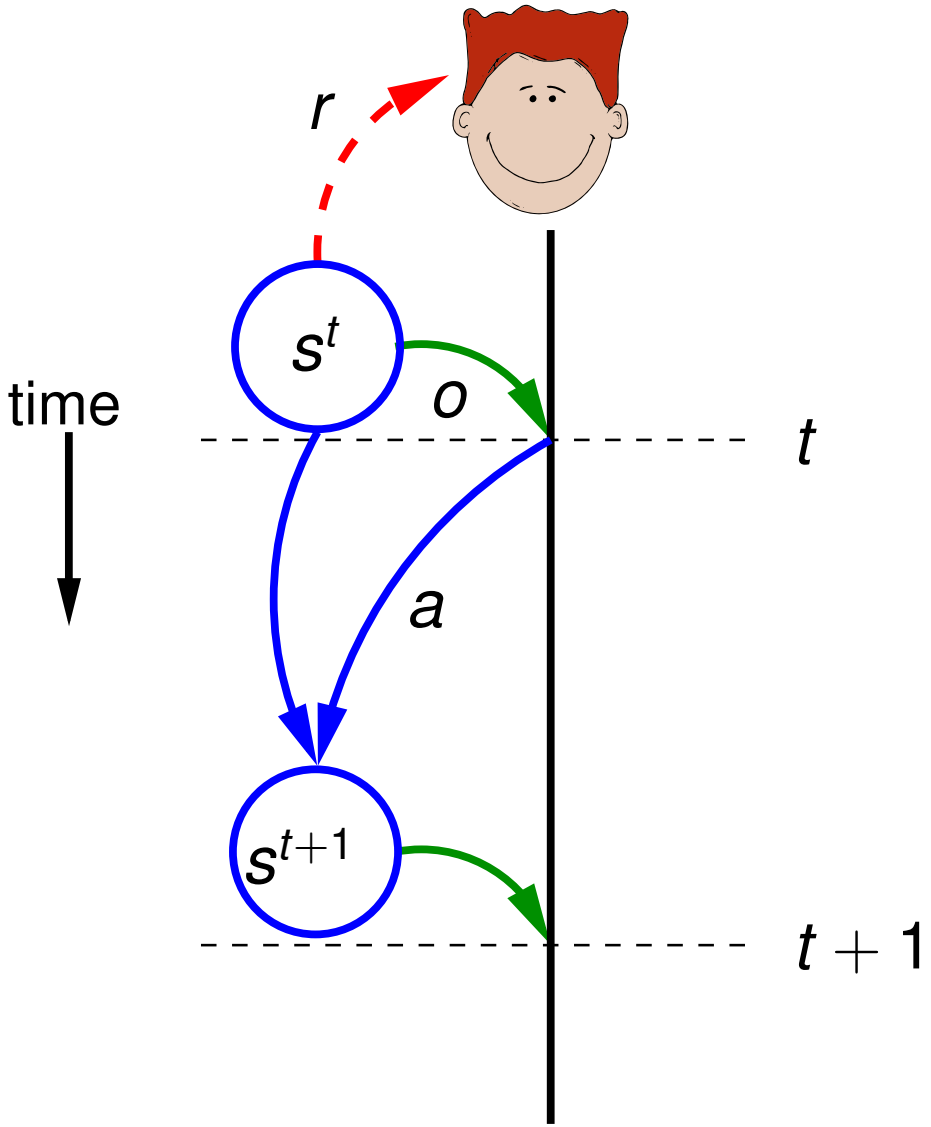
Single-agent POMDPs



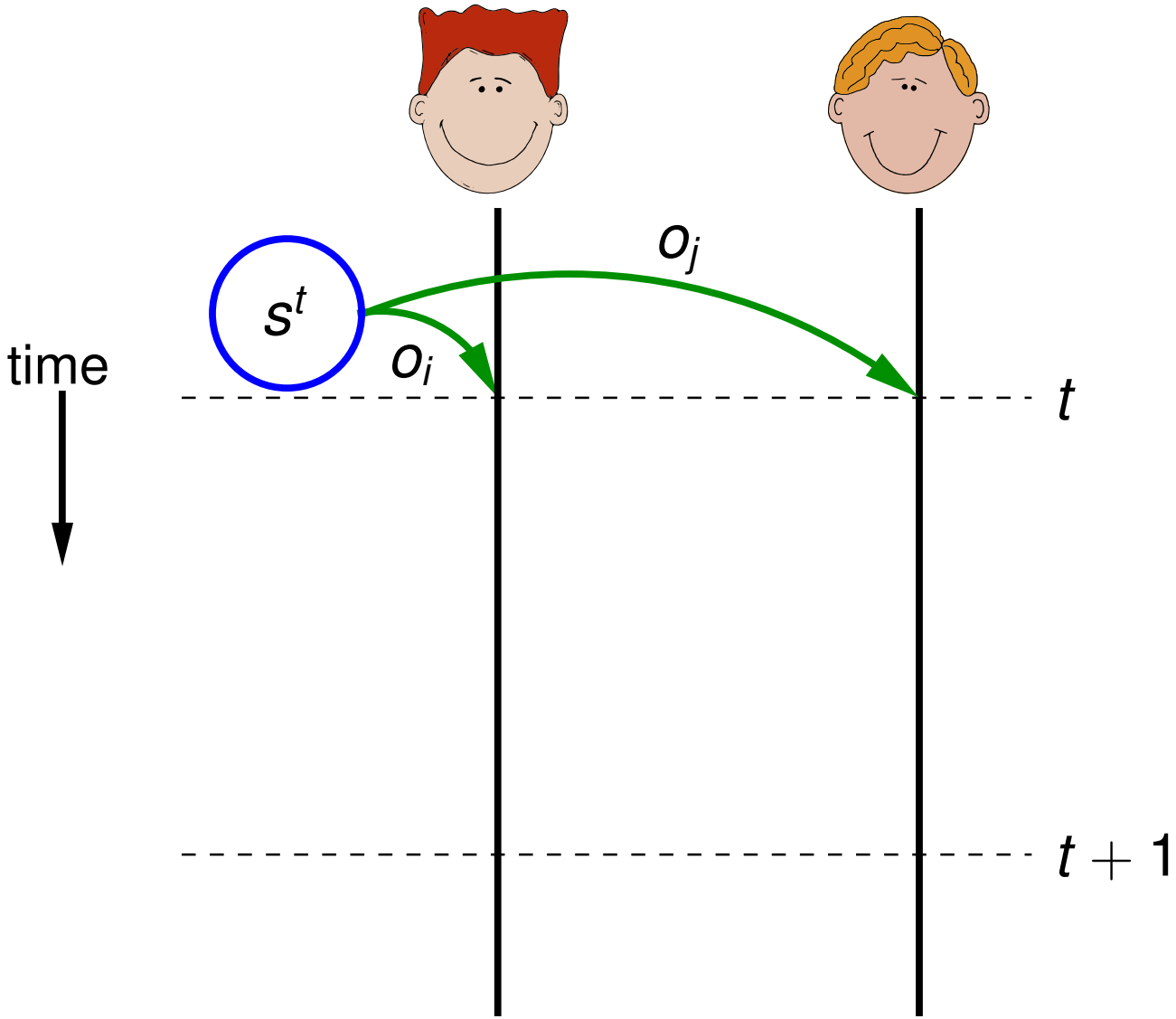
Single-agent POMDPs



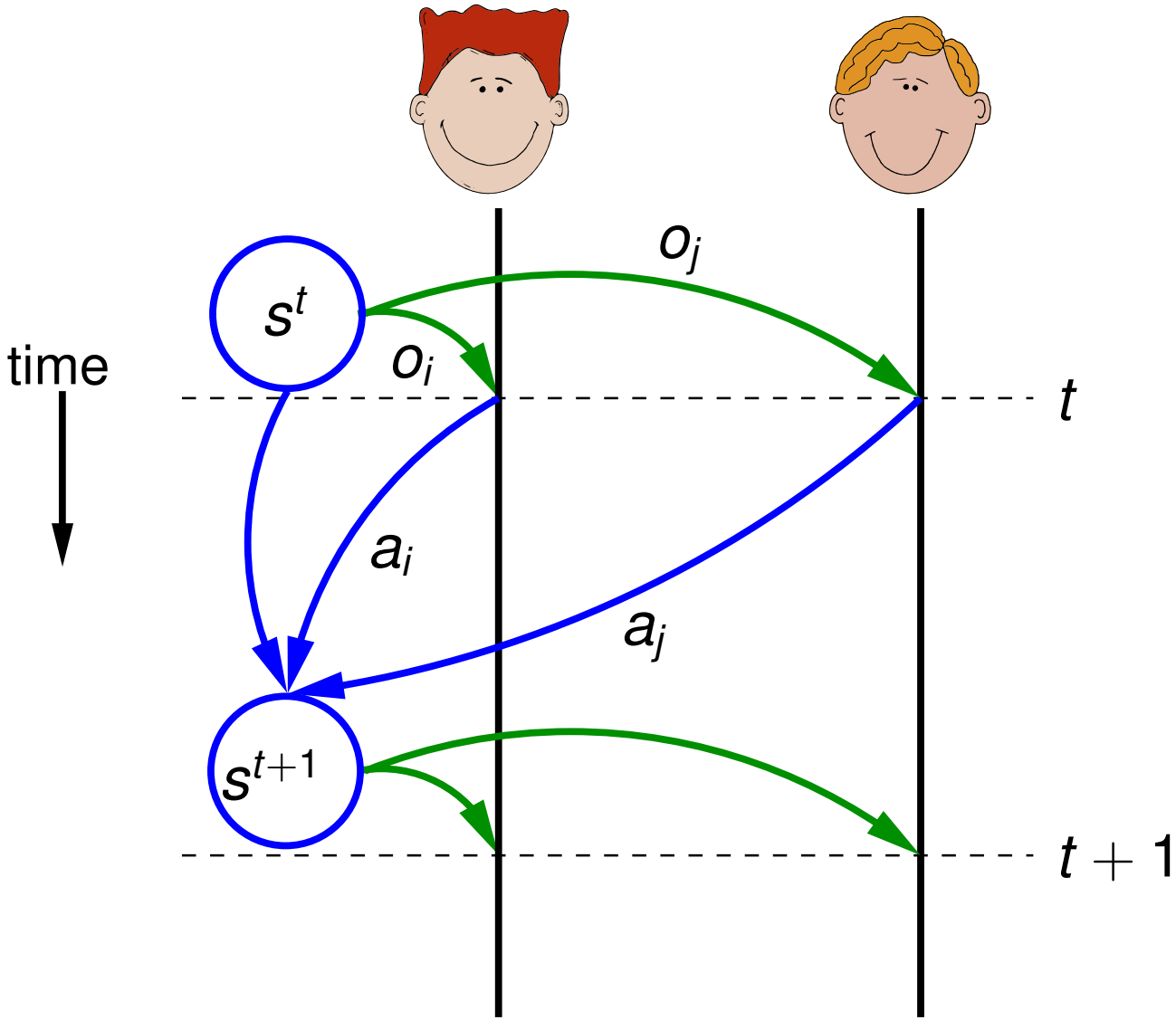
Single-agent POMDPs



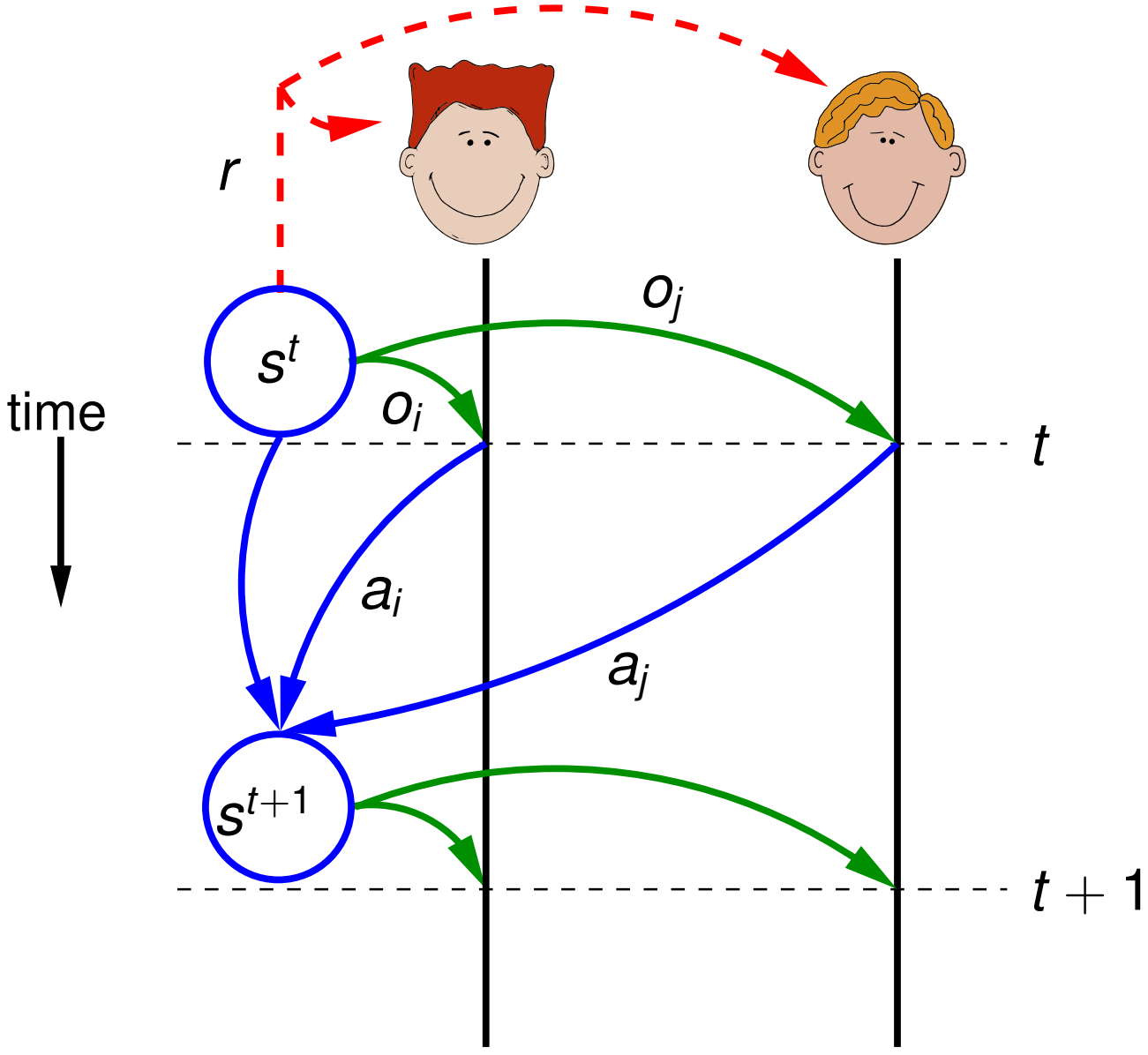
Dec-POMDPs



Dec-POMDPs



Dec-POMDPs



POMDPs vs. Dec-POMDPs

Frameworks for acting optimally given:

- Limited sensing.
- Stochastic environments.

POMDPs vs. Dec-POMDPs

Frameworks for acting optimally given:

- Limited sensing.
- Stochastic environments.

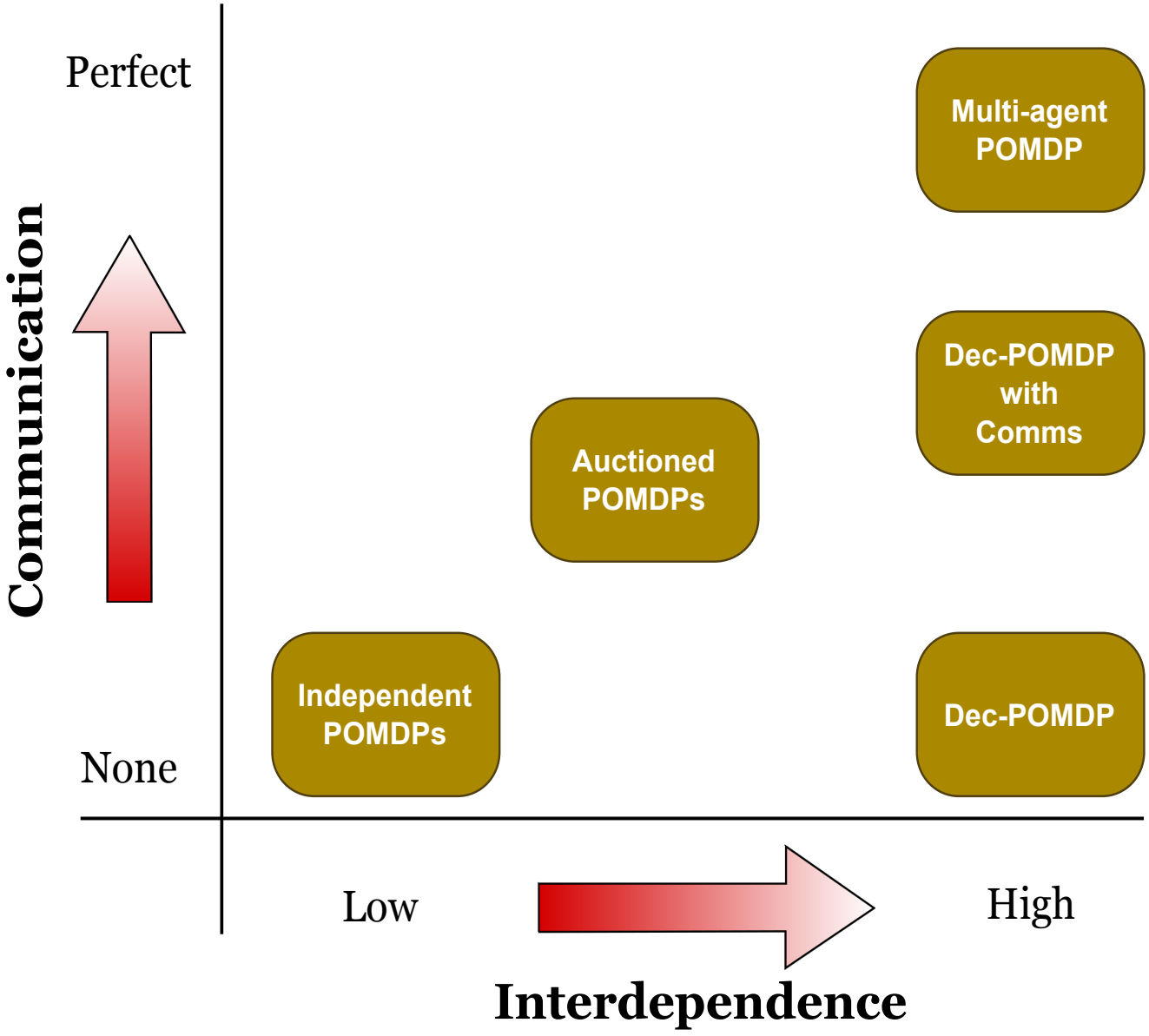
Dec-POMDPs:

- Decentralized execution.
- Usually centralized, off-line planning.
- No common state estimate, no joint belief.
- Optimal policies based on individual observation histories.
- NEXP-Complete, doubly-exponential in the horizon.

Adding communication

- Implicit communication in Dec-POMDPs.
  - Agent actions affect the state which affects other agents' observations.
- Explicit communication can be added as well.
  - Equivalent to Dec-POMDP (Goldman and Zilberstein, 2004).
  - Information sharing $\rightarrow$ can reduce complexity
- Issues
  - What to communicate and to whom?
  - When to coordinate?
- Semantics predominantly predefined.

Communication and interdependence



Communication – Outline

1. Optimizing semantics (Spaan et al., 2006)
   - Learning the meaning of messages

Communication – Outline

1. Optimizing semantics (Spaan et al., 2006)
   - Learning the meaning of messages
2. Multiagent POMDPs with delayed communication (Spaan et al., 2008), (Oliehoek and Spaan, 2012a)
   - Planning with stochastically delayed information sharing

Communication – Outline

1. Optimizing semantics (Spaan et al., 2006)
   - Learning the meaning of messages
2. Multiagent POMDPs with delayed communication (Spaan et al., 2008), (Oliehoek and Spaan, 2012a)
   - Planning with stochastically delayed information sharing
3. Offline communication policies for factored multiagent POMDPs (Messias et al., 2011)
   - Planning when communication can be reduced

Communication – Outline

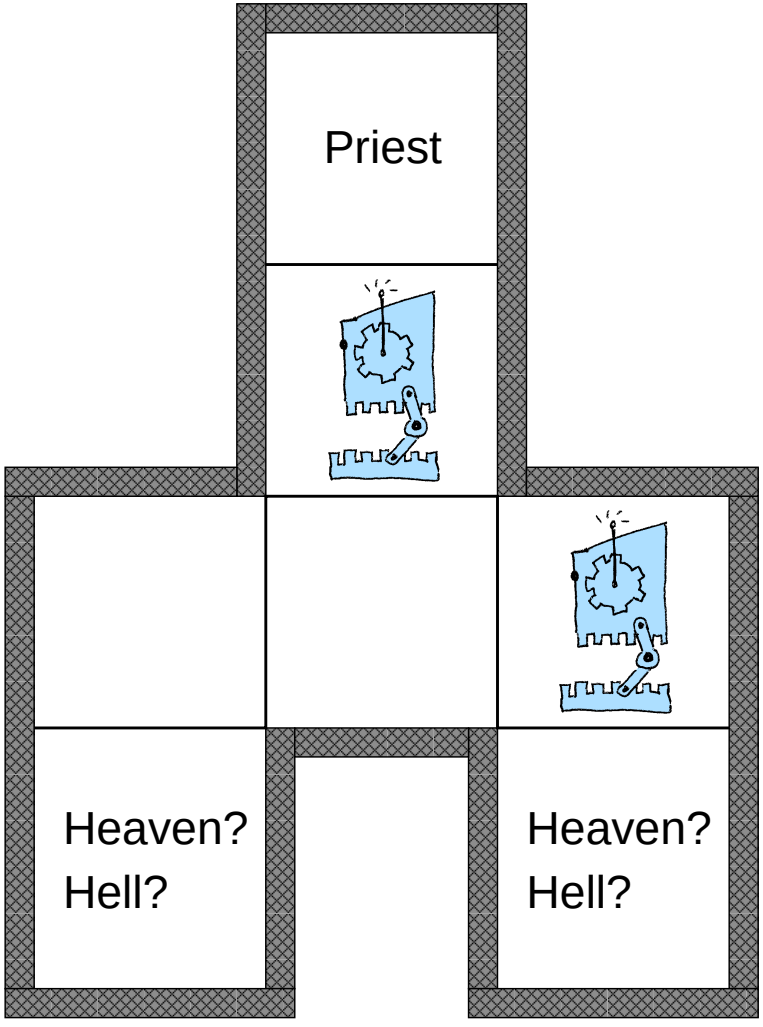
1. Optimizing semantics (Spaan et al., 2006)
   - Learning the meaning of messages
2. Multiagent POMDPs with delayed communication (Spaan et al., 2008), (Oliehoek and Spaan, 2012a)
   - Planning with stochastically delayed information sharing
3. Offline communication policies for factored multiagent POMDPs (Messias et al., 2011)
   - Planning when communication can be reduced
4. Event-driven models (Messias et al., 2013)
   - Planning for asynchronous execution

Optimizing semantics

Learning the meaning of messages

Example heaven or hell

- Task: meet in heaven.
- Priest knows the location of heaven.



Proposed model (Spaan et al., 2006)

Features:

Decentralized: Each agent considers only its own local state plus some uncontrollable state features, shared by all agents.

$\Rightarrow$ avoid computing an (approximate) solution of the centralized planning problem

Proposed model (Spaan et al., 2006)

Features:

Decentralized: Each agent considers only its own local state plus some uncontrollable state features, shared by all agents.

$\Rightarrow$ avoid computing an (approximate) solution of the centralized planning problem

Communication Semantics of sending a particular message are part of the optimization problem.

$\Rightarrow$ communication is an integral part of an agent's reasoning, not an add-on.

Proposed model (Spaan et al., 2006)

Features:

Decentralized: Each agent considers only its own local state plus some uncontrollable state features, shared by all agents.

$\Rightarrow$ avoid computing an (approximate) solution of the centralized planning problem

Communication Semantics of sending a particular message are part of the optimization problem.

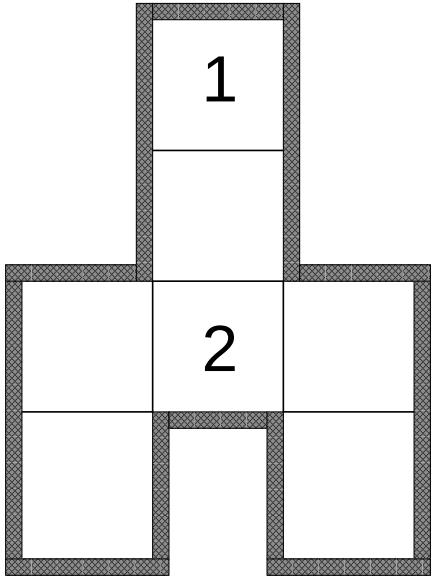
$\Rightarrow$ communication is an integral part of an agent's reasoning, not an add-on.

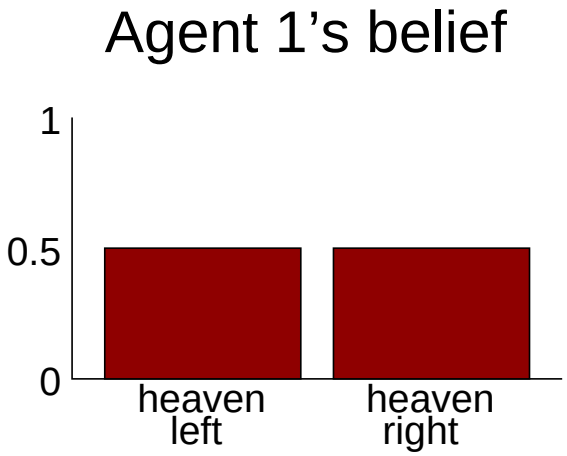
Environment The agents inhabit a stochastic environment that is only partially observable to them.

Proposed model: discussion

- Agent i's policy π_i maps a belief over its local state (s_0, s_i) to an action (as in a POMDP).
- A message sent by an agent i as part of its action is received by agent j at the next timestep as part of its observation.
- Agent j knows π_i
  - a message conveys information about the shared state S_0.
- Information is represented in agent i's observation model.

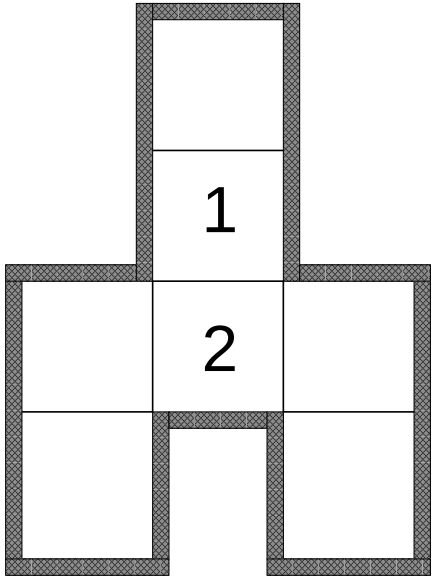
Communication example

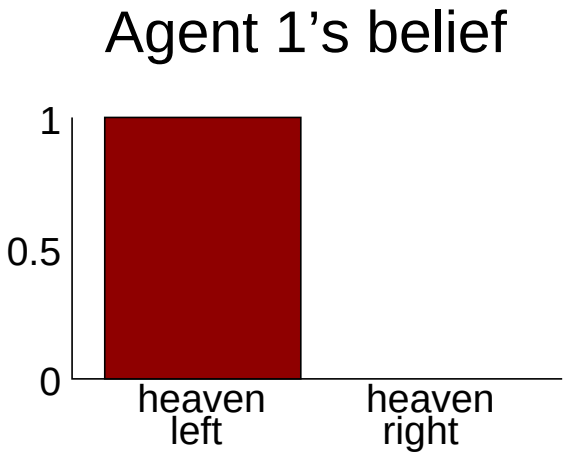




Agent 1 just moved to the Priest location.

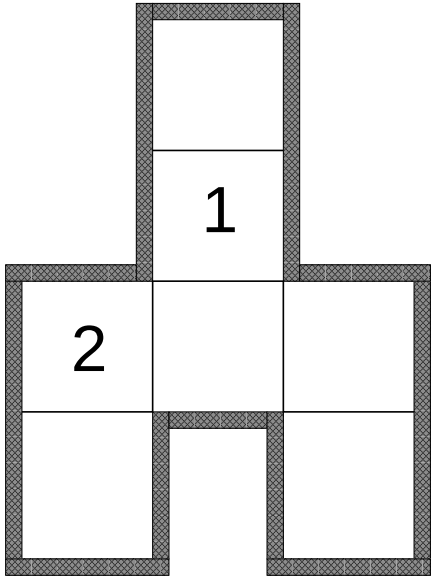
Communication example

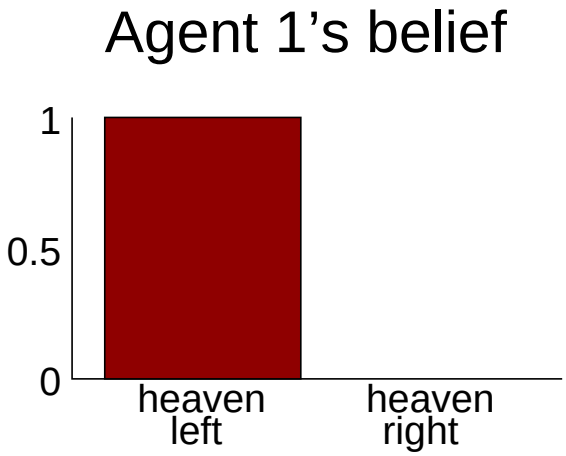


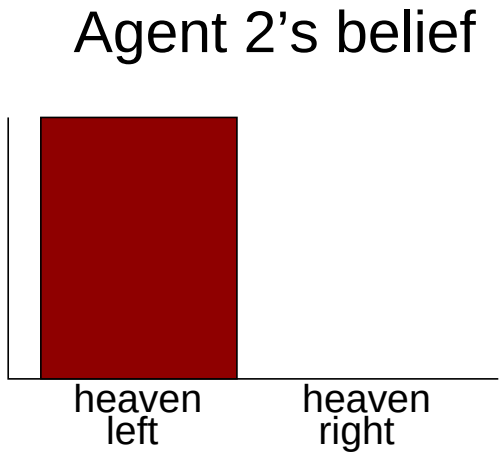


Agent 1 observes *heaven-left* and executes *south/send_1*.

Communication example







Agent 2 observes *no-walls/received_1* and executes *west*.

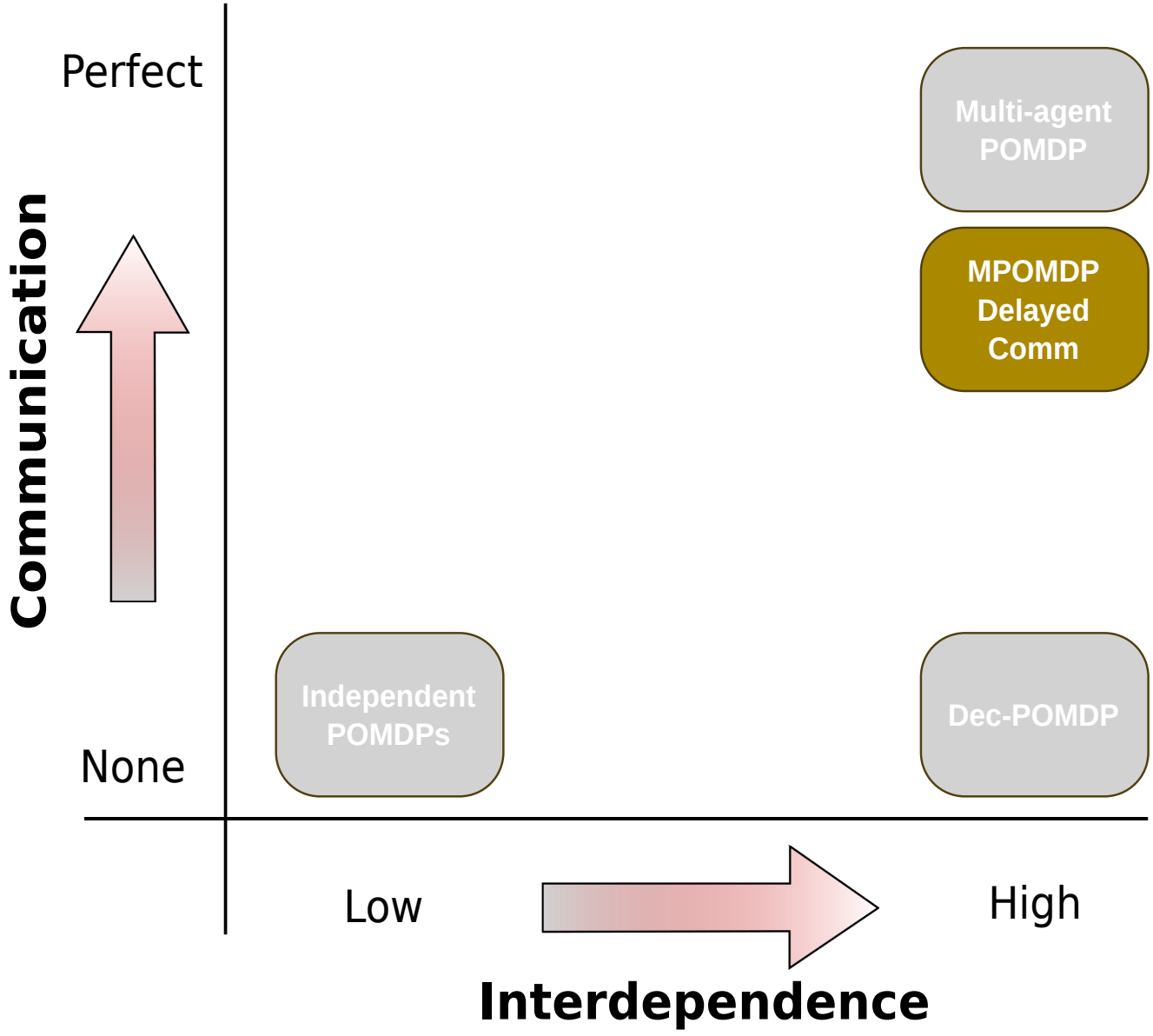
POMDP model for a single agent

- Fixed policies of other agents can be treated as part of the environment.
- Agent *j*'s policy π_j influences agent *i*'s POMDP model:
  - Observation model through communication.
  - Reward function through joint reward.
- Constructing a POMDP model for *i* requires statistics $p(s_j)$ and $p(a_j|s_j)$.
  - Approximated by simulating $\{\pi_i, \pi_j\}$.

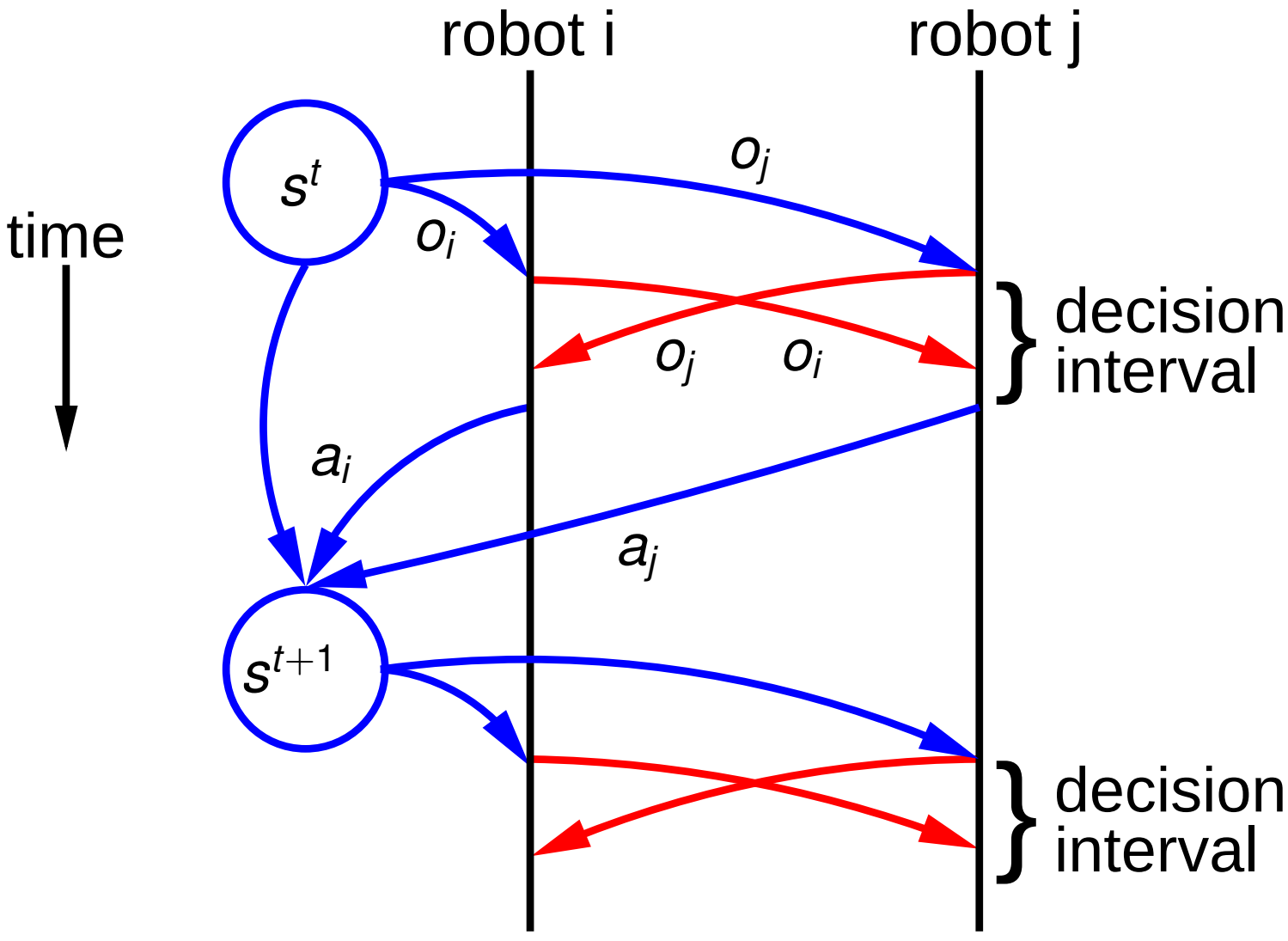
Delayed communication

Planning with stochastically delayed information sharing

Multiagent POMDPs with delayed communication

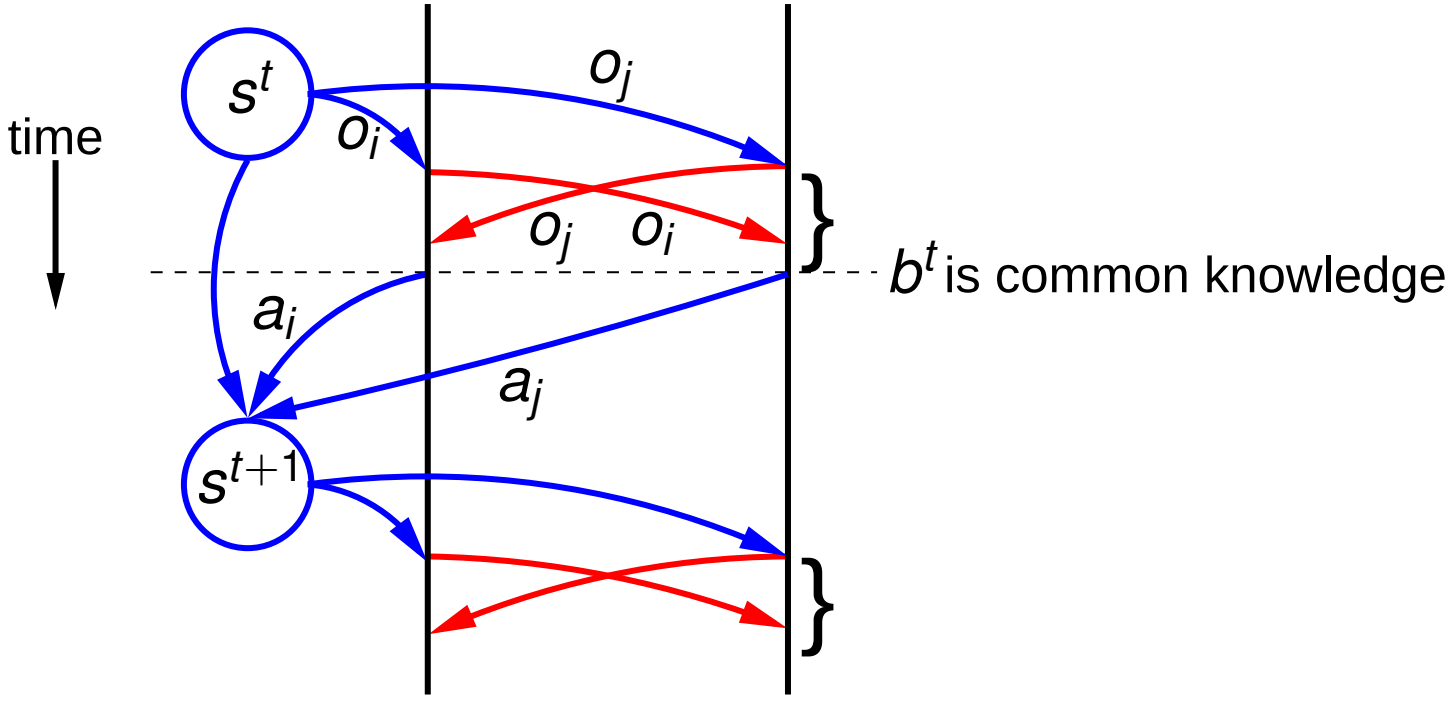


Instantaneous communication



Instantaneous communication

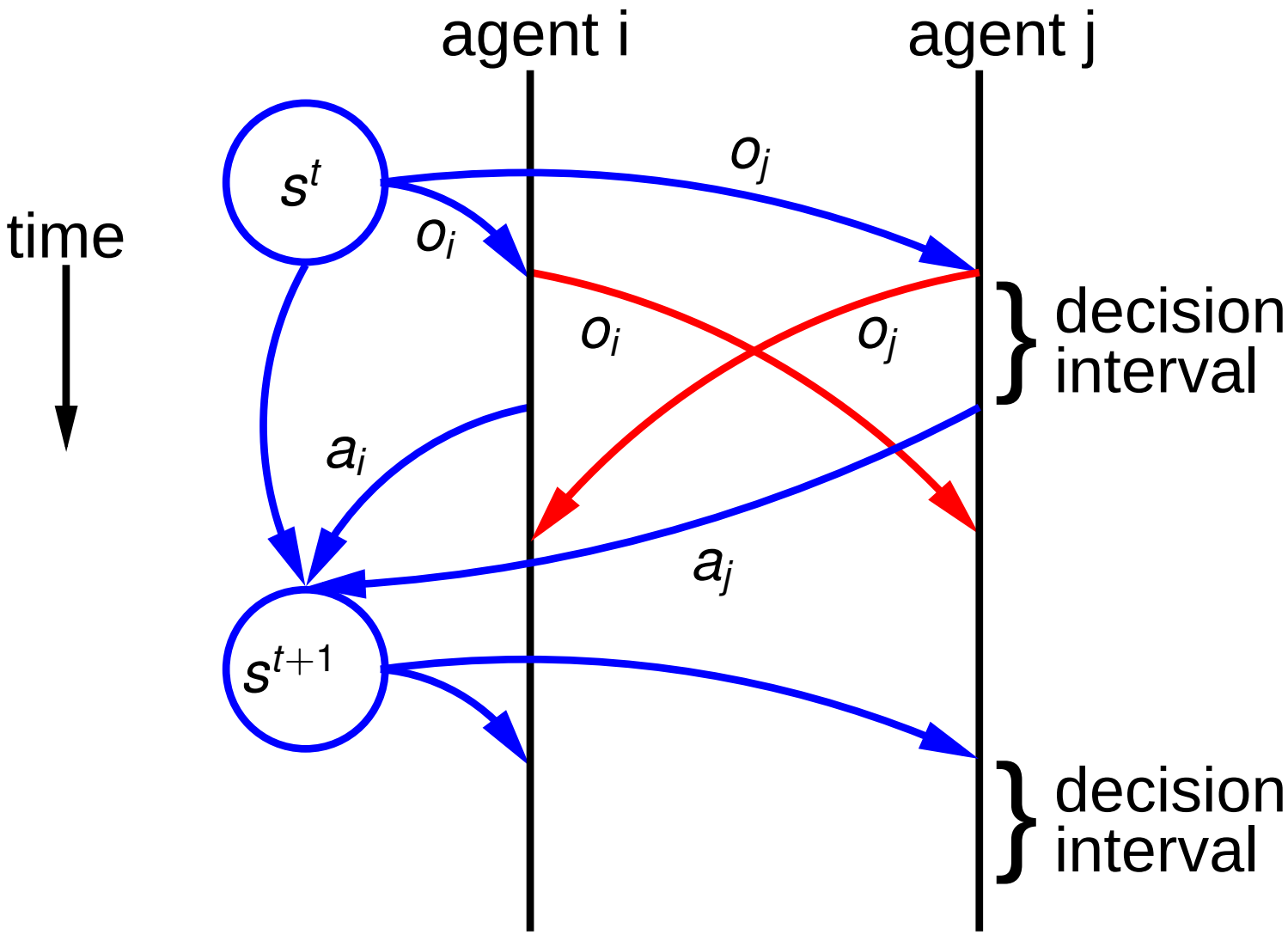
- Instantaneous, perfect communication reduces a Dec-POMDP to a POMDP. (Pynadath and Tambe, 2002)
- Observation sharing $\rightarrow$ each agent knows belief b^t at time t.
- Piecewise-linear and convex (PWLC) value function.
- But this synchronization step takes time.



Delayed communication

- But what if communication is not instantaneous?
- Delayed communication is useful in several settings:
  - Smart protection schemes
  - Multi-robot systems
  - Distributed video surveillance
  - Upper bound on Dec-POMDP value function (Oliehoek et al., 2008)

Delayed communication



Fixed delay communication

- Each agent knows the last commonly known $b^{t-k}, k > 0$.
- If $k = 1$, the optimal value function is PWLC. (Hsu and Marcus, 1982)
  - Equal to the Q_{BG} value function (Oliehoek et al., 2008)
- Otherwise the value function is not separable (Varaiya and Walrand, 1978)
  - It is not a function over the joint belief space.
- But what if the delay can vary?

Stochastic communication delays (Spaan et al., 2008)

- Probability that synchronization succeeds within a particular stage i: $p^{\mathrm{iTD}}(s)$
- Optimal value function:

$$Q^*_{\mathrm{SD}} = R + p^{0\mathrm{TD}} F_{0\mathrm{TD}} + p^{1\mathrm{TD}} F_{1\mathrm{TD}} + p^{2\mathrm{TD}} F_{2\mathrm{TD}} + \ldots,$$

  where F_{iTD} is the exp. future reward given delay i.
- We assume during planning that delay is at most 1 step:

$$p^D = p^{1\mathrm{TD}} + p^{2\mathrm{TD}} + \cdots = 1 - p^{0\mathrm{TD}},$$

  and define an approximate value function as

$$\widetilde{Q}^*_{\mathrm{SD}} = R + p^{0\mathrm{TD}} F_{0\mathrm{TD}} + p^D F_{1\mathrm{TD}}.$$

- We prove that $\widetilde{Q}^*_{\mathrm{SD}}$ is PWLC over the joint belief space.

Algorithms

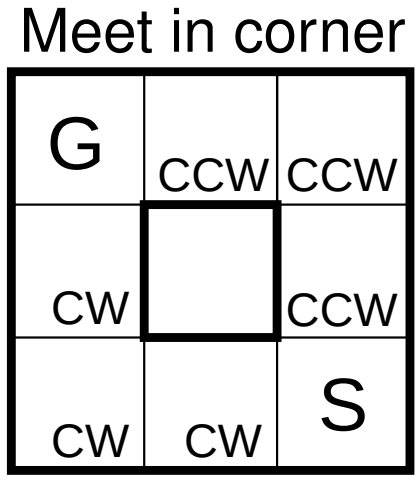
Algorithms

- Point-based approximate POMDP techniques transfer. (Spaan et al., 2008)
- Exhaustive backups of the 1TD delay value function can be sped up by tree-based pruning. (Oliehoek and Spaan, 2012b)
- For $k > 1$, we propose an online algorithm similar to Dec-COMM. (Roth et al., 2005)

Simulation results

- The policies consider potential future communication capabilities.

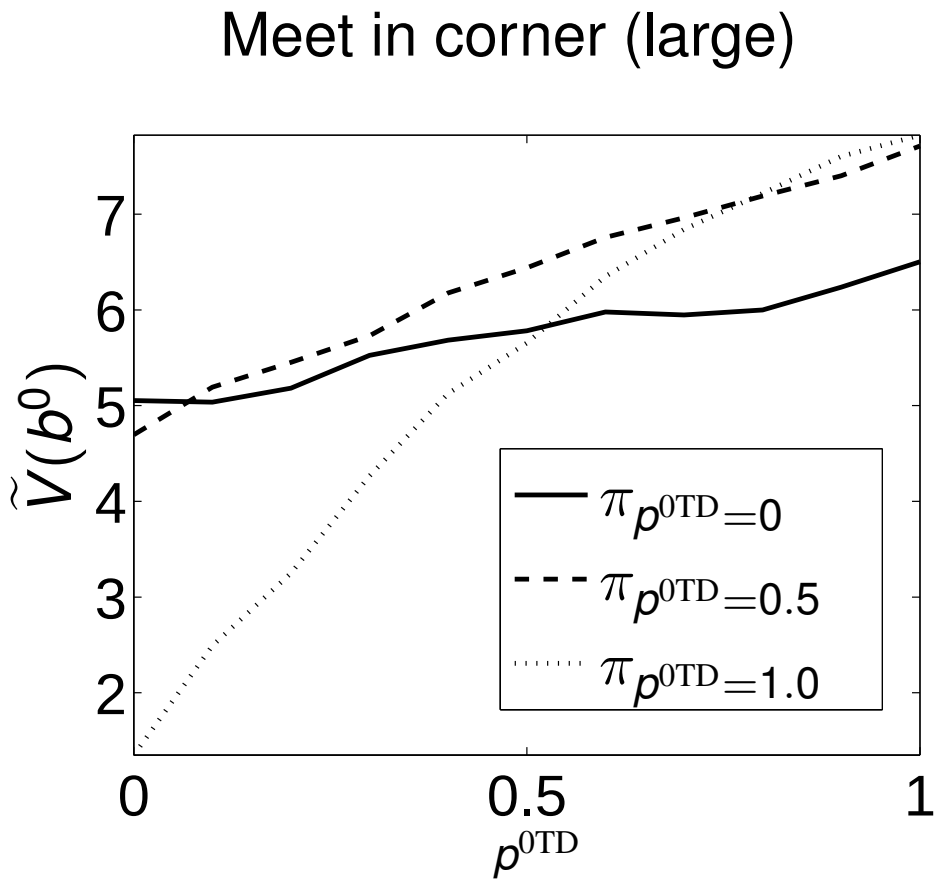
Experimental results



- The CW route is cheaper, and with uniform $p^{0\mathrm{TD}}$ it is chosen.
- We set $p^{0\mathrm{TD}}(s) = 0, \forall s \in \mathrm{CW}$ and to 1 everywhere else.
- Now the agents choose the CCW route, leading to higher payoff.

Experimental results

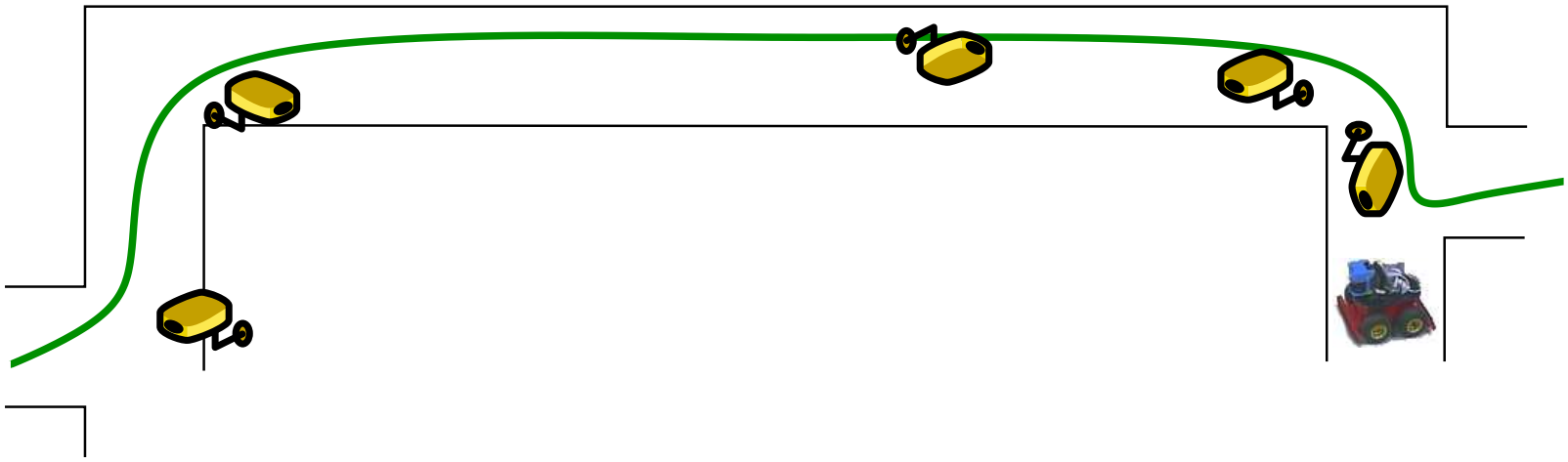
Policies computed for a specific value of $p^{0\text{TD}}$, but evaluated under different values of $p^{0\text{TD}}$.



Offline communication policies

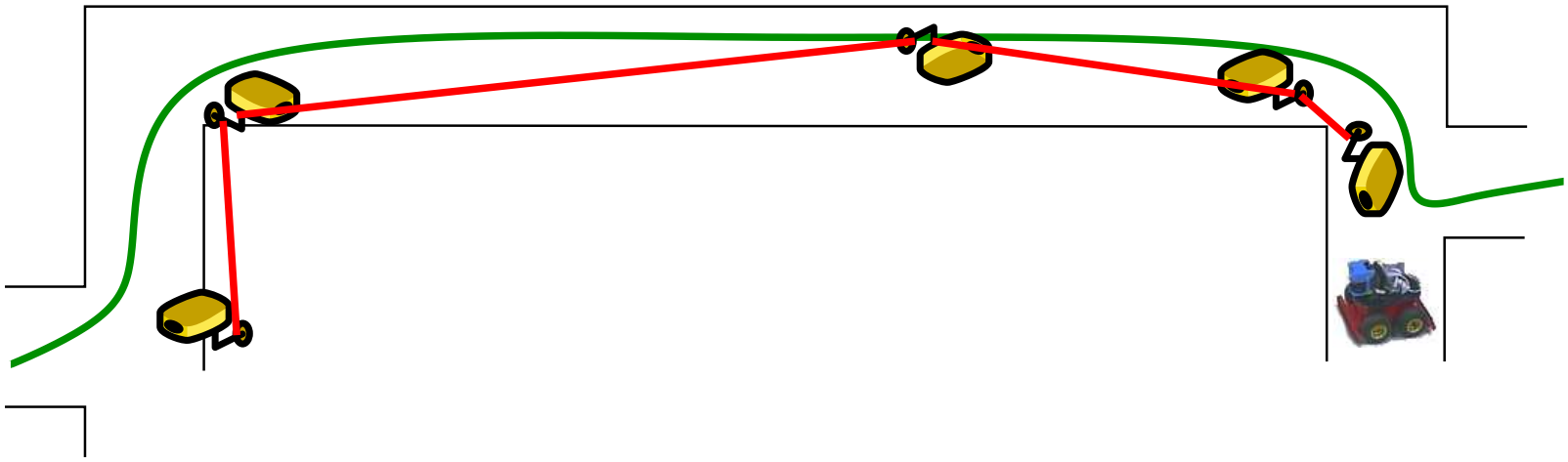
Planning when communication can be reduced

Exploiting local interactions



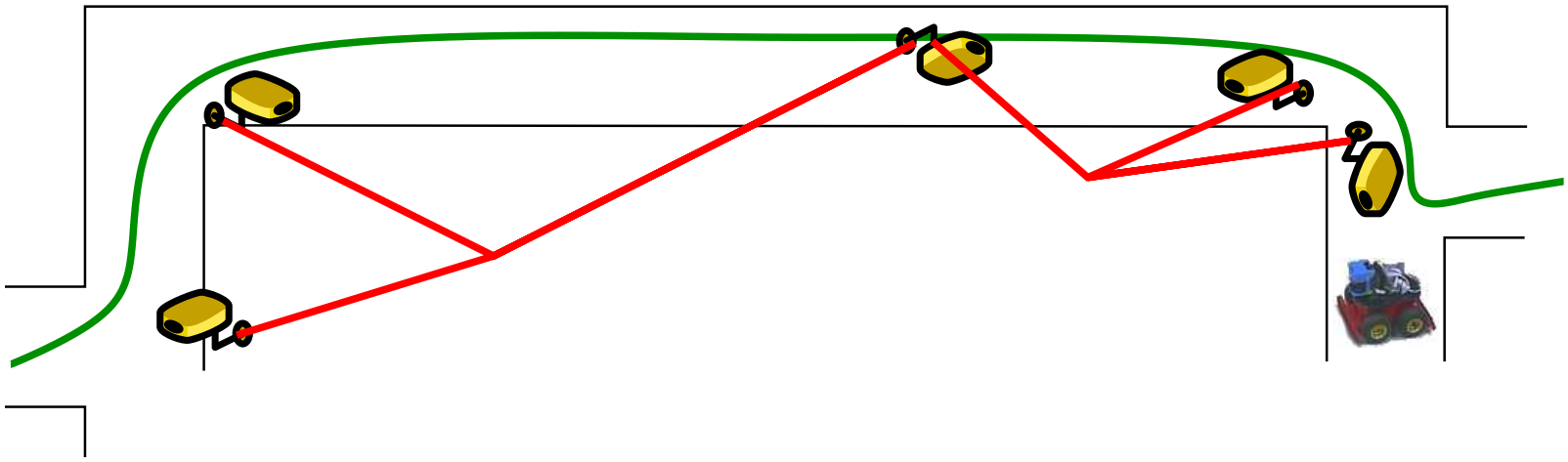
- Real-world domains have structure that can be exploited.
  - Factored models.
- Localized interactions can lead to reduced communication.

Exploiting local interactions



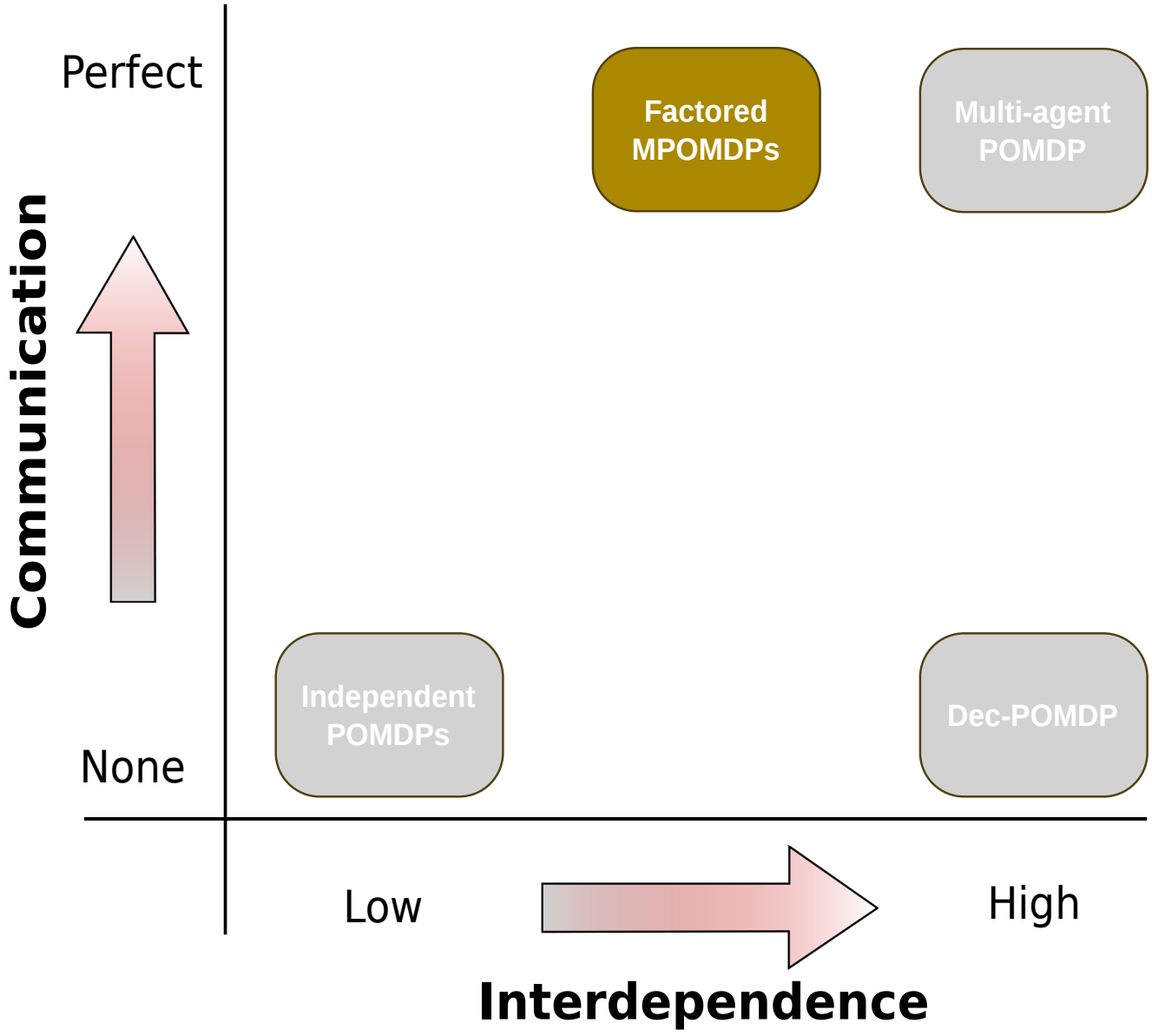
- Real-world domains have structure that can be exploited.
  - Factored models.
- Localized interactions can lead to reduced communication.

Exploiting local interactions



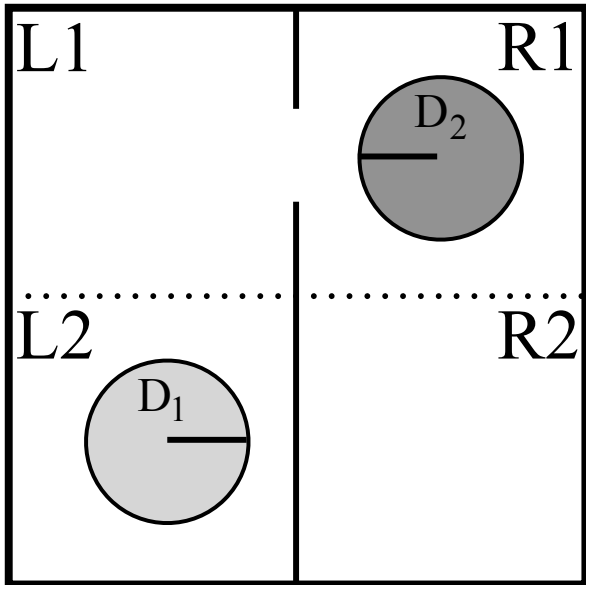
- Real-world domains have structure that can be exploited.
  - Factored models.
- Localized interactions can lead to reduced communication.

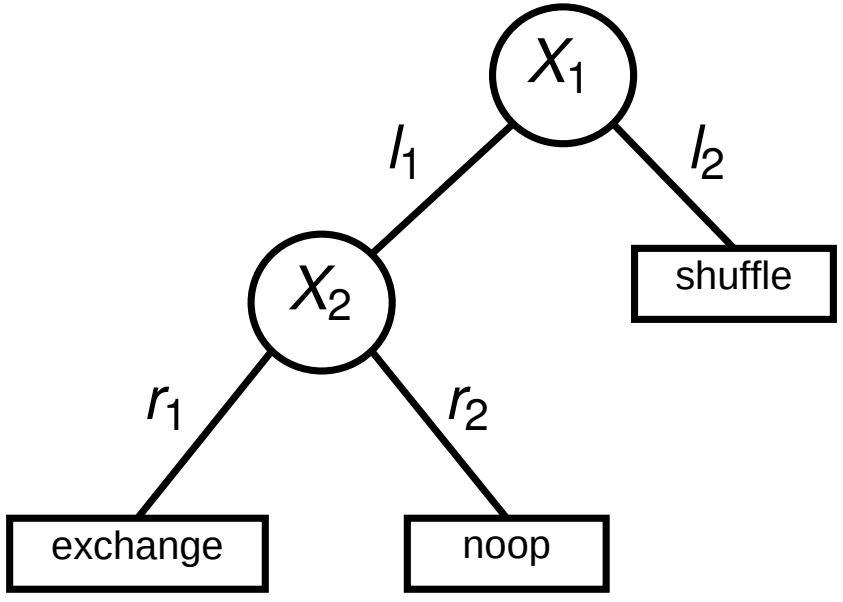
Factored MPOMDPs



Relay example

- State space: $S = X_1 \times X_2$ where $X_1 = \{l_1, l_2\}$ $X_2 = \{r_1, r_2\}$;
- Actions: $A_1 = A_2 = \{\text{shuffle, exchange, sense/noop}\}$;
- Observations: $O_1 = O_2 = \{\text{door, wall, idle}\}$;
- Agents should *exchange* when both are near the doorway.
- Unsuccessful exchanges are penalized.



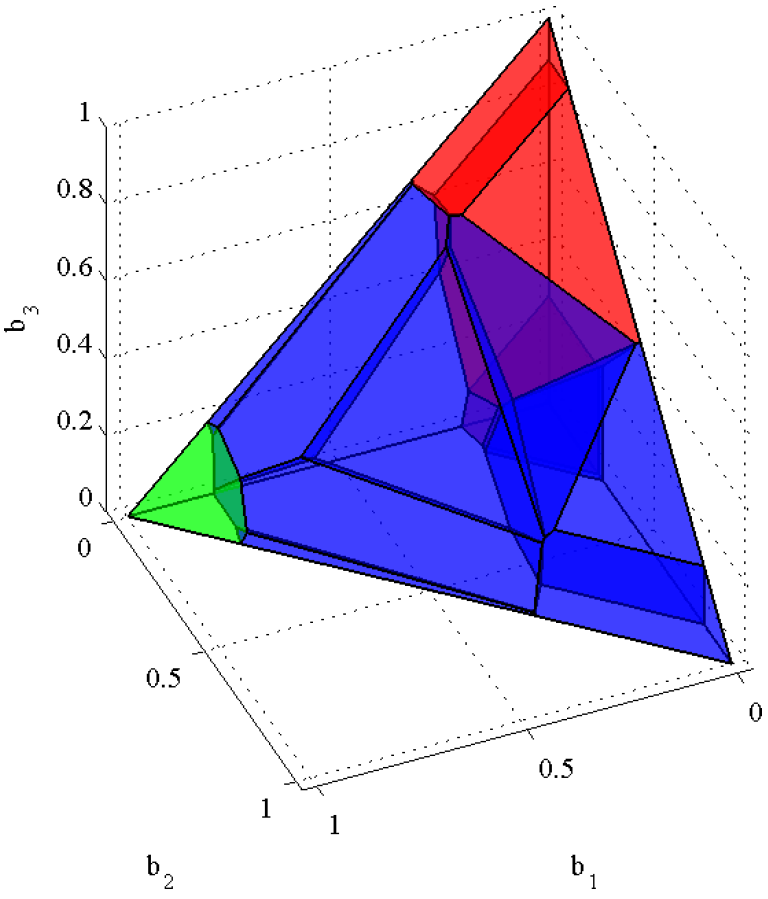


Factored Multiagent POMDPs

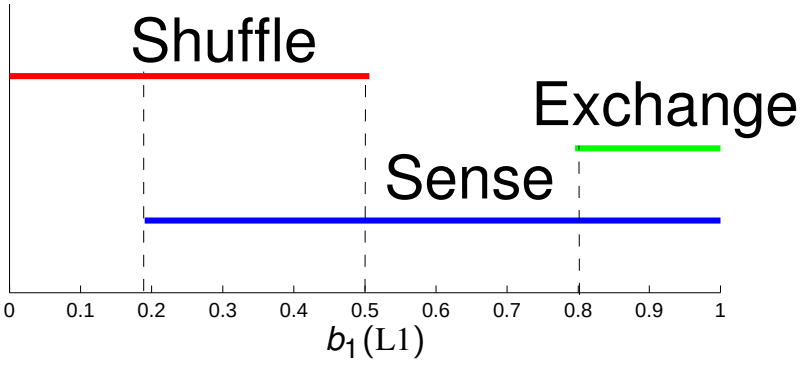
- In Multiagent POMDPs, joint beliefs map to joint actions.
- It is advantageous to factor the belief state itself. (Messias et al., 2011)
  - Agents maintain belief states over locally relevant factors.
  - Joint belief is approximated in factorized form.
- Multiagent POMDP policies are mapped to individual belief spaces
  - Query communication is used when other agents' factors are required.
- Extends ideas of Roth et al. (2007) to the MPOMDP setting.

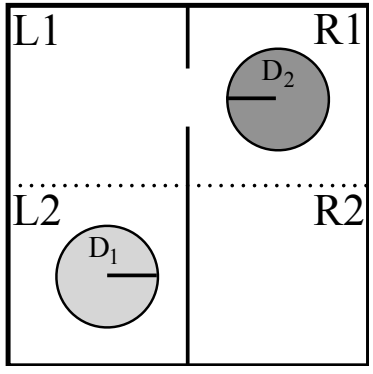
Linear and local supports of the value function

Linear supports:



Local supports:





shuffle; exchange; sense/noop.

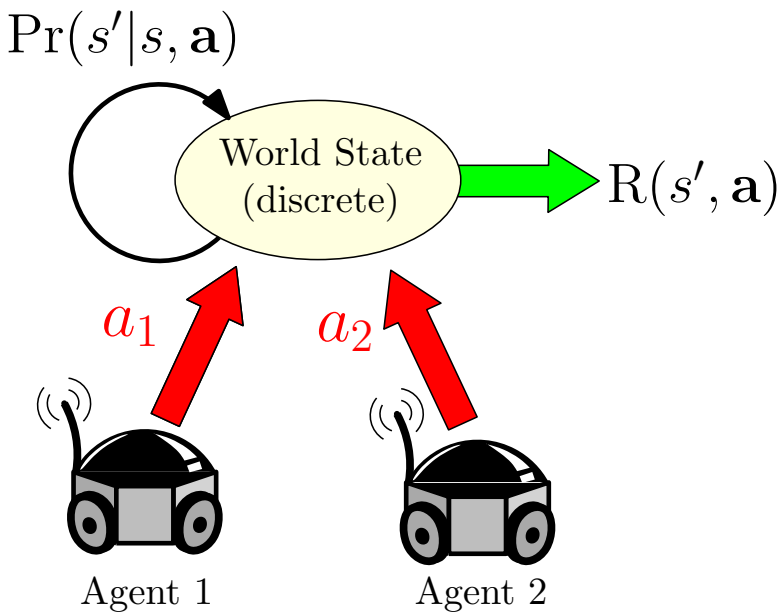
Event-driven models

Planning for asynchronous execution

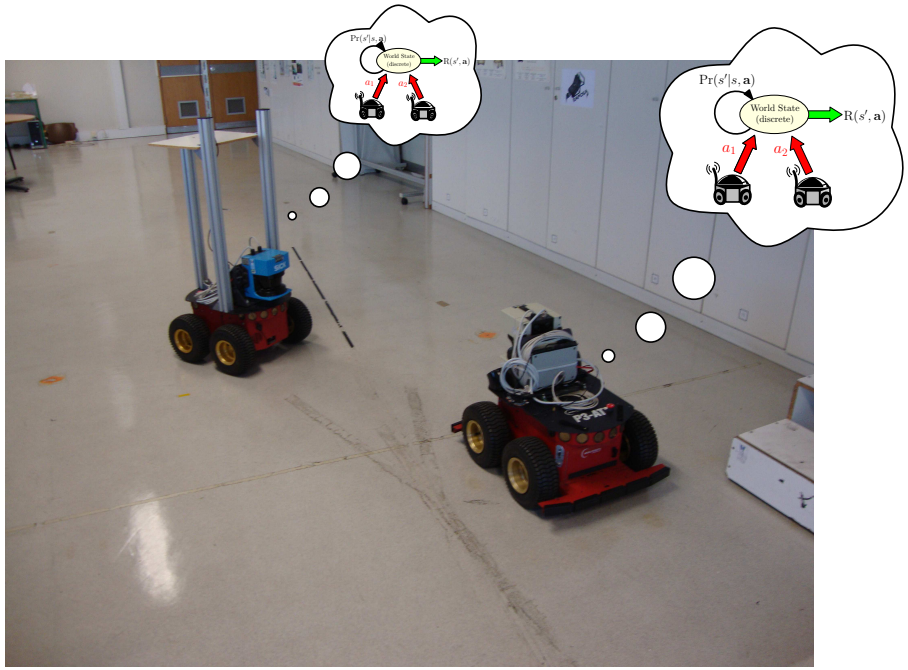
Discrete Multiagent Markov Decision Processes

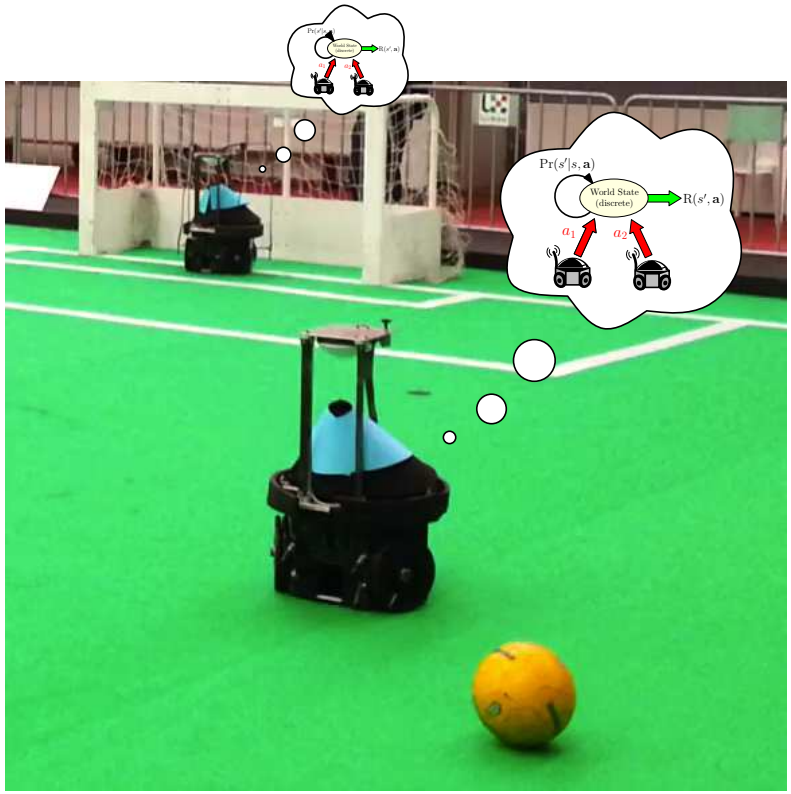
Often multiagent decision-theoretic models are discrete:

- Discrete state space $\mathcal{S}$
- Discrete action space $\mathcal{A}$
- Policies are step-based, **abstract w.r.t. time**: $\pi(s) = \{\delta_1(s), \ldots, \delta_h(s)\}$
- Communication required if only local states can be observed.

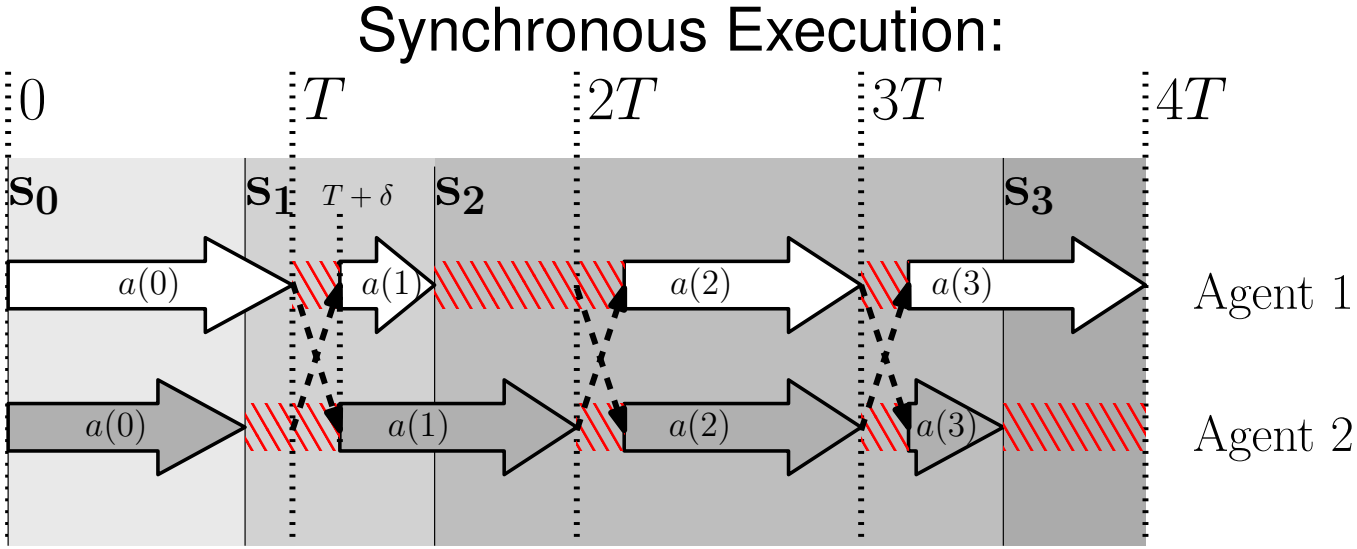


In the Real World...



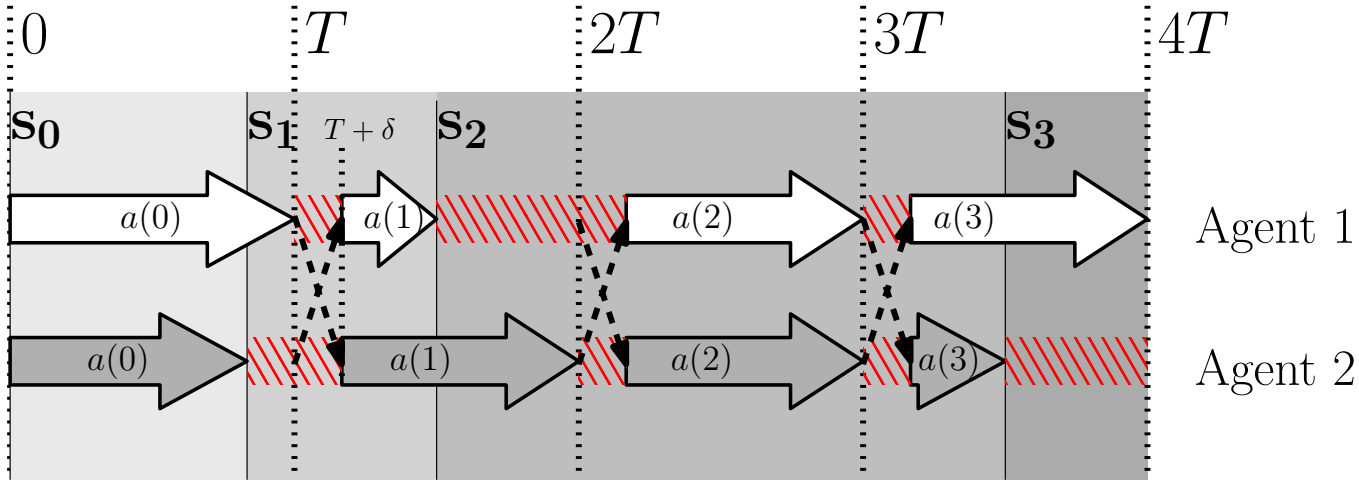


Real-Time Execution Strategies

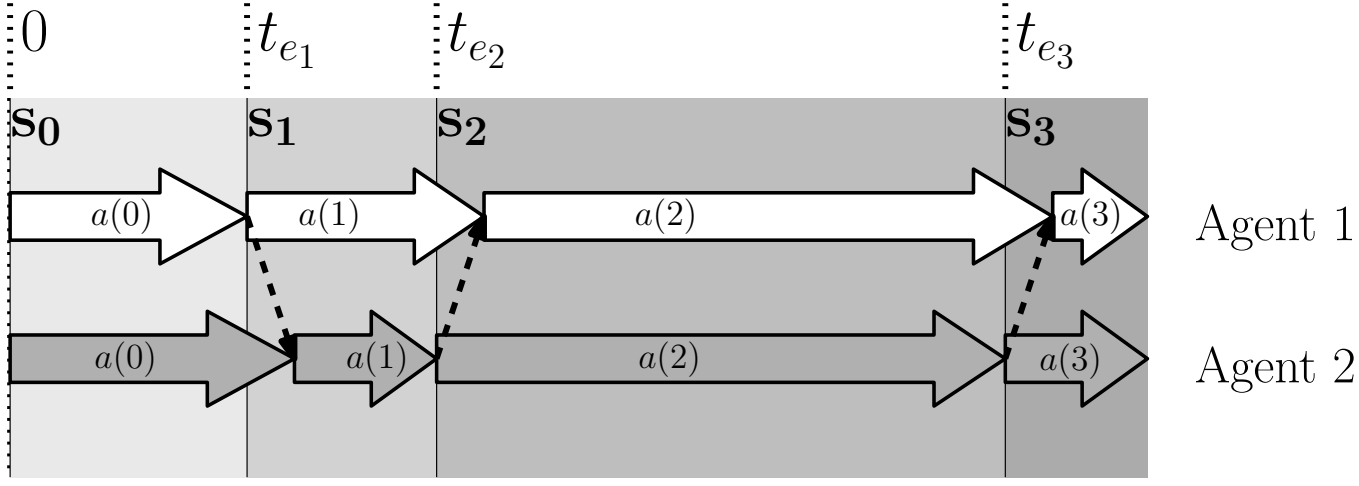


Real-Time Execution Strategies

Synchronous Execution:



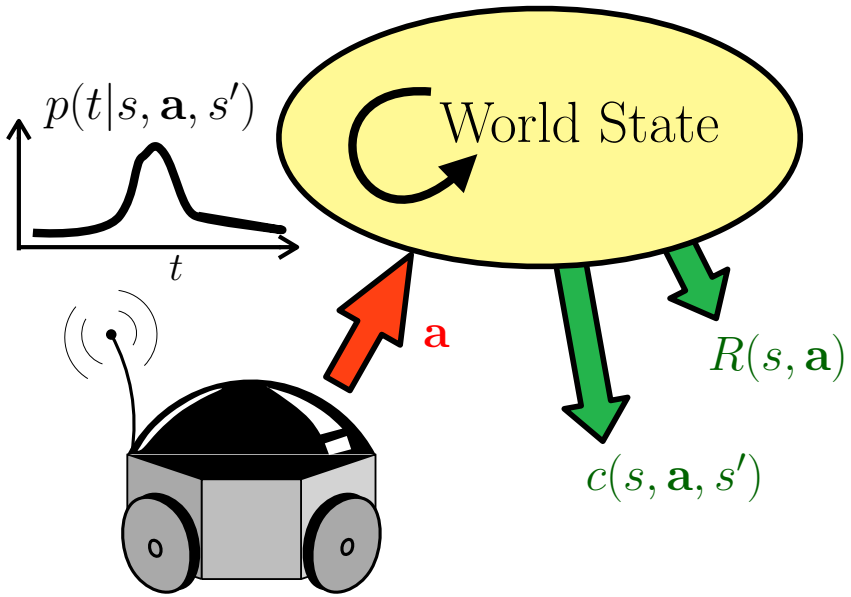
Event-Driven (Asynchronous) Execution:



Semi-Markov Decision Processes

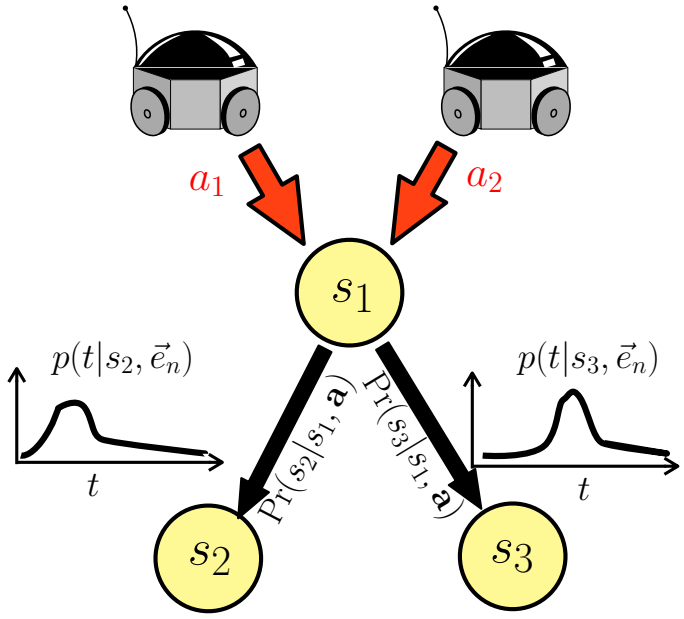
SMDP: Extension of a discrete MDP with a temporal distributions $f(s, \mathbf{a}, s')$:

$$\begin{aligned} p(t, s'|s, \mathbf{a}) &= p(t|s, \mathbf{a}, s')Pr(s'|s, \mathbf{a}) \\ &= f(s, \mathbf{a}, s')T(s, \mathbf{a}, s') \end{aligned}$$



Generalized Semi-Markov Decision Processes

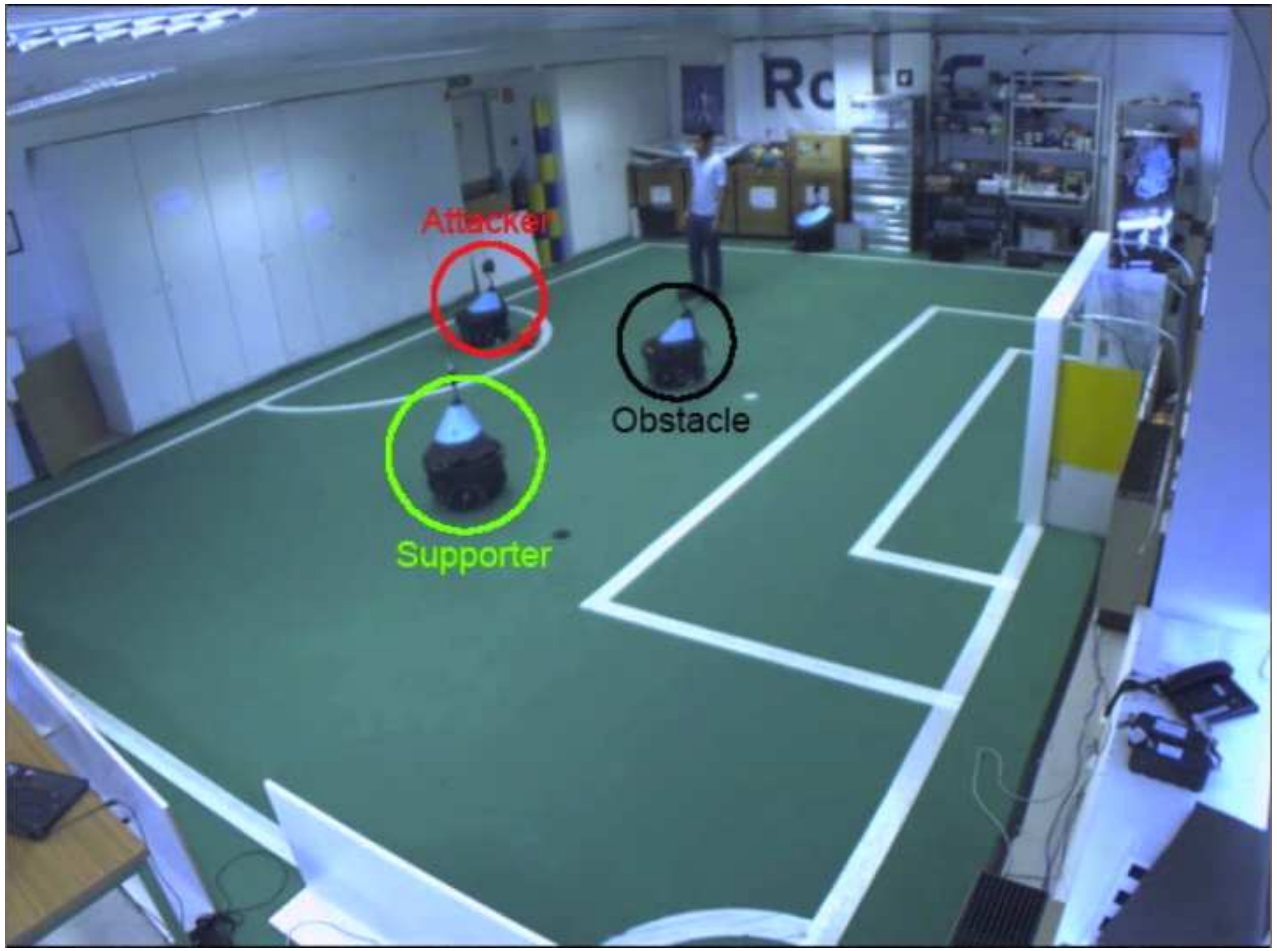
- Generalized SMDPs (Younes and Simmons, 2004)
  - State transitions abstracted as events, $e \in \mathcal{E}$.
  - Transition probabilities also depend on the history of events.
- We apply a GSMDP-based solution to a team of real robots. (Messias et al., 2013)
- Extension to Partially-Observable domains



Experimental Setup

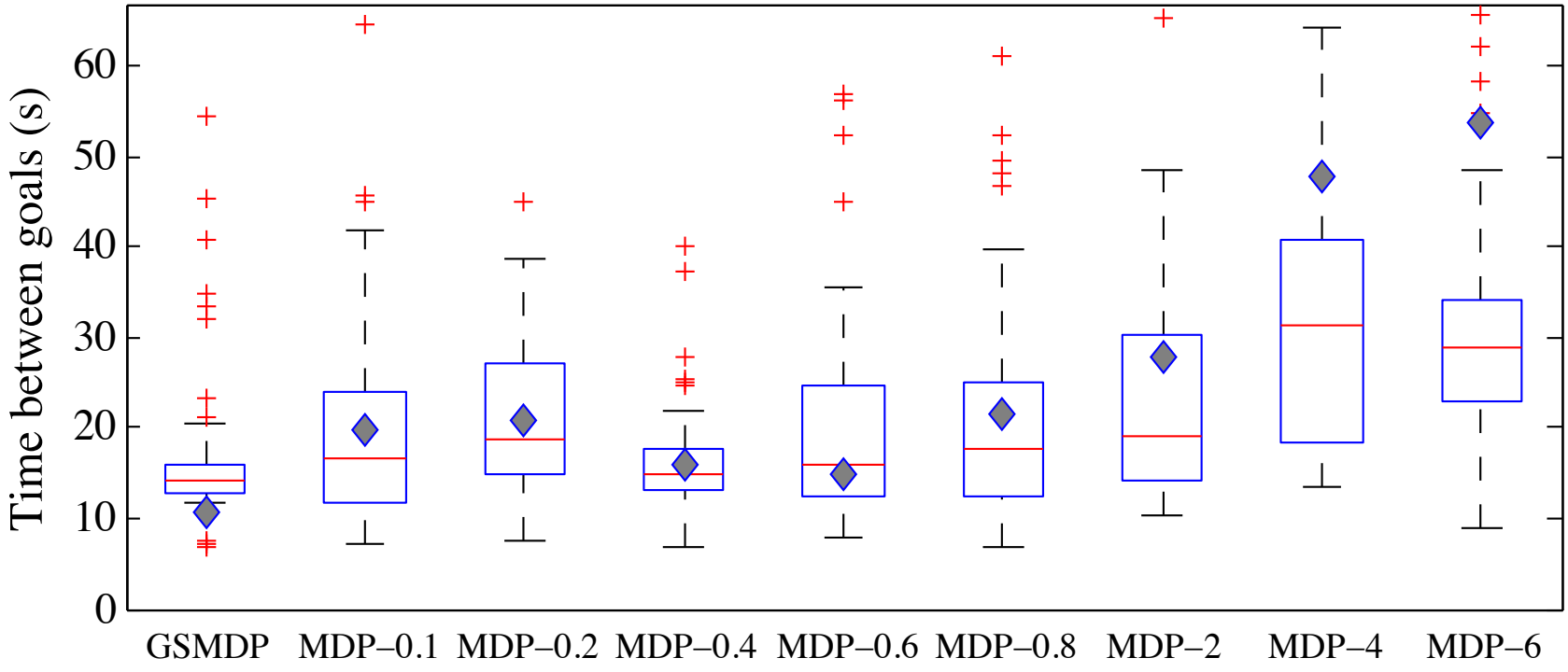
- A cooperative decision-making problem for two robots;
- Implemented both as a GSMDP and a synchronous MDP with configurable decision rate $\rightarrow |\mathcal{S}| = 126$, $|\mathcal{A}| = 36$;
- Results both with realistic physics-based simulation and with real robots.

Robot Results - Video



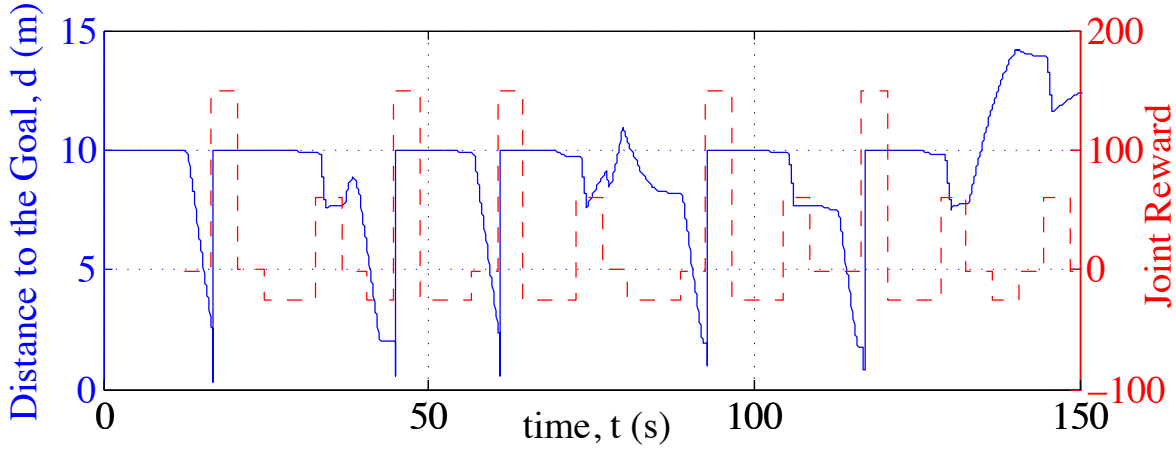
Combined Results - Data

Combined results:

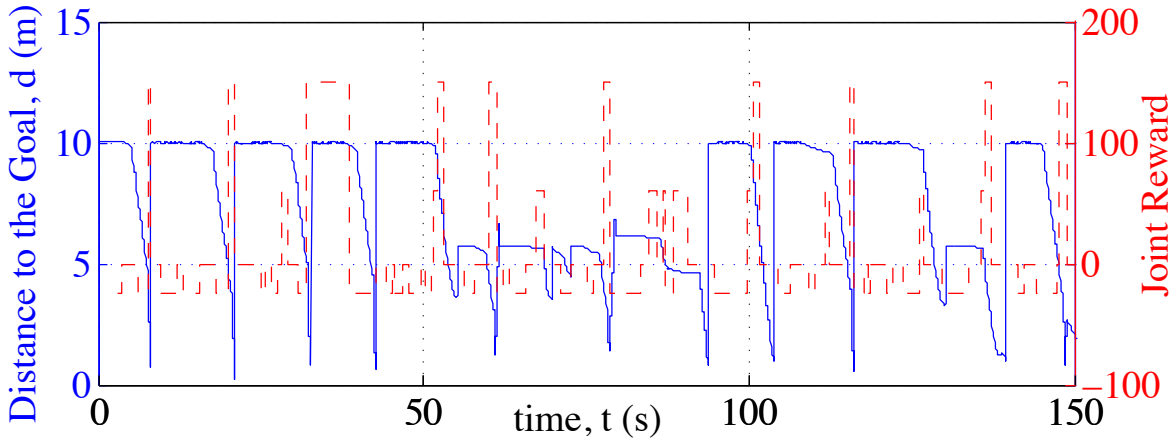


Simulation Results

MDP, T = 4*s*:



GSMDP



Models and algorithms using communication

- Analysis of possible communication models and complexity results (Pynadath and Tambe, 2002) (Goldman and Zilberstein, 2004)

Models and algorithms using communication

- Analysis of possible communication models and complexity results (Pynadath and Tambe, 2002) (Goldman and Zilberstein, 2004)
- Myopic communication in transition independent Dec-MDPs (Becker et al., 2009)

Models and algorithms using communication

- Analysis of possible communication models and complexity results (Pynadath and Tambe, 2002) (Goldman and Zilberstein, 2004)
- Myopic communication in transition independent Dec-MDPs (Becker et al., 2009)
- Reasoning about run-time synchronization decisions (Nair et al., 2004), (Roth et al., 2005)

Models and algorithms using communication

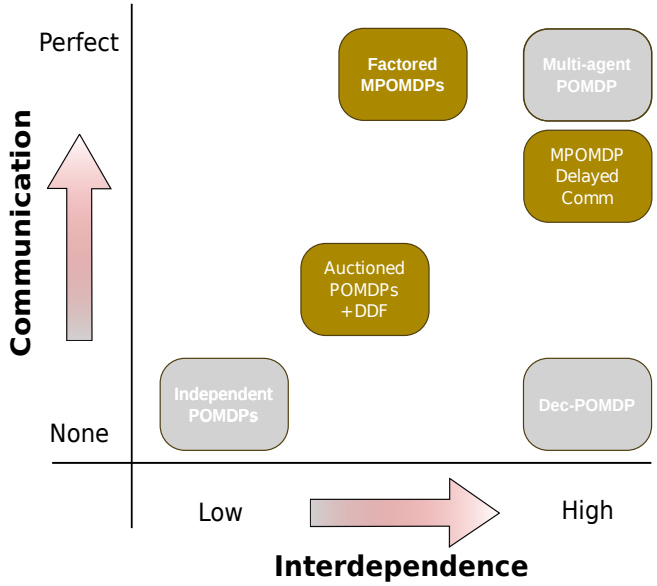
- Analysis of possible communication models and complexity results (Pynadath and Tambe, 2002) (Goldman and Zilberstein, 2004)
- Myopic communication in transition independent Dec-MDPs (Becker et al., 2009)
- Reasoning about run-time synchronization decisions (Nair et al., 2004), (Roth et al., 2005)
- Exploiting factored Dec-MDP representations (Roth et al., 2007)

Models and algorithms using communication

- Analysis of possible communication models and complexity results (Pynadath and Tambe, 2002) (Goldman and Zilberstein, 2004)
- Myopic communication in transition independent Dec-MDPs (Becker et al., 2009)
- Reasoning about run-time synchronization decisions (Nair et al., 2004), (Roth et al., 2005)
- Exploiting factored Dec-MDP representations (Roth et al., 2007)
- Multi-robot cooperation with auctioned POMDPs (Capitán et al., 2013)

Communication – Conclusions

- Multiagent planning under uncertainty can exploit communication.
- Real-world systems often can communicate, but not perfectly.
- Sparse dependencies can lead to sparse communication needs.

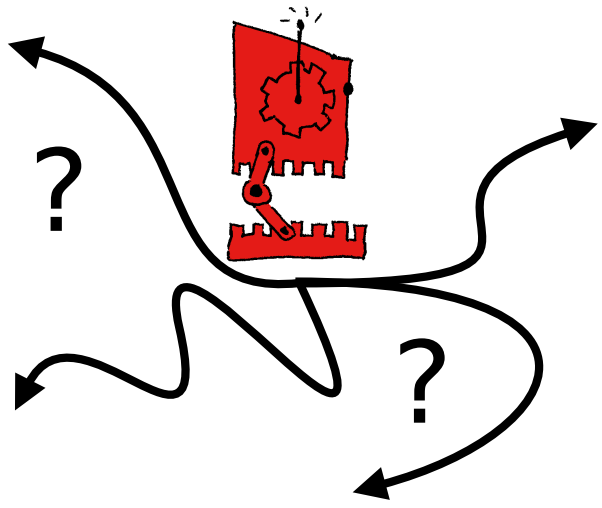


Particular issues:

1. Learning semantics
2. Deal with imperfect communication
3. Minimizing communication
4. Asynchronous execution

Conclusions

- Decision-theoretic models provide a principled framework for planning under uncertainty
  - Acting and sensing uncertainty
  - Single and multiple agents
  - Communication
- Software and solvers available



Current challenges

- Identifying structure in domains or policies, e.g., influences
- Considering richer communication models

Further reading

- Textbook on reinforcement learning
  - R. S. Sutton and A. G. Barto. “Reinforcement Learning: An Introduction”. MIT Press, 1998.
- Recent book containing chapters on many aspects of decision-theoretic planning:
  - M. Wiering and M. van Otterlo, editors, “Reinforcement Learning: State of the Art”, Springer, 2012.
- Software and other resources:

  MultiAgentDecisionProcess toolbox

  `http://masplan.org`

References I

R. Becker, A. Carlin, V. Lesser, and S. Zilberstein. Analyzing myopic approaches for multi-agent communications. *Computational Intelligence*, 25(1):31–50, 2009.

R. Bellman. *Dynamic programming*. Princeton University Press, 1957.

D. P. Bertsekas. *Dynamic Programming and Optimal Control*. Athena Scientific, Belmont, MA, 2nd edition, 2000.

B. Bonet. An epsilon-optimal grid-based algorithm for partially observable Markov decision processes. In *International Conference on Machine Learning*, 2002.

R. I. Brafman. A heuristic variable grid solution method for POMDPs. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence*, 1997.

J. Capitán, M. T. J. Spaan, L. Merino, and A. Ollero. Decentralized multi-robot cooperation with auctioned POMDPs. *International Journal of Robotics Research*, 32(6):650–671, 2013.

A. R. Cassandra, L. P. Kaelbling, and M. L. Littman. Acting optimally in partially observable stochastic domains. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, 1994.

A. R. Cassandra, L. P. Kaelbling, and J. A. Kurien. Acting under uncertainty: Discrete Bayesian models for mobile robot navigation. In *Proc. of International Conference on Intelligent Robots and Systems*, 1996.

A. R. Cassandra, M. L. Littman, and N. L. Zhang. Incremental pruning: A simple, fast, exact method for partially observable Markov decision processes. In *Proc. of Uncertainty in Artificial Intelligence*, 1997.

H. T. Cheng. *Algorithms for partially observable Markov decision processes*. PhD thesis, University of British Columbia, 1988.

A. W. Drake. *Observation of a Markov process through a noisy channel*. Sc.D. thesis, Massachusetts Institute of Technology, 1962.

C. V. Goldman and S. Zilberstein. Decentralized control of cooperative systems: Categorization and complexity analysis. *Journal of Artificial Intelligence Research*, 22:143–174, 2004.

E. A. Hansen. *Finite-memory control of partially observable systems*. PhD thesis, University of Massachusetts, Amherst, 1998a.

E. A. Hansen. Solving POMDPs by searching in policy space. In *Proc. of Uncertainty in Artificial Intelligence*, 1998b.

K. Hsu and S. I. Marcus. Decentralized control of finite state Markov processes. *IEEE Transactions on Automatic Control*, 27(2):426–431, 1982.

References II

L. P. Kaelbling, M. L. Littman, and A. R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101:99–134, 1998.

H. Kurniawati, D. Hsu, and W. Lee. SARSOP: Efficient point-based POMDP planning by approximating optimally reachable belief spaces. In *Robotics: Science and Systems*, 2008.

M. L. Littman, A. R. Cassandra, and L. P. Kaelbling. Learning policies for partially observable environments: Scaling up. In *International Conference on Machine Learning*, 1995.

W. S. Lovejoy. Computationally feasible bounds for partially observed Markov decision processes. *Operations Research*, 39(1):162–175, 1991.

J. V. Messias, M. T. J. Spaan, and P. U. Lima. Efficient offline communication policies for factored multiagent POMDPs. In *Advances in Neural Information Processing Systems*, pages 1917–1925, 2011.

J. V. Messias, M. T. J. Spaan, and P. U. Lima. GSMDPs for multi-robot sequential decision-making. In *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence*, pages 1408–1414, 2013.

G. E. Monahan. A survey of partially observable Markov decision processes: theory, models and algorithms. *Management Science*, 28(1), Jan. 1982.

R. Nair, M. Tambe, M. Roth, and M. Yokoo. Communication for improving policy computation in distributed POMDPs. In *Proc. of Int. Conference on Autonomous Agents and Multi Agent Systems*, 2004.

F. A. Oliehoek and M. T. J. Spaan. Tree-based solution methods for multiagent POMDPs with delayed communication. In *Proc. of the AAAI Conference on Artificial Intelligence*, pages 1415–1421, 2012a.

F. A. Oliehoek and M. T. J. Spaan. Tree-based pruning for multiagent POMDPs with delayed communication. In *Proc. of Int. Conference on Autonomous Agents and Multi Agent Systems*, pages 1229–1230, 2012b.

F. A. Oliehoek, M. T. J. Spaan, and N. Vlassis. Optimal and approximate Q-value functions for decentralized POMDPs. *Journal of Artificial Intelligence Research*, 32:289–353, 2008.

R. Parr and S. Russell. Approximating optimal policies for partially observable stochastic domains. In *Proc. Int. Joint Conf. on Artificial Intelligence*, 1995.

J. Pineau, G. Gordon, and S. Thrun. Point-based value iteration: An anytime algorithm for POMDPs. In *Proc. Int. Joint Conf. on Artificial Intelligence*, 2003.

References III

L. K. Platzman. A feasible computational approach to infinite-horizon partially-observed Markov decision problems. Technical Report J-81-2, School of Industrial and Systems Engineering, Georgia Institute of Technology, 1981. Reprinted in working notes AAAI 1998 Fall Symposium on Planning with POMDPs.

P. Poupart and C. Boutilier. Value-directed compression of POMDPs. In *Advances in Neural Information Processing Systems 15*. MIT Press, 2003.

P. Poupart and C. Boutilier. Bounded finite state controllers. In *Advances in Neural Information Processing Systems 16*. MIT Press, 2004.

M. L. Puterman. *Markov Decision Processes—Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., New York, NY, 1994.

D. V. Pynadath and M. Tambe. The communicative multiagent team decision problem: Analyzing teamwork theories and models. *Journal of Artificial Intelligence Research*, 16:389–423, 2002.

M. Roth, R. Simmons, and M. Veloso. Reasoning about joint beliefs for execution-time communication decisions. In *Proc. of Int. Conference on Autonomous Agents and Multi Agent Systems*, 2005.

M. Roth, R. Simmons, and M. Veloso. Exploiting factored representations for decentralized execution in multi-agent teams. In *Proc. of Int. Conference on Autonomous Agents and Multi Agent Systems*, 2007.

N. Roy, G. Gordon, and S. Thrun. Finding approximate POMDP solutions through belief compression. *Journal of Artificial Intelligence Research*, 23:1–40, 2005.

S. J. Russell and P. Norvig. *Artificial Intelligence: a modern approach*. Prentice Hall, 2nd edition, 2003.

J. K. Satia and R. E. Lave. Markovian decision processes with probabilistic observation of states. *Management Science*, 20(1):1–13, 1973.

G. Shani, R. I. Brafman, and S. E. Shimony. Forward search value iteration for POMDPs. In *Proc. Int. Joint Conf. on Artificial Intelligence*, 2007.

D. Silver and J. Veness. Monte-Carlo planning in large POMDPs. In *Advances in Neural Information Processing Systems 23*, 2010.

S. Singh, T. Jaakkola, and M. Jordan. Learning without state-estimation in partially observable Markovian decision processes. In *International Conference on Machine Learning*, 1994.

R. D. Smallwood and E. J. Sondik. The optimal control of partially observable Markov decision processes over a finite horizon. *Operations Research*, 21:1071–1088, 1973.

References IV

T. Smith and R. Simmons. Heuristic search value iteration for POMDPs. In *Proc. of Uncertainty in Artificial Intelligence*, 2004.

E. J. Sondik. *The optimal control of partially observable Markov processes*. PhD thesis, Stanford University, 1971.

M. T. J. Spaan and N. Vlassis. Perseus: Randomized point-based value iteration for POMDPs. *Journal of Artificial Intelligence Research*, 24:195–220, 2005.

M. T. J. Spaan, G. J. Gordon, and N. Vlassis. Decentralized planning under uncertainty for teams of communicating agents. In *Proc. of Int. Conference on Autonomous Agents and Multi Agent Systems*, pages 249–256, 2006.

M. T. J. Spaan, F. A. Oliehoek, and N. Vlassis. Multiagent planning under uncertainty with stochastic communication delays. In *Proc. of Int. Conf. on Automated Planning and Scheduling*, pages 338–345, 2008.

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

P. Varaiya and J. Walrand. On delayed sharing patterns. *IEEE Transactions on Automatic Control*, 23(3):443–445, 1978.

Y. Virin, G. Shani, S. E. Shimony, and R. Brafman. Scaling up: Solving POMDPs through value based clustering. In *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence*, 2007.

H. Younes and R. Simmons. Solving generalized semi-Markov decision processes using continuous phase-type distributions. In *Proceedings of the Nineteenth National Conference on Artificial Intelligence*, pages 742–747, 2004.

N. L. Zhang and W. Liu. Planning in stochastic domains: problem characteristics and approximations. Technical Report HKUST-CS96-31, Department of Computer Science, The Hong Kong University of Science and Technology, 1996.

R. Zhou and E. A. Hansen. An improved grid-based approximation algorithm for POMDPs. In *Proc. Int. Joint Conf. on Artificial Intelligence*, 2001.