# Reasoning and Decision-Making under Uncertainty

10. Session:

Solving MPDs and POMDPs

Prof. Dr.-Ing. Stefan Kopp
Center of Excellence „Cognitive Interaction Technology“
AG Sociable Agents

CITEC

Sociable Agents

# Markov Decision Problem

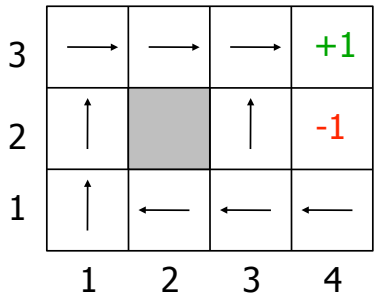
Can solve problems of the sort: given my current information, what actions in which order achieve a MEU result?

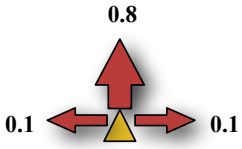
Definition of decision problems in *fully observable* environments:

1. Initial state $s_0$
2. Transition model T(s,a,s‘) = **P**(s‘|a,s)
3. Reward function: R(s) or R(s,a,s‘), additive

*Question*: How to compute the optimal action policy that ensures maximum expected utility?

# Policy (Reactive/Closed-Loop Strategy)

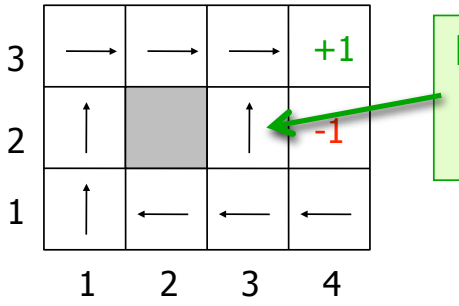



A **policy Π** is a mapping from *states* to *actions*, i.e. recommends an action given a state *s*

A complete policy is defined for any possible world state.

# Optimal Policy




Note that [3,2] is a "dangerous" state that the optimal policy tries to avoid

The optimal policy Π* is the one that *always* yields a history (ending at a terminal state) with maximum expected utility

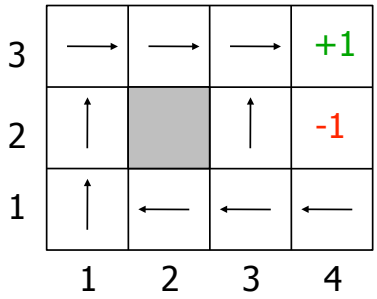
An optimal policy balances risks and rewards, depending on the value *R(s)* for non-terminal states

*Key question:* How to find an optimal policy?

# Example

Robot navigation example:

- $\mathbf{R}(s_n) = +1$ if $s_n = [4,3]$
- $\mathbf{R}(s_n) = -1$ if $s_n = [4,2]$
- $\mathbf{R}(s_i) = -0.04$ if $i = 0, \ldots, n-1$




History $h = (s_0, s_1, \ldots, s_n)$

Utility of history: $\mathbf{U}(s_0, s_1, \ldots, s_n) = \Sigma\, \mathbf{R}(s_i)$

*But:* Cannot simply search all possibly open-ended histories

# Calculating the optimal policy

*Idea*: Calculate the utility of all states first and use this information to select the optimal action when in a state, simply by using the MEU principle to follow the optimal policy:

→ Optimal policy: $\Pi^*(s_i) = \text{argmax}_a \Sigma_{s'} \mathbf{T}(s_i, a, s')\mathbf{U}(s')$

- one-step look-ahead using U(s)

*Q:* How to determine the utilities of all states (terminal and intermediate)?

# Calculating the optimal policy

*In general:* Utility of a state = expected utility of the (possibly open-ended) history that might follow under some policy Π

$$U^{\Pi}(s_i) = E[\sum_{t=0..\infty} \gamma^t R(s_t) \mid \Pi, s_0 = s_i]$$

- $R(s)$ = short-term reward for *being in* s
- $U(s)$ = long-term reward from s *onwards*

This history can be unbounded. We thus use discounting to make infinite horizon problems mathematically tractable: $0 \leq \gamma \leq 1$

Utility of an infinite history becomes finite:

$$\lim_{T\to\infty} \sum_i \gamma^i R(s_i) \leq \lim_{T\to\infty} R^{max} \sum_i \gamma^i = R^{max} (\gamma / 1-\gamma)$$

➜ Utility of states: $U(s) = R(s) + \gamma \max_a \sum_{s'} T(s, a, s') U(s')$

# Calculating the optimal policy

*Observation:* There is a direct relationship between utility of a state and the utility of its neighbor states:

- Utility of a state = reward for the current state plus expected discounted utility of next state, assuming the agent acts optimally

**Bellman equation:** $U(s_i) = R(s_i) + \gamma \max_a \sum_{s'} T(s_i, a, s') U(s')$

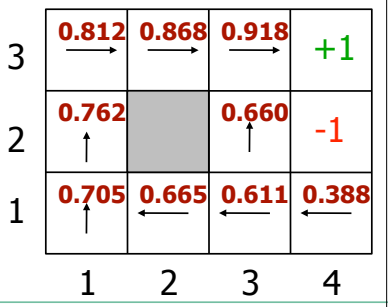
*Example*: In state (1,1) action „Up" is best, hence

$\mathbf{U(1,1)}$= -0.04 + γ(0.8*U(1,2)+0.1*U(2,1)+0.1*$\mathbf{U(1,1)}$)

=-0.04+0.8*0.762+0.1*0.655+ 0.1*0.705

= 0.7056

➜ recursive update as actions may have no effect



# Value Iteration

For *n* states, there are *n* equations $U(s_i)$ with *n* unknowns, but non-linear because of „max"-operator → iterative approach:

- initialize the utilities, calc r.h.s. of the equation, and update l.h.s. and all other utilities, until an equilibrium is reached.

*Algorithm*:

- Initialization: of the utility of each *non-terminal* state s
- Iteration step: for t = 0, 1, 2, … do

$$U_{t+1}(s) \leftarrow R(s) + \gamma \max_a \sum_{s'} T(s,a,s') U_t(s') \quad \text{(Bellman update)}$$

Converges to a unique solution of the Bellman eq., the correct utilities for the optimal policy

# Value Iteration

**function** VALUE-ITERATION($mdp, \epsilon$) **returns** a utility function
  **inputs**: $mdp$, an MDP with states $S$, transition model $T$, reward function $R$, discount $\gamma$
    $\epsilon$, the maximum error allowed in the utility of any state
  **local variables**: $U$, $U'$, vectors of utilities for states in $S$, initially zero
    $\delta$, the maximum change in the utility of any state in an iteration

  **repeat**
    $U \leftarrow U'$; $\delta \leftarrow 0$
    **for each** state $s$ **in** $S$ **do**
      $U'[s] \leftarrow R[s] + \gamma \max_a \sum_{s'} T(s,a,s')\, U[s']$
      **if** $|U'[s] - U[s]| > \delta$ **then** $\delta \leftarrow |U'[s] - U[s]|$
  **until** $\delta < \epsilon(1-\gamma)/\gamma$
  **return** $U$

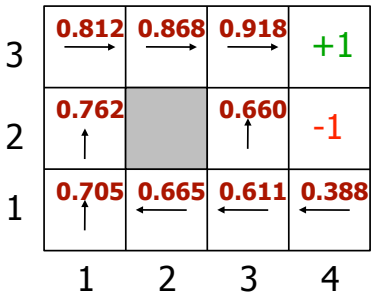
**Figure 17.4** The value iteration algorithm for calculating utilities of states. The termination condition is from Equation (17.8).

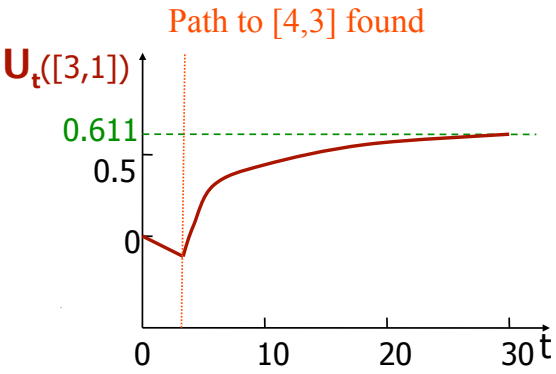
# Value Iteration

Note the importance of terminal states and connectivity of the state-transition graph

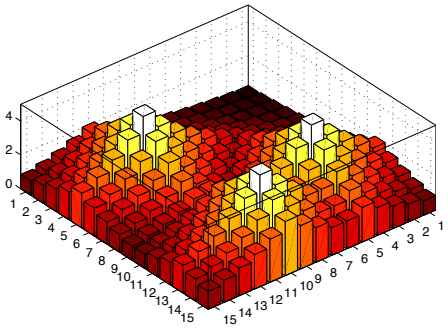
- Initialize the utility of each non-terminal state s to 0
- For t = 0, 1, 2, … do:

$$U_{t+1}(s) \leftarrow R(s) + \max_a \sum_{s'} T(s,a,s') U_t(s')$$




# Value Iteration



Value Iteration on a set of $225$ states, corresponding to a $15 \times 15$ two dimensional grid. Deterministic transitions to neighbours on the grid, $\{\text{stay}, \text{left}, \text{right}, \text{up}, \text{down}\}$. There are three goal states, each with utility 1 – all other states have utility 0. Plotted is the value $v(s)$ for $\gamma = 0.9$ after 30 updates of Value Iteration, where the states index a point on the $x - y$ grid. The optimal decision for any state on the grid is to go to the neighbouring state with highest value.

# Value Iteration with discounting

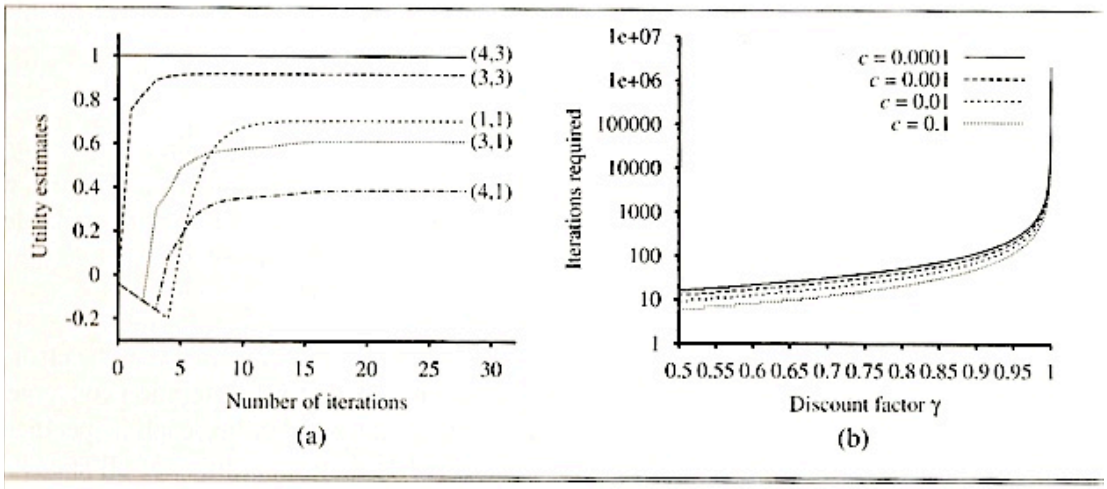



**Figure 17.5** (a) Graph showing the evolution of the utilities of selected states using value iteration. (b) The number of value iterations $k$ required to guarantee an error of at most $\epsilon = c \cdot R_{\max}$, for different values of $c$, as a function of the discount factor $\gamma$.



# Policy Iteration

*Idea*: iteration works even with inaccurate utilities, when one action is clearly better than all others. Thus we can alternate two steps:

1. policy evaluation: calculate U(s) of each state under a policy Π
2. policy improvement: set new Π‘ using one-step look-ahead from U(s)

*Algorithm*:

- Pick a policy Π at random
- Repeat:
  - 1. Compute the utility of each state for Π
    $U_{t+1}(s) \leftarrow R(s) + \gamma \sum_{s'} T(s,\Pi(s), s') U_t(s')$
  - 2. Compute a new policy Π' given these utilities
    $\Pi'(s) = \arg\max_a \sum_{s'} T(s,a,s') U(s')$
  - If Π' = Π then return Π

Simpler, linear equations because no „max“ step. Can be solved in $O(n^3)$ for $n$ states

Fixpoint of Bellman eq., optimal policy

# Policy evaluation

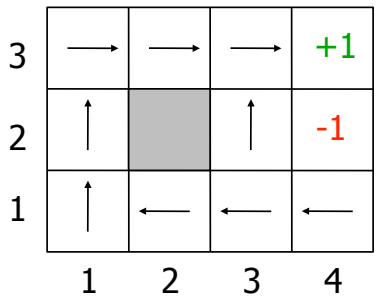
*Example at time t:*

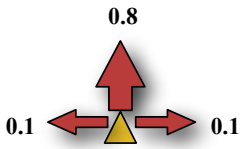
→ $\Pi_t(1,1)$=Up, $\Pi_t(1,2)$=Up, ...

→ Simplified Bellman eq.:

- $U_t(1,1) = 0.8U_t(1,2) + 0.1U_t(2,1) + 0.1U_t(1,1)$
- $U_t(1,2) = 0.8U_t(1,3) + 0.1U_t(1,2)$
- .....

*n linear* equations with *n* unknowns, solvable in $O(n^3)$






# Policy Iteration

*Remarks*:

- When no improvement, U is fixpoint and the policy it optimal
- There are finitely many policies and each step yields a better policy, so policy iteration must terminate
- Policy evaluation (step 1.) uses an iterative update to estimate the current U using a simplified Bellman update
- Possible to update U or policy only for a subset of states (instead of all), using either kind of updating (policy improvement, simplified value iteration): asynchronous policy iteration
  - could update only those states that are likely reached
  - more efficient, can be guaranteed to converge

## *Matlab example*

D. Barber (2012). Bayesian Reasoning and Machine Learning, Cambridge Univ. Press.

→ BRML toolkit



# Partially Observable Markov Decision Problems (POMDP)

MDPs: fully observable environments and Markovian transition models

„POMDP" account in addition for partially observable environments: Which state is the agent in? Utility of s? Optimal action?

A POMDP is given by

1. Initial state $s_0$
2. Transition model T(s,a,s') = **P**(s'|a,s)
3. Reward function R(s) or R(s,a,s'), additive
4. Observation (measurement) model: O(s,o) = **P**(o|s)
   sensing operation in state s, returns multiple observations o, with a probability distribution

# POMDPs

*Idea in a nutshell:*
Belief state b(s) = prob. distribution over all possible states
- *Example*: in 4x3-world = point in 11-dim continuous space

Define the agent's policy over its belief state: $\Pi^*(b)$
That is, actions depend on *beliefs*, not the state the agent is in!

POMDP decision cycle:
1. Given current belief state b, execute $a=\Pi^*(b)$
2. Get new observations (measurements) o
3. Bayesian update of belief states:
   $b'(s') = \alpha\, O(s',o) \sum_s T(s,a,s')b(s)$

**How to determine $\Pi^*(b)$ ?**



# POMDPs

Often, slightly different notation. In the following:
- generic value function V instead of utility U
- continuous belief domain (i.e. use integral instead of sum), with single belief *b*
- replace *a (action)* by *u*, and *s (state)* by *x*

➜ POMDPs compute a value function over belief spaces:

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

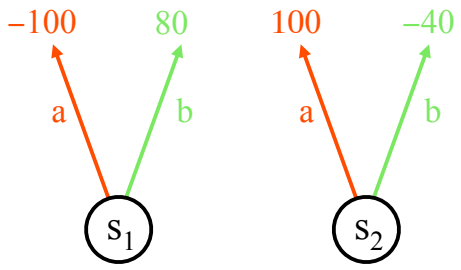
# POMDPs

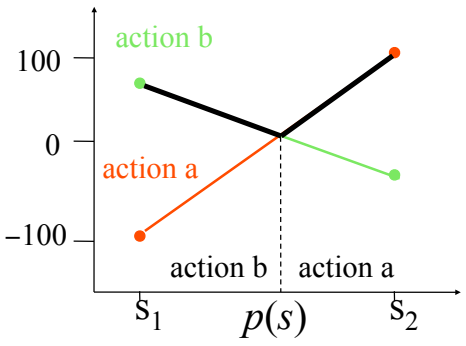
*Problems*:

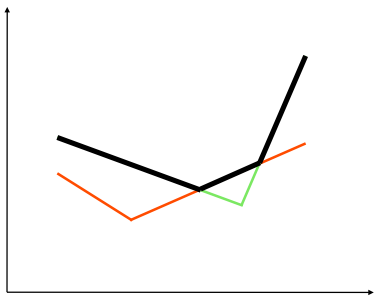
- Each belief is a probability distribution, thus, each value in a POMDP is a function of an entire probability distribution. This is problematic, since probability distributions are continuous.
- Additionally, huge complexity of belief spaces.
- For finite worlds with finite state, action, and measurement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



# POMDPs

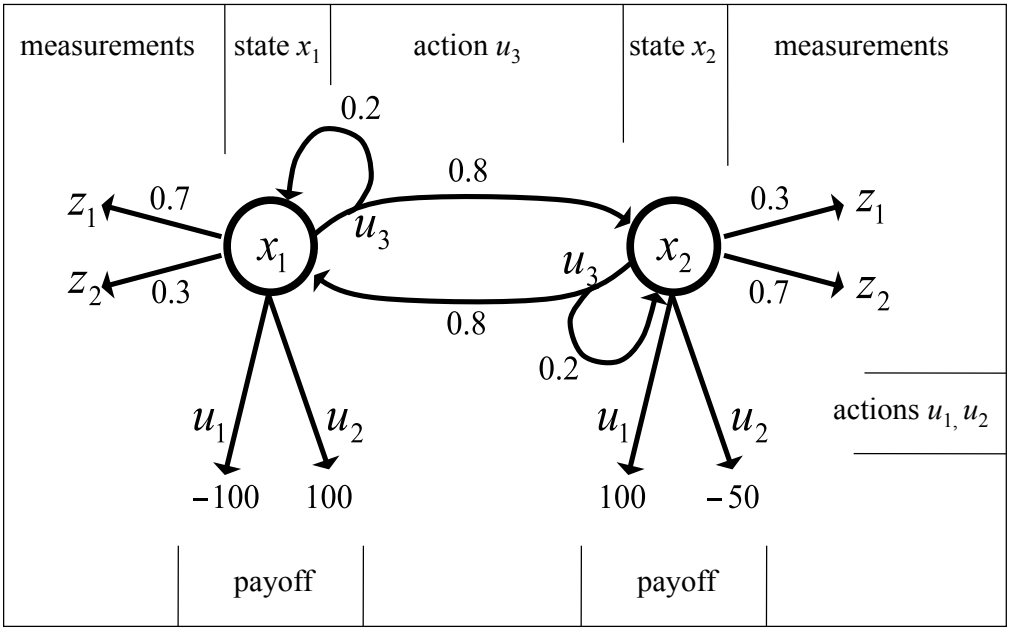


[Sondik 72, Littman, Kaelbling, Cassandra '97]

# Example






# Example

*Parameters*: The actions u1 and u2 are terminal actions. The action u3 is a sensing action that potentially leads to a state transition. The horizon is finite and γ=1.

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

$$
\begin{aligned}
p(x_1'|x_1, u_3) &= 0.2 & \qquad p(x_2'|x_1, u_3) &= 0.8 \\
p(x_1'|x_2, u_3) &= 0.8 & \qquad p(z_2'|x_2, u_3) &= 0.2
\end{aligned}
$$

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

# Payoffs in POMDPs

In MDPs, the payoff (or reward) depends on the state of the system.

But in POMDPs, the true system state is not exactly known. Thus, we compute the expected payoff from belief state, by integrating over all states:

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

*Example*:

If certain we are in x1 and execute u1, we receive a reward of -100

If certain we are in x2 and execute u1, the reward is +100. In between, linear combination weighted by probabilities

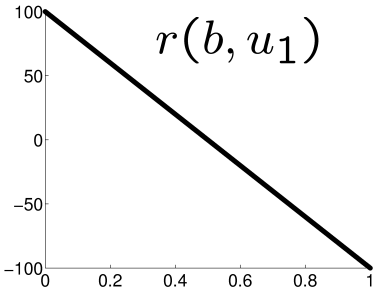
$$\begin{aligned} r(b,u_1) &= -100\, p_1 + 100\, p_2 \\ &= -100\, p_1 + 100\, (1-p_1) \end{aligned}$$

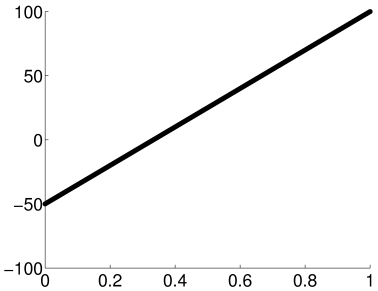
$$r(b,u_2) = 100\, p_1 - 50\, (1-p_1)$$

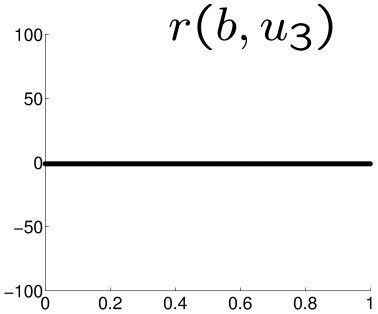
$$r(b,u_3) = -1$$

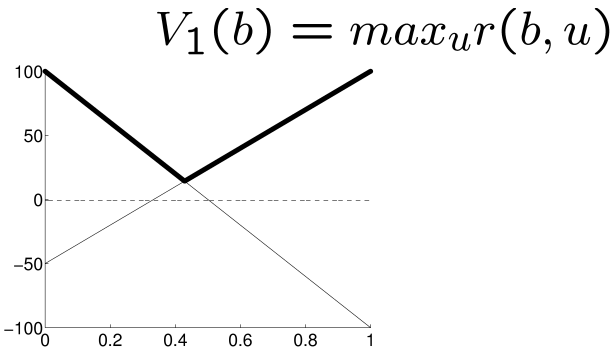


# *Example*: Payoff



# 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

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

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 $V_1(b)$.

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



# Increasing the Time Horizon

If we go over to a time horizon of T=2, the agent can also consider the sensing action u3.

Suppose we perceive z1 for which p(z1 | x1)=0.7 and p(z1| x2)=0.3.

Given the observation z1 we update the belief using Bayes rule.

Thus $V_1(b \mid z1)$ is given by

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

$$= \frac{1}{p(z_1)} \max \left\{ \begin{array}{cc} -70\, p_1 & +30\,(1-p_1) \\ 70\, p_1 & -15\,(1-p_1) \end{array} \right\}$$

# Expected Value after Measuring

Since we do not know in advance what the measurement will be ($z_1$, $z_2$), we have to compute the expected belief

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



# Resulting Value Function

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

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

# State Transitions (Prediction)

When the agent selects u3 its state potentially changes.

When computing the value function, we have to take these potential state changes into account.

$$\begin{aligned} p_1' &= E_x[p(x_1 \mid x, u_3)] \\ &= \sum_{i=1}^{2} p(x_1 \mid x_i, u_3) p_i \\ &= 0.2p_1 + 0.8(1 - p_1) \\ &= 0.8 - 0.6p_1 \end{aligned}$$



# Resulting Value Function

What's the resulting Value Function after executing u3 ?

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

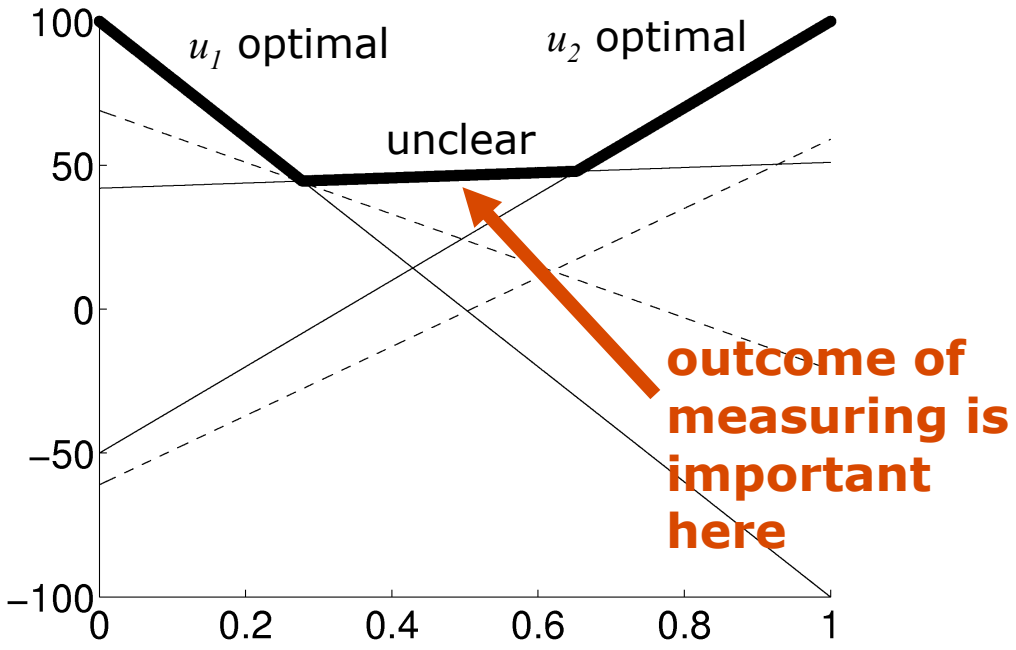
$$\bar{V}_1(b \mid u_3) = \max \left\{ \begin{array}{cc} 60\, p_1 & -60\,(1 - p_1) \\ 52\, p_1 & +43\,(1 - p_1) \\ -20\, p_1 & +70\,(1 - p_1) \end{array} \right\}$$

Value Function for T=2

- Taking into account that the agent can either directly perform u1 or u2, or first u3 and then u1 or u2, we obtain (after pruning)

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

# Resulting Value Function $V_2(b)$



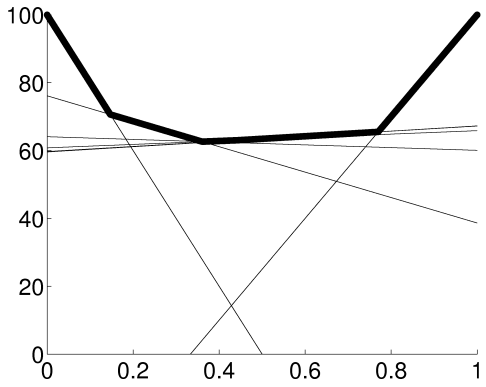


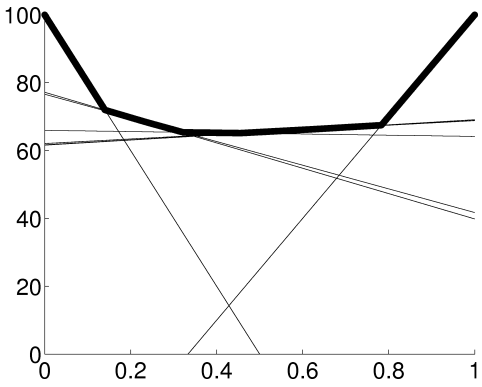

# Deep Horizons and Pruning

We have now completed a full backup in belief space.

This process can be applied recursively.

The value functions for T=10 and T=20 are



# 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 10547,864 linear functions.

At T=30 we have 10561,012,337 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 and are still an active research area. So far only applied successfully to very small state spaces with small numbers of possible observations and actions. In other words: Finding efficient state representations is a key to solve (PO)MDPs!