Decision Making under Uncertainty

Matthijs Spaan

Delft University of Technology
Delft, The Netherlands

March 5, 2013

Outline

This lecture:

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

This afternoon: start of lab course.

Next week:

1. Multiagent planning
2. Selected further topics

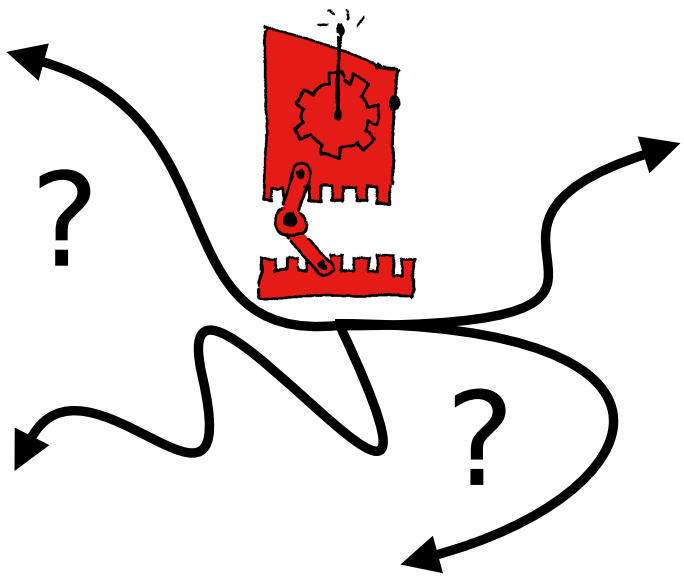
Important:

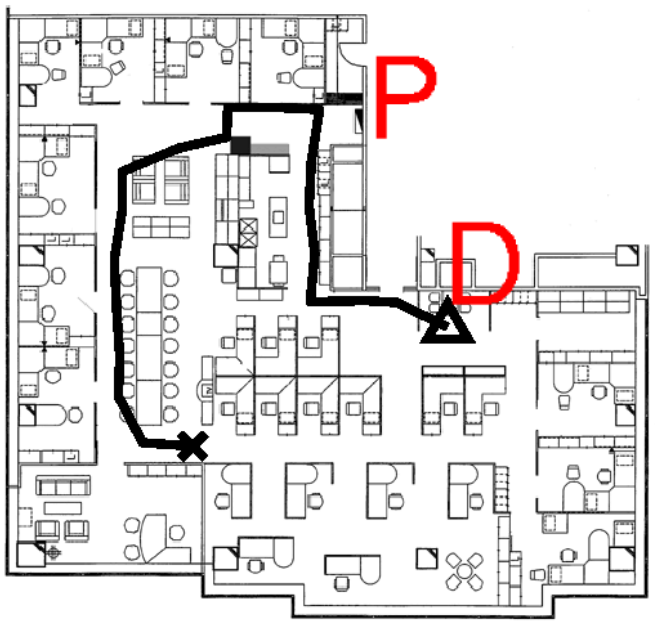
- To pass this module: lab report.
- To pass the course: pass all modules.

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.



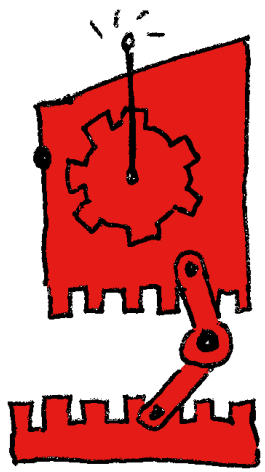


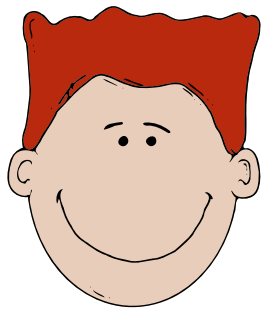
Applications

- Resource planning
- Maintenance
- Queue management
- Medical decision making

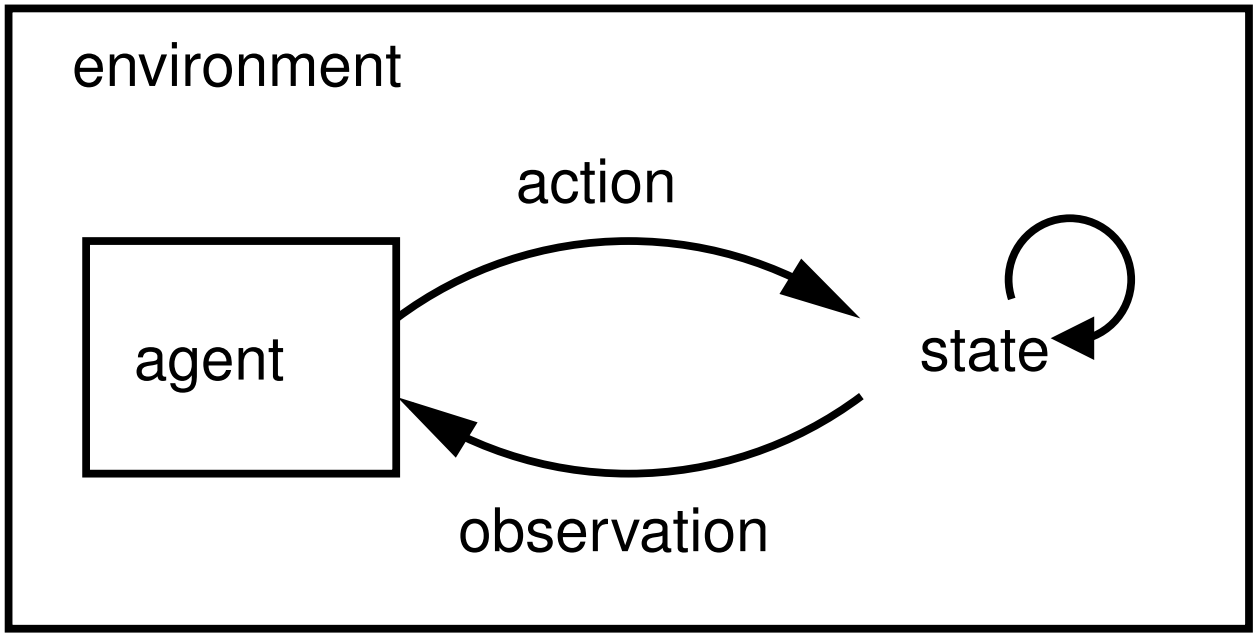
Agents

- 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.

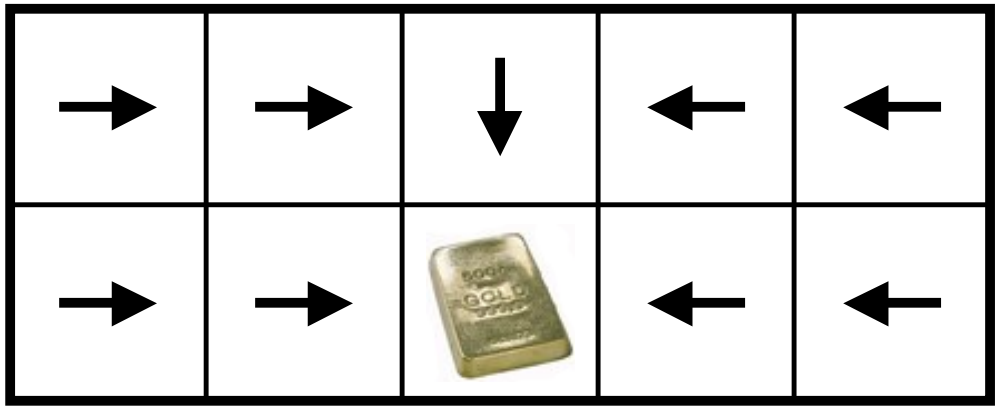
Classic planning

- 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.

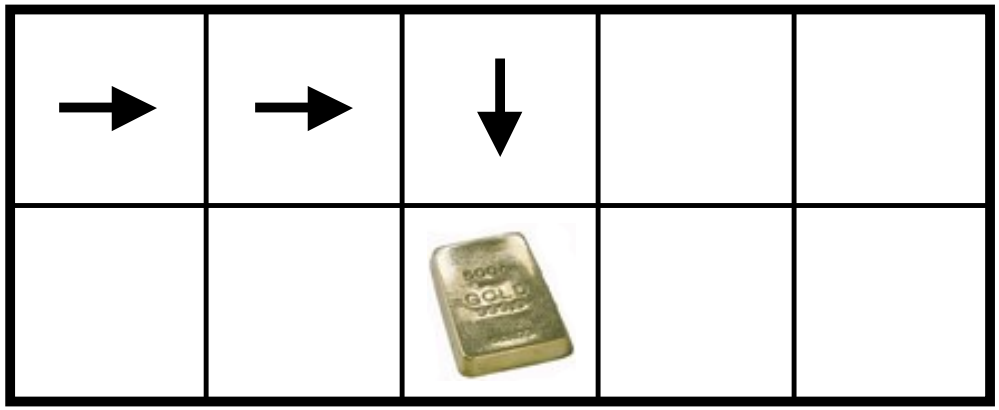
Decision-theoretic planning

−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:

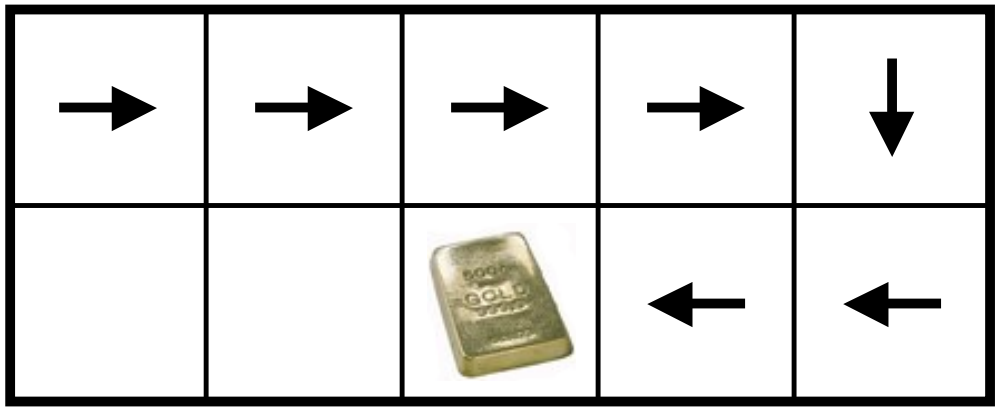
9.7	9.8	9.9		
		10		

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:

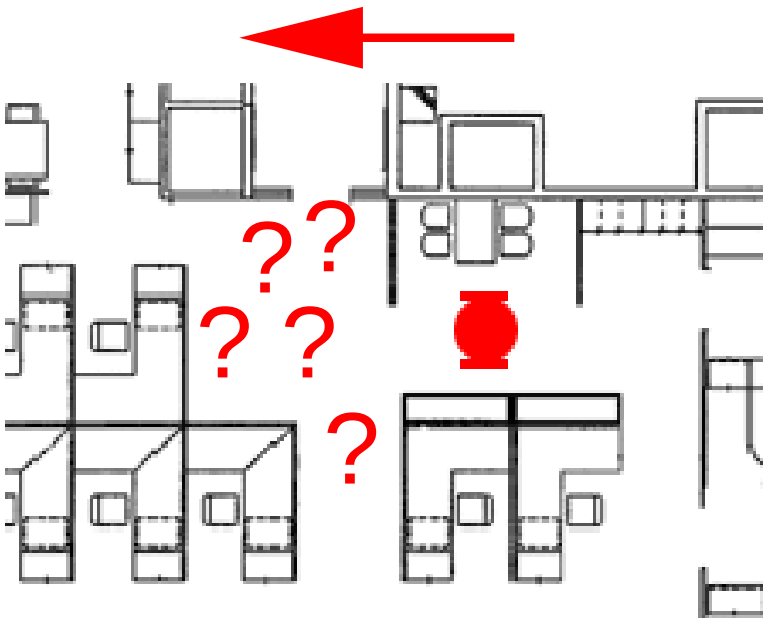
Sequential decisions: problems are formulated as a sequence of "independent" decisions;

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

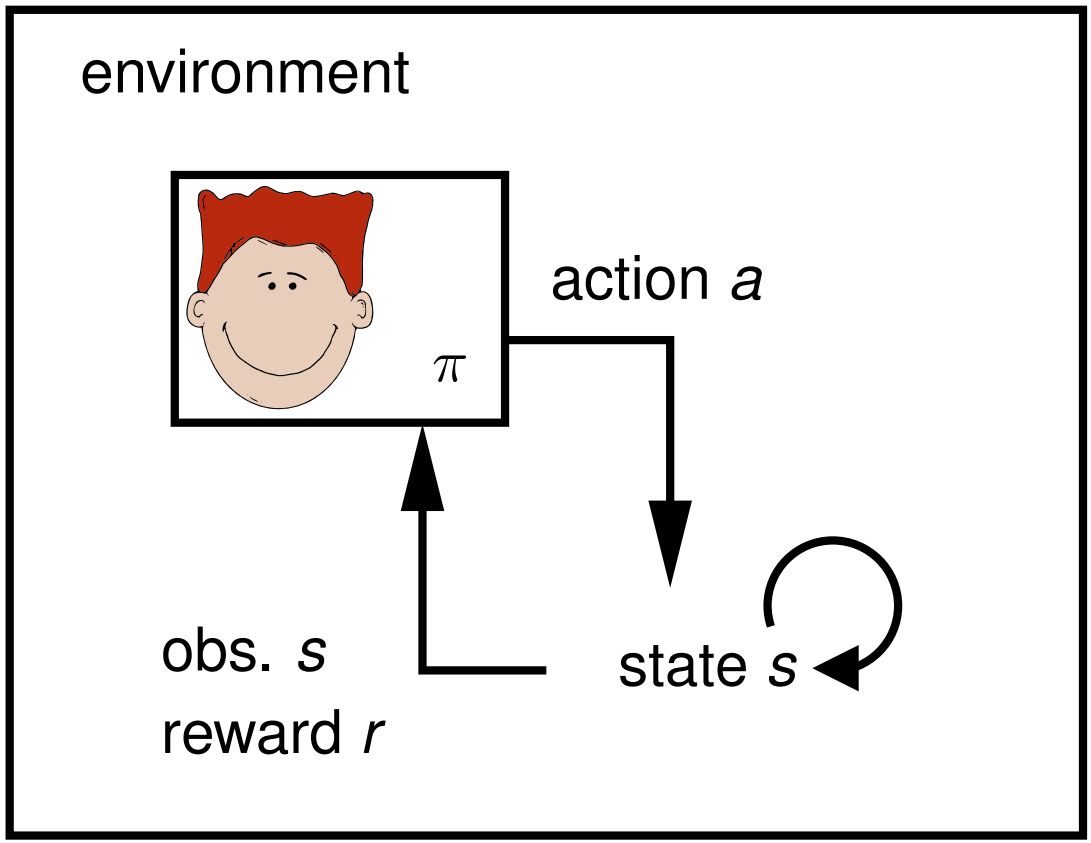
Evaluative feedback: use of a reinforcement signal as performance measure (reinforcement learning);

Transition model

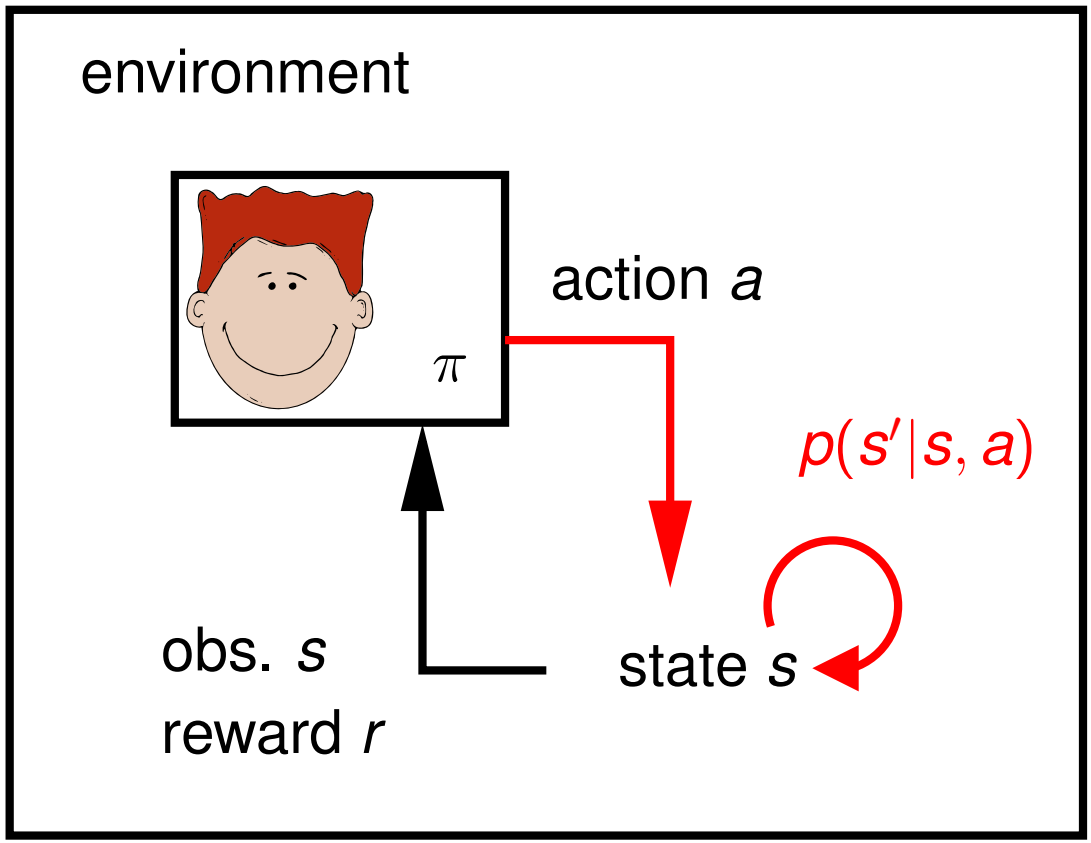
- For instance, robot motion is inaccurate.
- Transitions between states are **stochastic**.
- $p(s'|s, a)$ is the probability to jump from state s to state s' after taking action a.



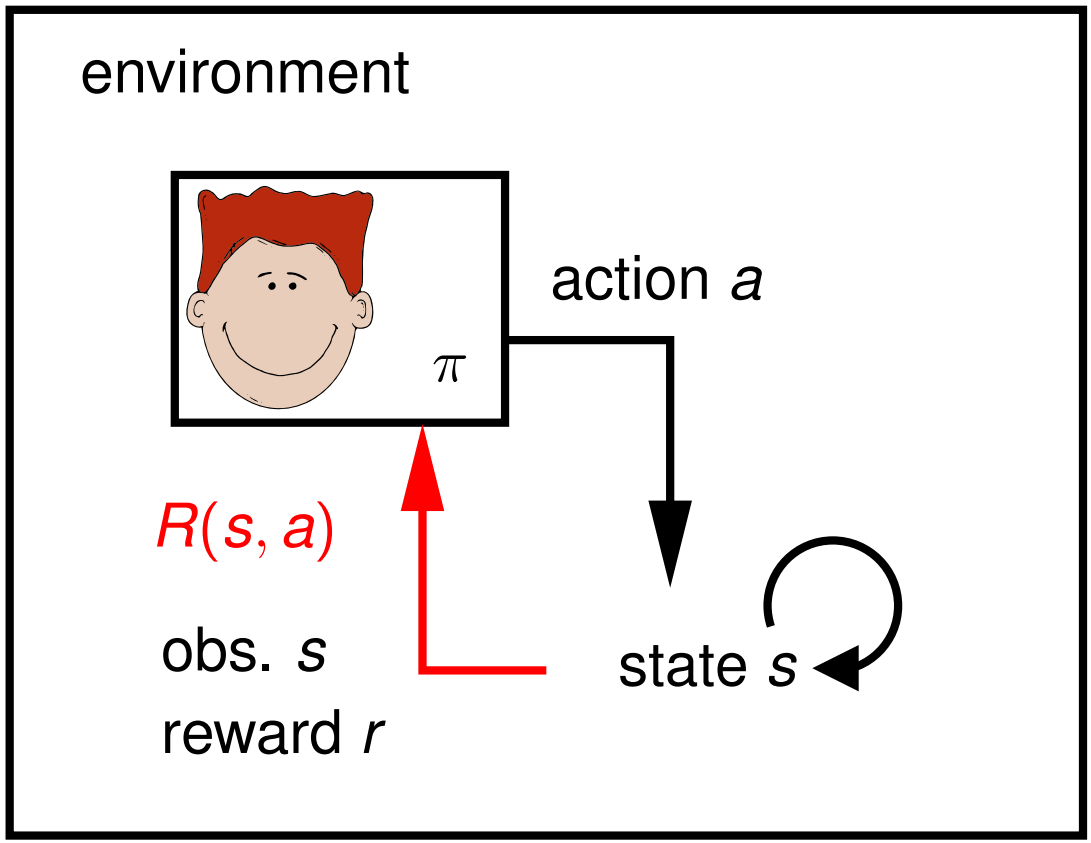
MDP Agent



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):

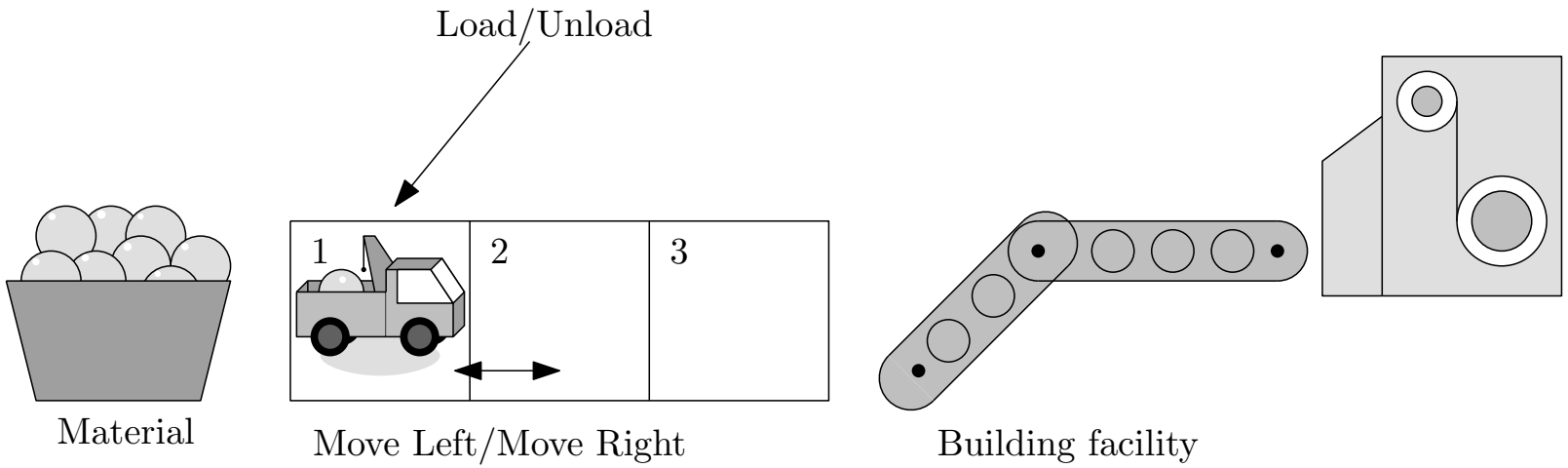
- Time t is discrete.
- State space S.
- Set of actions A.
- Reward function $R : S \times A \mapsto \mathbb{R}$.
- Transition model $p(s'|s, a)$, $T_a : S \times A \mapsto \Delta(S)$.
- Initial state s_0 is drawn from $\Delta(S)$.

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

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

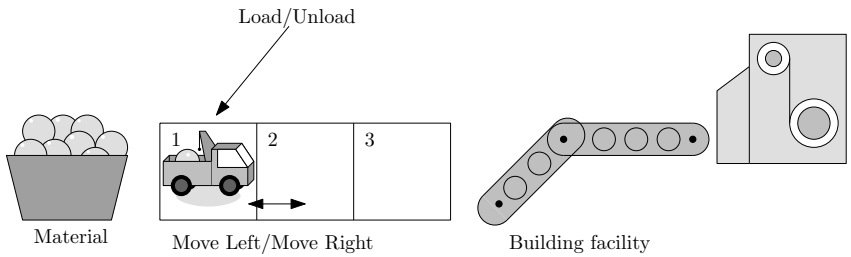
A simple problem

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: $A = \{\text{Left, Right, Load, Unload}\}$;

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:

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

- 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 (1)

Initialize Q arbitrarily, e.g., $Q(s,a) = 0, \forall s \in S, a \in A$
repeat
  $\delta \leftarrow 0$
  for all $s \in S, a \in A$ **do**
    $v \leftarrow Q(s,a)$
    $Q(s,a) \leftarrow R(s,a) + \gamma \sum_{s' \in S} p(s'|s,a) \max_{a' \in A} Q(s',a')$
    $\delta \leftarrow \max(\delta, |v - Q(s,a)|)$
  end for
until $\delta < \epsilon$
Return Q

Value iteration (2)

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.

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

- Real agents cannot directly observe the state.
- Sensors provide partial and noisy information about the world.

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

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

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

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

Beyond MDPs

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

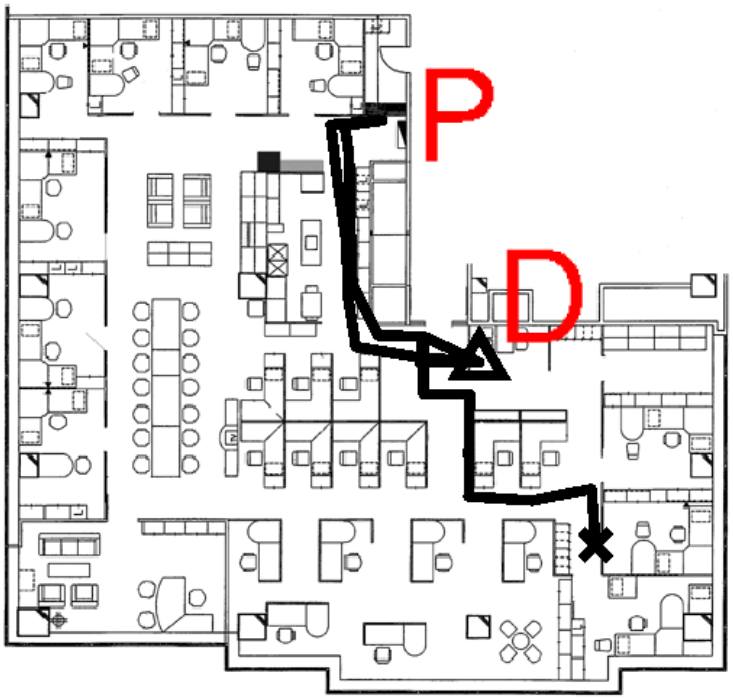
- Diagnosis (medical, maintenance)
- Robot navigation
- Tutoring
- Dialog systems
- Vision-based robotics
- Fault recovery

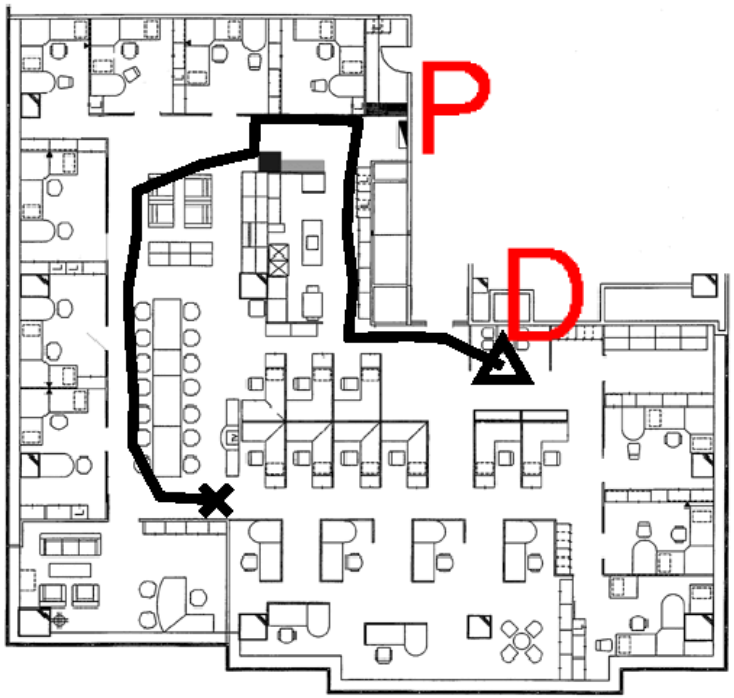
Beyond MDPs

- 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

A partially observable problem





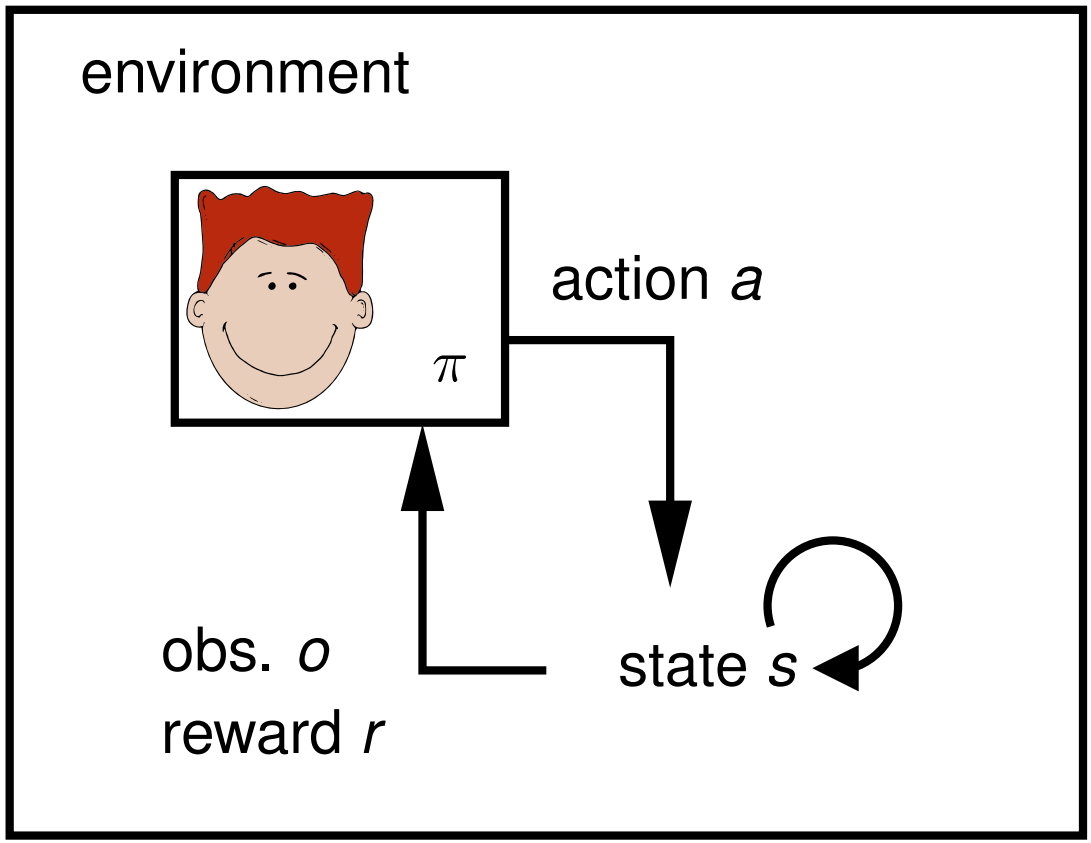
Task: start at random position ($\times$) $\rightarrow$ pick up mail at P $\rightarrow$ deliver mail at D ($\triangle$).

Characteristics: motion noise, perceptual aliasing.

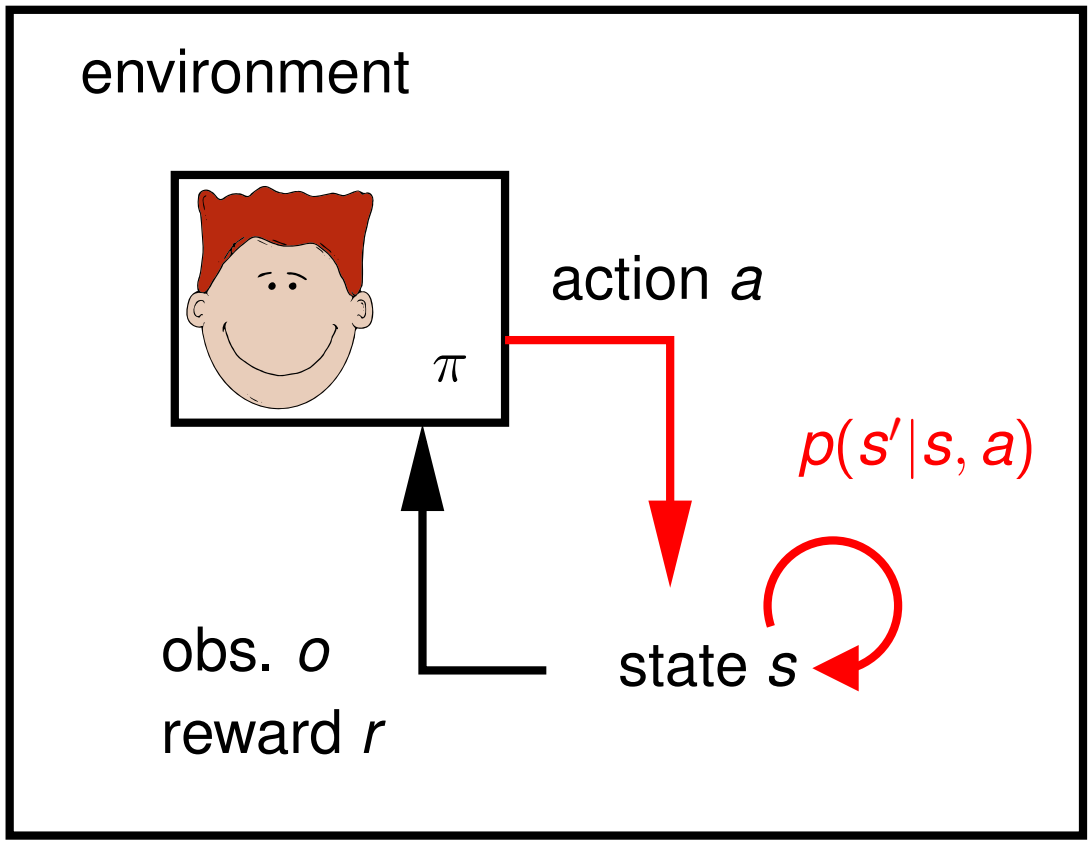
Observation model

- 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.

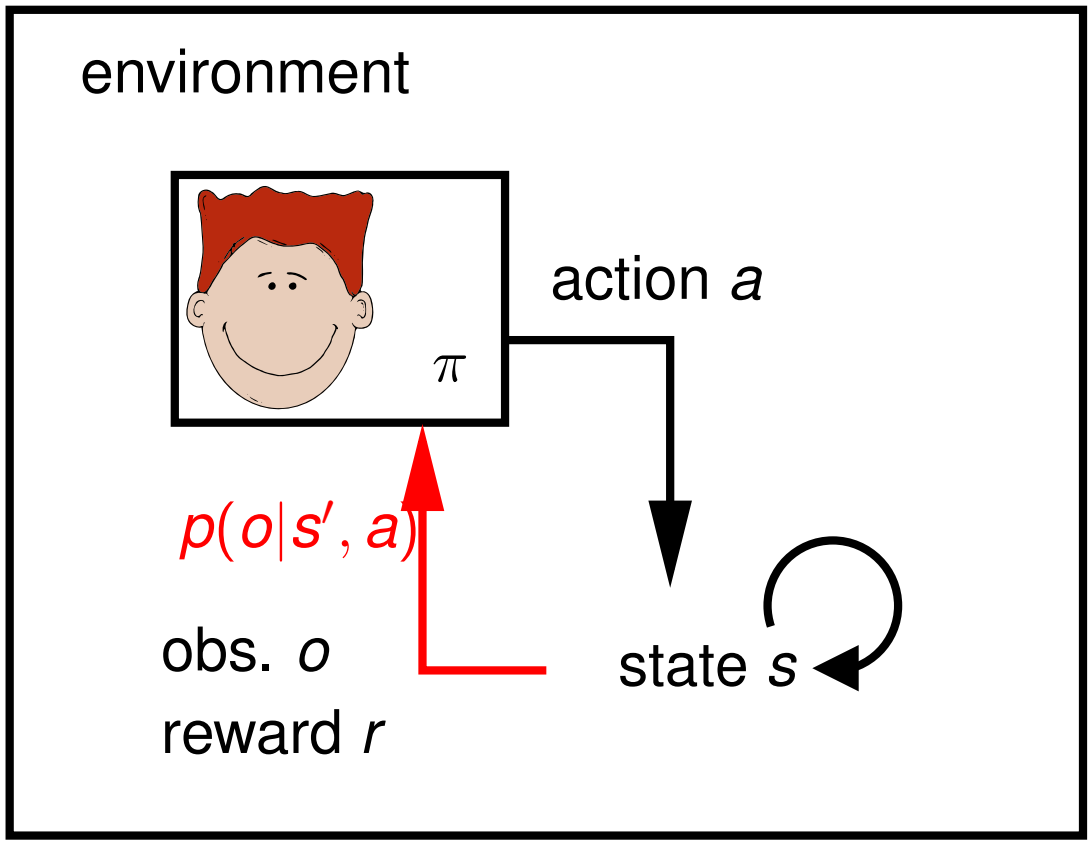
POMDP Agent



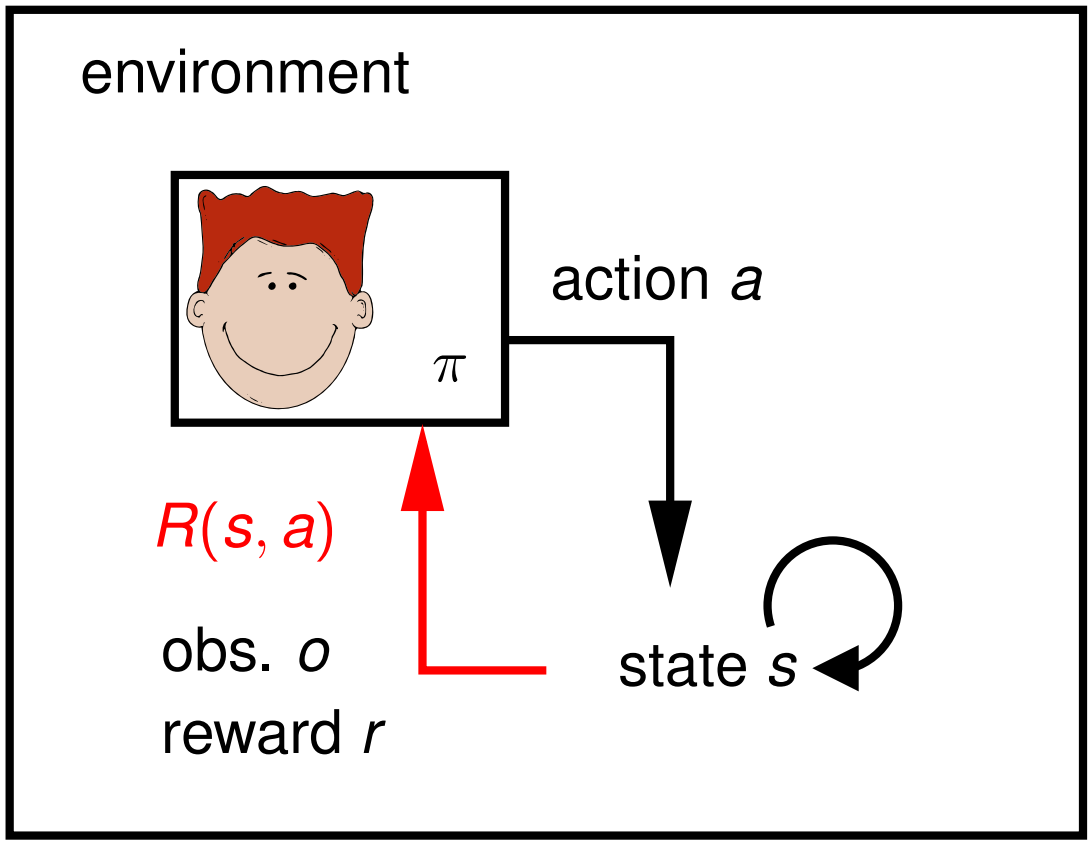
POMDP Agent



POMDP Agent



POMDP Agent



POMDPs

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

- Framework for agent planning under uncertainty.
- 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.

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.

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.

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.

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.

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.

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.

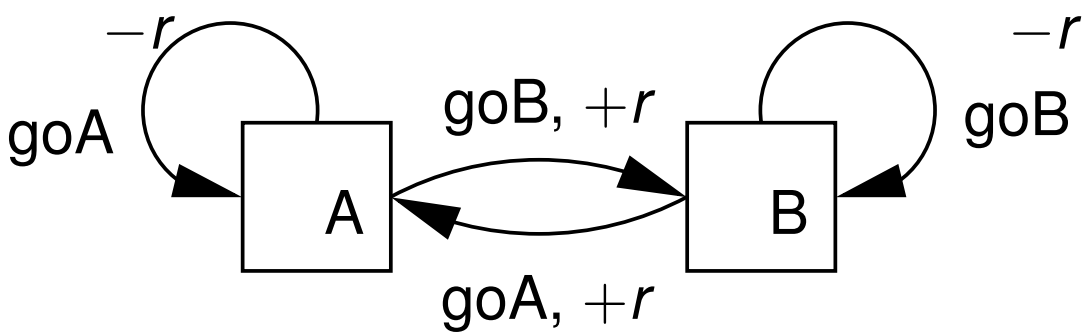
POMDPs

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.

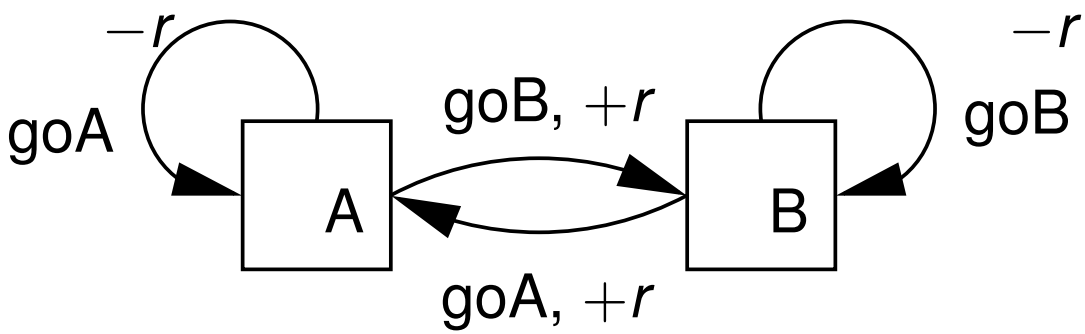
Memory

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



Memory

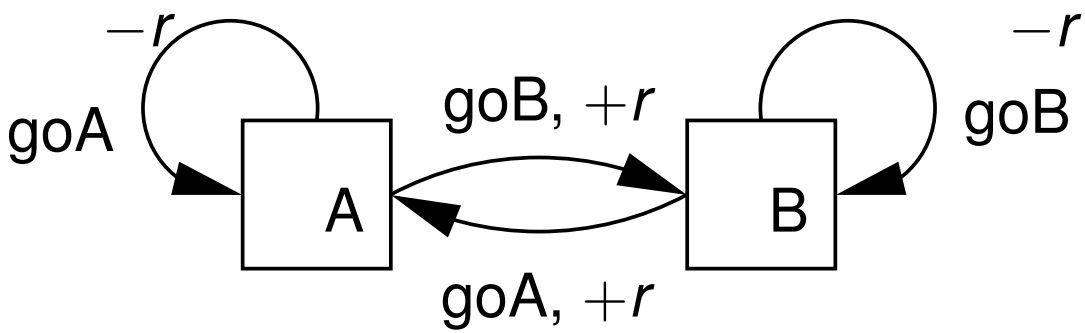
- In POMDPs memory is required for optimal decision making.
- In this non-observable example (Singh et al., 1994):



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

Memory

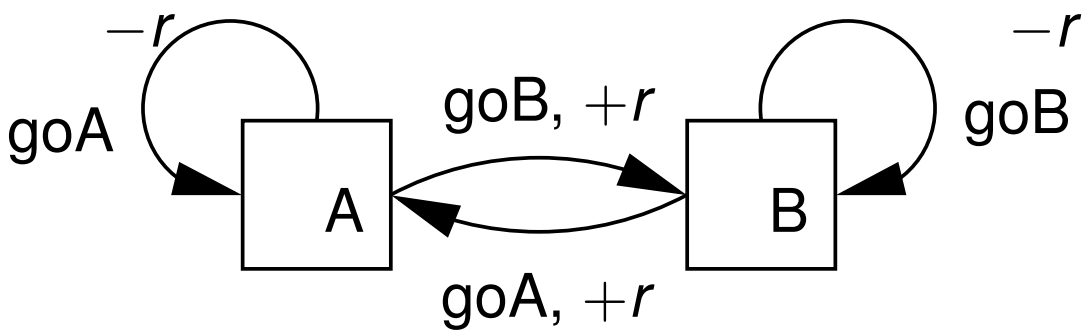
- In POMDPs memory is required for optimal decision making.
- In this non-observable example (Singh et al., 1994):



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

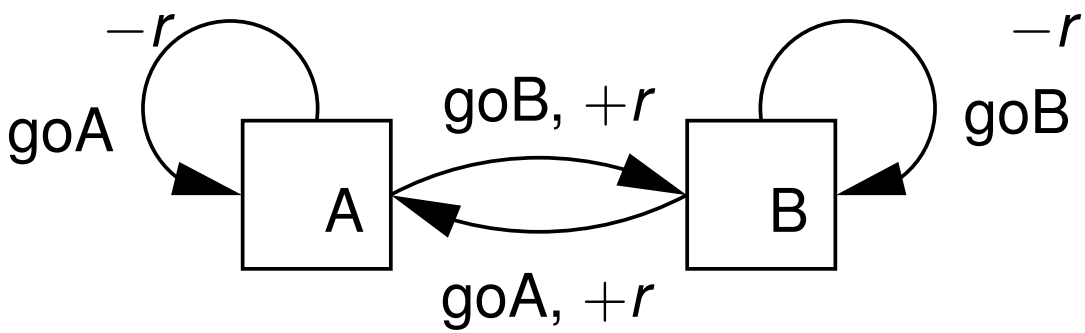
- In POMDPs memory is required for optimal decision making.
- In this non-observable example (Singh et al., 1994):



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

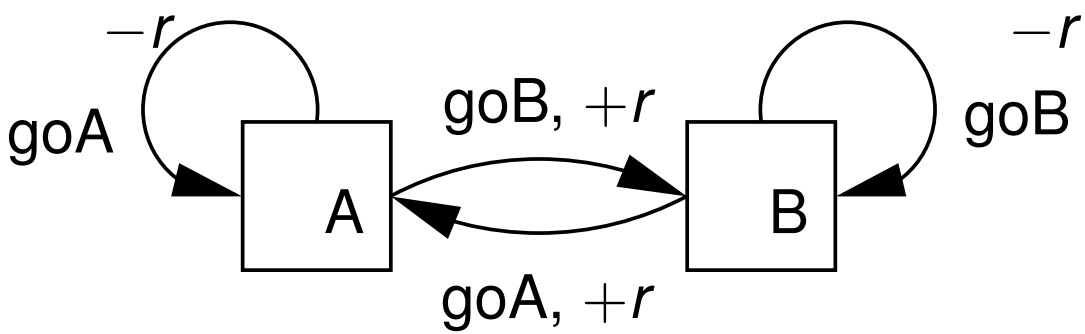
- In POMDPs memory is required for optimal decision making.
- In this non-observable example (Singh et al., 1994):



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:

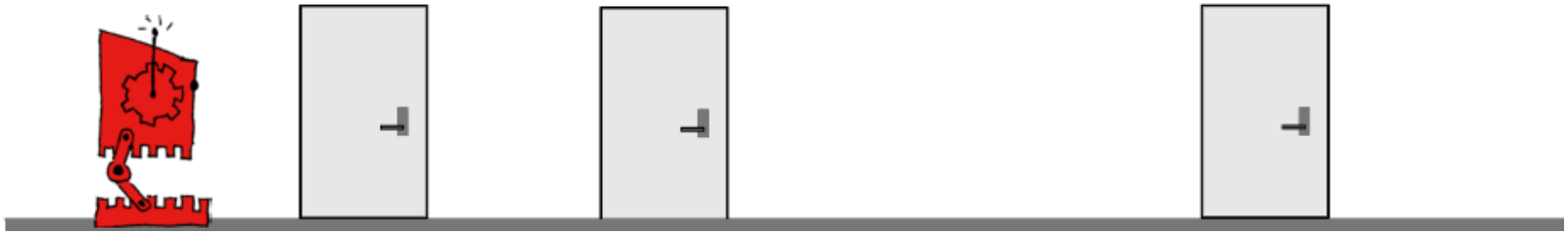
- The agent maintains a **belief** $b(s)$ of being at state s.
- After action $a \in A$ and observation $o \in O$ the belief $b(s)$ can be updated using Bayes' rule:

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

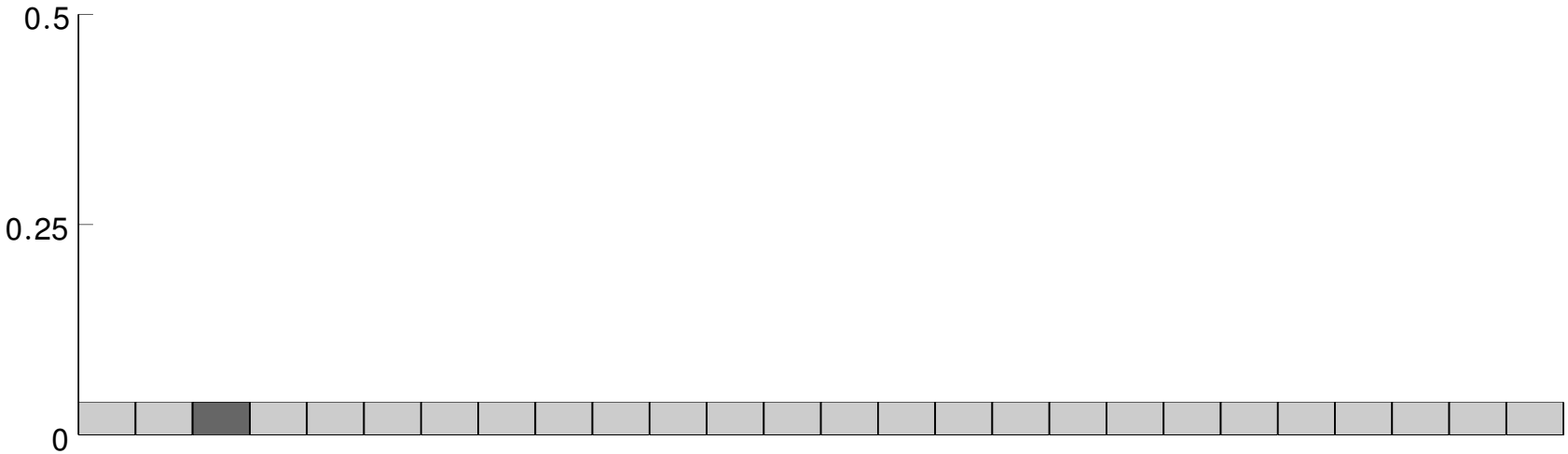
- The belief vector is a **Markov** signal for the planning task.

Belief update example

True situation:



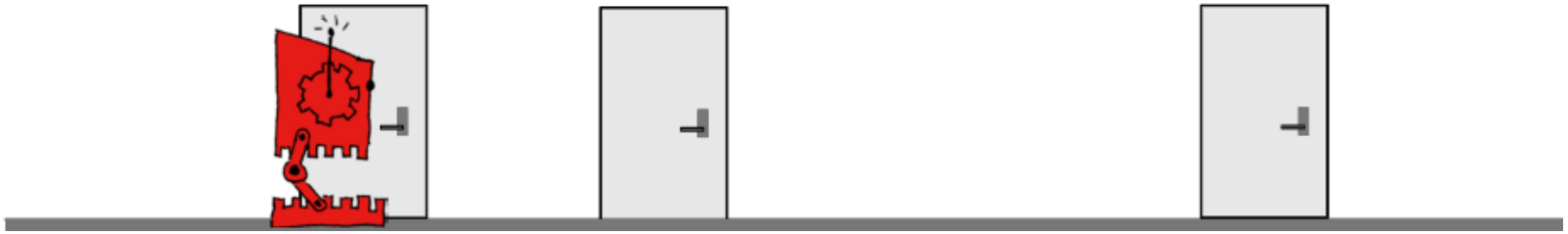
Robot's belief:



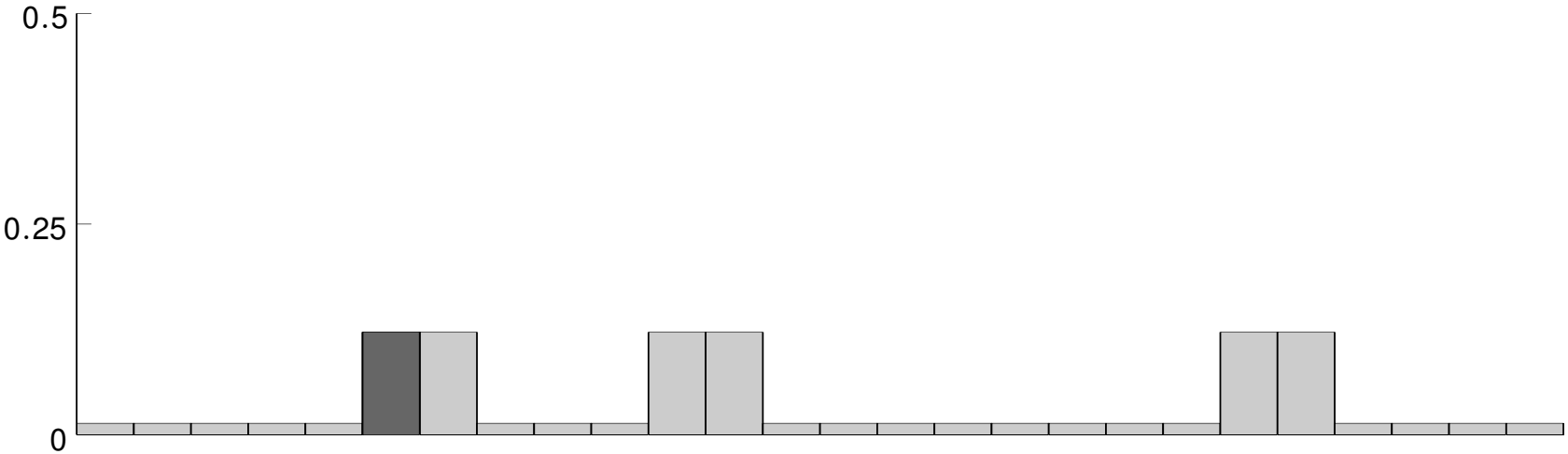
- Observations: *door* or *corridor*, 10% noise.
- Action: moves 3 (20%), 4 (60%), or 5 (20%) states.

Belief update example

True situation:



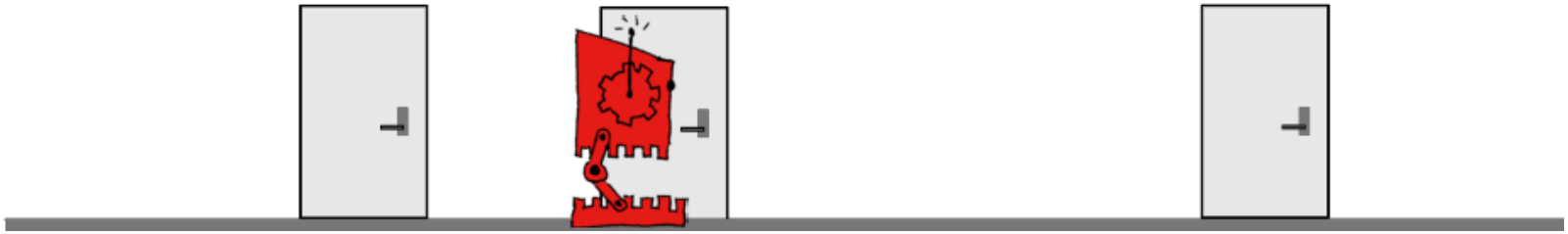
Robot's belief:



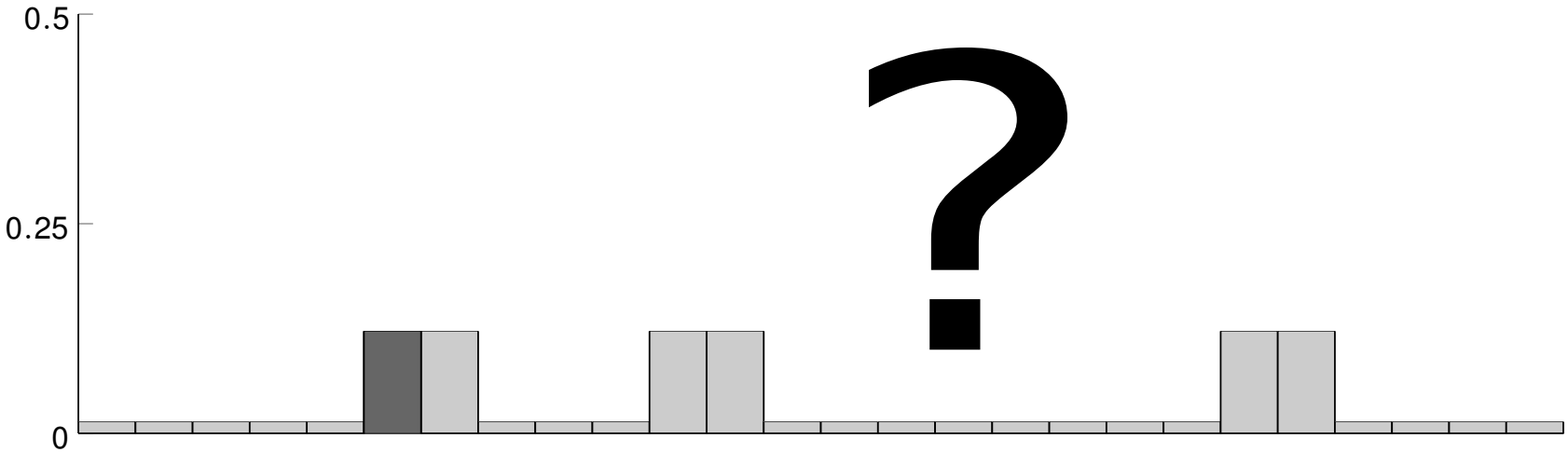
- Observations: **door** or *corridor*, 10% noise.
- Action: moves 3 (20%), 4 (60%), or 5 (20%) states.

Belief update example

True situation:



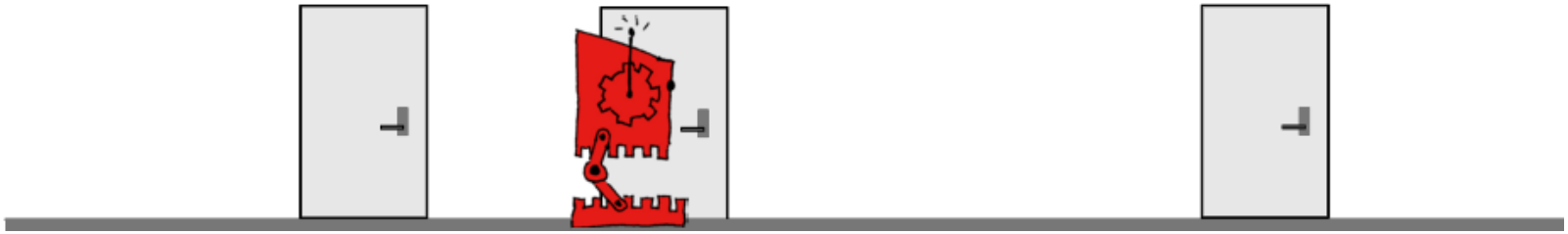
Robot's belief:



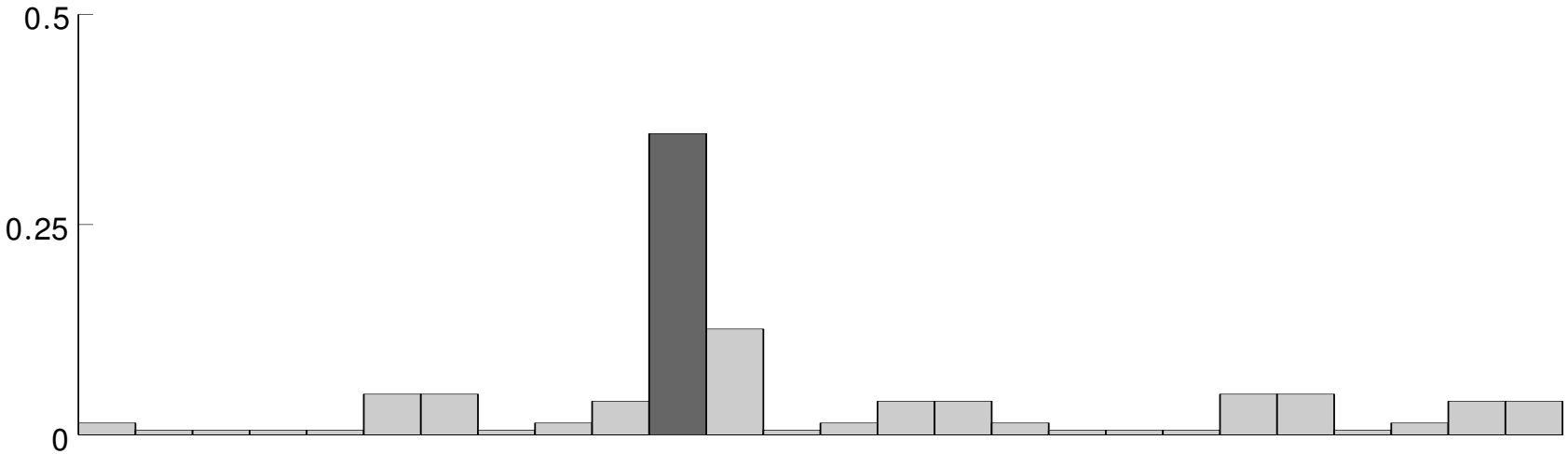
- Observations: **door** or *corridor*, 10% noise.
- Action: moves 3 (20%), 4 (60%), or 5 (20%) states.

Belief update example

True situation:



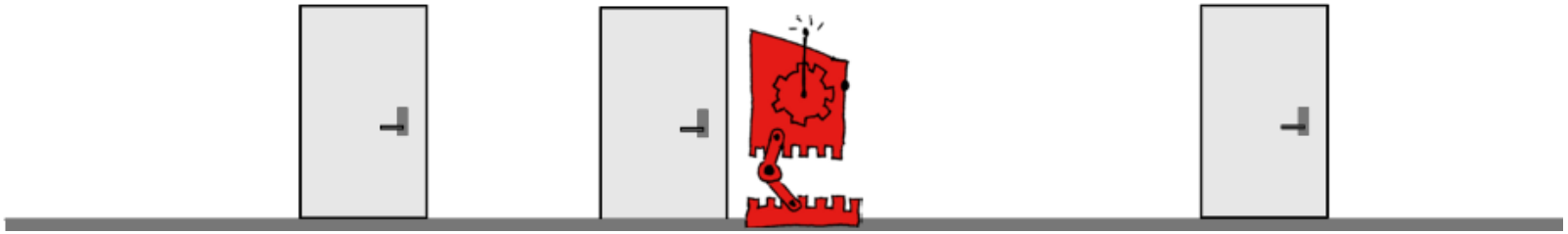
Robot's belief:



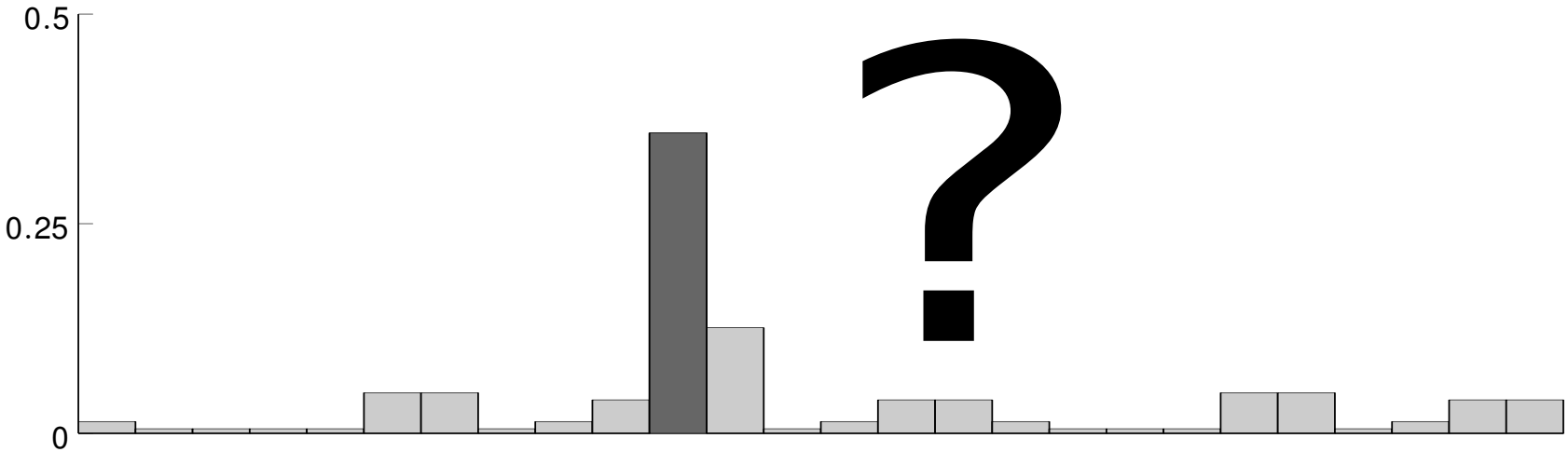
- Observations: **door** or *corridor*, 10% noise.
- Action: moves 3 (20%), 4 (60%), or 5 (20%) states.

Belief update example

True situation:



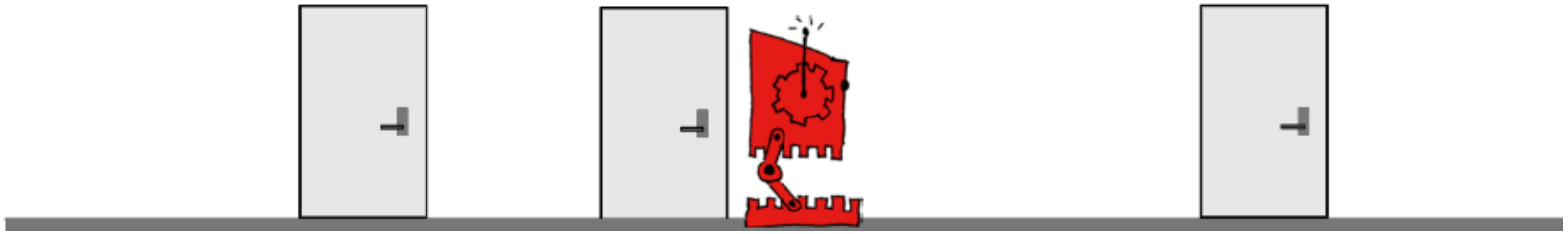
Robot's belief:



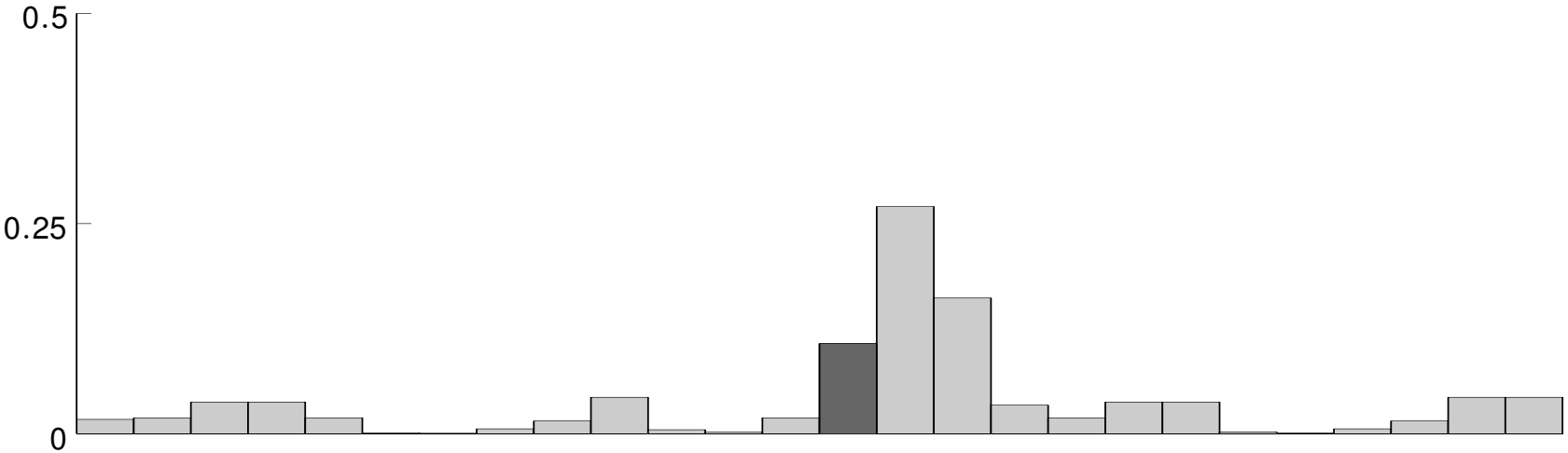
- Observations: *door* or **corridor**, 10% noise.
- Action: moves 3 (20%), 4 (60%), or 5 (20%) states.

Belief update example

True situation:



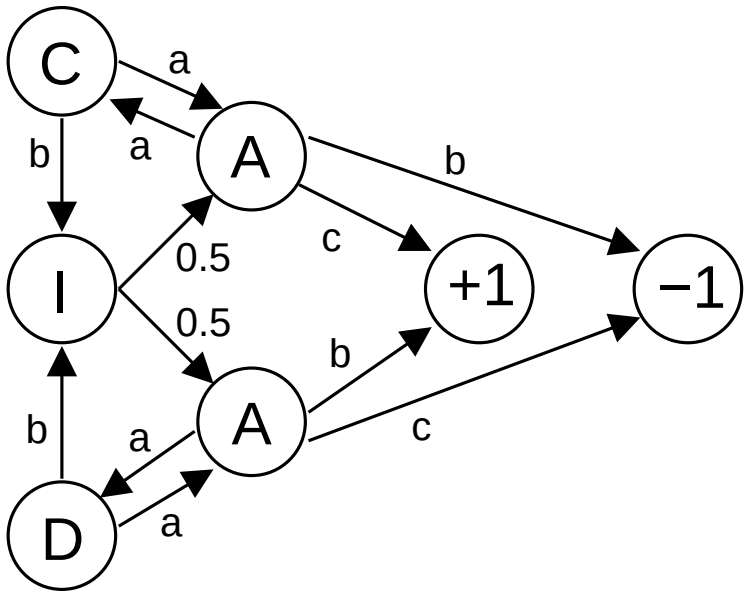
Robot's belief:



- 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 $[0, 1]^{|S|-1}$.
- 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.

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.

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.

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.
- 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).

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).

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).

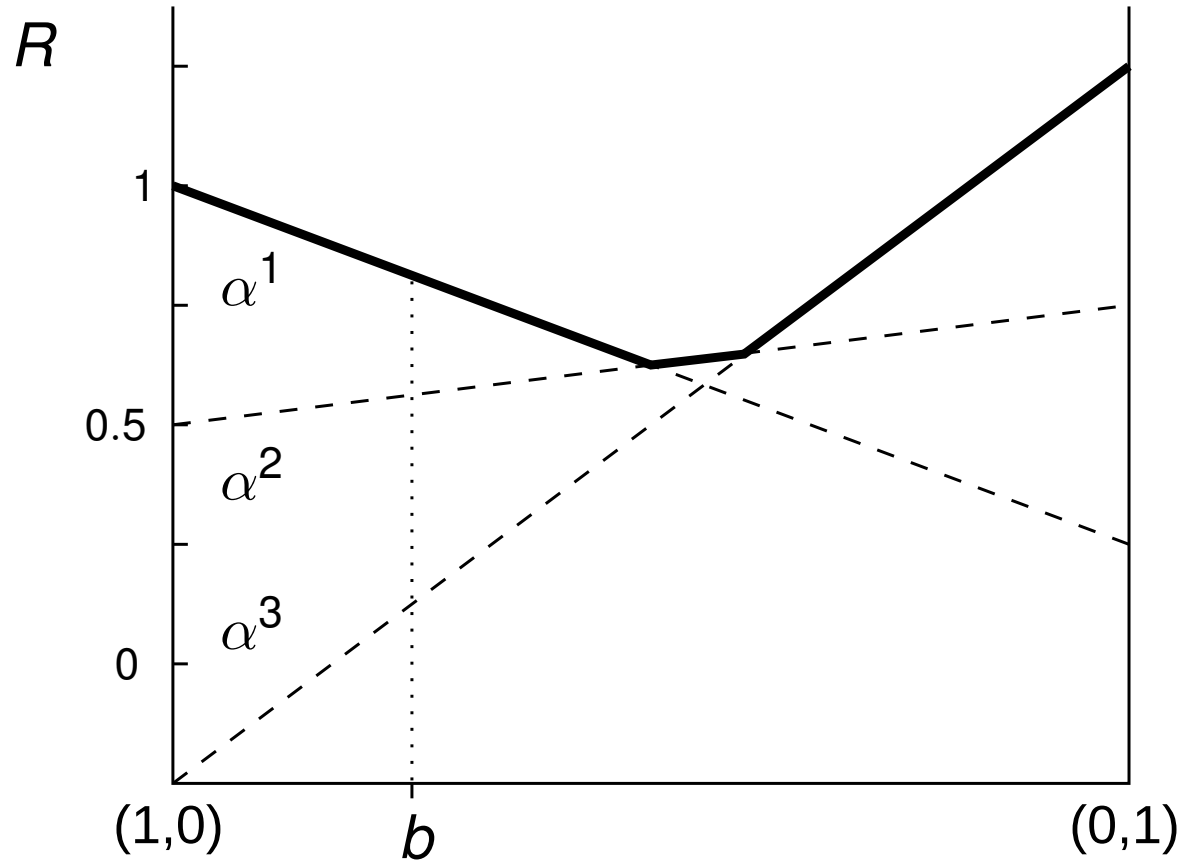
Solving POMDPs

- 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

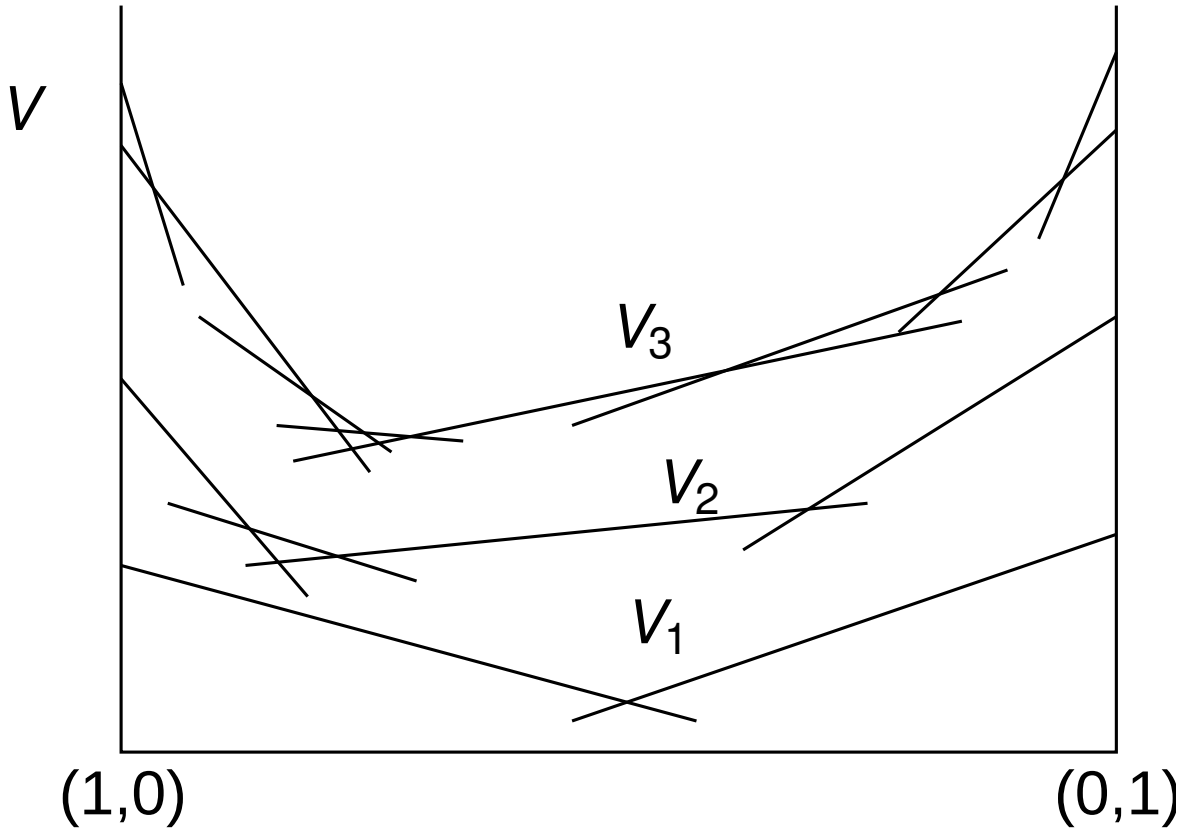
- Like V_0, V_n is as well piecewise linear and convex.
- Rewards $R(b, a) = b \cdot R(s, a)$ are linear functions of b. Note that the value of a point b satisfies:

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

  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).

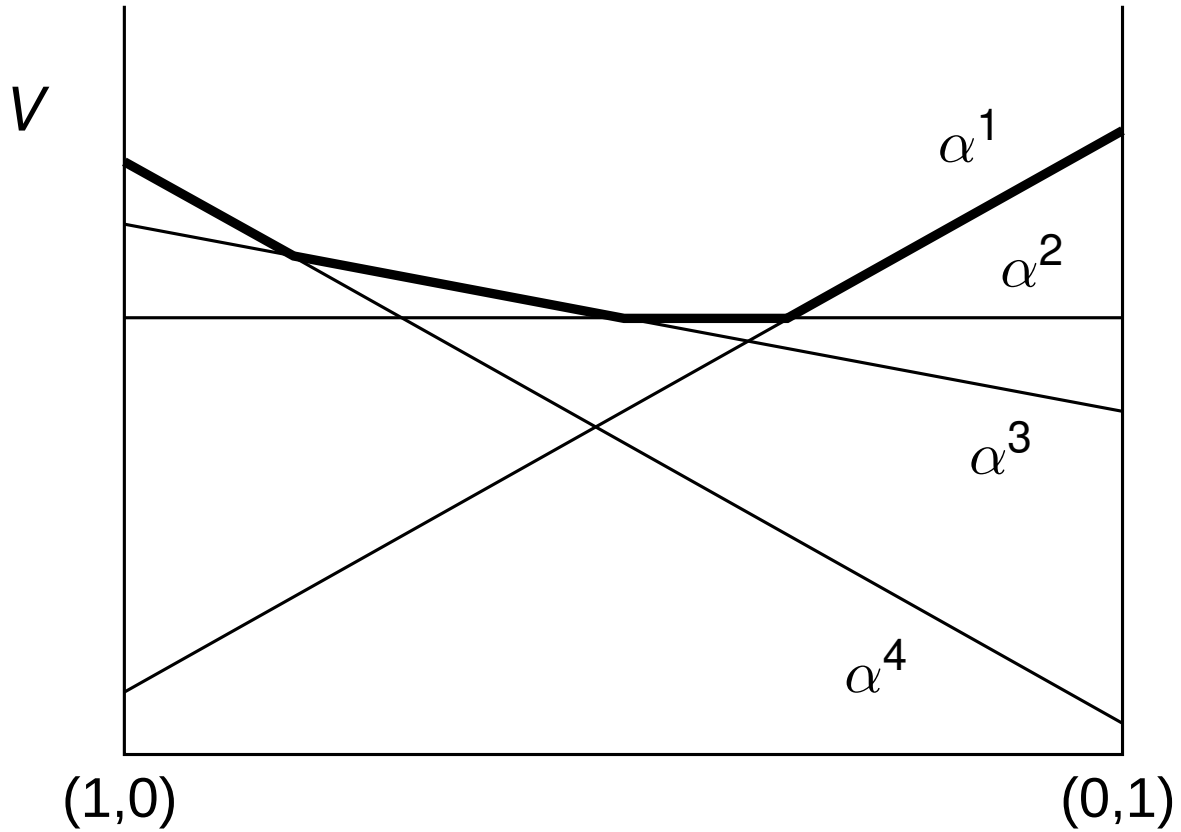
Exact value iteration

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

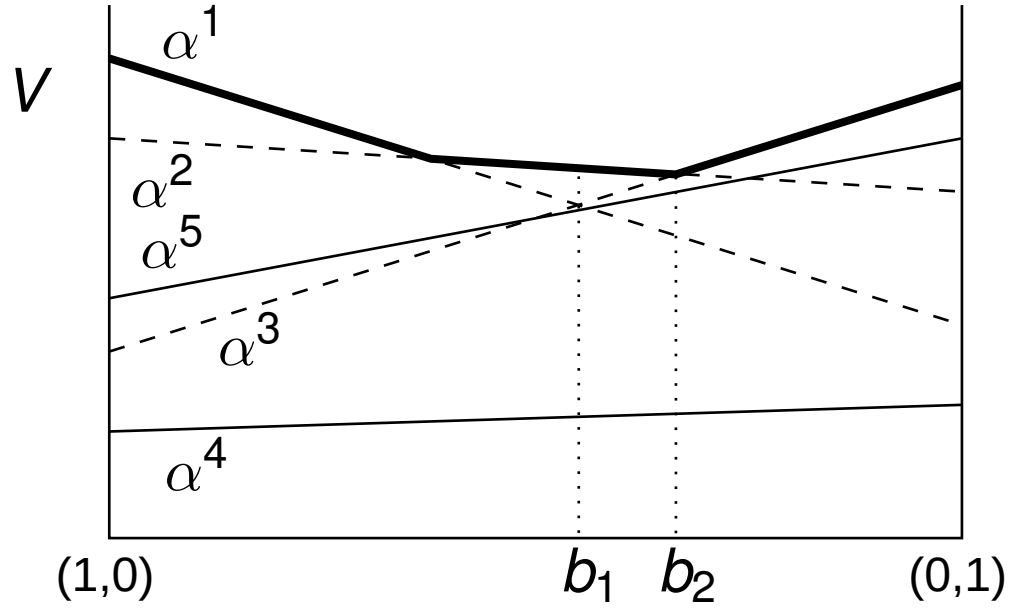


Optimal value functions

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).
- 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 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 $\alpha_{n+1}^b = \texttt{backup}(b)$.

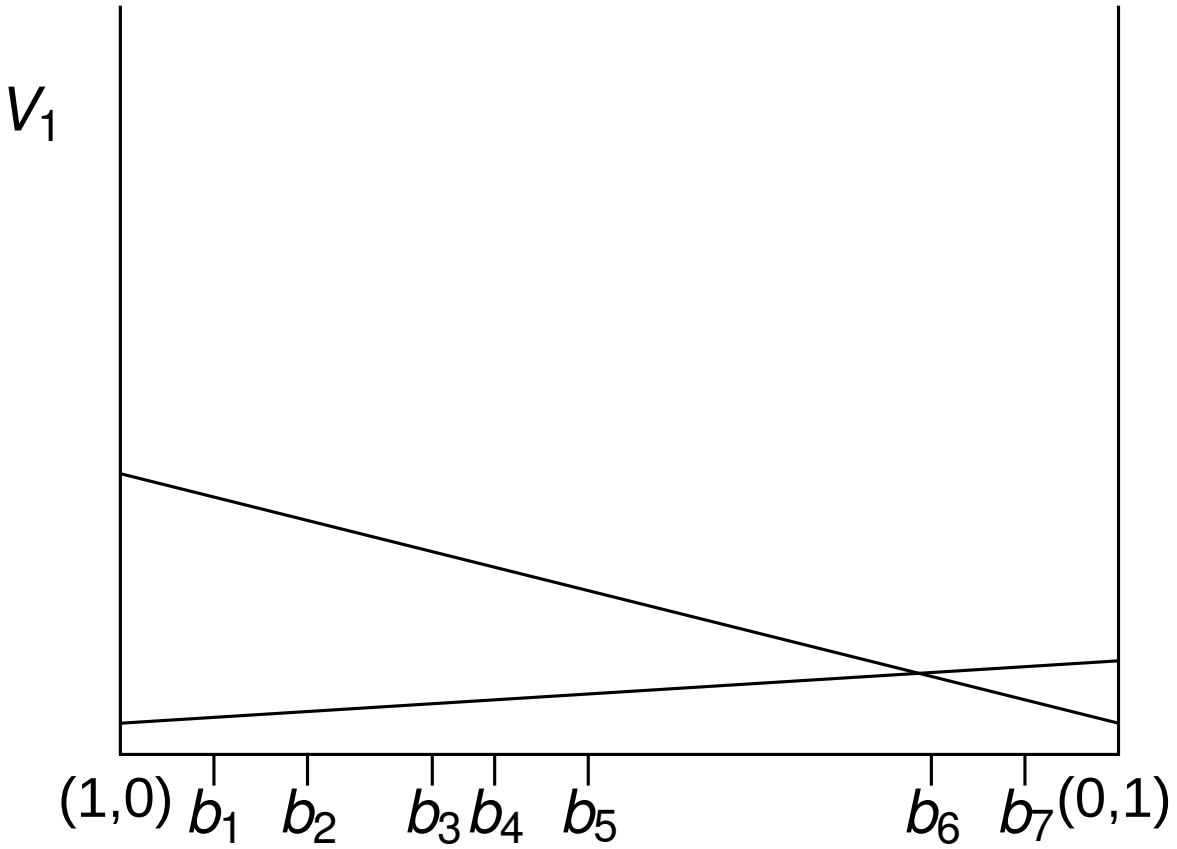
Point-based (approximate) methods

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

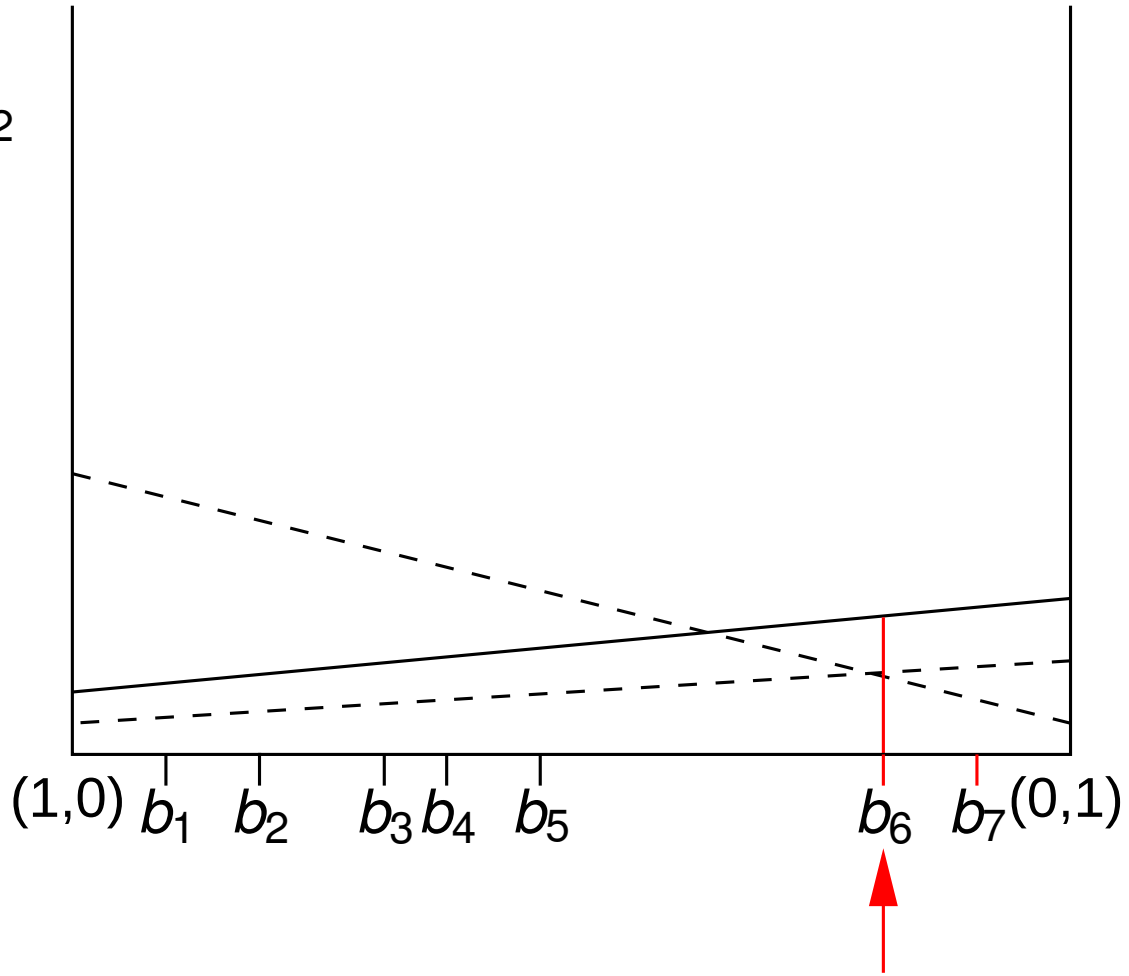
Idea: at every backup stage improve the value of all $b \in B$.



(Spaan and Vlassis, 2005)

PERSEUS: randomized point-based VI

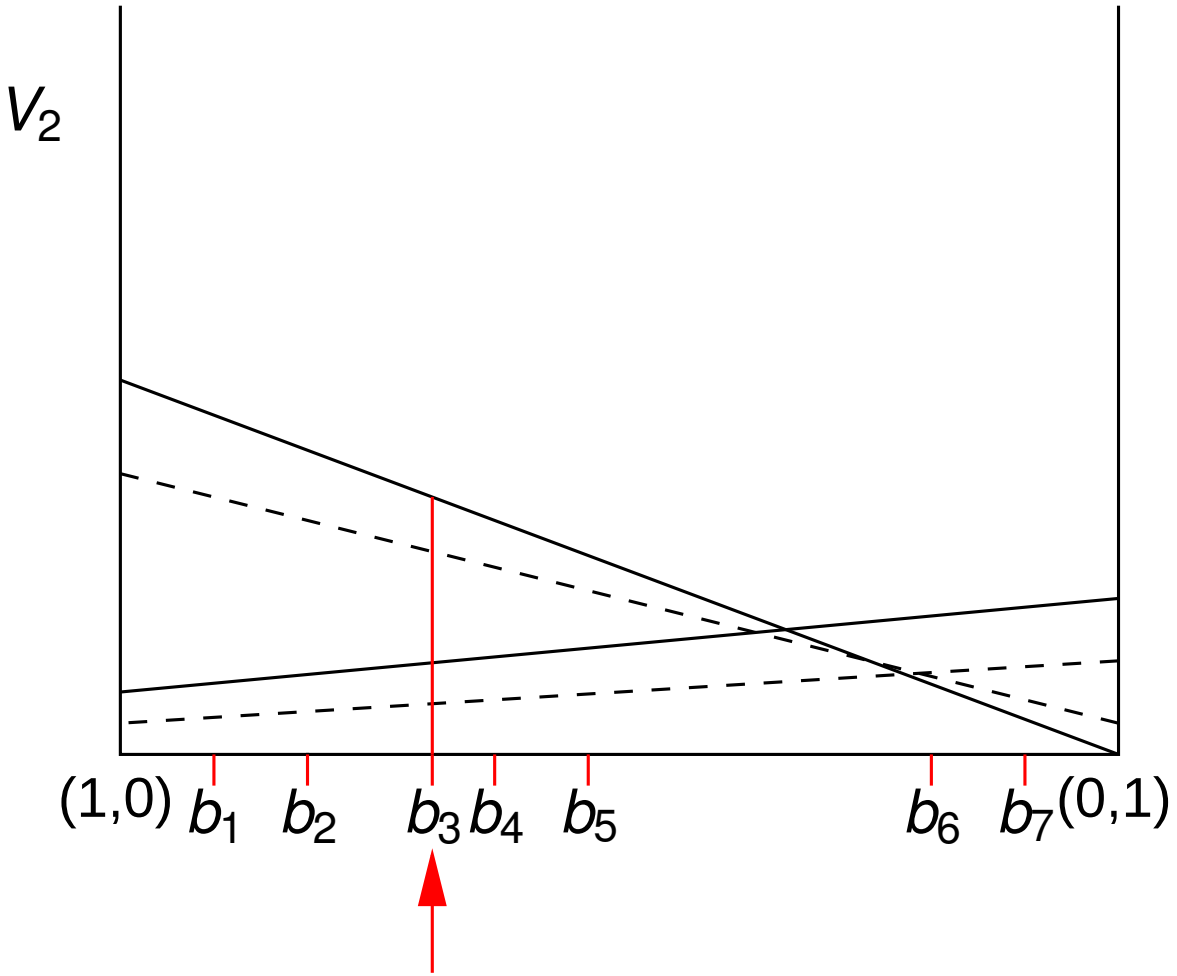
Idea: at every backup stage improve the value of all $b \in B$.



(Spaan and Vlassis, 2005)

PERSEUS: randomized point-based VI

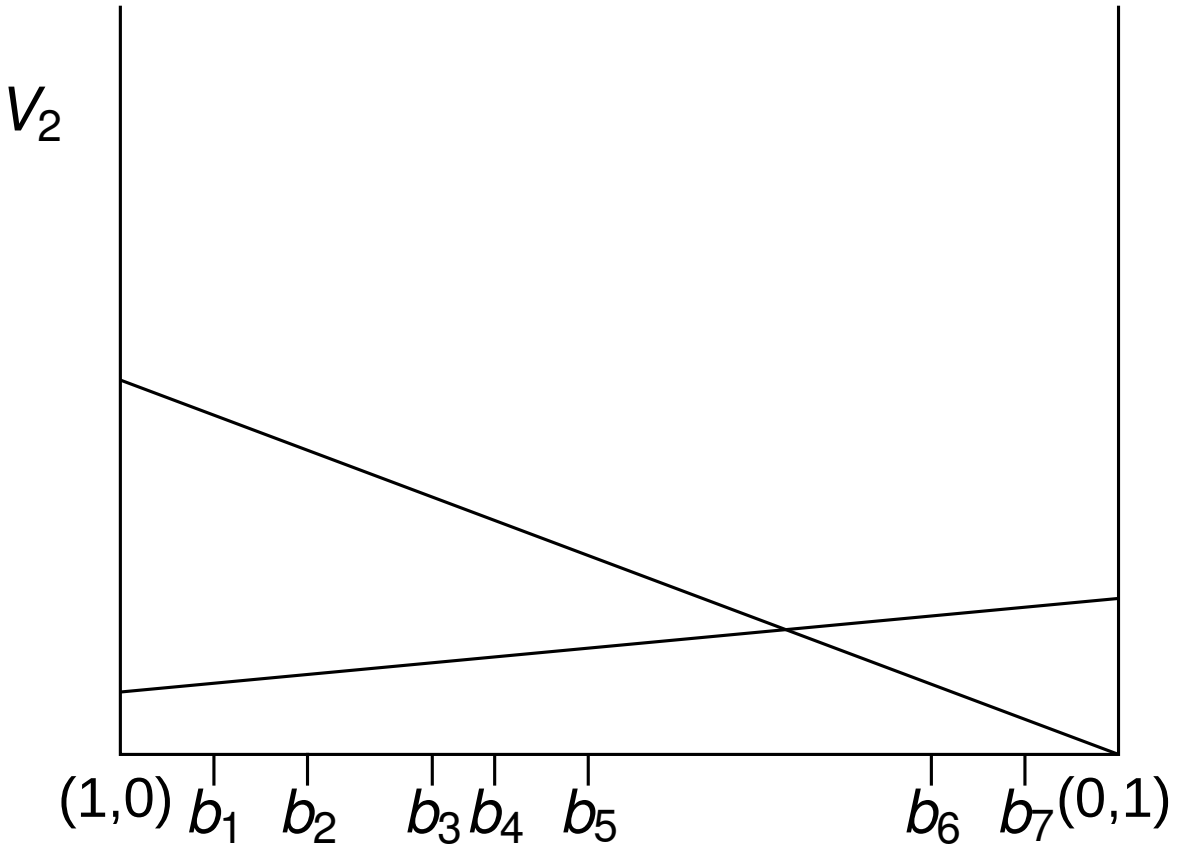
Idea: at every backup stage improve the value of all $b \in B$.



(Spaan and Vlassis, 2005)

PERSEUS: randomized point-based VI

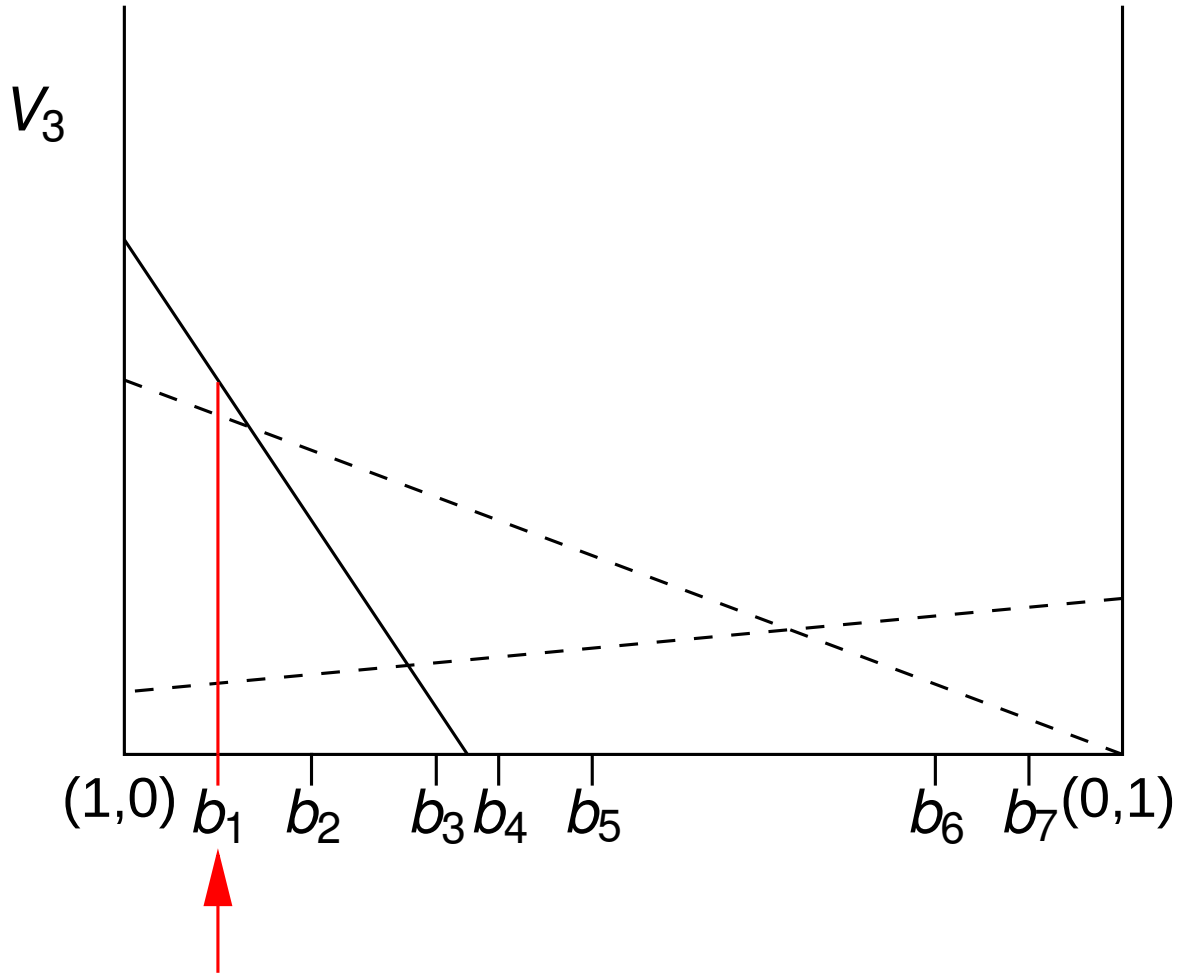
Idea: at every backup stage improve the value of all $b \in B$.



(Spaan and Vlassis, 2005)

PERSEUS: randomized point-based VI

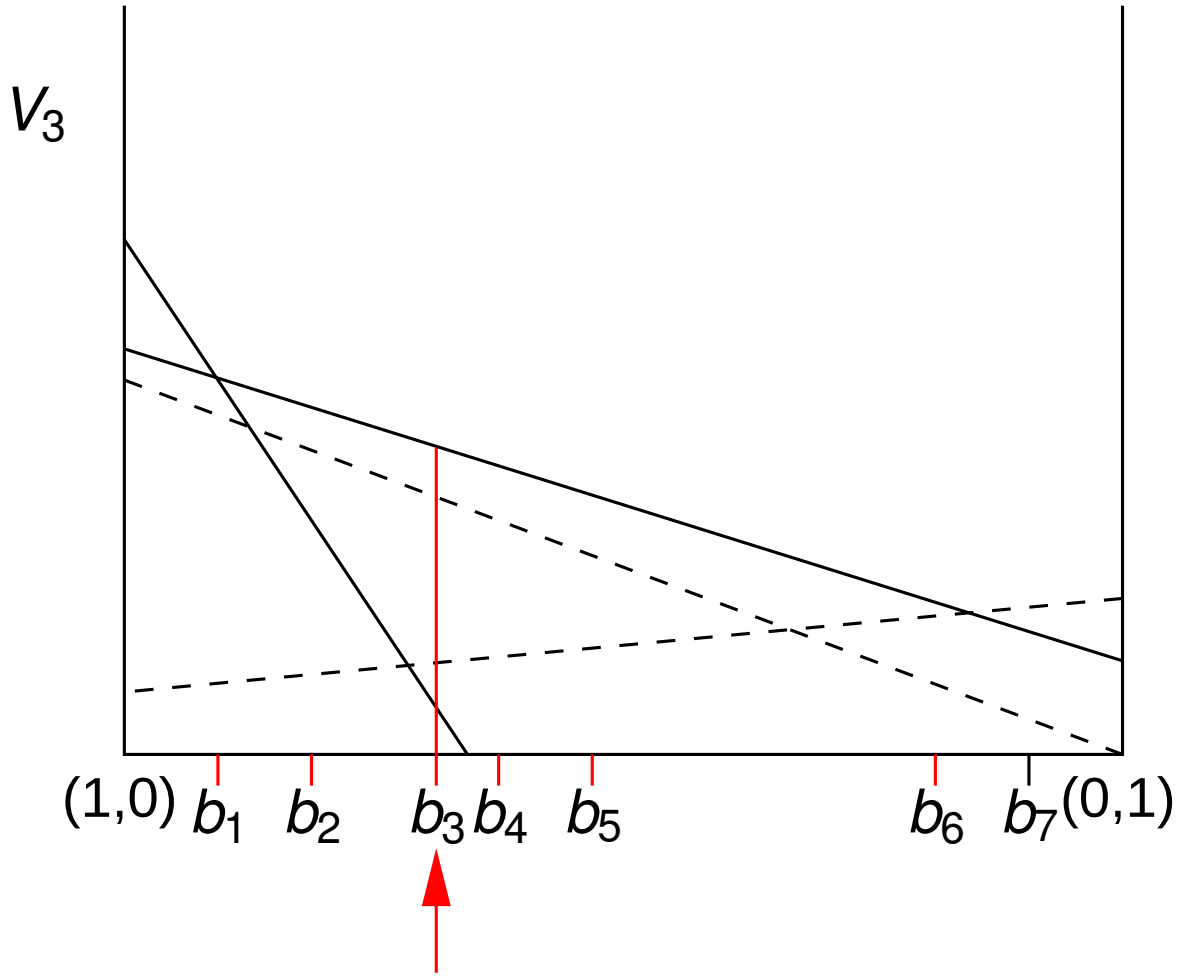
Idea: at every backup stage improve the value of all $b \in B$.



(Spaan and Vlassis, 2005)

PERSEUS: randomized point-based VI

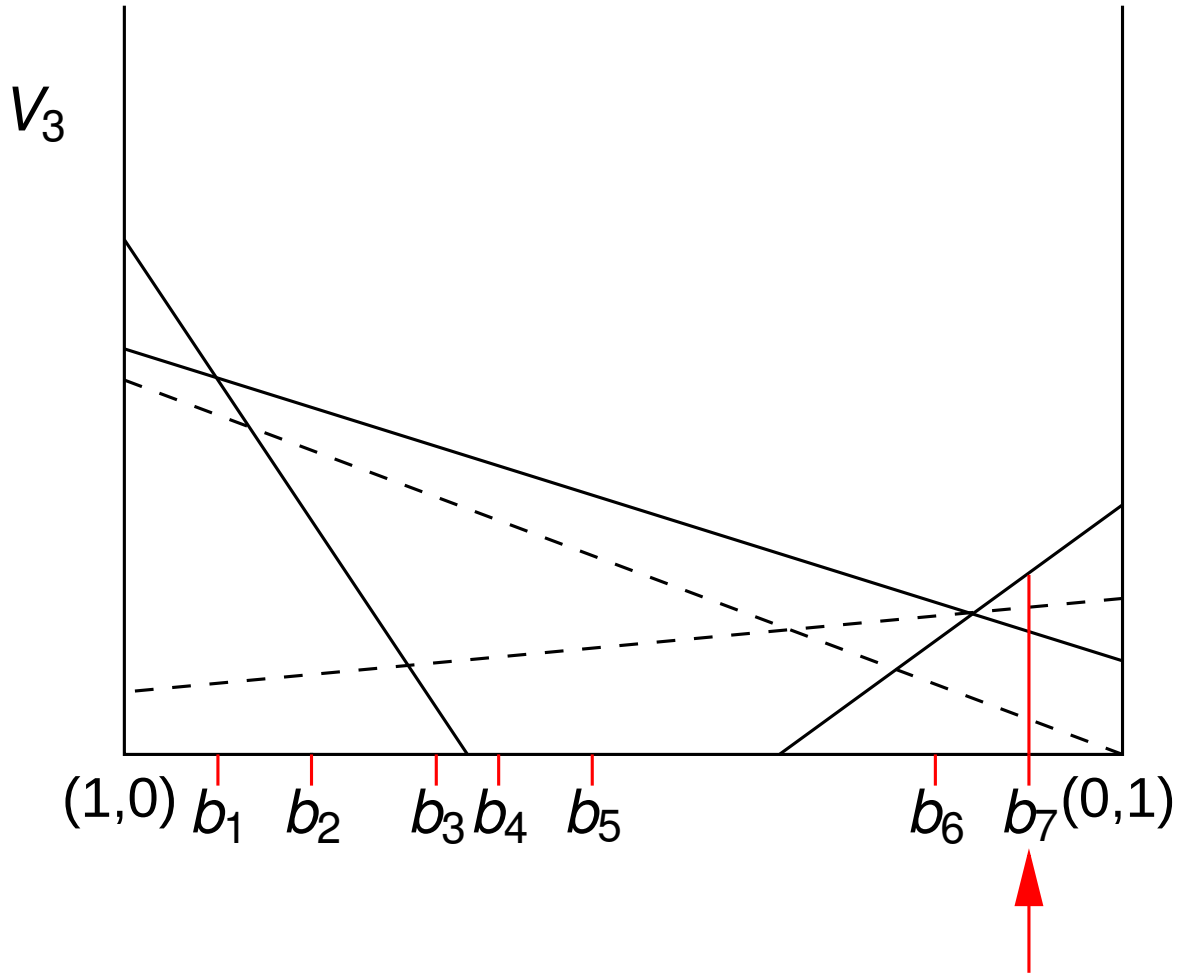
Idea: at every backup stage improve the value of all $b \in B$.



(Spaan and Vlassis, 2005)

PERSEUS: randomized point-based VI

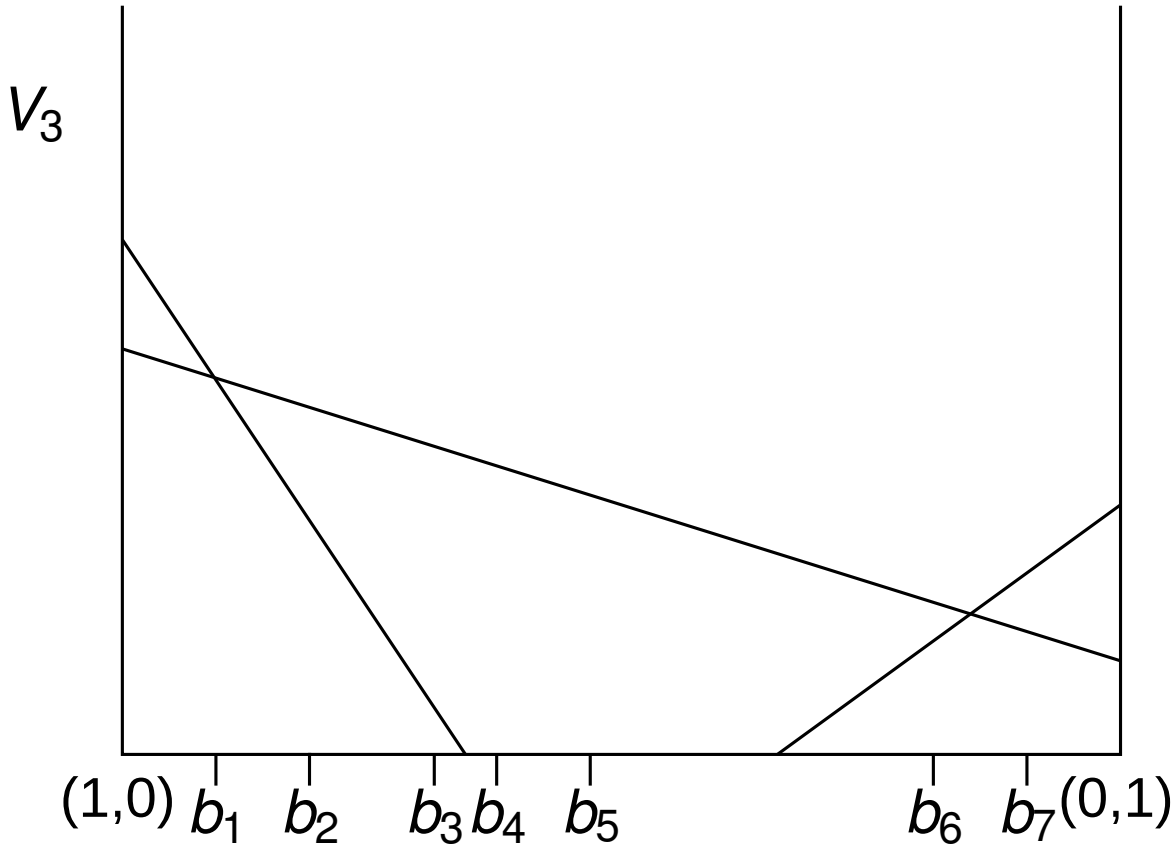
Idea: at every backup stage improve the value of all $b \in B$.



(Spaan and Vlassis, 2005)

PERSEUS: randomized point-based VI

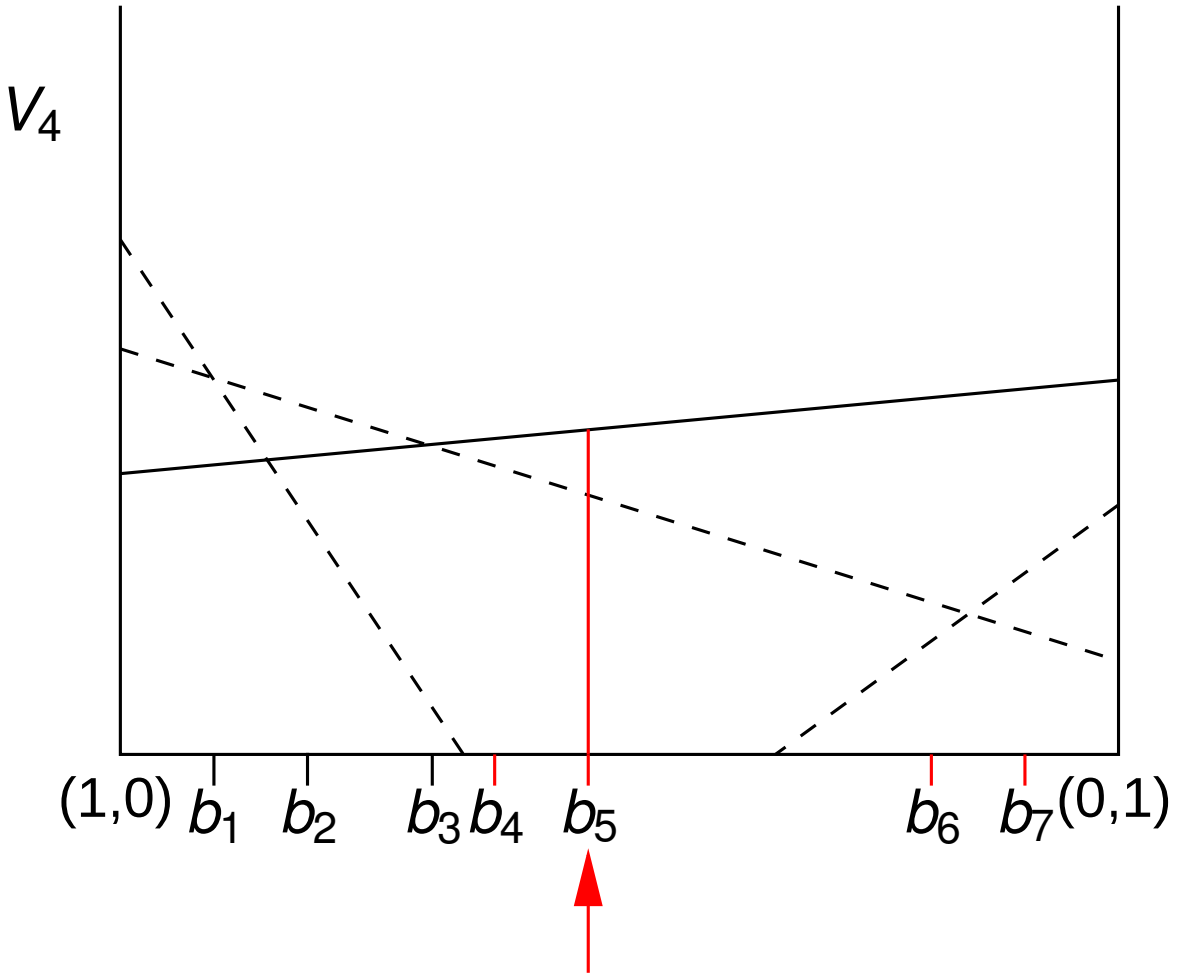
Idea: at every backup stage improve the value of all $b \in B$.



(Spaan and Vlassis, 2005)

PERSEUS: randomized point-based VI

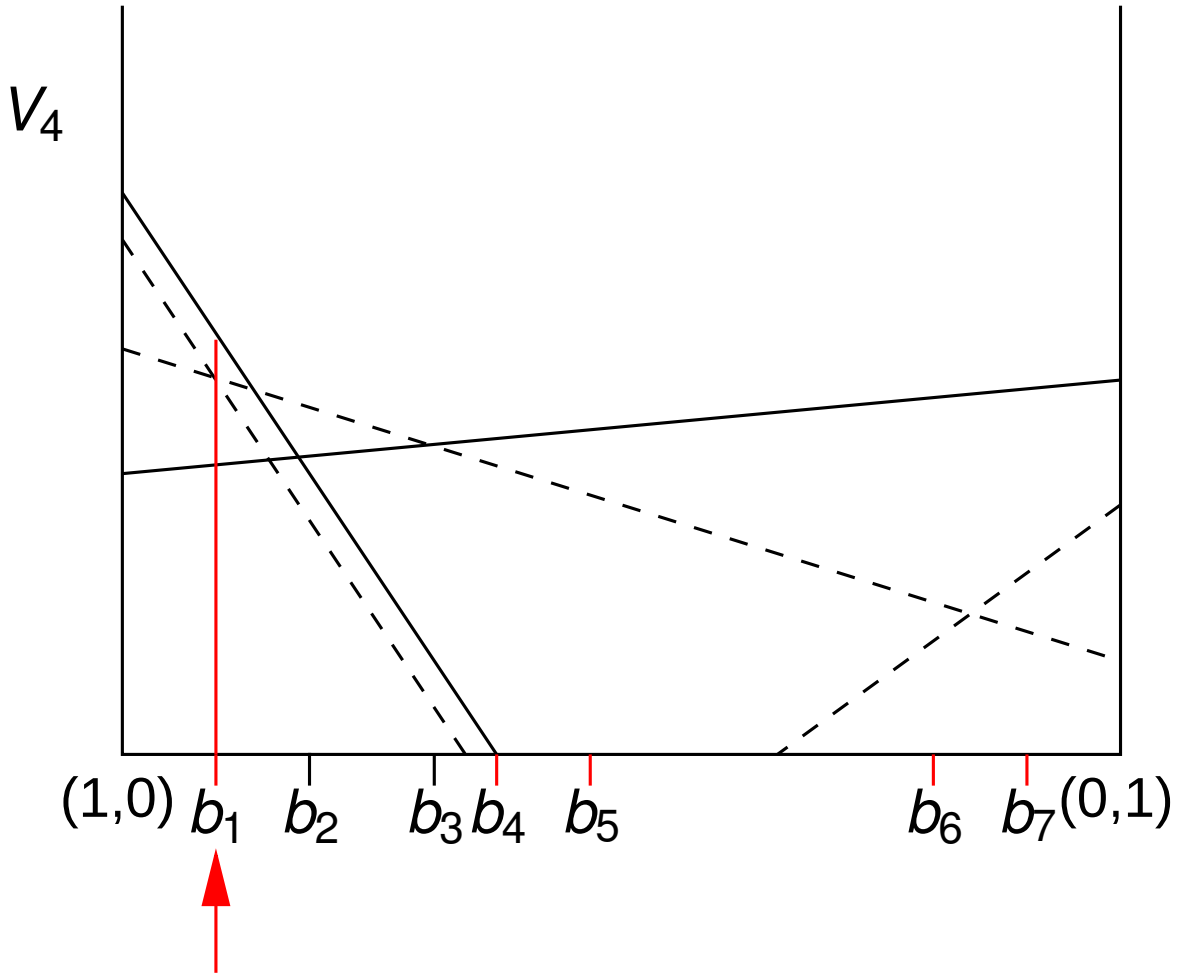
Idea: at every backup stage improve the value of all $b \in B$.



(Spaan and Vlassis, 2005)

PERSEUS: randomized point-based VI

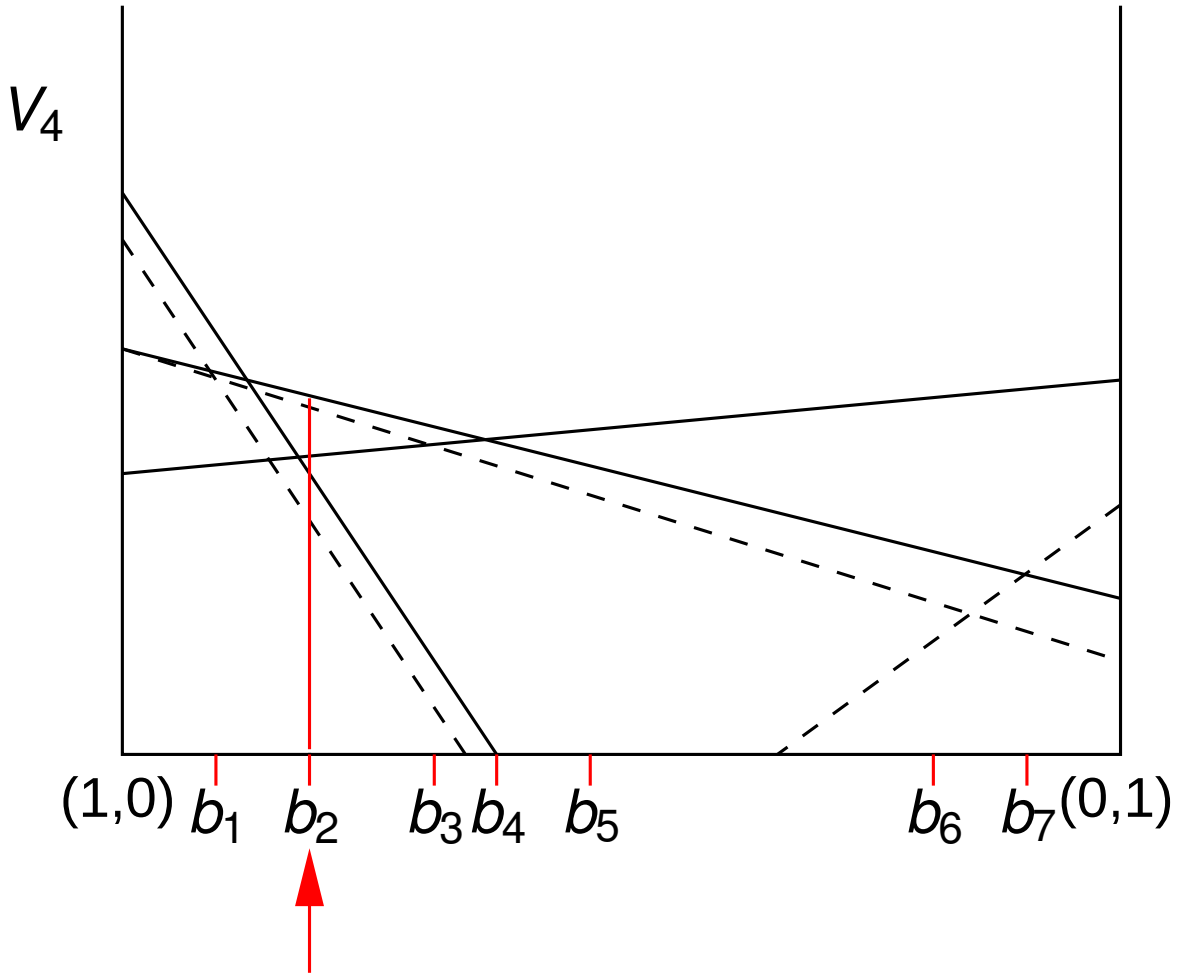
Idea: at every backup stage improve the value of all $b \in B$.



(Spaan and Vlassis, 2005)

PERSEUS: randomized point-based VI

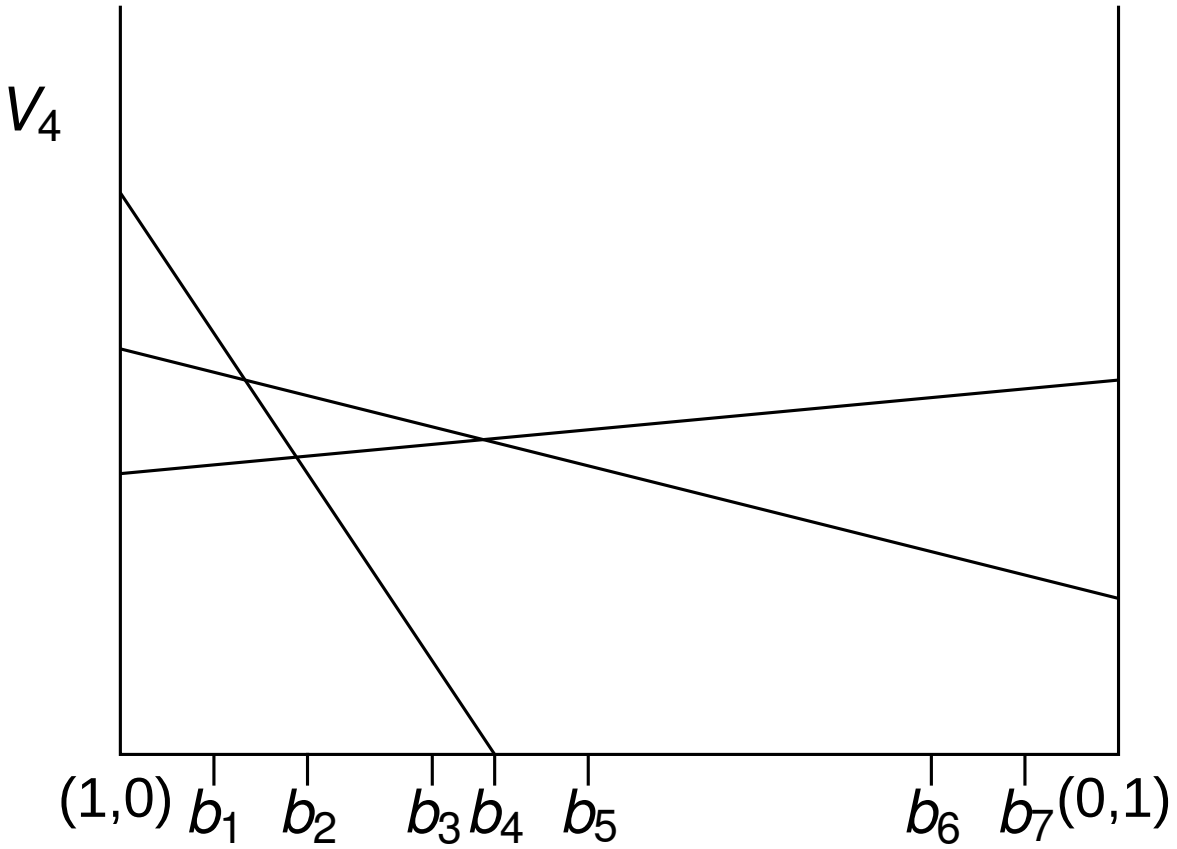
Idea: at every backup stage improve the value of all $b \in B$.



(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)

High dimensional sensor readings

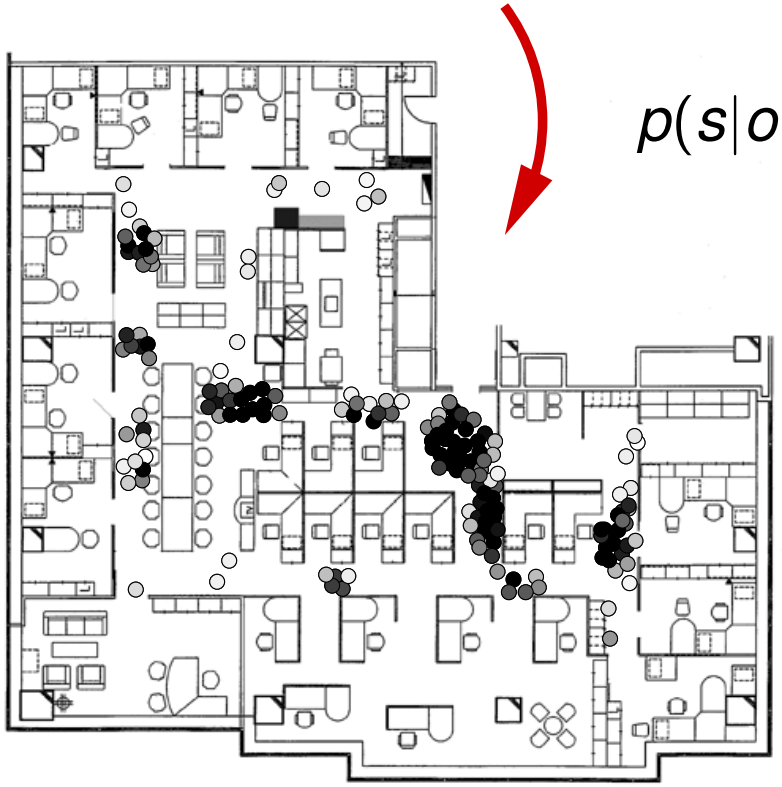
Omnidirectional camera images.

Example images $\Rightarrow$

Dimension reduction:

- Collect a database of images and record their location.
- Apply Principal Component Analysis on the image data.
- Project each image to the first 3 eigenvectors, resulting in a 3D feature vector for each image.

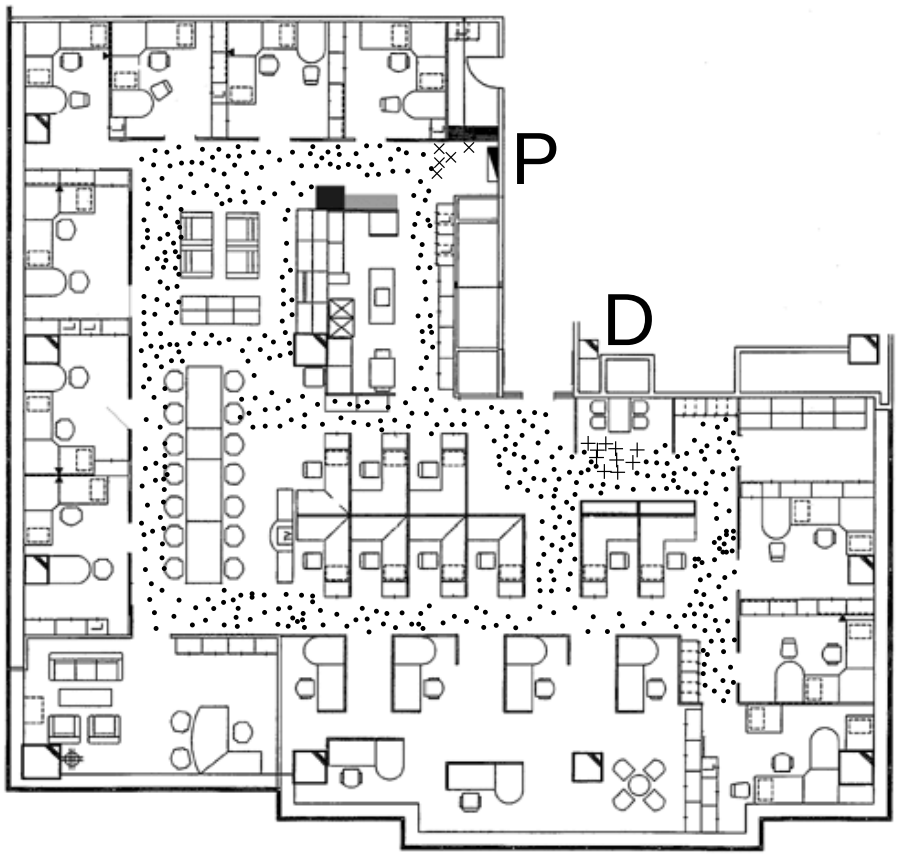
Observation model



$p(s|o)$

- We cluster the feature vectors into 10 prototype observations.
- We compute a discrete observation model $p(o|s, a)$ by a histogram operation.

States, actions and rewards



- State: $s = (x, j)$ with x the robot's location and j the mail bit.
- Grid X into 500 locations.
- Actions: $\{\uparrow, \rightarrow, \downarrow, \leftarrow,$ *pickup*, *deliver*$\}$.
- Positive reward: only upon successful mail delivery.

(Spaan and Vlassis, 2004)

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 (MDPs, POMDPs, Dec-POMDPs):
  - Marco Wiering and Martijn van Otterlo, editors, “Reinforcement Learning: State of the Art”, Springer, 2012.

References I

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. 1997.

A. R. Cassandra, L. P. Kaelbling, and M. L. Littman. Acting optimally in partially observable stochastic domains. 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.

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.

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.

TU Delft

References II

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

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.

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.

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.

TU Delft

References III

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.

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. A point-based POMDP algorithm for robot planning. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 2399–2404, New Orleans, Louisiana, 2004.

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

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

Y. Virin, G. Shani, S. E. Shimony, and R. Brafman. Scaling up: Solving POMDPs through value based clustering. 2007.

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.