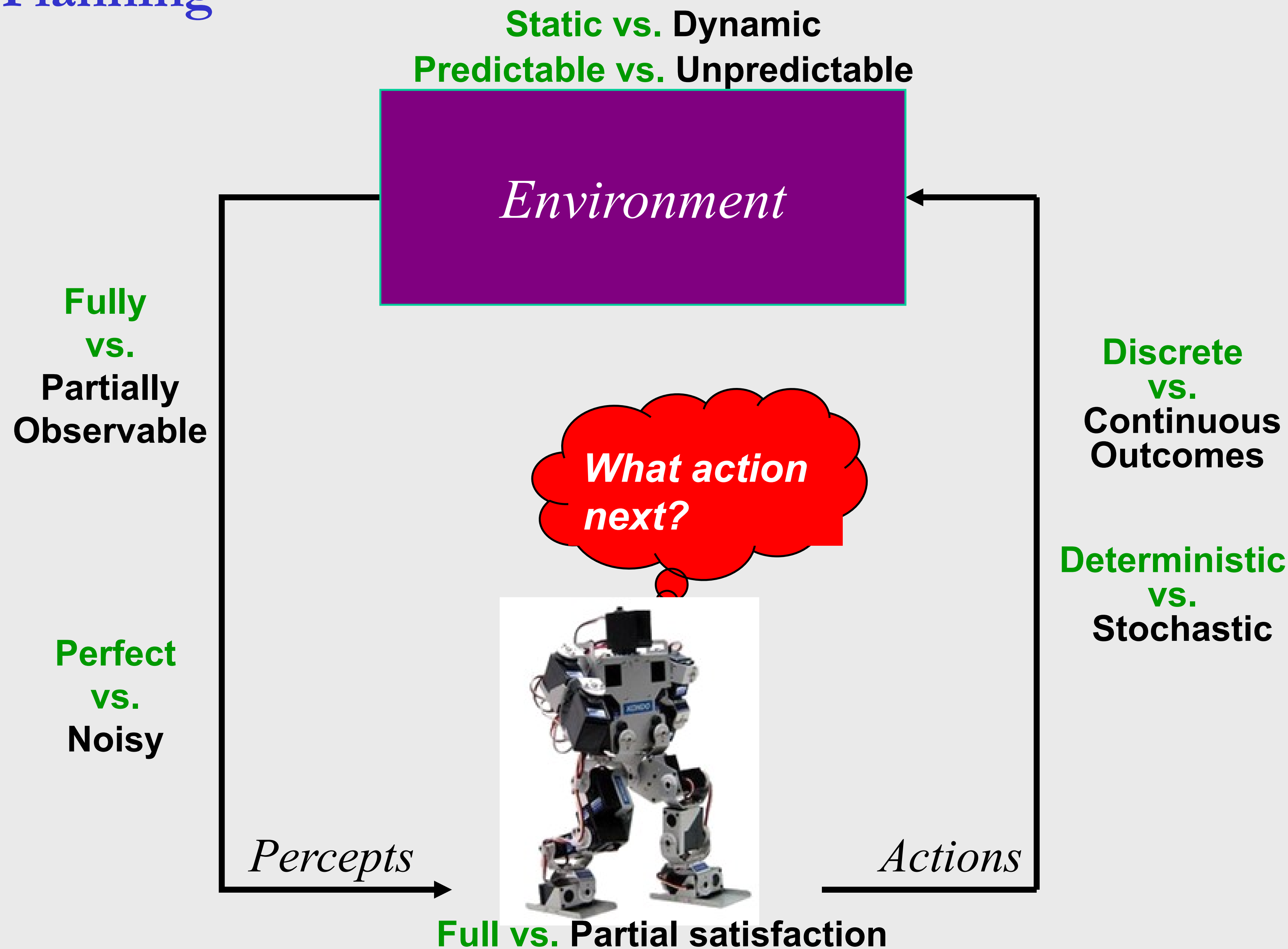
CSE-571
AI-based Mobile Robotics

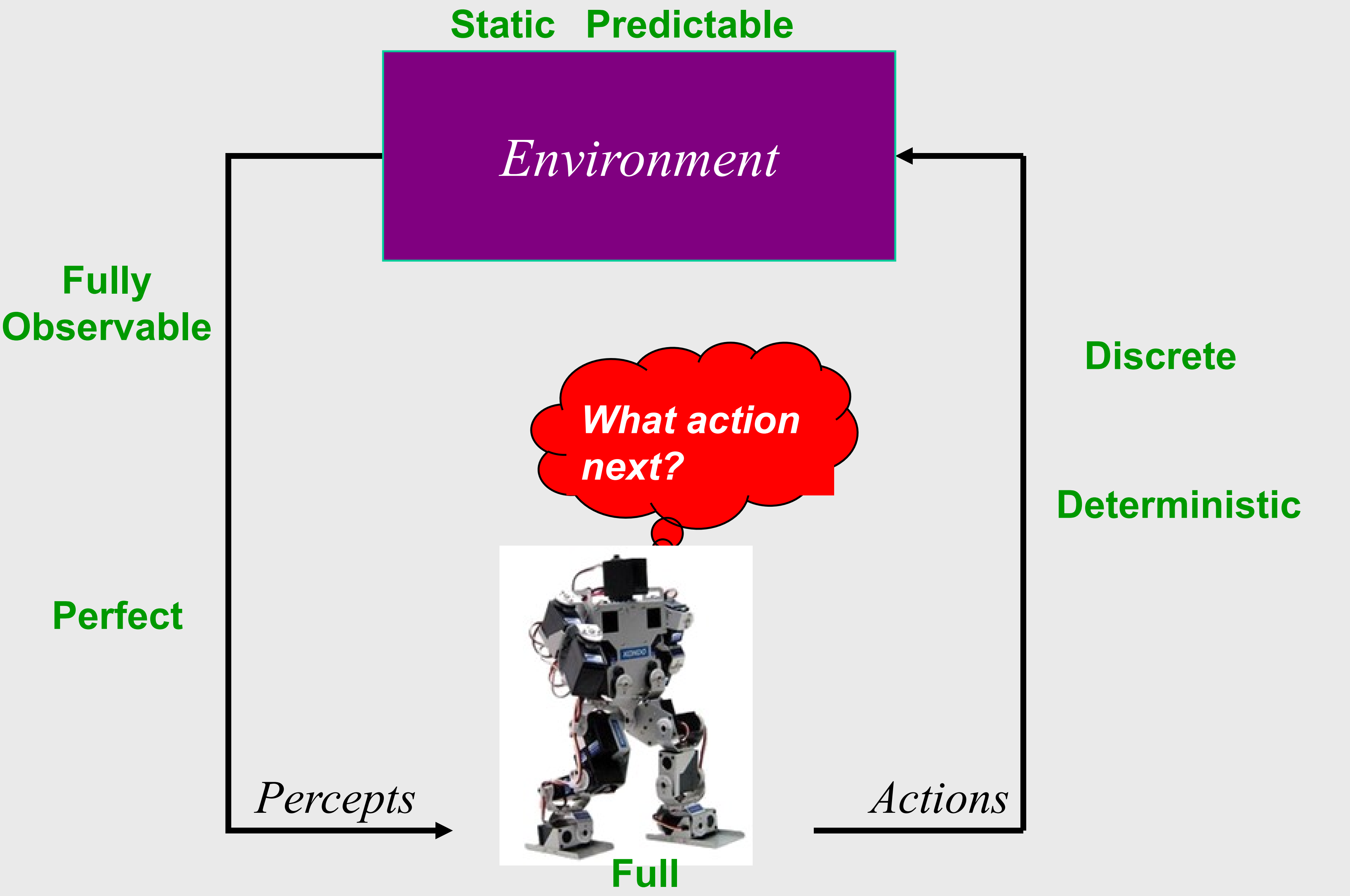
Planning and Control:

Markov Decision Processes

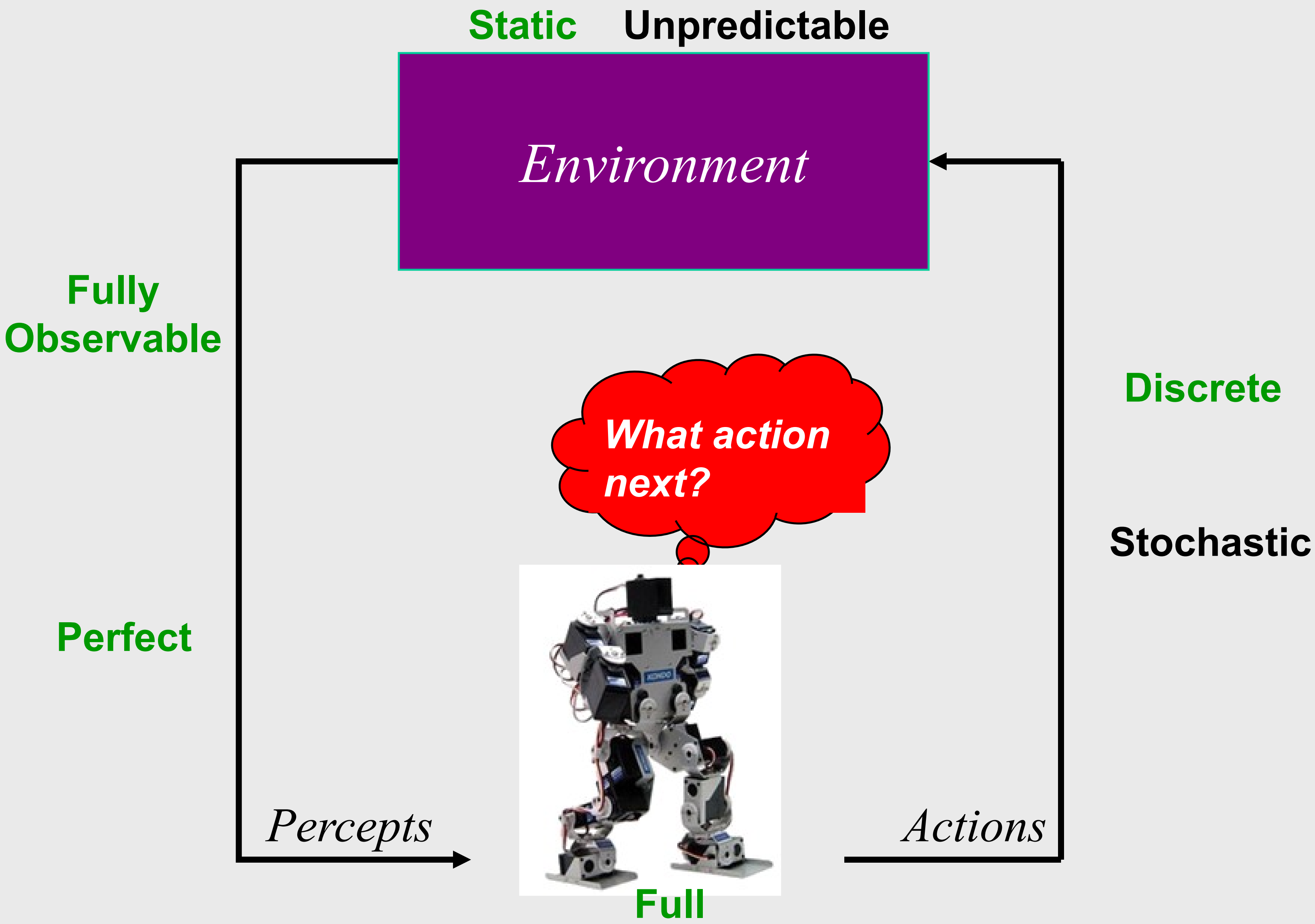
Planning


Static vs. Dynamic
Predictable vs. Unpredictable
Environment
Fully vs. Partially Observable
Perfect vs. Noisy
What action next?
Discrete vs. Continuous Outcomes
Deterministic vs. Stochastic
Percepts
Actions
Full vs. Partial satisfaction

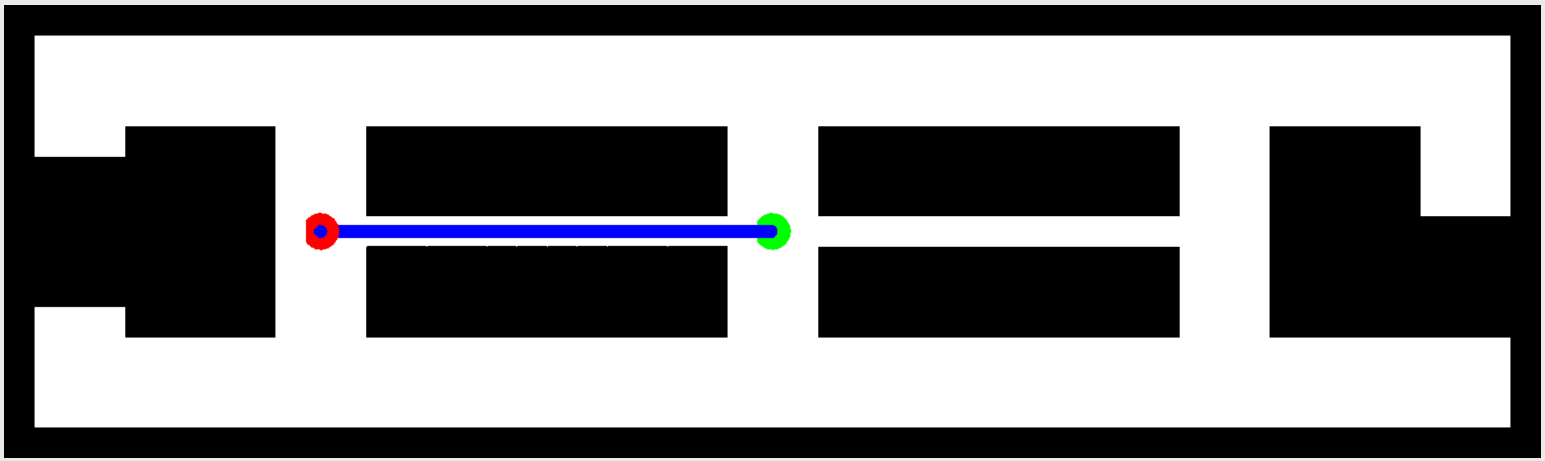
Classical Planning



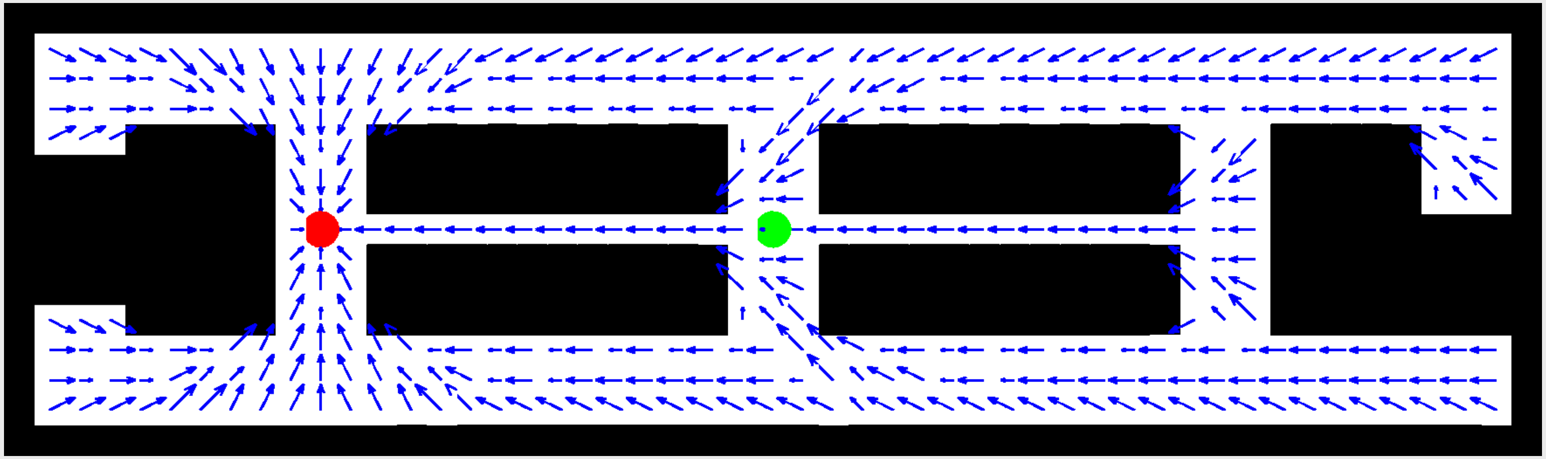
Stochastic Planning

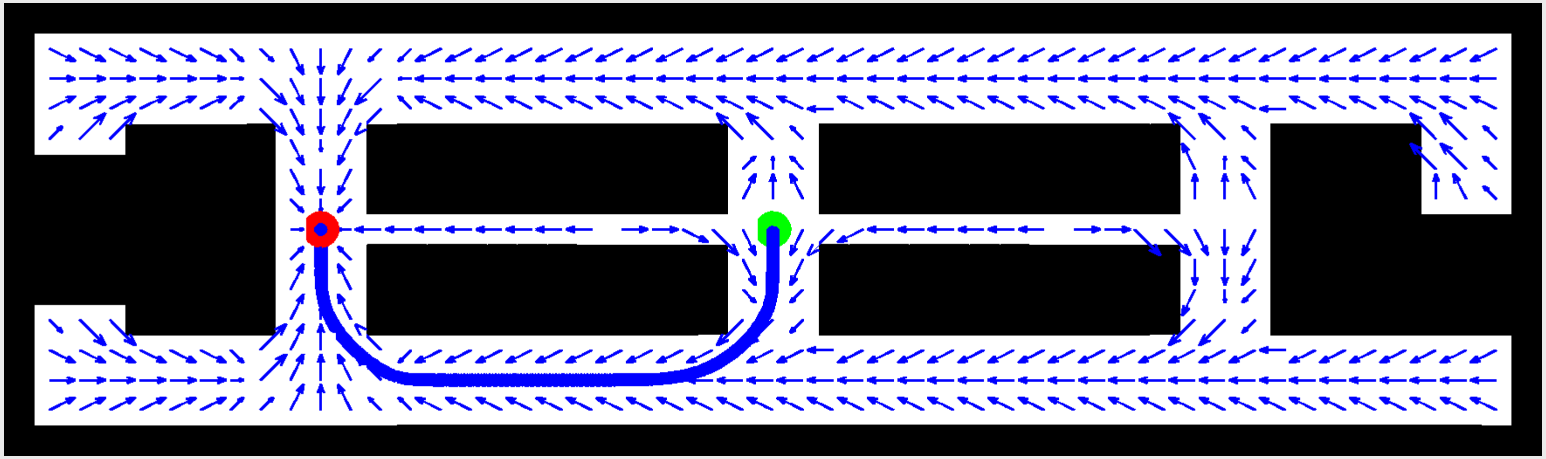


Deterministic, fully observable

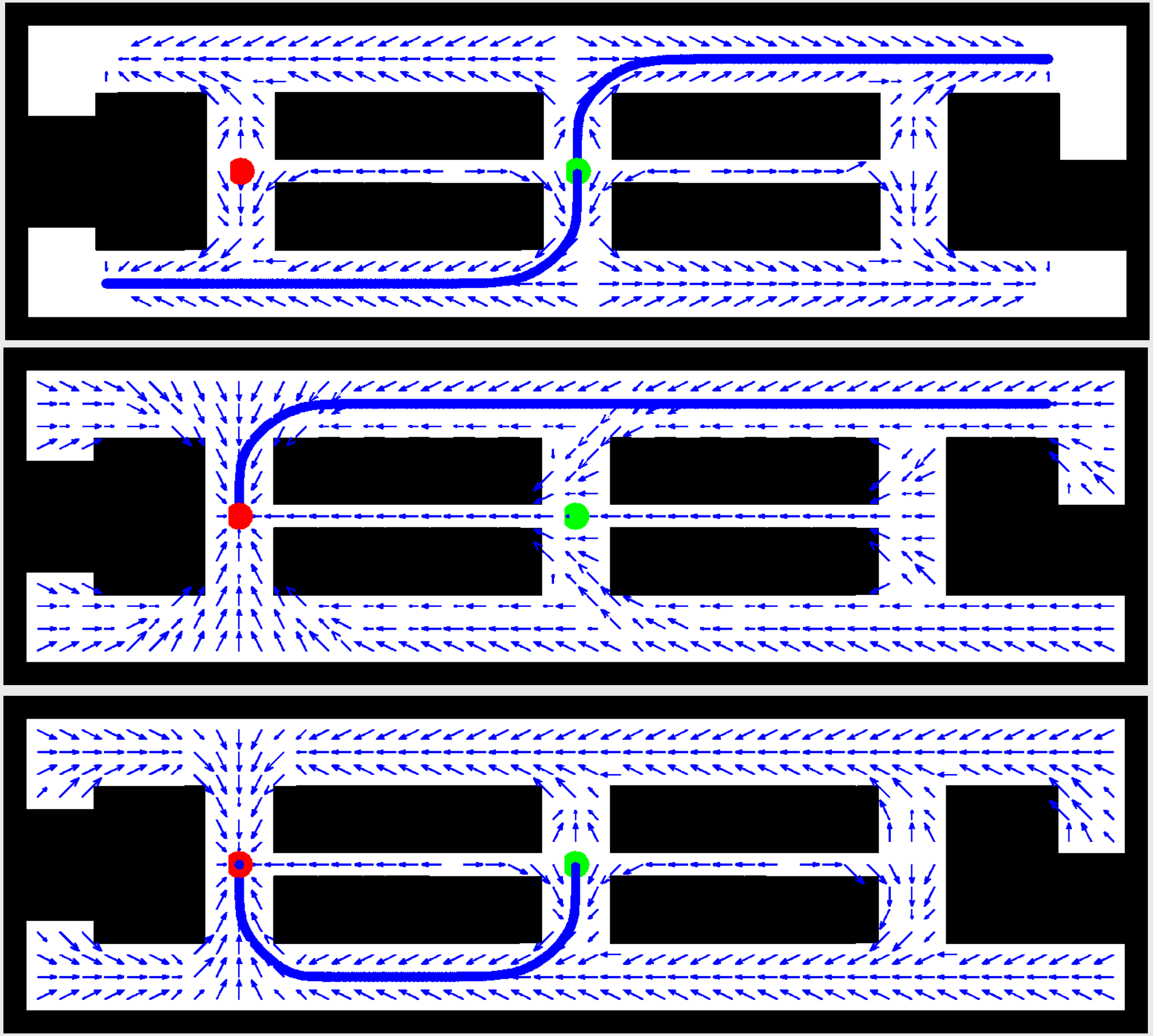


Stochastic, Fully Observable





Stochastic, Partially Observable



Markov Decision Process (MDP)

- $\mathcal{S}$: A set of states
- $\mathcal{A}$: A set of actions
- $\mathcal{P}r(s'|s,a)$: transition model
- $\mathcal{C}(s,a,s')$: cost model
- $\mathcal{G}$: set of goals
- s_0: start state
- γ: discount factor
- $\mathcal{R}(s,a,s')$: reward model

Role of Discount Factor (γ)

- Keep the total reward/total cost finite
  - useful for infinite horizon problems
  - sometimes indefinite horizon: if there are deadends
- Intuition (economics):
  - Money today is worth more than money tomorrow.

- Total reward: $r_1 + \gamma r_2 + \gamma^2 r_3 + \ldots$
- Total cost: $c_1 + \gamma c_2 + \gamma^2 c_3 + \ldots$

Objective of a Fully Observable MDP

- Find a policy $\pi: \mathcal{S} \rightarrow \mathcal{A}$

- which optimises
  - minimises $\begin{bmatrix} \text{discounted} \\ \text{or} \\ \text{undiscount.} \end{bmatrix}$ expected cost to reach a goal
  - maximises expected reward
  - maximises expected (reward-cost)

- given a ____ horizon
  - finite
  - infinite
  - indefinite

- assuming full observability

Examples of MDPs

- Goal-directed, Indefinite Horizon, Cost Minimisation MDP
  - $<\mathcal{S}, \mathcal{A}, \mathcal{P}r, \mathcal{C}, \mathcal{G}, s_0>$
- Infinite Horizon, Discounted Reward Maximisation MDP
  - $<\mathcal{S}, \mathcal{A}, \mathcal{P}r, \mathcal{R}, \gamma>$
  - Reward = $\sum_t \gamma^t \mathbf{r}_t$
- Goal-directed, Finite Horizon, Prob. Maximisation MDP
  - $<\mathcal{S}, \mathcal{A}, \mathcal{P}r, \mathcal{G}, s_0, T>$

Bellman Equations for MDP_1

- $<\mathcal{S}, \mathcal{A}, \mathcal{P}r, \mathcal{C}, \mathcal{G}, s_0>$
- Define J*(s) {optimal cost} as the minimum expected cost to reach a goal from this state.
- J* should satisfy the following equation:

$$\boxed{\begin{aligned} J^*(s) &= 0 \ \ if \ \ s \in \mathcal{G} \\ J^*(s) &= \min_{a \in Ap(s)} \underbrace{\sum_{s' \in \mathcal{S}} \mathcal{P}r(s'|s,a)\left[\mathcal{C}(s,a,s') + J^*(s')\right]}_{Q^*(s,a)} \end{aligned}}$$

Bellman Equations for MDP_2

- $<\mathcal{S}, \mathcal{A}, \mathcal{P}r, \mathcal{R}, s_0, \gamma>$
- Define **V*(s)** {optimal **value**} as the **maximum** expected **discounted reward** from this state.
- V* should satisfy the following equation:

$$\boxed{V^*(s) \;=\; \max_{a \in Ap(s)} \sum_{s' \in \mathcal{S}} \mathcal{P}r(s'|s,a)\left[\mathcal{R}(s,a,s') + \gamma V^*(s')\right]}$$

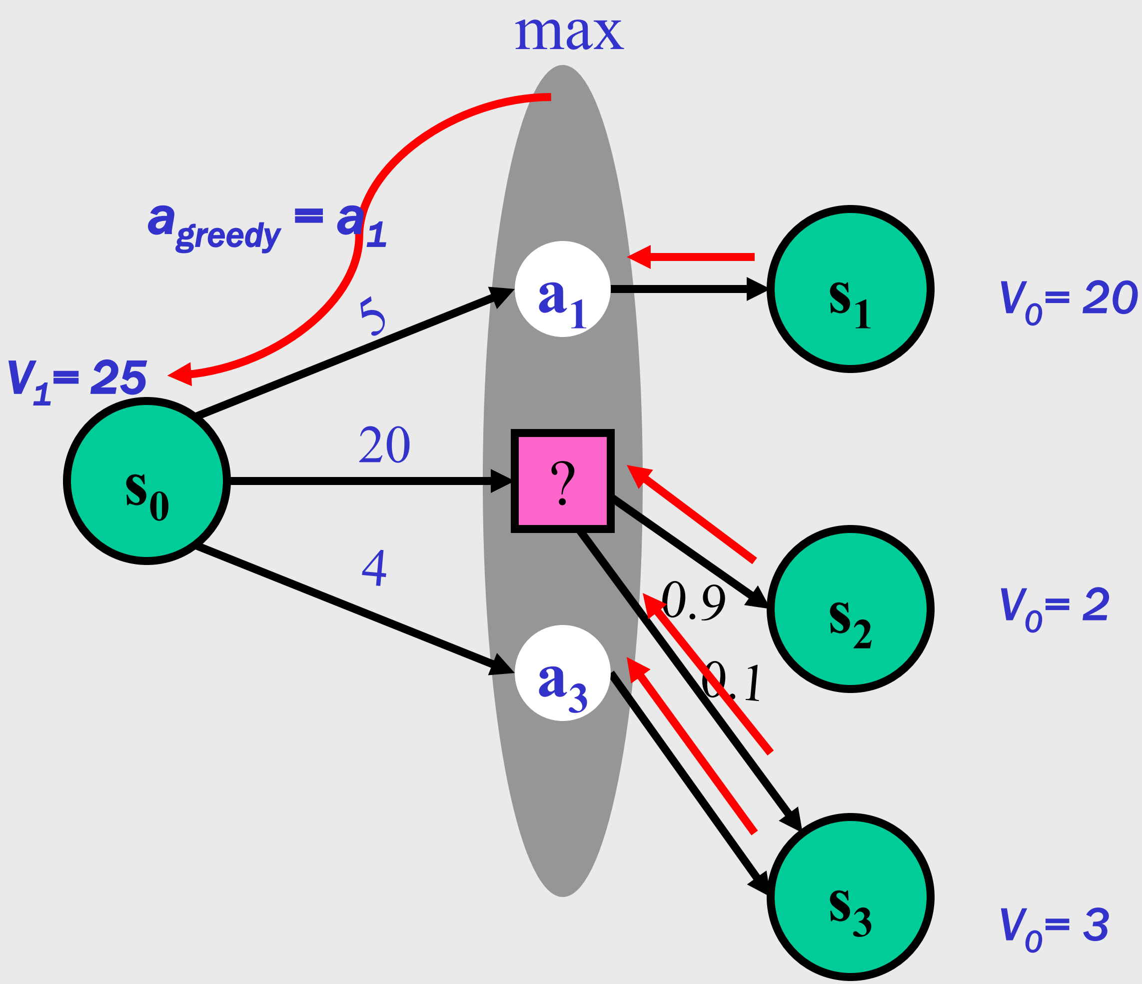
Bellman Backup

- Given an estimate of V* function (say V_n)
- Backup V_n function at state s
  - calculate a new estimate (V_{n+1}) :

$$\boxed{\begin{aligned} Q_{n+1}(s,a) &= \sum_{s' \in \mathcal{S}} Pr(s'|s,a)\left[\mathcal{R}(s,a,s') + \gamma V_n(s')\right] \\ V_{n+1}(s) &= \max_{a \in Ap(s)} \left[Q_{n+1}(s,a)\right] \end{aligned}}$$

- $Q_{n+1}(s,a)$: value/cost of the strategy:
  - execute action a in s, execute π_n subsequently
  - $\pi_n = \text{argmax}_{a \in Ap(s)} Q_n(s,a)$ (greedy action)

Bellman Backup



$Q_1(s,a_1) = 20 + 5$

$Q_1(s,a_2) = 20 + 0.9 \times 2$
$+ 0.1 \times 3$

$Q_1(s,a_3) = 4 + 3$

Value iteration [Bellman'57]

- assign an arbitrary assignment of V_0 to each non-goal state.
- repeat
  - for all states s
    - compute $V_{n+1}(s)$ by Bellman backup at s.
- until $\max_s |V_{n+1}(s) - V_n(s)| < \epsilon$

Iteration n+1

Residual(s)

ε-convergence

Complexity of value iteration

- One iteration takes $O(|\mathcal{A}||\mathcal{S}|^2)$ time.
- Number of iterations required
  - poly($|\mathcal{S}|$,$|\mathcal{A}|$,$1/(1-\gamma)$)
- Overall:
  - the algorithm is polynomial in state space
  - thus exponential in number of state variables.

Policy Computation

$$\begin{aligned}\pi^*(s) &= \operatorname*{argmax}_{a \in Ap(s)} Q^*(s,a) \\ &= \operatorname*{argmax}_{a \in Ap(s)} \sum_{s' \in \mathcal{S}} \mathcal{P}r(s'|s,a)\left[\mathcal{R}(s,a,s') + \gamma V^*(s')\right]\end{aligned}$$

Optimal policy is stationary and time-independent.

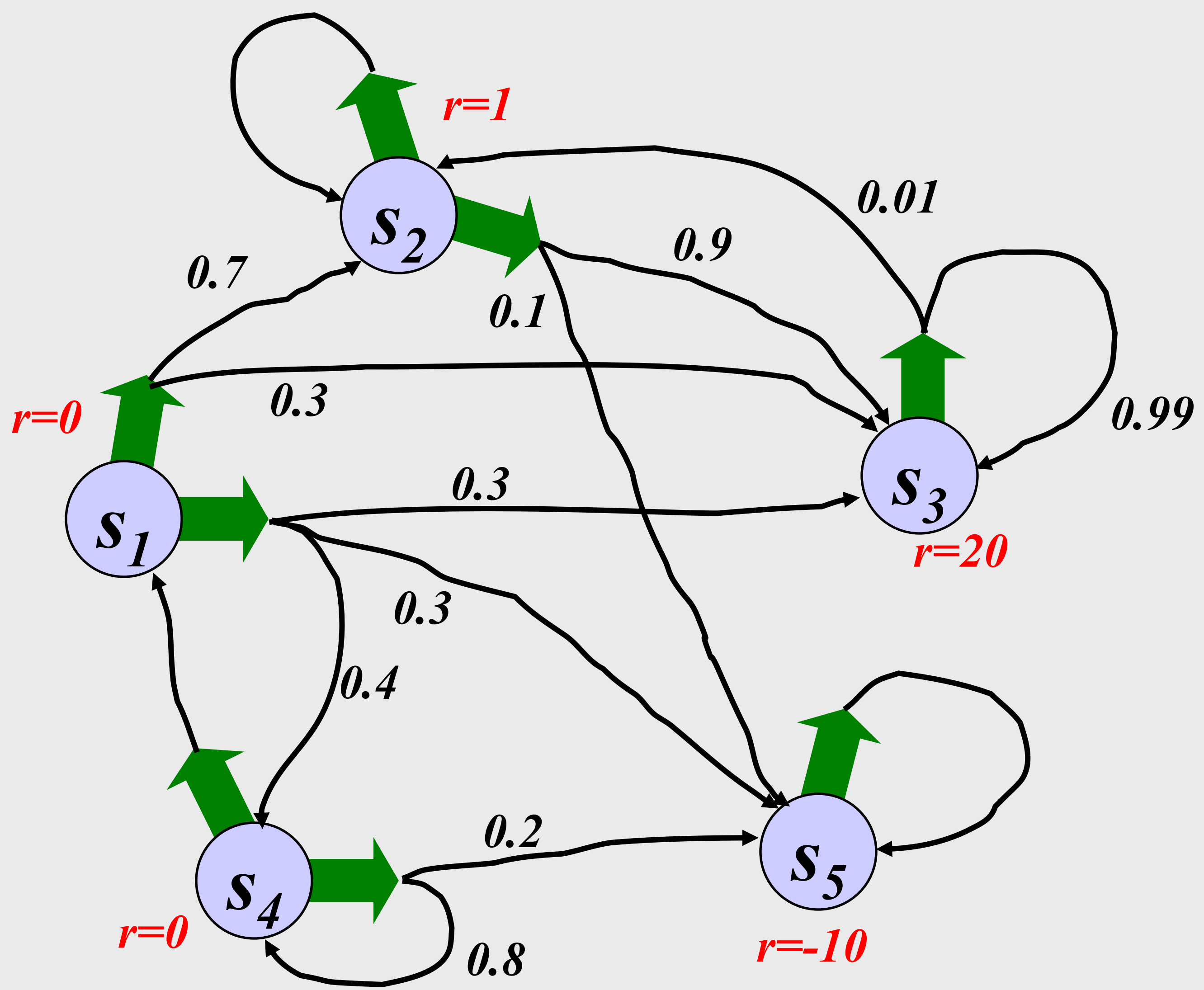
- for infinite/indefinite horizon problems

Policy Evaluation

$$\boxed{V_\pi(s) = \sum_{s' \in \mathcal{S}} \mathcal{P}r(s'|s,\pi(s))\left[\mathcal{R}(s,\pi(s),s') + \gamma V_\pi(s')\right]}$$

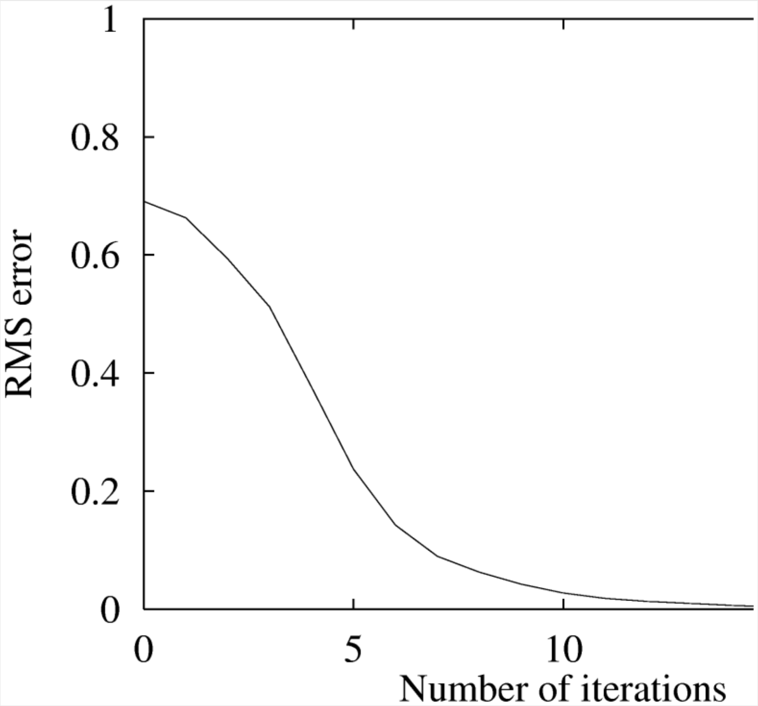
A system of linear equations in $|\mathcal{S}|$ variables.

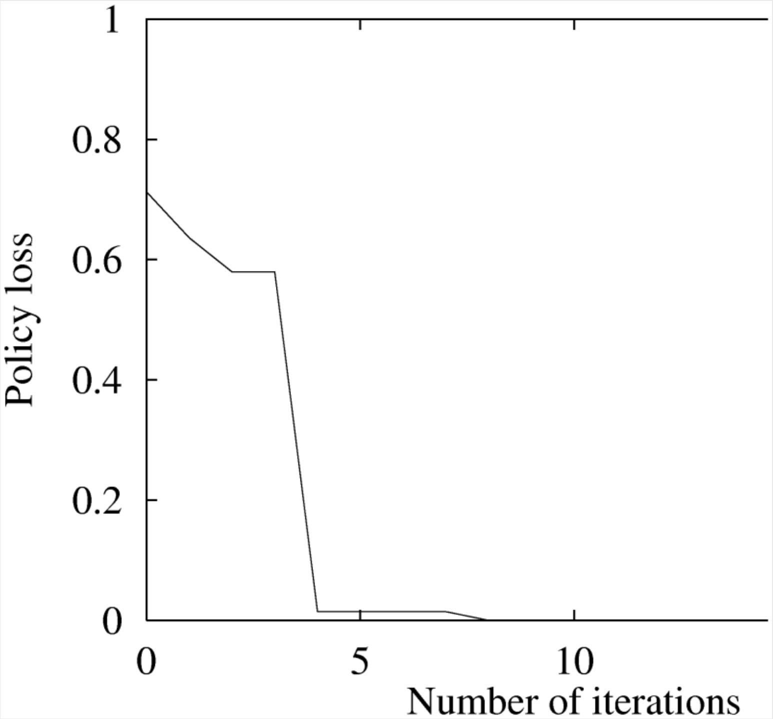
Markov Decision Process (MDP)



Value Function and Policy

- Value residual and policy residual





Changing the Search Space

- Value Iteration
  - Search in value space
  - Compute the resulting policy

- Policy Iteration [Howard'60]
  - Search in policy space
  - Compute the resulting value

Policy iteration [Howard'60]

- assign an arbitrary assignment of π_0 to each state.
- repeat
  - compute V_{n+1}: the evaluation of π_n
  - for all states s

    compute $\pi_{n+1}(s)$: $\text{argmax}_{a\in Ap(s)} Q_{n+1}(s,a)$
- until $\pi_{n+1} = \pi_n$

costly: $O(n^3)$

Modified Policy Iteration

approximate by value iteration using fixed policy

Advantage

- searching in a finite (policy) space as opposed to uncountably infinite (value) space ⇒ convergence faster.
- all other properties follow!

LP Formulation

minimise $\sum_{s \in \mathcal{S}} V^*(s)$

under constraints:

for every s, a

$V^*(s) \geq \mathcal{R}(s) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}r(s'|a,s) V^*(s')$

A big LP. So other tricks used to solve it!

Hybrid MDPs

Hybrid Markov decision process:

Markov state = $(n, \mathbf{x})$, where n is the discrete component (set of fluents) and $\mathbf{x} \in \Re^l$.

Bellman's equation:

$$V_n^{t+1}(\mathbf{x}) = \max_{a \in A_n(\mathbf{x})} \left[\sum_{n' \in N} \Pr(n' \mid n, \mathbf{x}, a) \int_{\mathbf{x}' \in \mathbf{X}} \Pr(\mathbf{x}' \mid n, \mathbf{x}, a, n') \left(R_{n'}(\mathbf{x}') + V_{n'}^{t}(\mathbf{x}') \right) d\mathbf{x}' \right]$$

Hybrid MDPs

Hybrid Markov decision process:

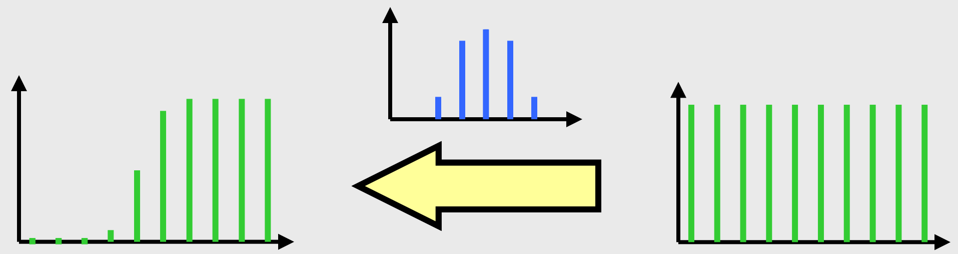
Markov state = $(n, \mathbf{x})$, where n is the discrete component (set of fluents) and $\mathbf{x} \in \Re^l$.

Bellman's equation:

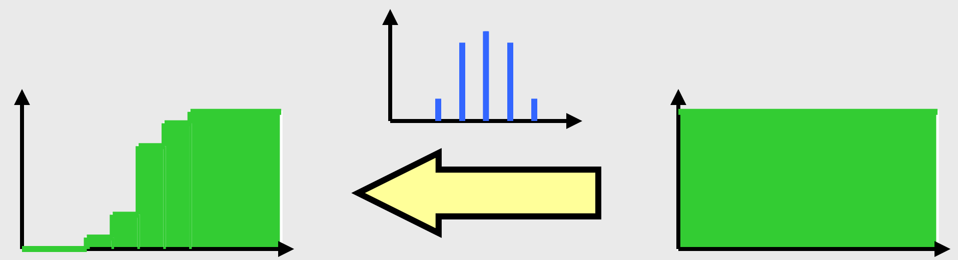
$$V_n^{t+1}(\mathbf{x}) = \max_{a \in A_n(\mathbf{x})} \left[\sum_{n' \in N} \Pr(n' | n, \mathbf{x}, a) \int_{\mathbf{x}' \in \mathbf{X}} \Pr(\mathbf{x}' | n, \mathbf{x}, a, n') \left(R_{n'}(\mathbf{x}') + V_{n'}^{t}(\mathbf{x}') \right) d\mathbf{x}' \right]$$

Convolutions

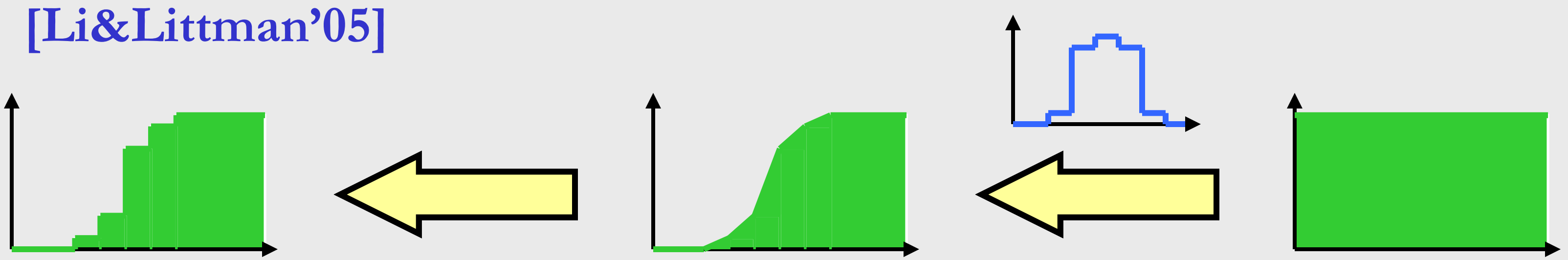
discrete-discrete



constant-discrete
[Feng et.al.'04]



constant-constant
[Li&Littman'05]



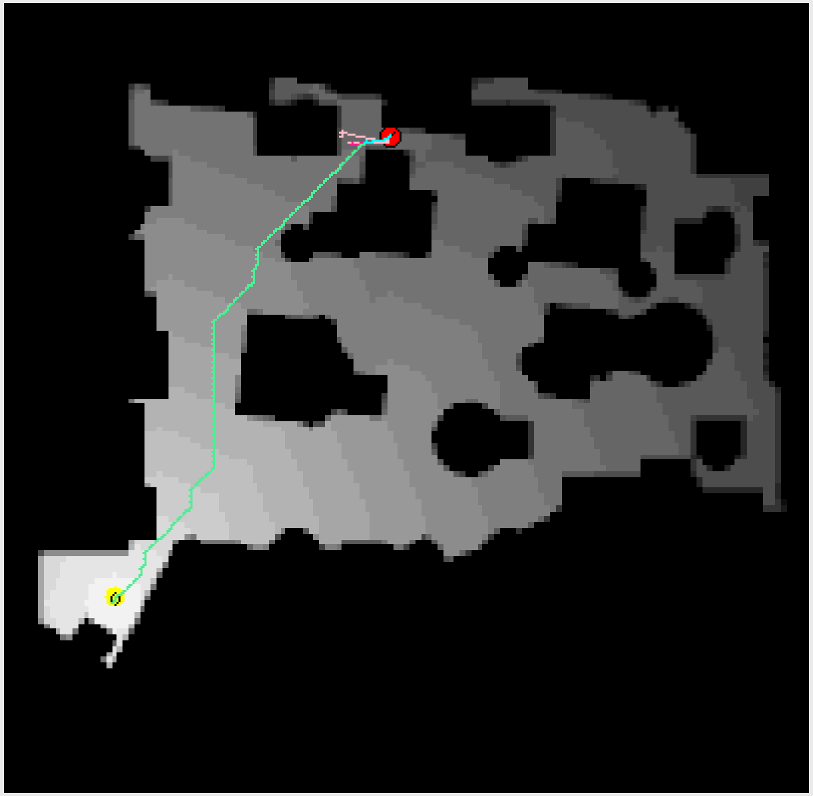
Result of convolutions

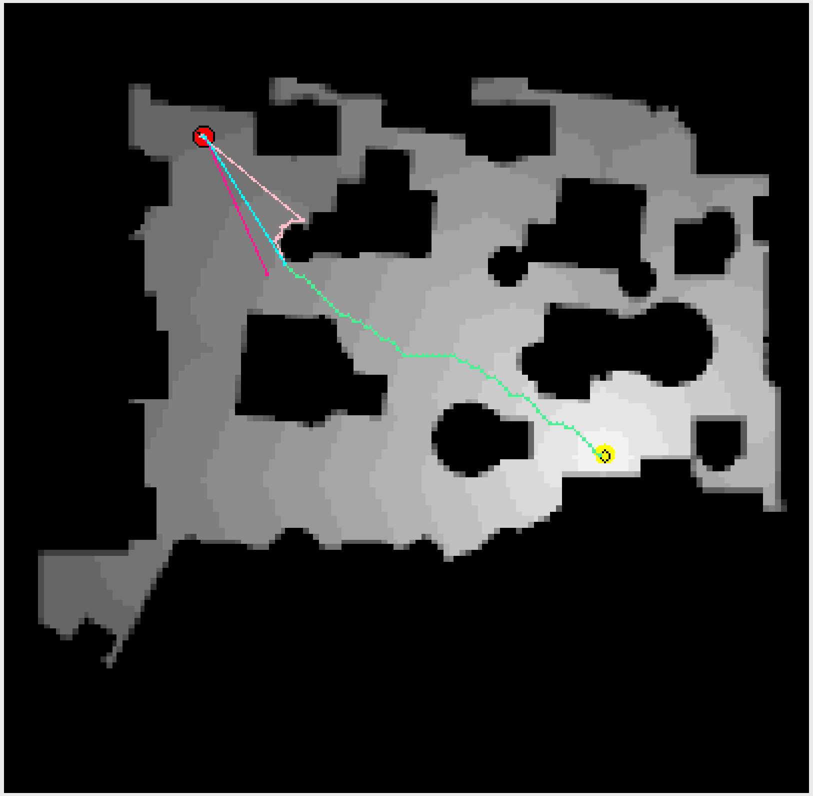
value function

probability density function

	discrete	constant	linear
discrete	discrete	constant	linear
constant	constant	linear	quadratic
linear	linear	quadratic	cubic

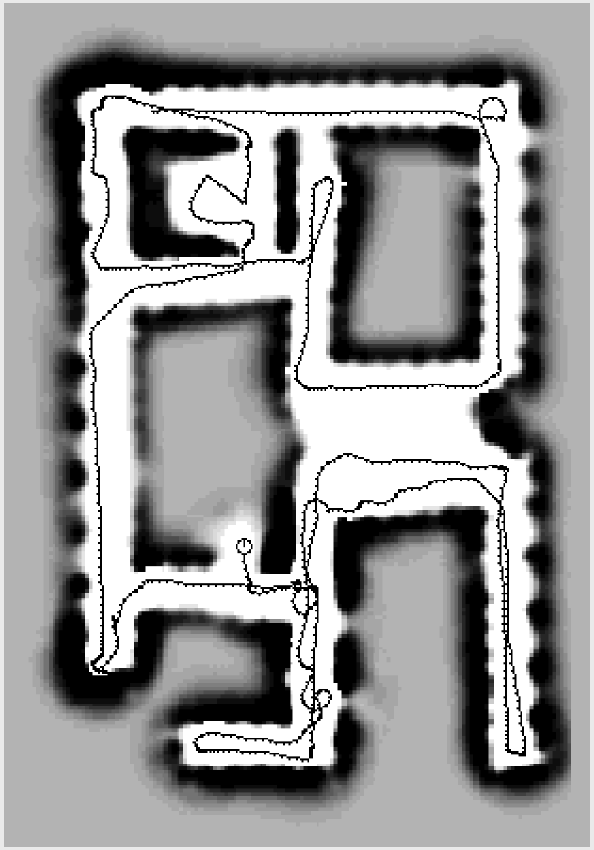
Value Iteration for Motion Planning
(assumes knowledge of robot's location)

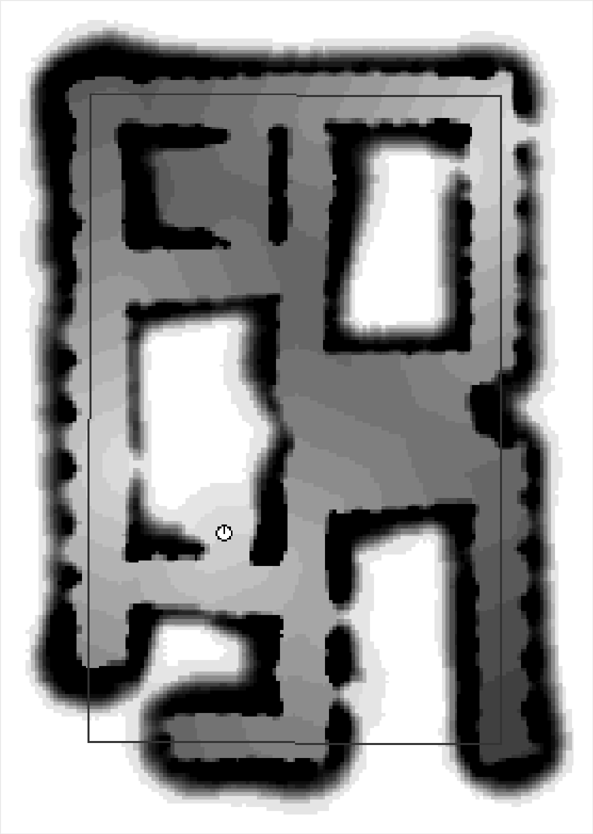




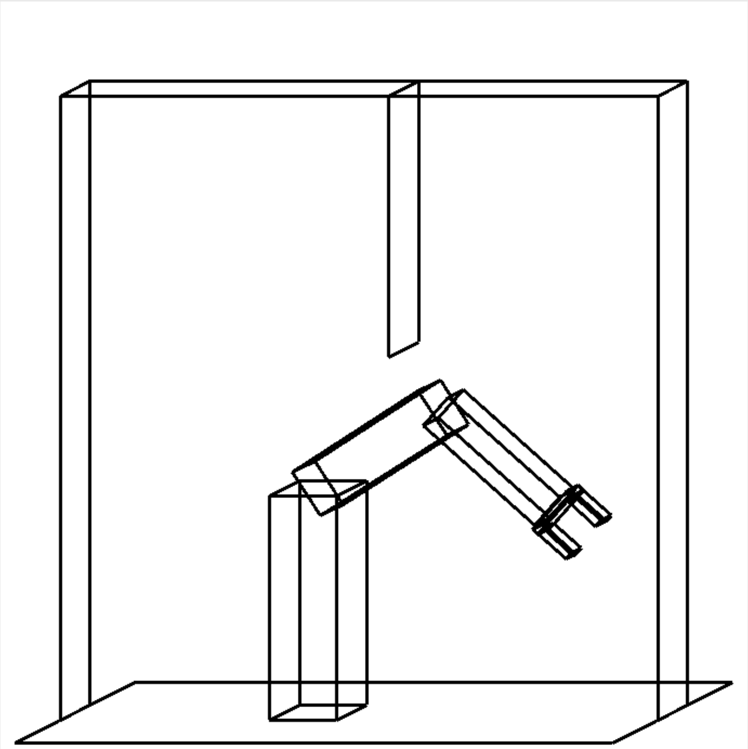
Frontier-based Exploration

- Every unknown location is a target point.

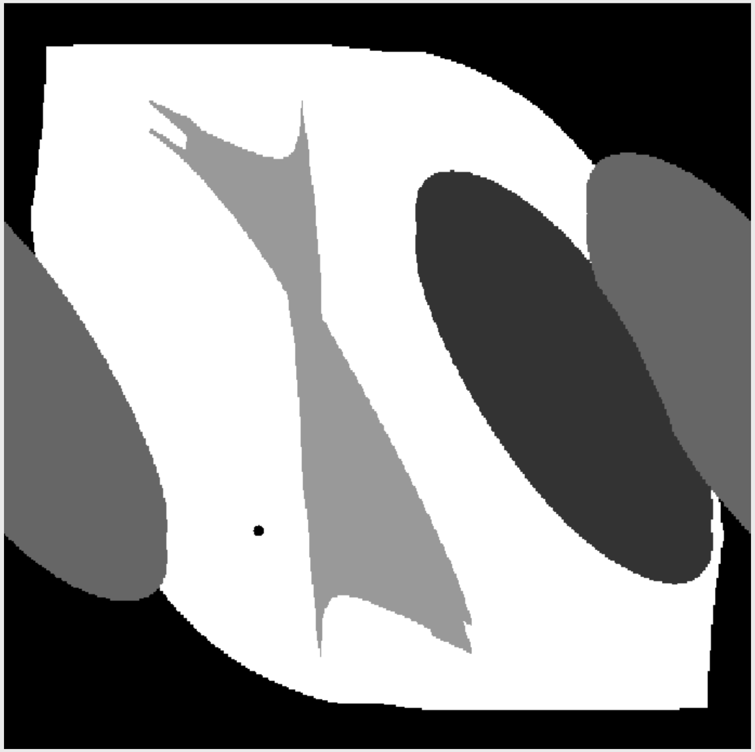




Manipulator Control

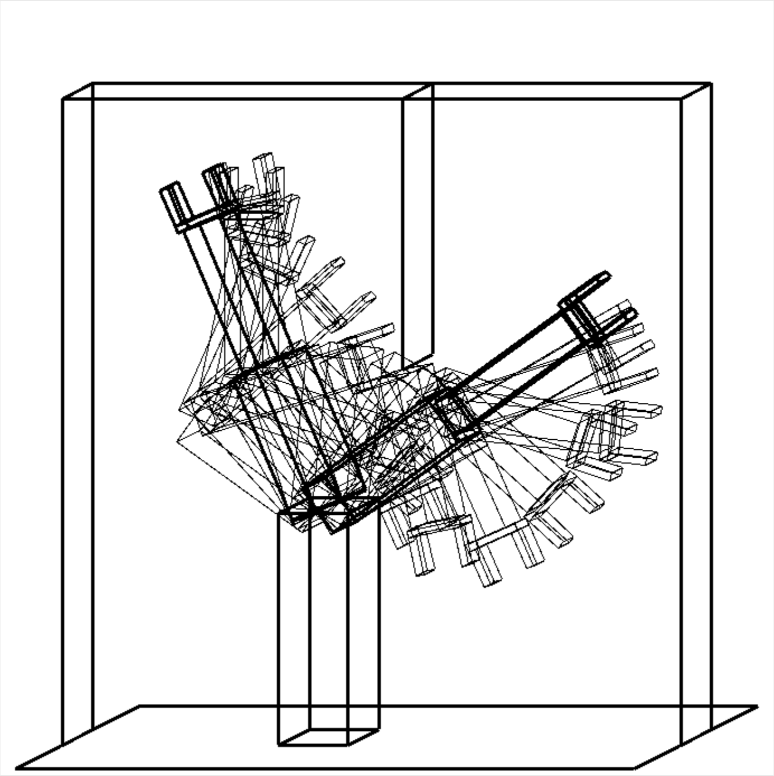


Arm with two joints

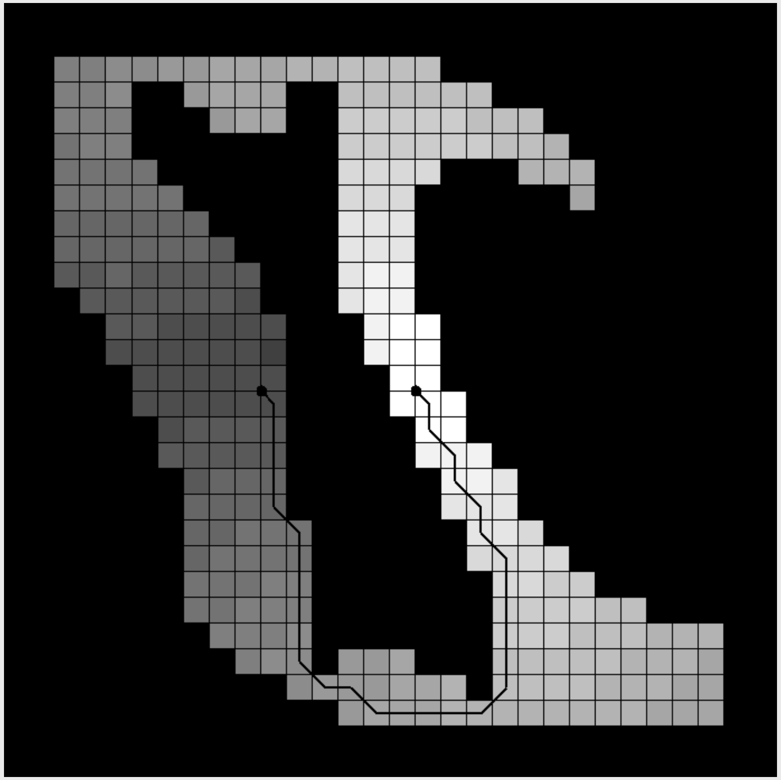


Configuration space

Manipulator Control Path

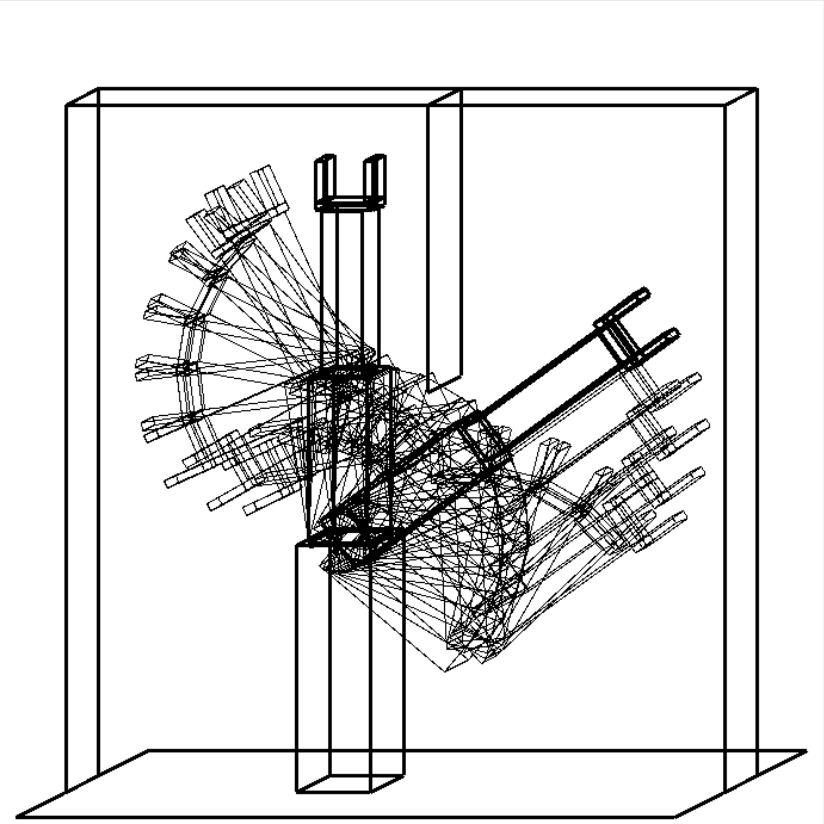


State space

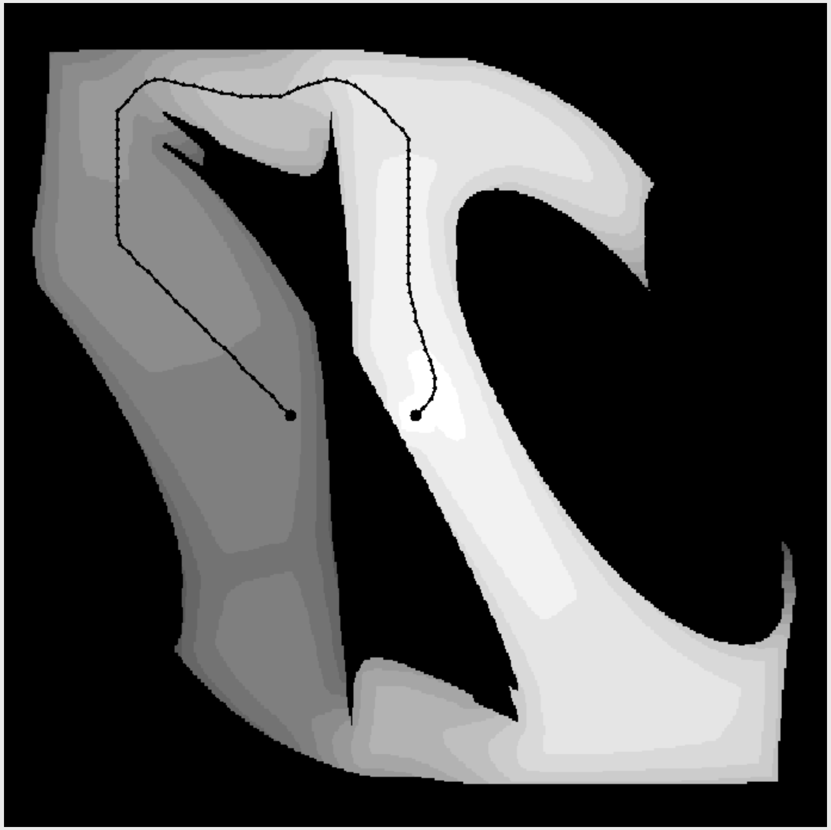


Configuration space

Manipulator Control Path



State space



Configuration space

Collision Avoidance via Planning

- Potential field methods have local minima
- Perform efficient path planning in the local perceptual space
- Path costs depend on length and closeness to obstacles

[Konolige, Gradient method]

Paths and Costs

- Path is list of points $P=\{p_1, p_2, \ldots p_k\}$
- p_k is only point in goal set
- Cost of path is separable into intrinsic cost at each point along with adjacency cost of moving from one point to the next

$$F(P) = \sum_i I(p_i) + \sum_i A(p_i, p_{i+1})$$

- Adjacency cost typically Euclidean distance
- Intrinsic cost typically occupancy, distance to obstacle

Navigation Function

- Assignment of potential field value to every element in configuration space [Latombe, 91].
- Goal set is always downhill, no local minima.
- Navigation function of a point is cost of minimal cost path that starts at that point.

$$N_k = \min_{P_k} F(P_k)$$

Computation of Navigation Function

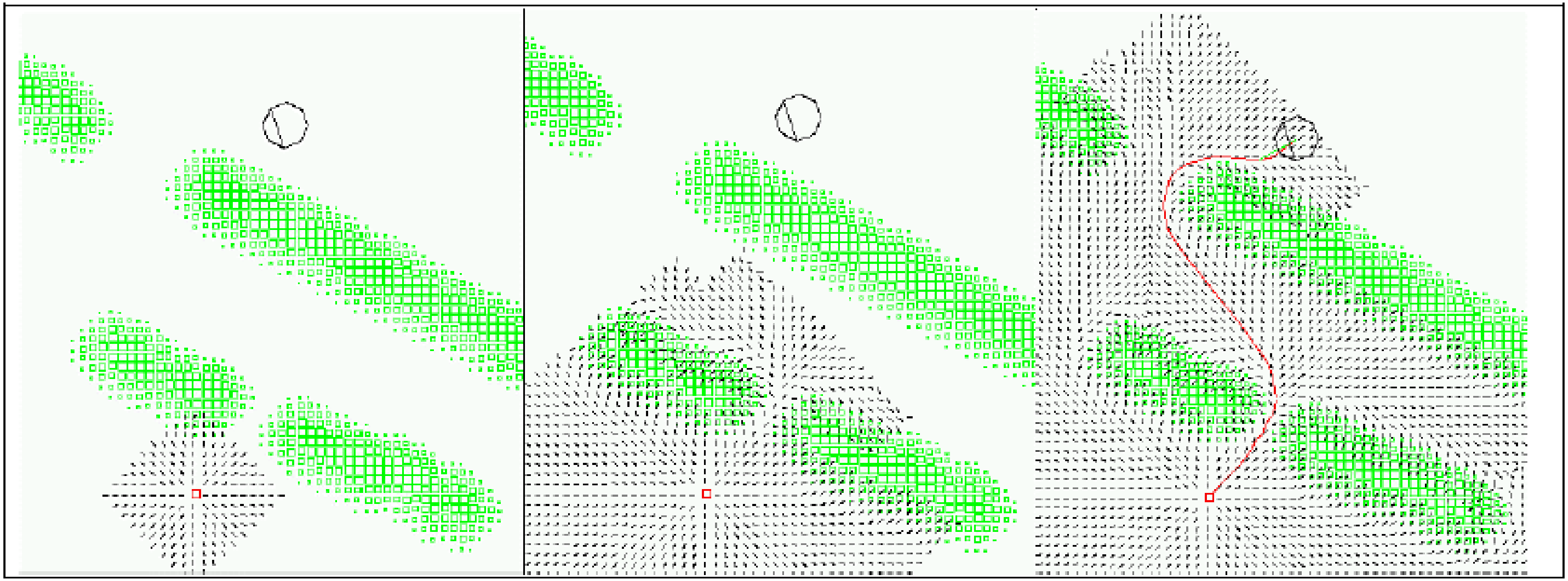


Figure 2. Three stages of the LPN algorithm, starting from a single goal point. The rectangles indicate the intrinsic cost of a point. The gradient direction at each point is shown as a short black line. The images shown are at 10, 30, and 70 iterations. The interpolated path from the robot to the goal is shown in the last image.

Challenges

- Where do we get the state space from?
- Where do we get the model from?
- What happens when the world is slightly different?
- Where does reward come from?

- **Continuous state variables**
- **Continuous action space**

How to solve larger problems?

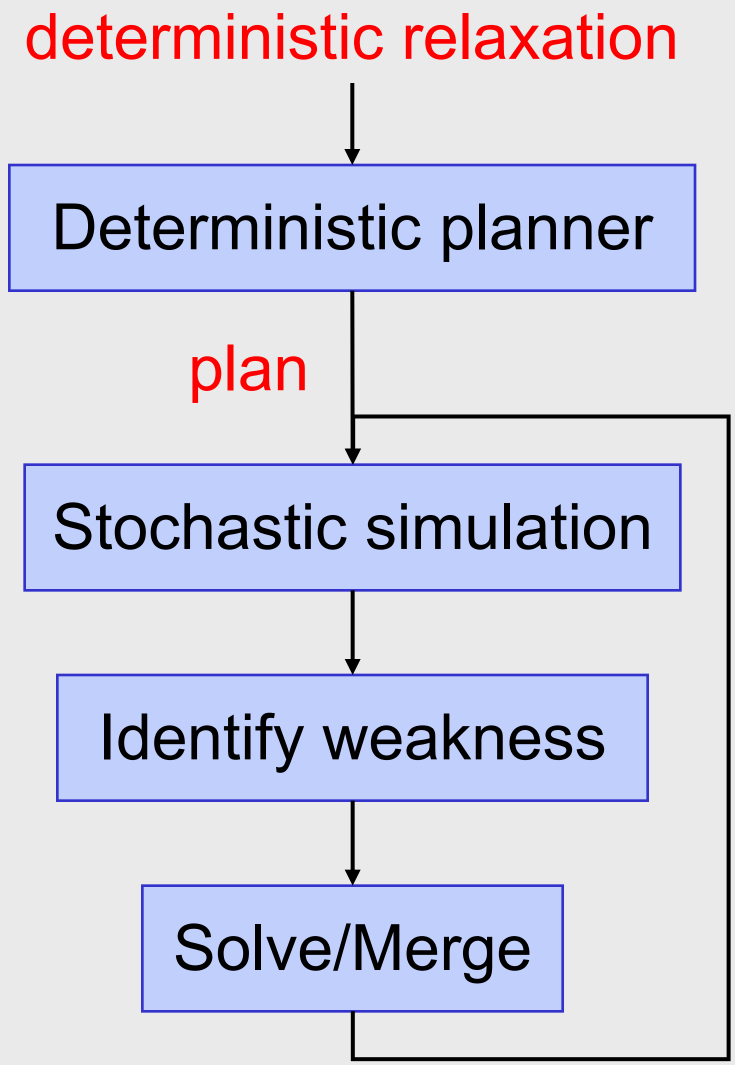
- If deterministic problem
  - Use dijkstra's algorithm
- If no back-edge
  - Use backward Bellman updates
- Prioritize Bellman updates
  - to maximize information flow
- If known initial state
  - Use dynamic programming + heuristic search
  - LAO*, RTDP and variants
- Divide an MDP into sub-MDPs are solve the hierarchy
- Aggregate states with similar values
- Relational MDPs

Approximations: n-step lookahead

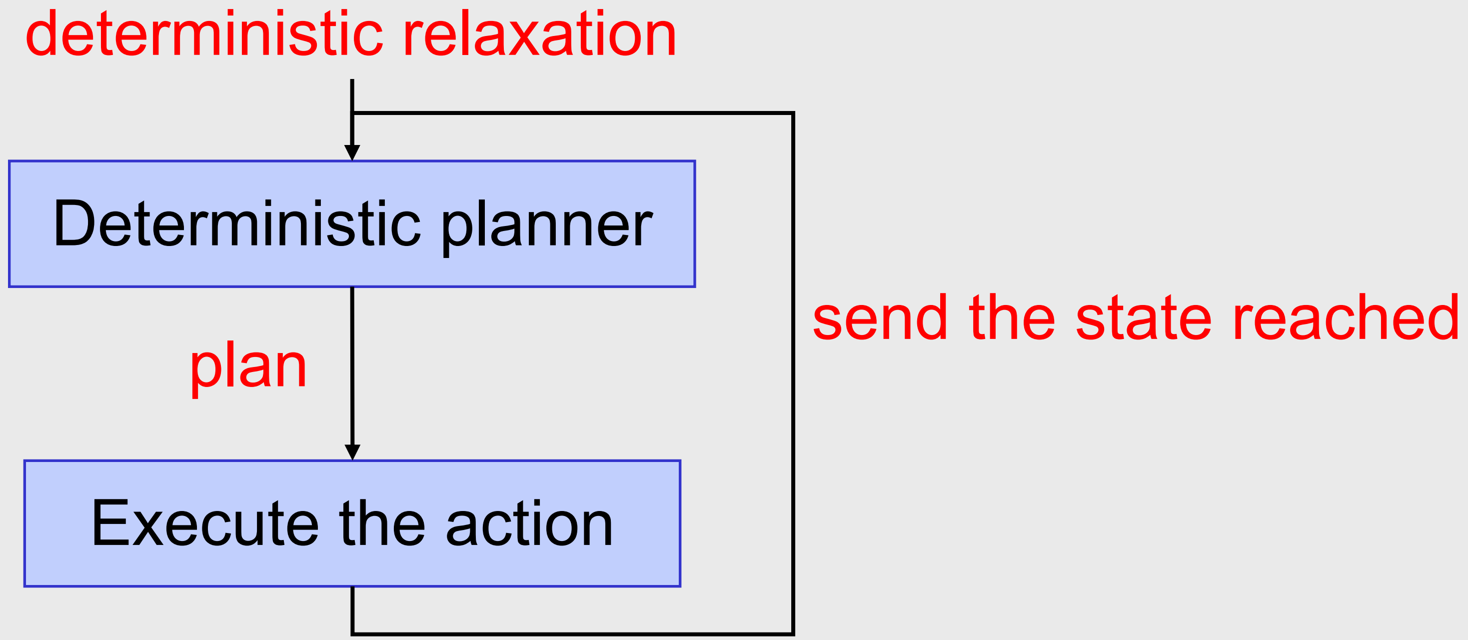
- n=1 : greedy
  - $\pi_1(s) = \text{argmax}_a \mathcal{R}(s,a)$

- n-step lookahead
  - $\pi_n(s) = \text{argmax}_a V_n(s)$

Approximation: Incremental approaches



Approximations: Planning and Replanning



CSE-571
AI-based Mobile Robotics

Planning and Control:

(1) Reinforcement Learning
(2) Partially Observable Markov Decision Processes

Reinforcement Learning

- Still have an MDP
  - Still looking for policy π
- New twist: don't know $\mathcal{P}$r and/or $\mathcal{R}$
  - i.e. don't know which states are good
  - And what actions do

- Must actually try actions and states out to learn

Model based methods

- Visit different states, perform different actions
- Estimate $\mathcal{P}$r and $\mathcal{R}$

- Once model built, do planning using V.I. or other methods

- Cons: require _huge_ amounts of data

Model free methods

- TD learning
- Directly learn Q*(s,a) values

$$Q^*(s,a) \;=\; \sum_{s' \in \mathcal{S}} \mathcal{P}r(s'|s,a)\left[\mathcal{R}(s,a,s') + \gamma V^*(s')\right]$$

$$Q^*(s,a) \;=\; \sum_{s' \in \mathcal{S}} \mathcal{P}r(s'|s,a)\left[\mathcal{R}(s,a,s') + \gamma max_{a'} Q^*(s',a')\right]$$

- sample = $\mathcal{R}$(s,a,s') + $\gamma \max_{a'} Q_n(s',a')$
- Nudge the old estimate towards the new sample
- $Q_{n+1}(s,a) \leftarrow (1-\alpha)Q_n(s,a) + \alpha$[sample]

Properties

- Converges to optimal if
  - If you explore enough
  - If you make learning rate (α) small enough
  - But not decrease it too quickly

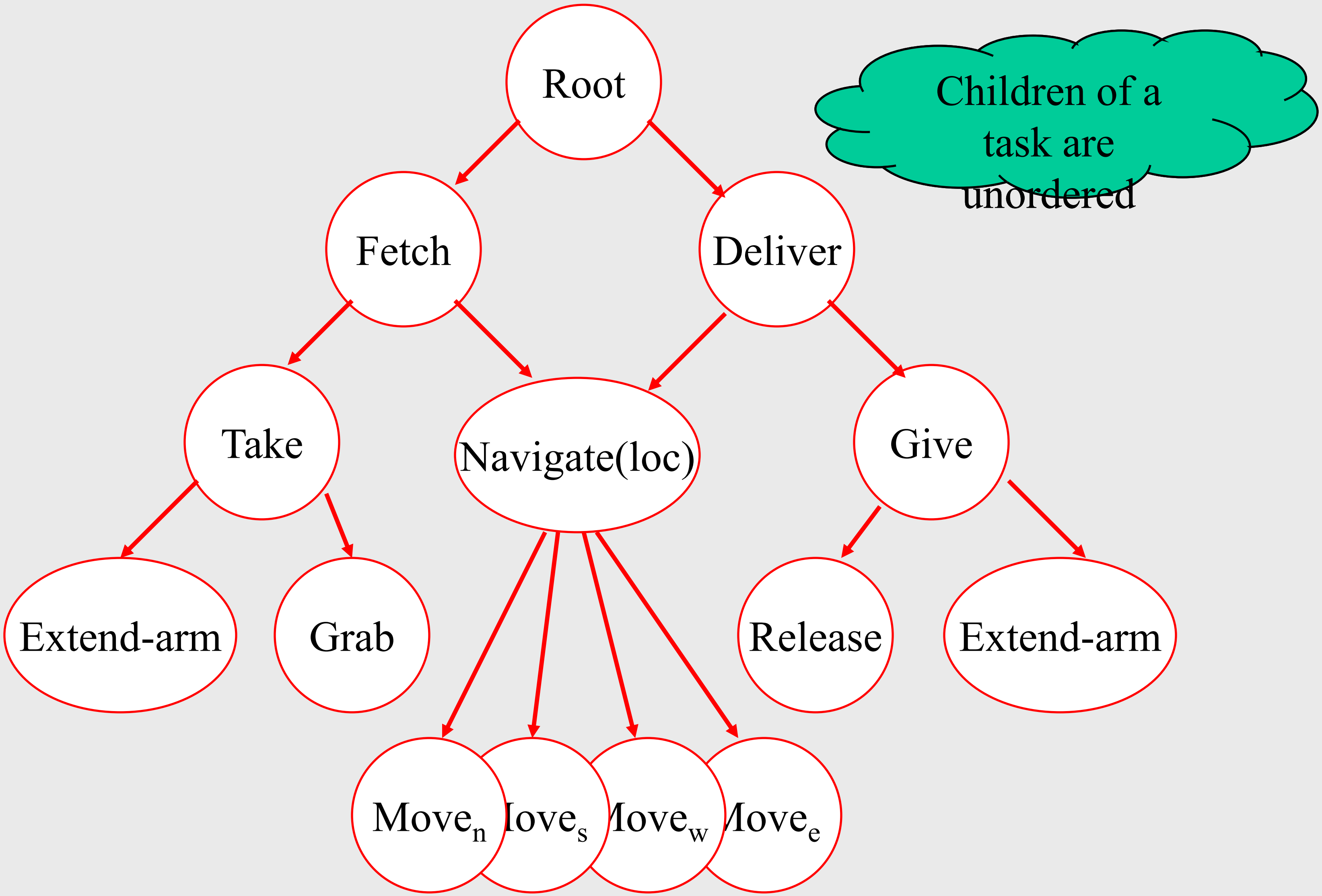
Exploration vs. Exploitation

- ε-greedy
  - Each time step flip a coin
  - With prob ε, action randomly
  - With prob 1-ε take the current greedy action
- Lower ε over time to increase exploitation as more learning has happened

Q-learning

- Problems
  - Too many states to visit during learning
  - Q(s,a) is a BIG table
- We want to generalize from small set of training examples
- Solutions
  - Value function approximators
  - Policy approximators
  - Hierarchical Reinforcement Learning

Task Hierarchy: MAXQ Decomposition [Dietterich'00]



MAXQ Decomposition

- Augment the state s by adding the subtask i: [s,i].
- Define $\mathcal{C}$([s,i],j) as the reward received in i after j finishes.
- $\mathcal{Q}$([s,Fetch],Navigate(prr)) = $\mathcal{V}$([s,Navigate(prr)])+$\mathcal{C}$([s,Fetch],Navigate(prr))

  Reward received while navigating

  Reward received after navigation

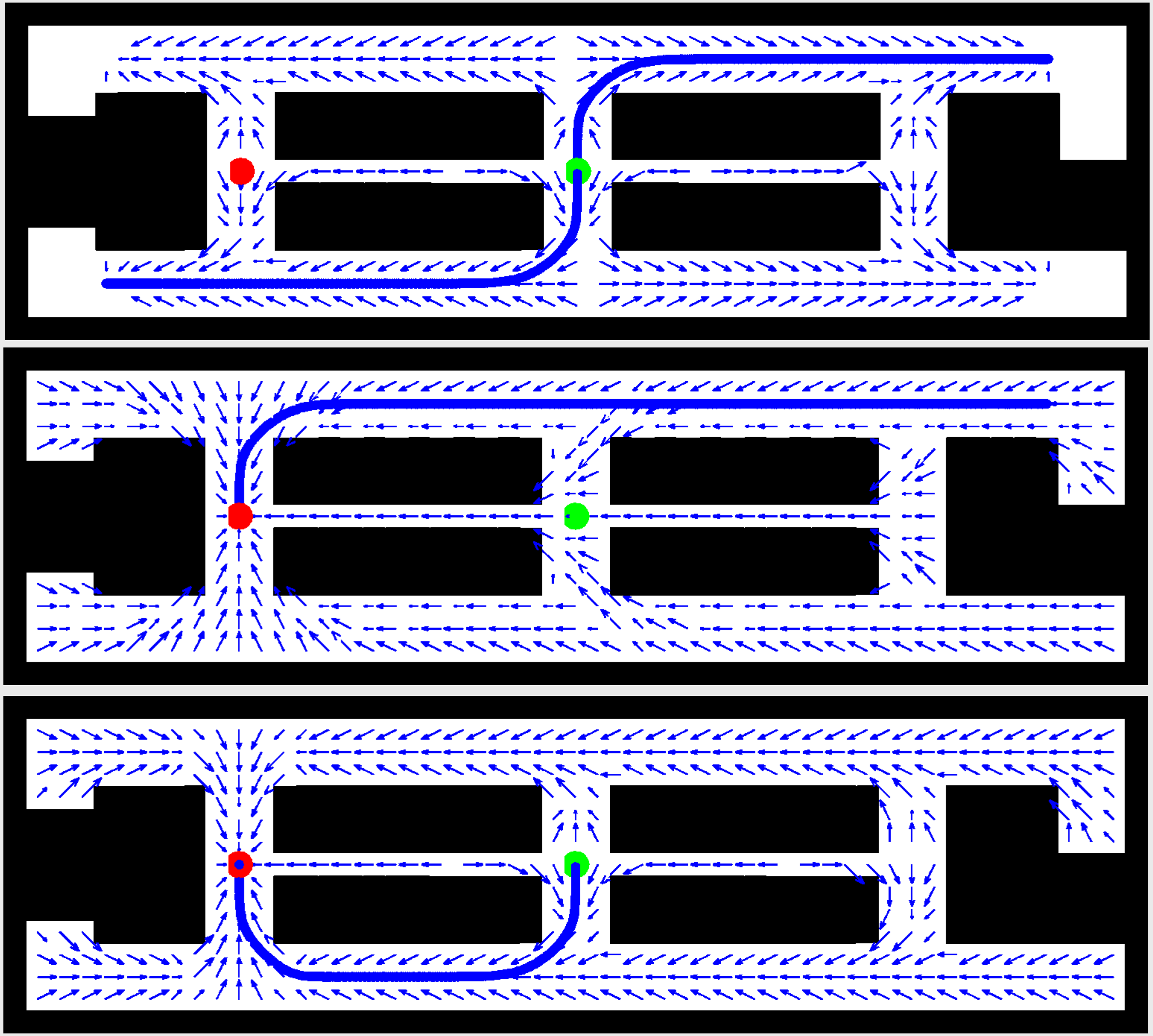
- Express $\mathcal{V}$ in terms of $\mathcal{C}$
- Learn $\mathcal{C}$, instead of learning $\mathcal{Q}$

MAXQ Decomposition (contd)

$$\mathcal{C}_{n+1}(s,i,a) \leftarrow (1-\alpha)\mathcal{C}_n(s,i,a) + \alpha\gamma^N \left[max_{a'} V(s',a') + C_n(s',i,a')\right]$$

- State Abstraction
  - Finding irrelevant actions
  - Finding funnel actions

POMDPs: Recall example



Partially Observable Markov Decision Processes

POMDPs

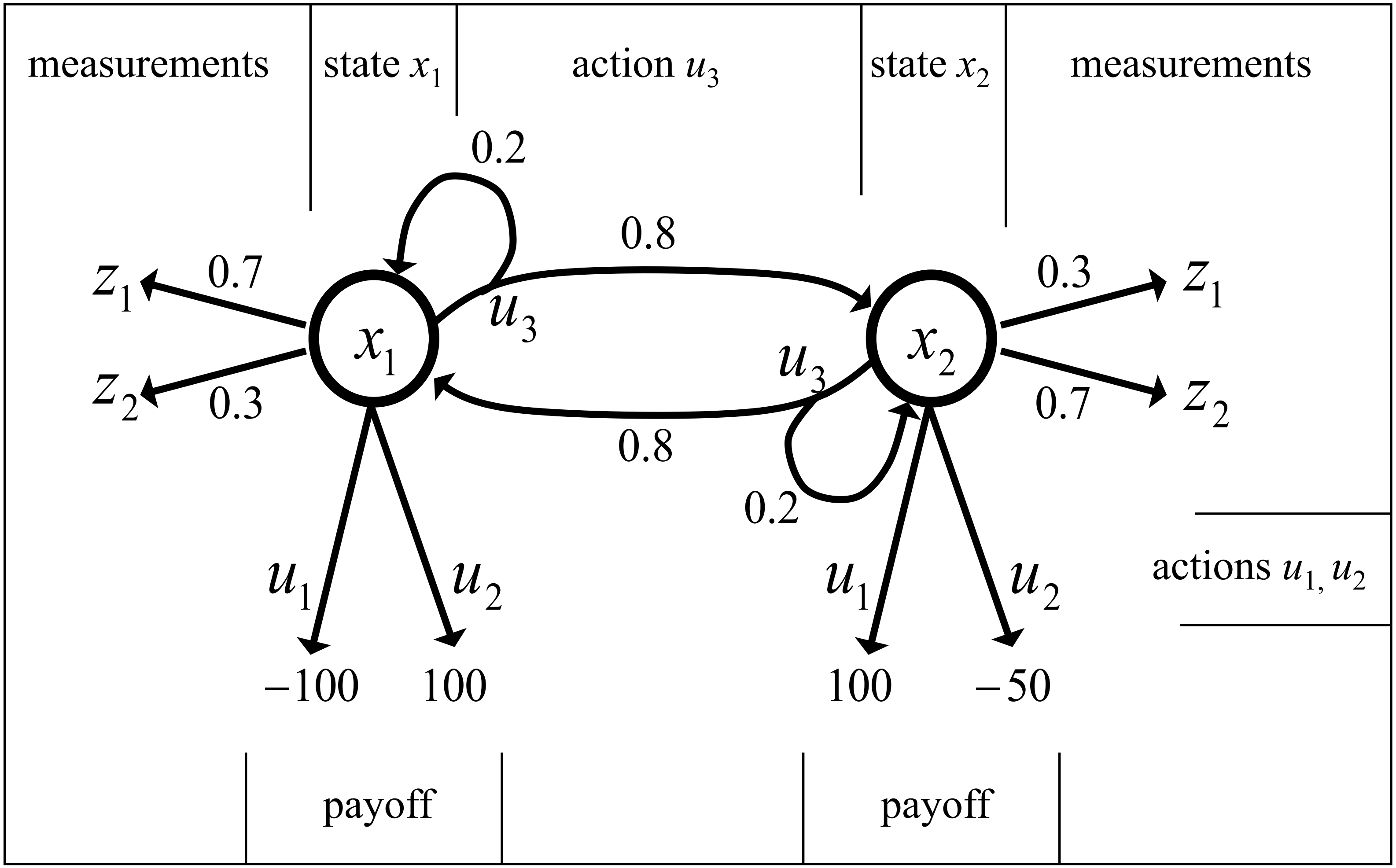
- In POMDPs we apply the very same idea as in MDPs.
- **Since the state is not observable**, the agent has to **make its decisions based on** the belief state which is a **posterior distribution over states**.
- Let b be the belief of the agent about the state under consideration.
- POMDPs compute a **value function over belief space**:

$$V_T(b) \;=\; \gamma \max_u \left[r(b,u) + \int V_{T-1}(b') p(b' \mid u, b) \, db' \right]$$

Problems

- Each belief is a probability distribution, thus, each value in a **POMDP is a function of an entire probability distribution**.
- **This is problematic, since probability distributions are continuous**.
- Additionally, we have to deal with the **huge complexity of belief spaces**.
- For **finite worlds** with finite state, action, and measurement spaces and finite horizons, however, we can **effectively represent the value functions by piecewise linear functions**.

An Illustrative Example



The Parameters of the Example

- The actions u_1 and u_2 are terminal actions.
- The action u_3 is a sensing action that potentially leads to a state transition.
- The horizon is finite and γ=1.

$$\begin{aligned} r(x_1, u_1) &= -100 & \quad r(x_2, u_1) &= +100 \\ r(x_1, u_2) &= +100 & \quad r(x_2, u_2) &= -50 \\ r(x_1, u_3) &= -1 & \quad r(x_2, u_3) &= -1 \end{aligned}$$

$$\begin{aligned} p(x'_1 | x_1, u_3) &= 0.2 & \quad p(x'_2 | x_1, u_3) &= 0.8 \\ p(x'_1 | x_2, u_3) &= 0.8 & \quad p(z'_2 | x_2, u_3) &= 0.2 \end{aligned}$$

$$\begin{aligned} p(z_1 | x_1) &= 0.7 & \quad p(z_2 | x_1) &= 0.3 \\ p(z_1 | x_2) &= 0.3 & \quad p(z_2 | x_2) &= 0.7 \end{aligned}$$

Payoff in POMDPs

- In MDPs, the payoff (or return) depended on the state of the system.
- In POMDPs, however, the true state is not exactly known.
- Therefore, we compute the **expected payoff** by **integrating ov**

$$
\begin{aligned}
r(b,u) &= E_x[r(x,u)] \\
&= \int r(x,u)p(x)\,dx \\
&= p_1\, r(x_1,u) + p_2\, r(x_2,u)
\end{aligned}
$$

Payoffs in Our Example (1)

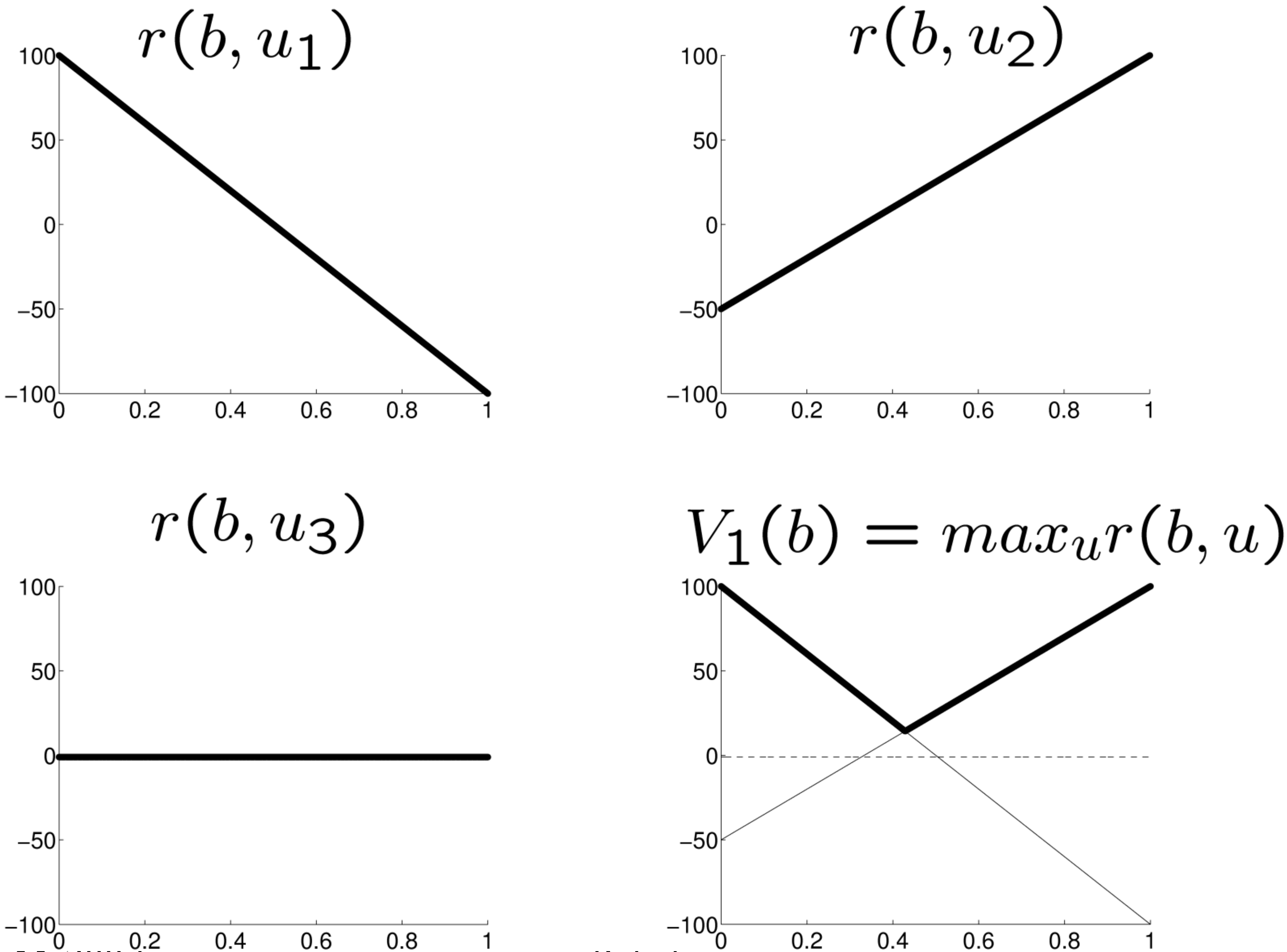
- If we are totally certain that we are in state x_1 and execute action u_1, we receive a reward of -100
- If, on the other hand, we definitely know that we are in x_2 and execute u_1, the reward is +100.
- In between it is the linear combination of the extreme values weighted by the probabilities

$$\begin{aligned} r(b, u_1) &= -100\, p_1 + 100\, p_2 \\ &= -100\, p_1 + 100\, (1 - p_1) \end{aligned}$$

$$r(b, u_2) = 100\, p_1 - 50\, (1 - p_1)$$

$$r(b, u_3) = -1$$

Payoffs in Our Example (2)



The Resulting Policy for T=1

- Given we have a finite POMDP with T=1, we would use $V_1(b)$ to determine the optimal policy.
- In our example, the optimal policy for T=1 is

$$\pi_1(b) = \begin{cases} u_1 & \text{if } p_1 \leq \frac{3}{7} \\ \\ u_2 & \text{if } p_1 > \frac{3}{7} \end{cases}$$

- This is the upper thick graph in the diagram.

Piecewise Linearity, Convexity

- The resulting value function $V_1(b)$ is the maximum of the three functions at each point

$$
\begin{aligned}
V_1(b) &= \max_u r(b,u) \\
&= \max \left\{ \begin{array}{rr} -100\, p_1 & +100\,(1-p_1) \\ 100\, p_1 & -50\,(1-p_1) \\ -1 & \end{array} \right\}
\end{aligned}
$$

- It is piecewise linear and convex.

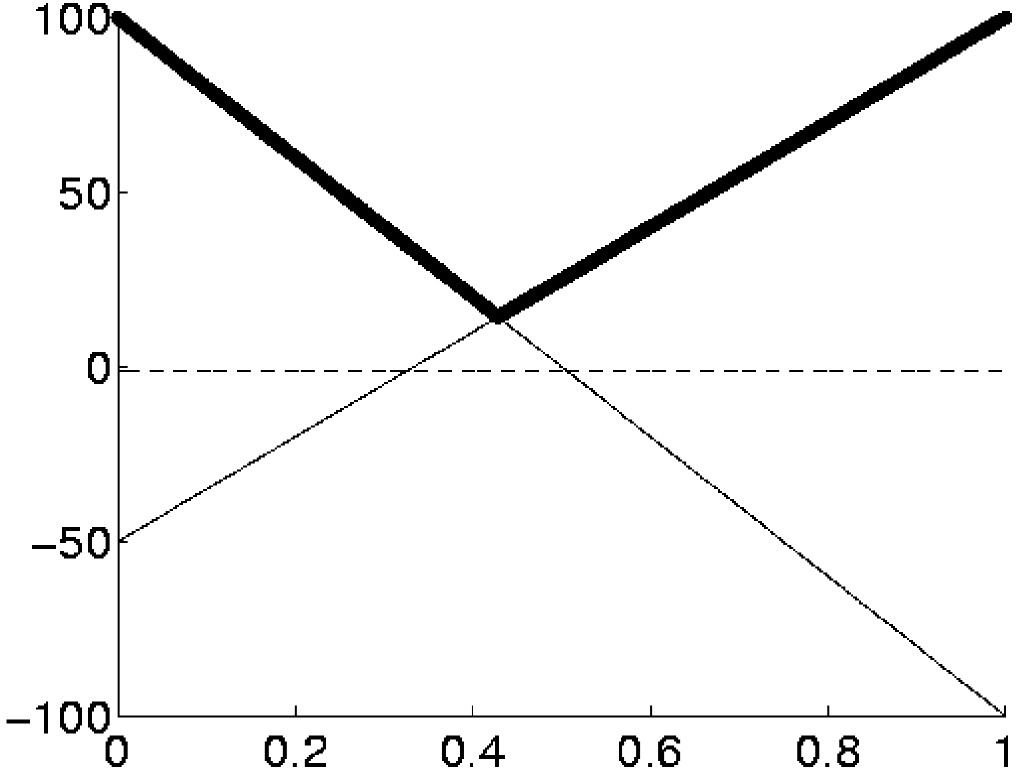
Pruning

- If we carefully consider $V_1(b)$, we see that only the first two components contribute.
- The third component can therefore safely be pruned away from $V_1(b)$.

$$V_1(b) \;=\; \max \left\{ \begin{array}{rr} -100\, p_1 & +100\,(1-p_1) \\ 100\, p_1 & -50\,(1-p_1) \end{array} \right\}$$

Increasing the Time Horizon

- Assume the robot can make an observation before deciding on an action.



$V_1(b)$

Increasing the Time Horizon

- Assume the robot can make an observation before deciding on an action.
- Suppose the robot perceives z_1 for which $p(z_1 \mid x_1)=0.7$ and $p(z_1 \mid x_2)=0.3$.
- Given the observation z_1 we update the belief using Bayes rule.

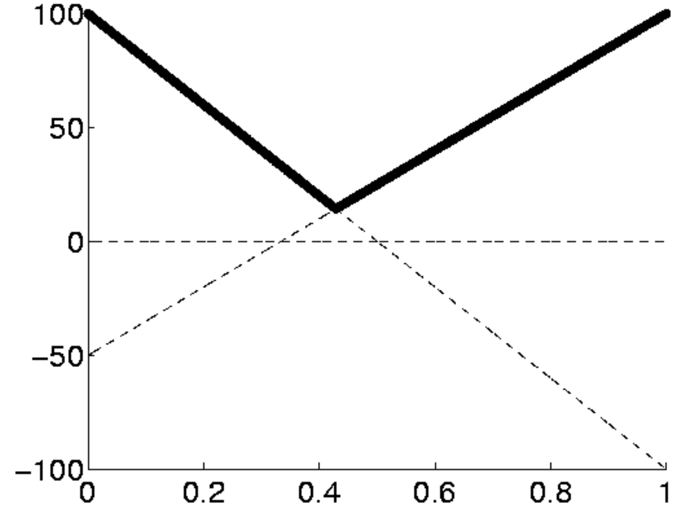
$$p'_1 = \frac{0.7\, p_1}{p(z_1)}$$

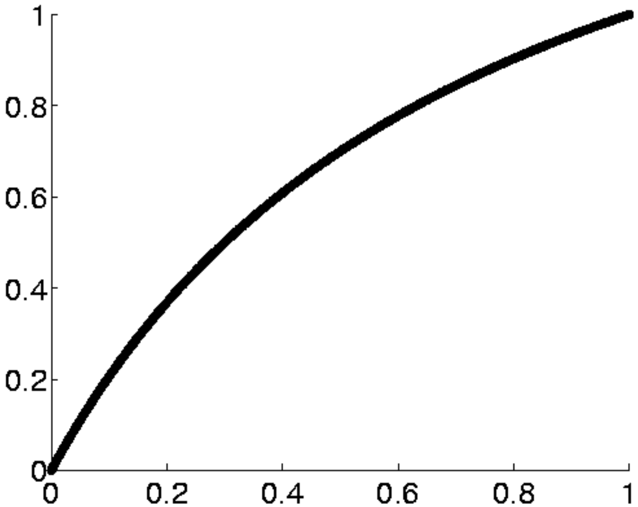
$$p'_2 = \frac{0.3(1-p_1)}{p(z_1)}$$

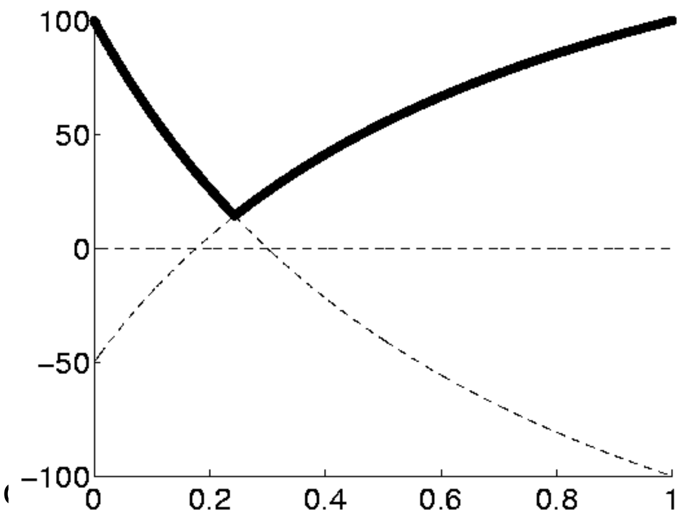
$$p(z_1) = 0.7\, p_1 + 0.3(1-p_1) = 0.4\, p_1 + 0.3$$

Value Function

$V_1(b)$







$b'(b|z_1)$

$V_1(b|z_1)$

Increasing the Time Horizon

- Assume the robot can make an observation before deciding on an action.
- Suppose the robot perceives z_1 for which $p(z_1 \mid x_1)=0.7$ and $p(z_1 \mid x_2)=0.3$.
- Given the observation z_1 we update the belief using Bayes rule.
- Thus $V_1(b \mid z_1)$ is given by

$$
\begin{aligned}
V_1(b \mid z_1) &= \max \left\{ \begin{array}{ll} -100 \cdot \frac{0.7\, p_1}{p(z_1)} & +100 \cdot \frac{0.3\,(1-p_1)}{p(z_1)} \\ \\ 100 \cdot \frac{0.7\, p_1}{p(z_1)} & -50 \cdot \frac{0.3\,(1-p_1)}{p(z_1)} \end{array} \right\} \\
&= \frac{1}{p(z_1)} \max \left\{ \begin{array}{ll} -70\, p_1 & +30\,(1-p_1) \\ 70\, p_1 & -15\,(1-p_1) \end{array} \right\}
\end{aligned}
$$

Expected Value after Measuring

- Since we do not know in advance what the next measurement will be, we have to compute the expected belief

$$\bar{V}_1(b) = E_z[V_1(b \mid z)] = \sum_{i=1}^{2} p(z_i) V_1(b \mid z_i)$$

$$= \sum_{i=1}^{2} p(z_i) V_1\left(\frac{p(z_i \mid x_1) p_1}{p(z_i)}\right)$$

$$= \sum_{i=1}^{2} V_1\left(p(z_i \mid x_1) p_1\right)$$

Expected Value after Measuring

- Since we do not know in advance what the next measurement will be, we have to compute the expected belief

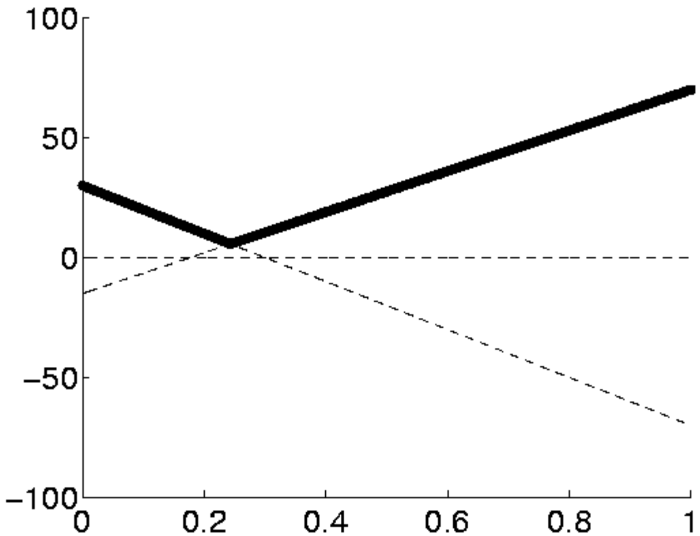
$$
\begin{aligned}
\bar{V}_1(b) &= E_z[V_1(b \mid z)] \\
&= \sum_{i=1}^{2} p(z_i)\, V_1(b \mid z_i) \\
&= \max \left\{ \begin{array}{rr} -70\, p_1 & +30\,(1-p_1) \\ 70\, p_1 & -15\,(1-p_1) \end{array} \right\} \\
&\quad + \max \left\{ \begin{array}{rr} -30\, p_1 & +70\,(1-p_1) \\ 30\, p_1 & -35\,(1-p_1) \end{array} \right\}
\end{aligned}
$$

Resulting Value Function

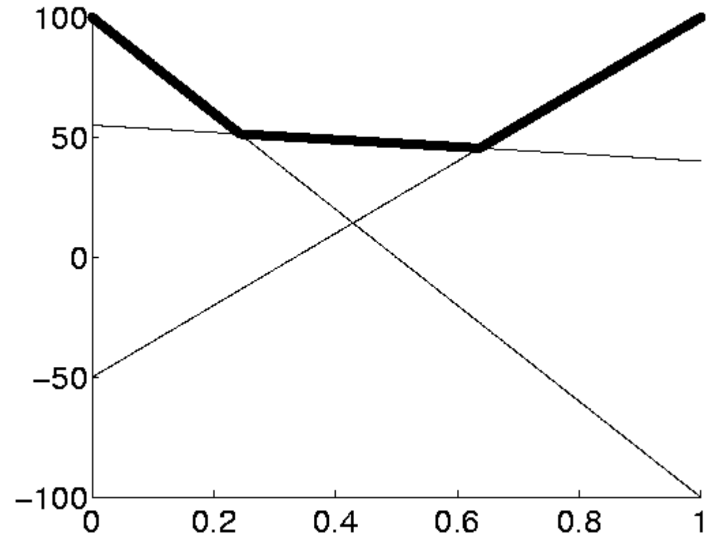
- The four possible combinations yield the following function which then can be simplified and pruned.

$$
\begin{aligned}
\bar{V}_1(b) &= \max \left\{ \begin{array}{llll} -70\, p_1 & +30\, (1-p_1) & -30\, p_1 & +70\, (1-p_1) \\ -70\, p_1 & +30\, (1-p_1) & +30\, p_1 & -35\, (1-p_1) \\ +70\, p_1 & -15\, (1-p_1) & -30\, p_1 & +70\, (1-p_1) \\ +70\, p_1 & -15\, (1-p_1) & +30\, p_1 & -35\, (1-p_1) \end{array} \right\} \\
&= \max \left\{ \begin{array}{ll} -100\, p_1 & +100\, (1-p_1) \\ +40\, p_1 & +55\, (1-p_1) \\ +100\, p_1 & -50\, (1-p_1) \end{array} \right\}
\end{aligned}
$$

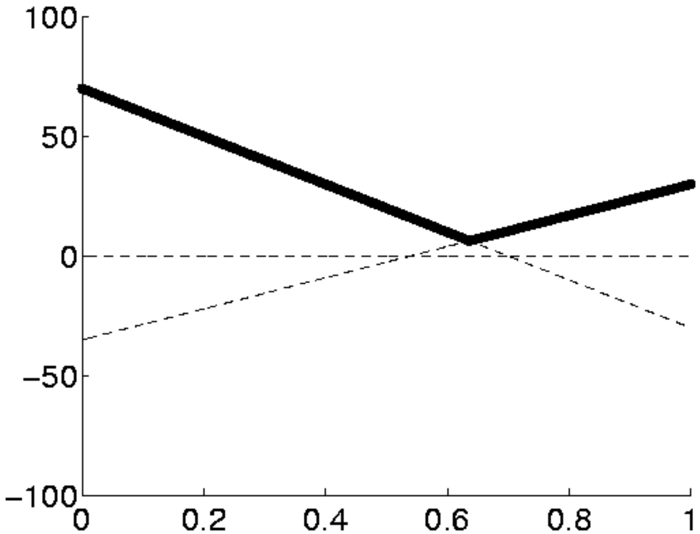
Value Function



$p(z_1)\ V_1(b|z_1)$



\bar{V}$_1$(b)



$p(z_2)\ V_2(b|z_2)$

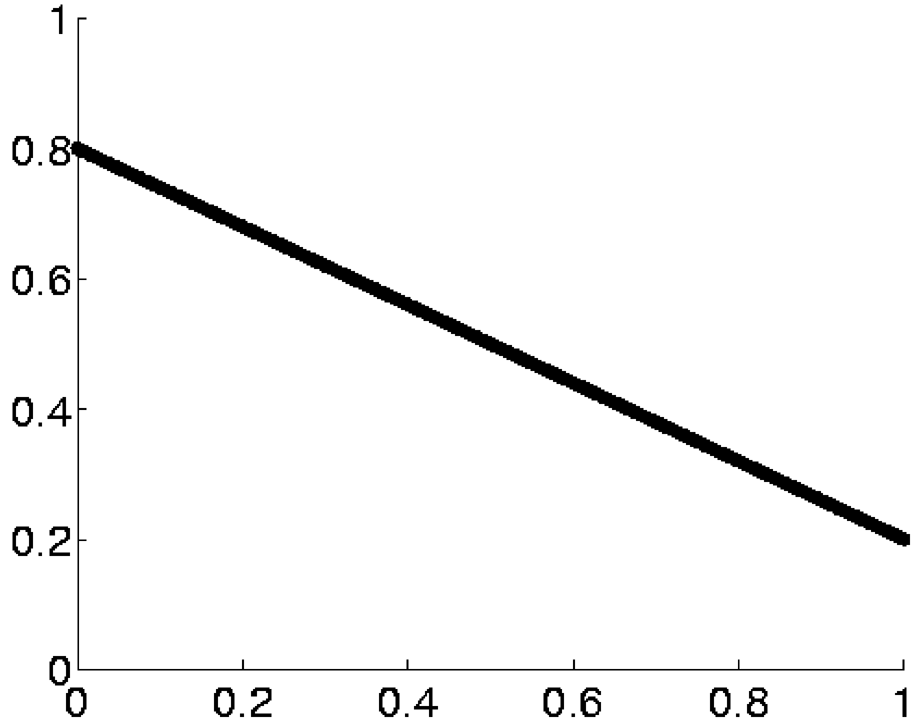
State Transitions (Prediction)

- When the agent selects u_3 its state potentially changes.
- When cor
  function,
  potential
  acc $p'_1 =$

  $=$

  $=$

  $= 0.8 - 0.6 p_1$



Resulting Value Function after executing u_3

- Taking the state transitions into account, we finally obtain.

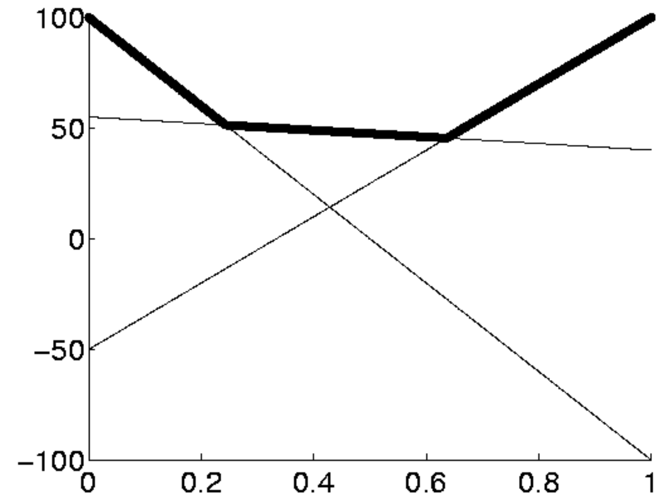
$$\bar{V}_1(b) = \max \left\{ \begin{array}{llll} -70\, p_1 & +30\,(1-p_1) & -30\, p_1 & +70\,(1-p_1) \\ -70\, p_1 & +30\,(1-p_1) & +30\, p_1 & -35\,(1-p_1) \\ +70\, p_1 & -15\,(1-p_1) & -30\, p_1 & +70\,(1-p_1) \\ +70\, p_1 & -15\,(1-p_1) & +30\, p_1 & -35\,(1-p_1) \end{array} \right\}$$

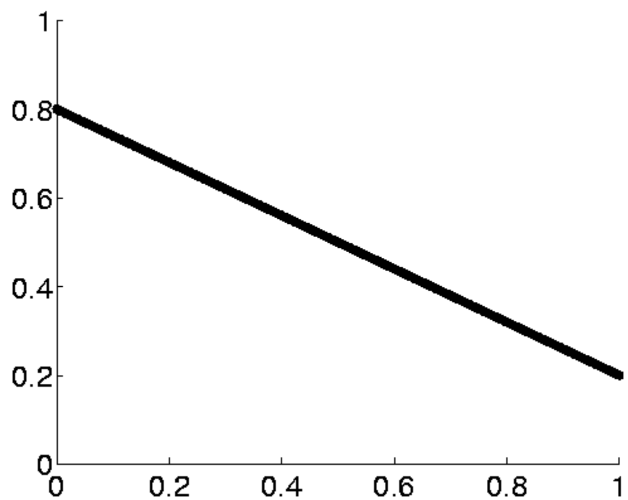
$$= \max \left\{ \begin{array}{ll} -100\, p_1 & +100\,(1-p_1) \\ +40\, p_1 & +55\,(1-p_1) \\ +100\, p_1 & -50\,(1-p_1) \end{array} \right\}$$

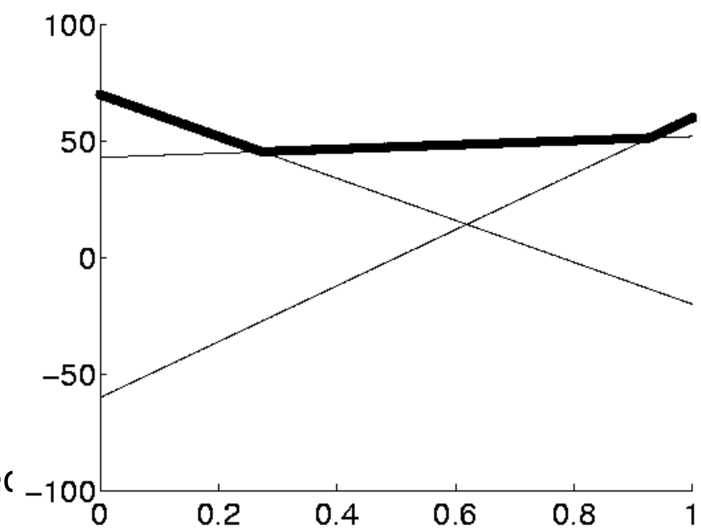
$$\bar{V}_1(b \mid u_3) = \max \left\{ \begin{array}{ll} 60\, p_1 & -60\,(1-p_1) \\ 52\, p_1 & +43\,(1-p_1) \\ -20\, p_1 & +70\,(1-p_1) \end{array} \right\}$$

Value Function after executing *u_3*

$\bar{V}_1(b)$







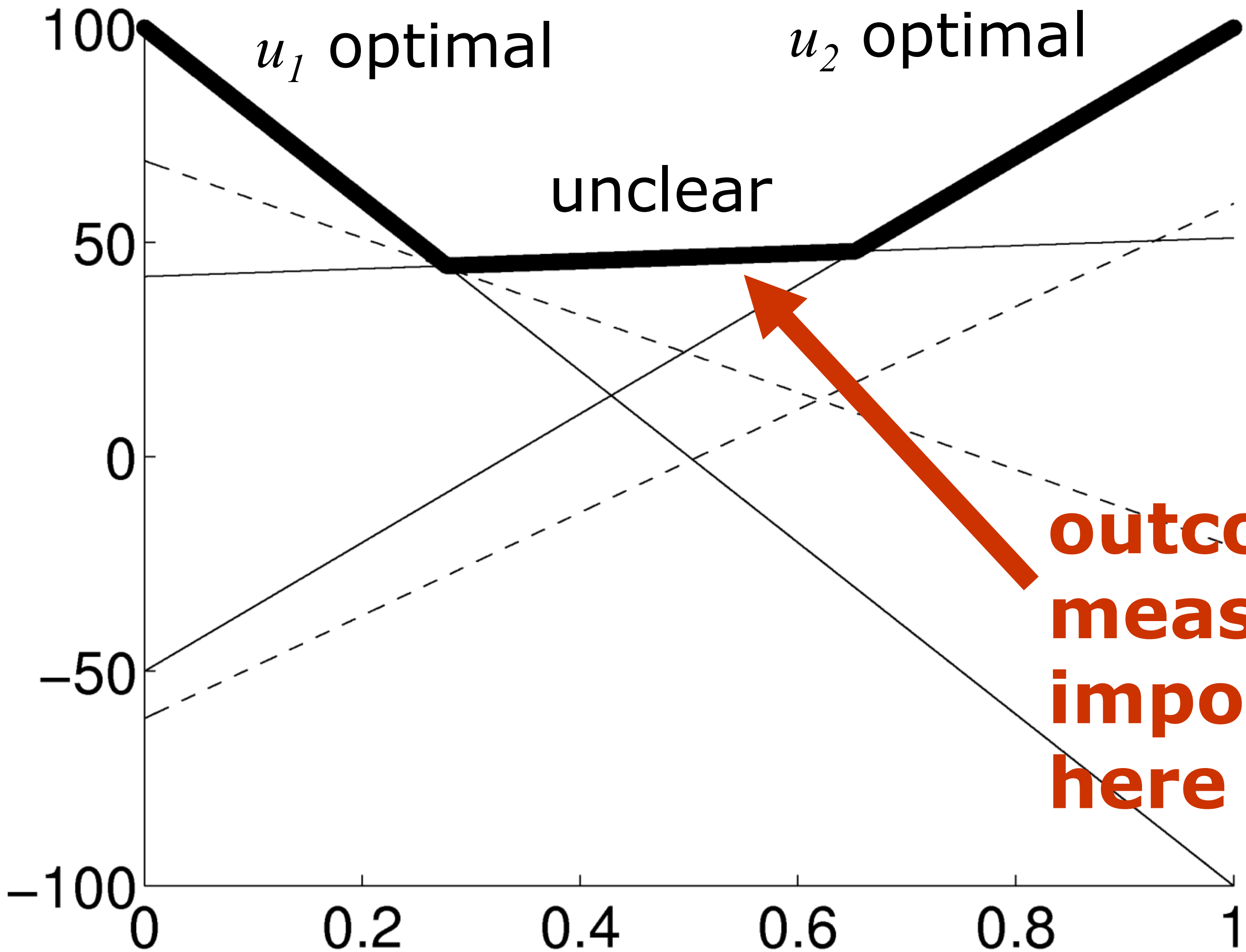
$\bar{V}_1(b|u_3)$

Value Function for T=2

- Taking into account that the agent can either directly perform u_1 or u_2 or first u_3 and then u_1 or u_2, we obtain (after pruning)

$$\bar{V}_2(b) \;=\; \max \left\{ \begin{array}{rr} -100\, p_1 & +100\,(1-p_1) \\ 100\, p_1 & -50\,(1-p_1) \\ 51\, p_1 & +42\,(1-p_1) \end{array} \right\}$$

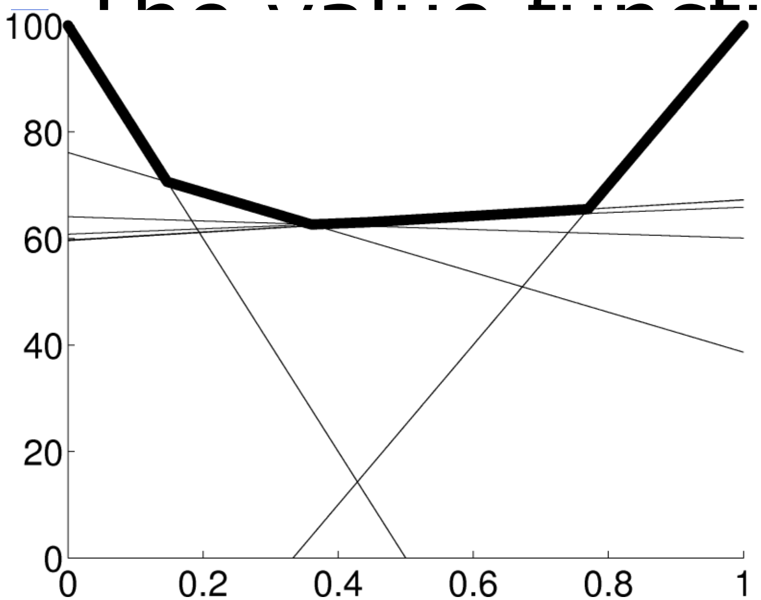
Graphical Representation of $V_2(b)$

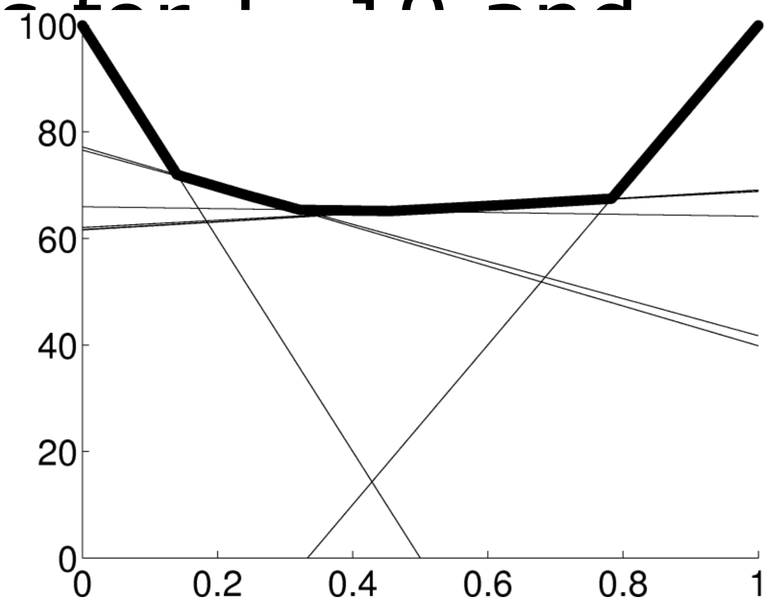


outcome of measuring is important here

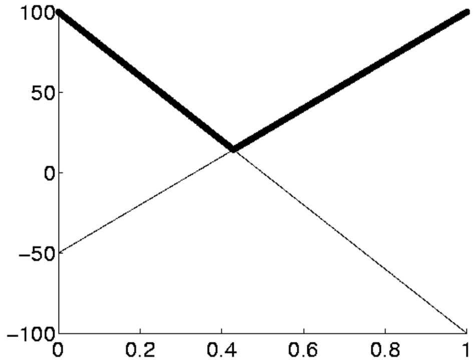
Deep Horizons and Pruning

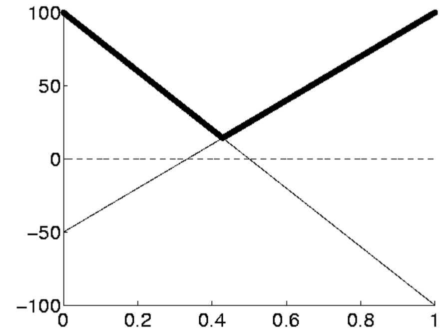
- We have now completed a full backup in belief space.
- This process can be applied recursively.
- The value functions for T=10 and

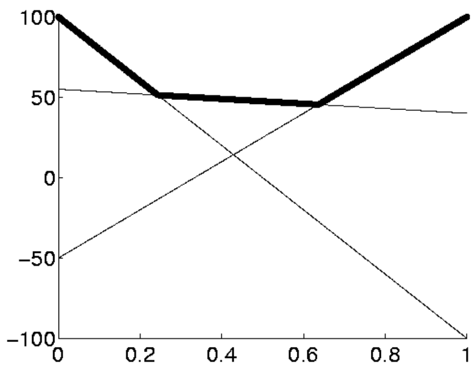


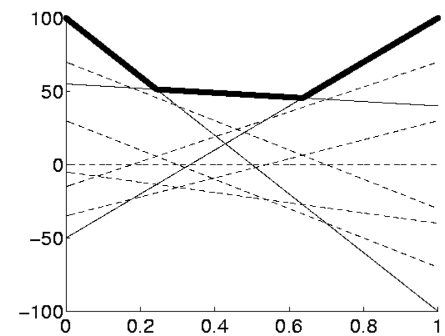


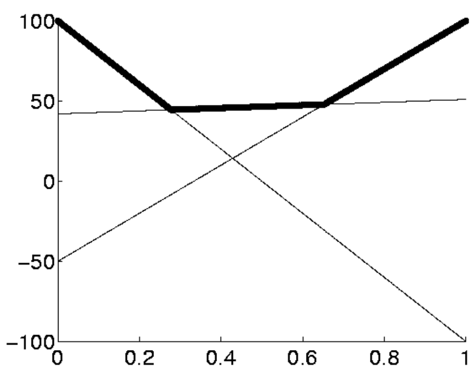
Deep Horizons and Pruning

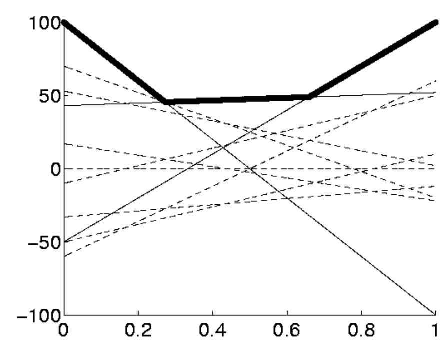












Algorithm POMDP(T):

$\Upsilon = (0, \ldots, 0)$
for $\tau = 1$ *to* T *do*
  $\Upsilon' = \emptyset$
  for all $(u'; v_1^k, \ldots, v_N^k)$ *in* Υ *do*
    for all control actions u *do*
      for all measurements z *do*
        for $j = 1$ *to* N *do*

$$v_{j,u,z}^k = \sum_{i=1}^{N} v_i^k \, p(z \mid x_i) \, p(x_i \mid u, x_j)$$

        endfor
      endfor
    endfor
  endfor
  for all control actions u *do*
    for all $k(1), \ldots, k(M) = (1, \ldots, 1)$ *to* $(|\Upsilon|, \ldots, |\Upsilon|)$ *do*
      for $i = 1$ *to* N *do*

$$v_i' = \gamma \left[r(x_i, u) + \sum_z v_{u,z,i}^{k(z)} \right]$$

      endfor
      add $(u; v_1', \ldots, v_N')$ *to* Υ'
    endfor
  endfor
  optional: prune Υ'
  $\Upsilon = \Upsilon'$
endfor
return Υ

Why Pruning is Essential

- Each **update introduces additional linear components** to V.
- Each **measurement squares the number of linear components**.
- Thus, an unpruned value function for T=20 includes more than $10^{547,864}$ linear functions.
- At T=30 we have $10^{561,012,337}$ linear functions.
- The pruned value functions at T=20, in comparison, contains only 12 linear components.
- The combinatorial explosion of linear components in the value function are the major reason why **POMDPs are impractical for most applications**.

POMDP Summary

- POMDPs compute the optimal action in partially observable, stochastic domains.
- For finite horizon problems, the resulting value functions are piecewise linear and convex.
- In each iteration the number of linear constraints grows exponentially.
- POMDPs so far have only been applied successfully to very small state spaces with small numbers of possible observations and actions.

POMDP Approximations

- Point-based value iteration
- QMDPs
- AMDPs

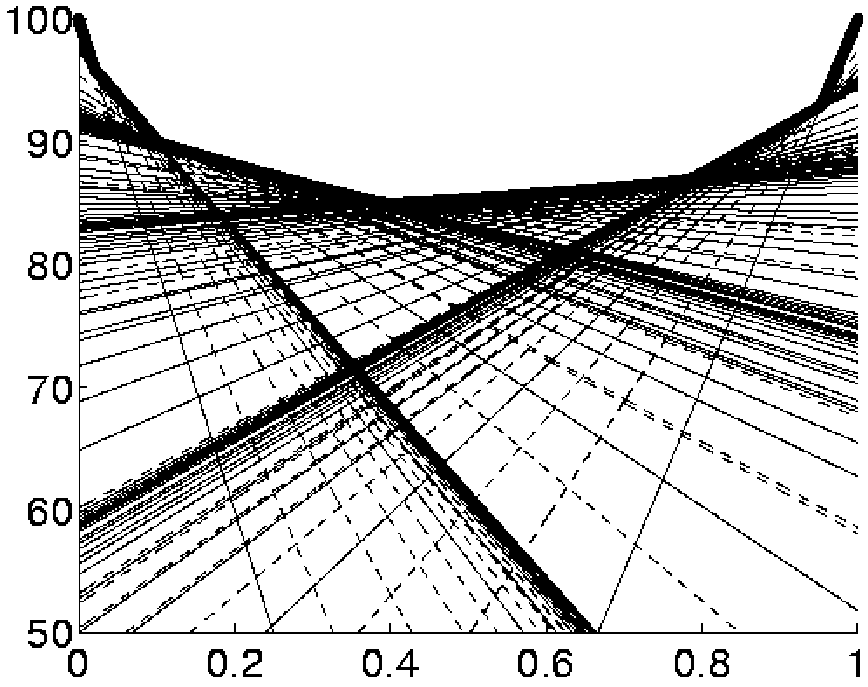
Point-based Value Iteration

- Maintains a set of example beliefs

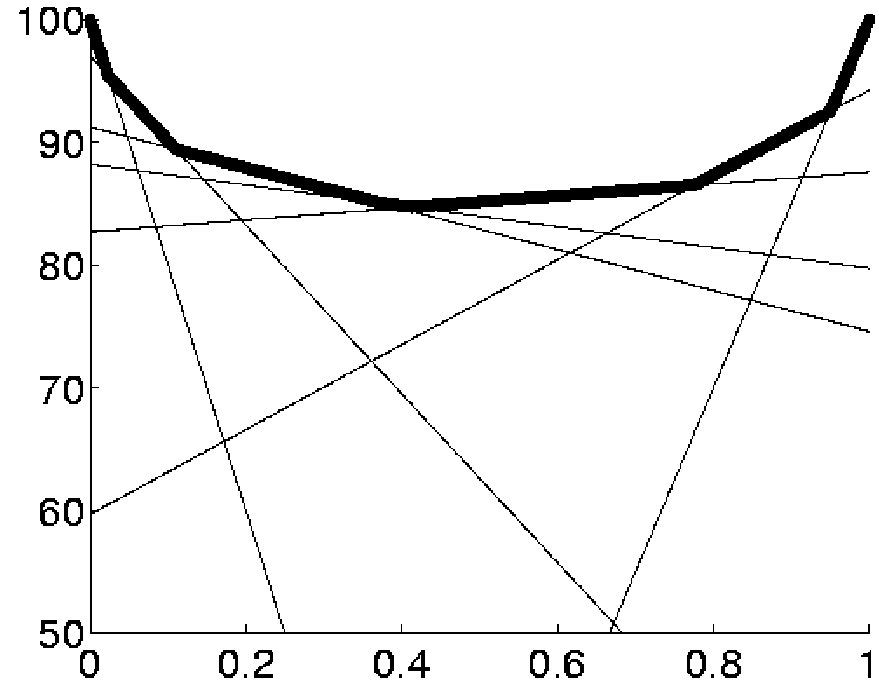
- Only considers constraints that maximize value function for at least one of the examples

Point-based Value Iteration

Value functions for T=30

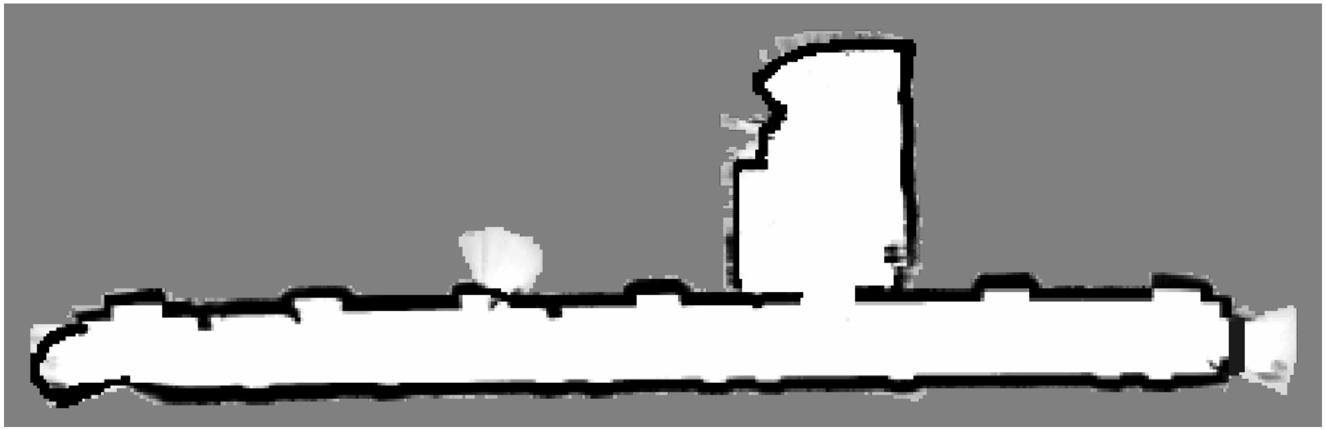


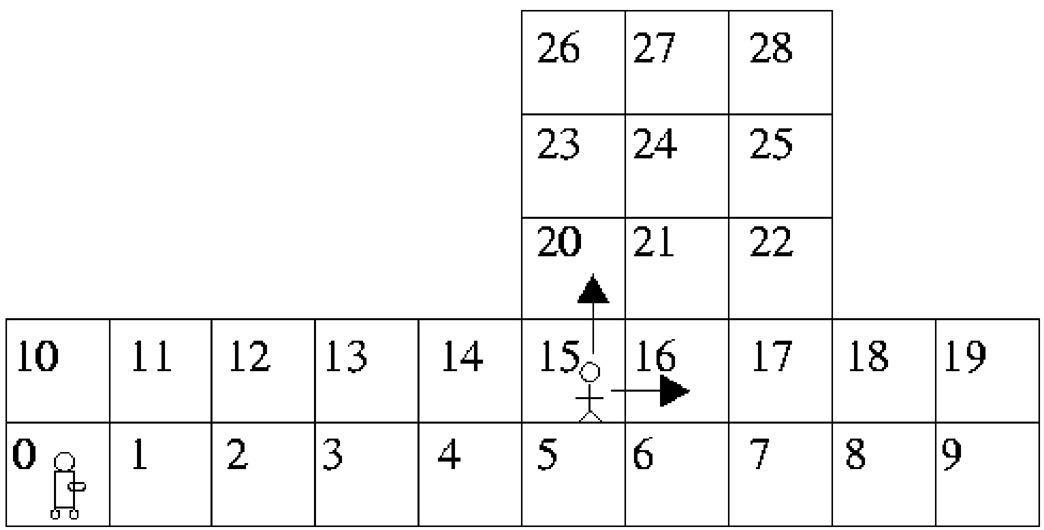
Exact value function



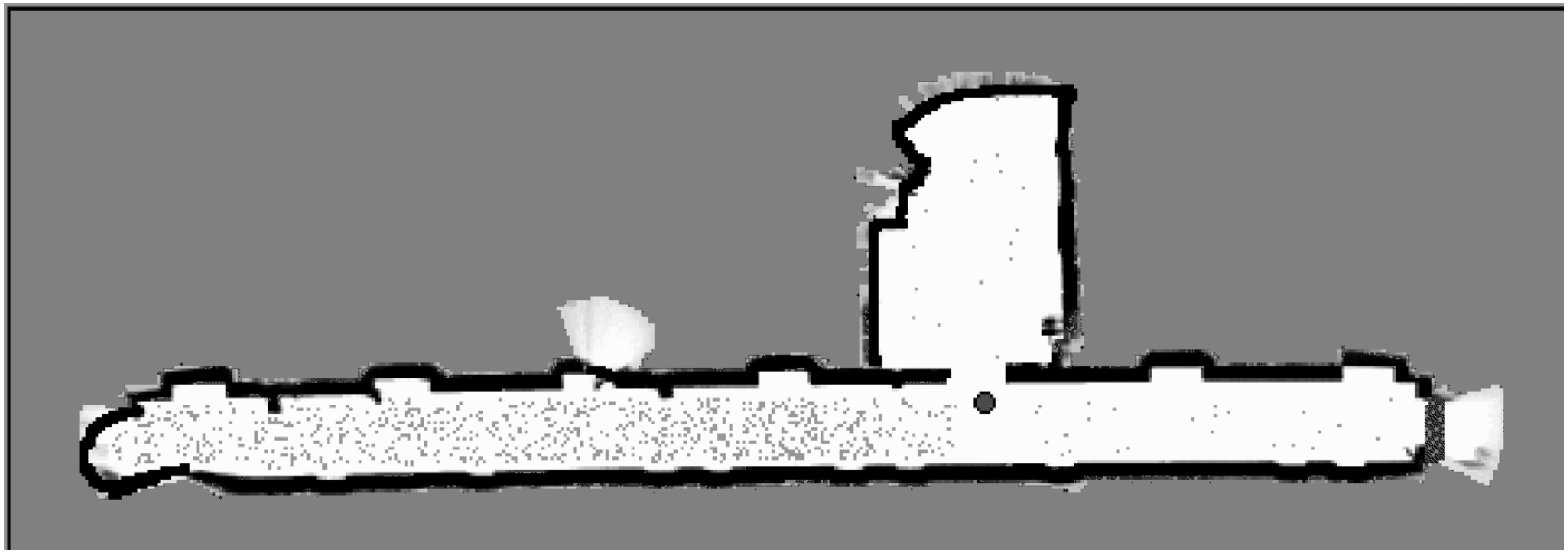
PBVI

Example Application





Example Application



QMDPs

- QMDPs only consider state uncertainty in the first step

- After that, the world becomes fully observable.

Algorithm QMDP($b = (p_1, \dots, p_N)$)**:**

$\hat{V}$ = MDP_discrete_value_iteration() // *see page 504*

for all control actions u *do*

$$Q(x_i, u) = r(x_i, u) + \sum_{j=1}^{N} \hat{V}(x_j)\, p(x_j \mid u, x_i)$$

endfor

return $\operatorname*{arg\,max}_u \sum_{i=1}^{N} p_i\, Q(x_i, u)$

Augmented MDPs

- Augmentation adds uncertainty component to state space, e.g.

$$\bar{b} = \begin{pmatrix} \arg\max_x b(x) \\ H_b(x) \end{pmatrix}, \qquad H_b(x) = -\int b(x) \log b(x) dx$$

- Planning is performed by MDP in augmented state space
- Transition, observation and payoff models have to be learned