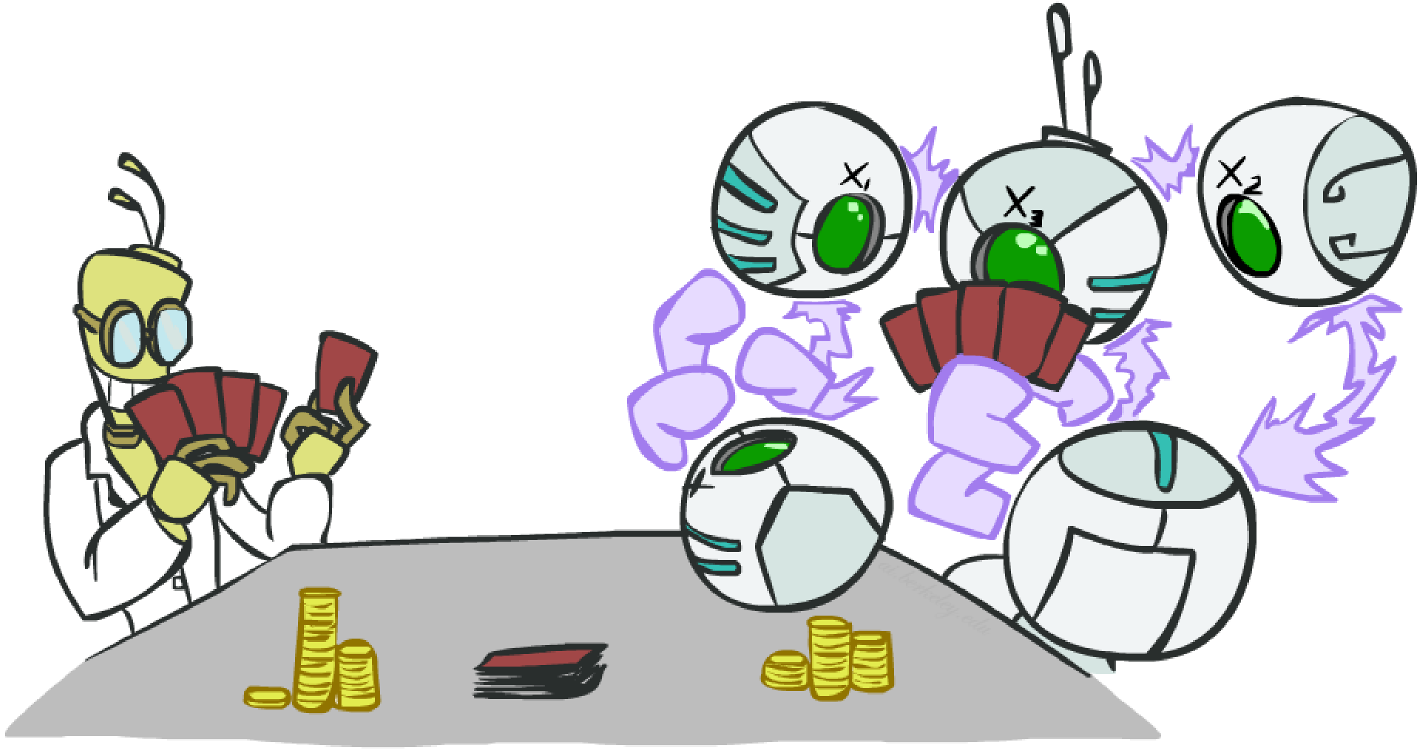
CS 6300: Artificial Intelligence

Partially Observable Markov Decision Processes



Instructor: Daniel Brown

University of Utah

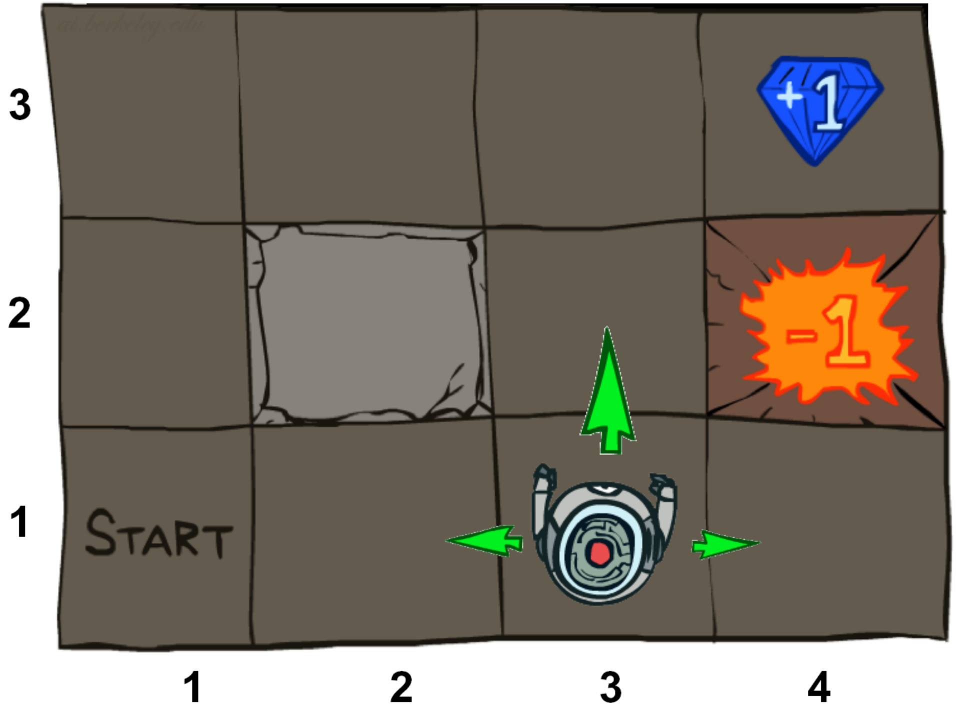
[Based on slides from Intro to AI at UC Berkeley and Geoff Hollinger at CMU]

Types of Markov Models

	System state is fully observable	System state is partially observable
System is autonomous	Markov chain	Hidden Markov model (HMM)
System is controlled	Markov decision process (MDP)	Partially observable Markov decision process (POMDP)

Markov Decision Processes

- An MDP is defined by:
  - A set of states s ∈ S
  - A set of actions a ∈ A
  - A transition function T(s, a, s')
    - Probability that a from s leads to s', i.e., P(s'| s, a)
    - Also called the model or the dynamics
  - A reward function R(s, a, s')
    - Sometimes just R(s), R(s,a), or R(s')
  - A start state distribution
  - Maybe a terminal state

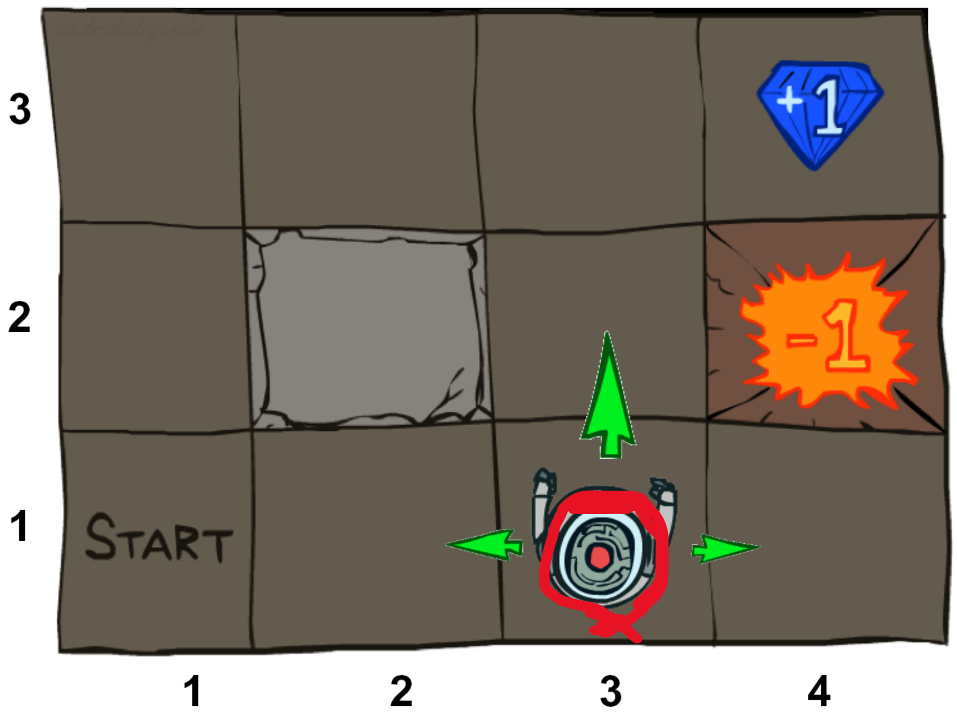


- MDPs are non-deterministic search problems
  - One way to solve them is with expectimax search
  - We'll have a new tool soon

[Demo – gridworld manual intro (L8D1)]

Partially Observable Markov Decision Processes

- A POMDP is defined by:
  - A set of states $s \in S$
  - A set of actions $a \in A$
  - A transition function T(s, a, s')
    - Probability that a from s leads to s', i.e., P(s'| s, a)
    - Also called the model or the dynamics
  - A reward function R(s, a, s')
    - Sometimes just R(s), R(s,a), or R(s')
  - A start state distribution
  - Maybe a terminal state
  - Observations Z
  - Emission Model O(s,z) = P(z|s)



- POMDPs are non-deterministic search problems **where you don't know where you are!**

Examples of POMPDs

MDP vs POMDP

- MDP
  - + Tractable to solve
  - + Relatively easy to specify
  - -Assumes perfect knowledge of state
- POMDP
  - +Models the real world
  - +Allows for information gathering actions
  - -Hugely intractable to solve optimally

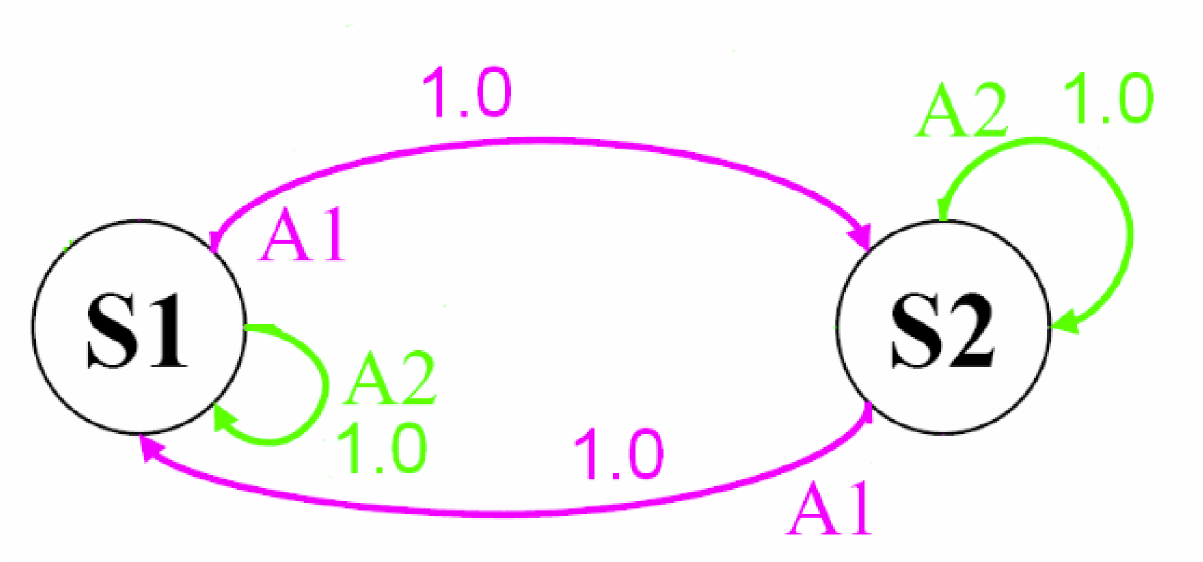
Quiz: Show POMDPs Generalize MDPs

- MDP: S,A,T,R
- POMDP: S,A,Z,T,R,O

- Z = S

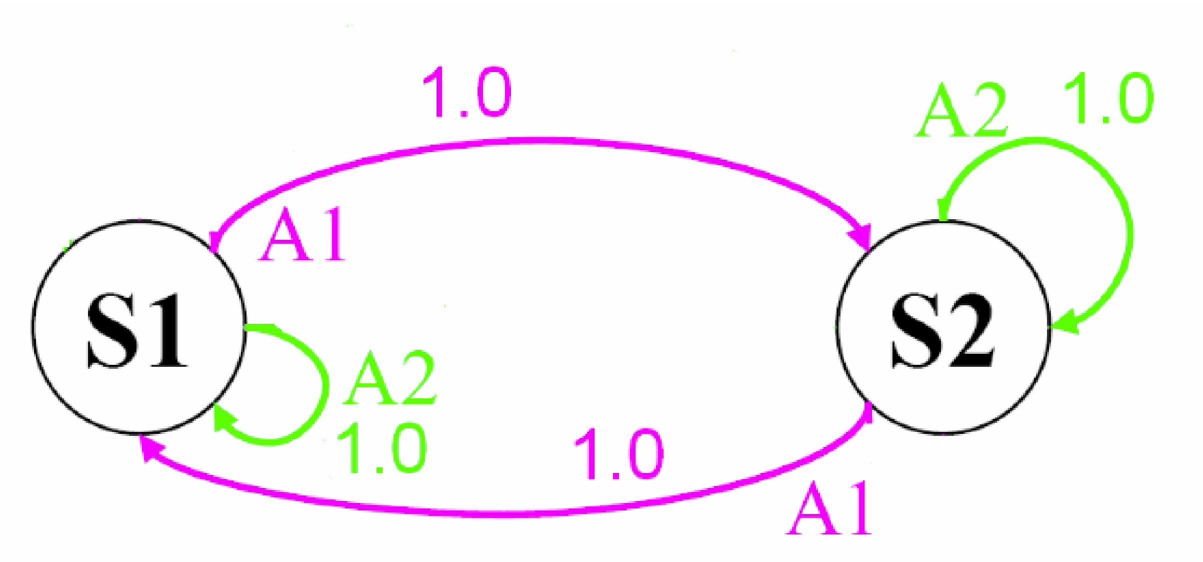
- O(s,z) = P(z|s) = 1 iff e == s

Simple Example



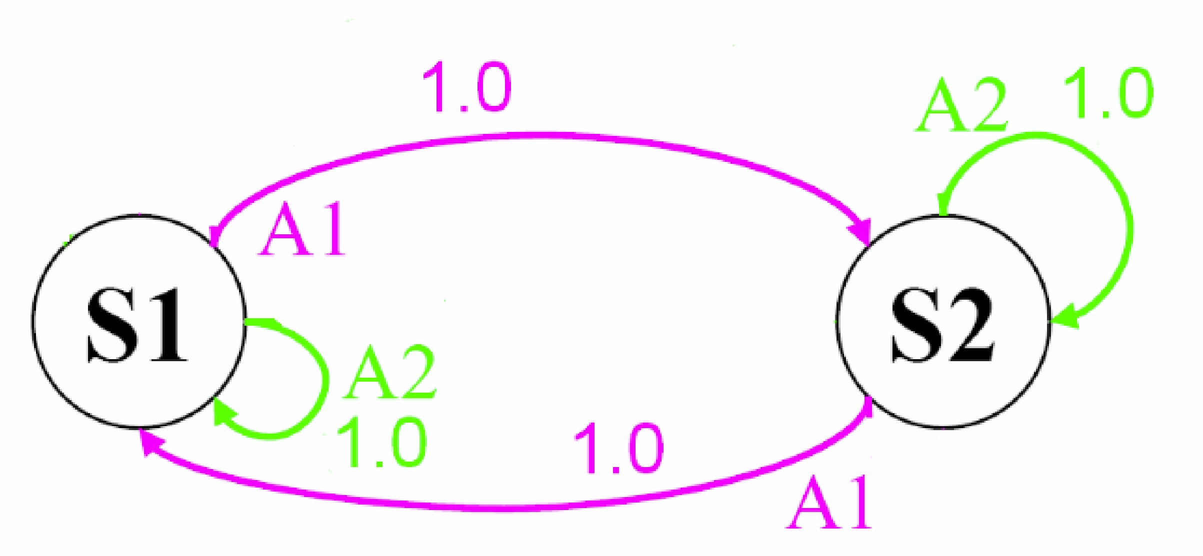
- Initial distribution: [0.9, 0.1]
- Discount factor: 0.5
- Reward: S1 = 10, S2 = 0
- Observations: S1 emits Z1 with prob 1.0, S2 emits Z2 with prob 1.0

Simple Example



- Initial distribution: [0.9, 0.1]
- Discount factor: 0.5
- Reward: S1 = 10, S2 = 0
- Observations: S1 emits Z1 with prob 0.75, S2 emits Z2 with prob 0.75

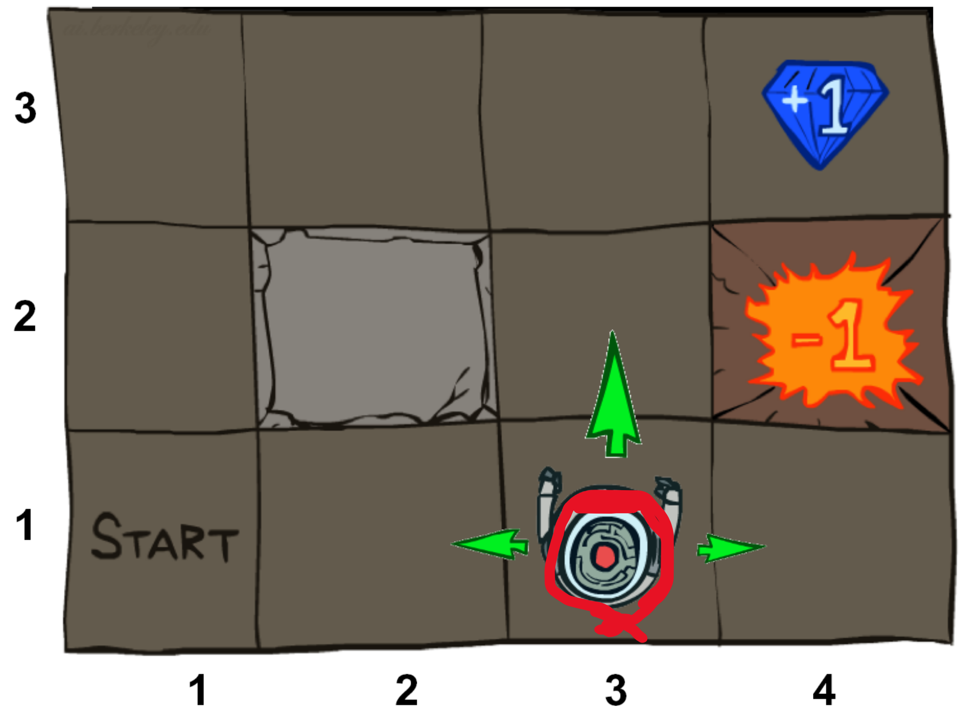
Simple Example



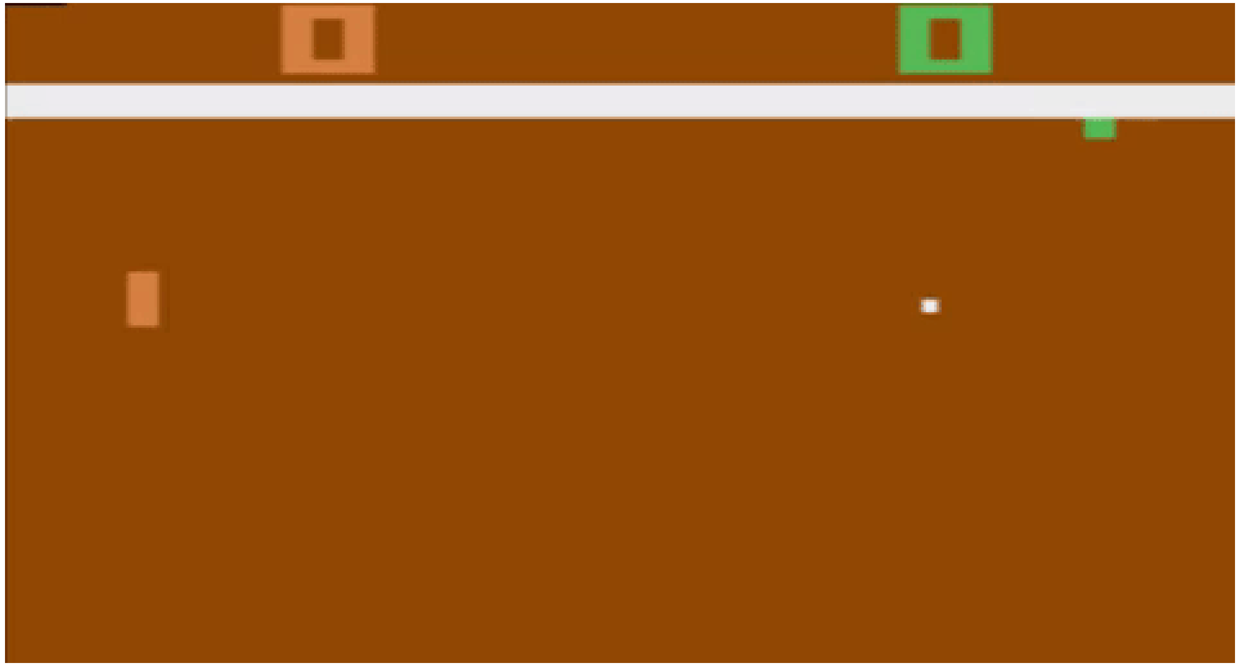
- Initial distribution: [0.5, 0.5]
- Discount factor: 0.5
- Reward: s1 = 10, s2 = 0
- Observations: s1 emits z1 with prob 0.5, s2 emits z2 with prob 0.5

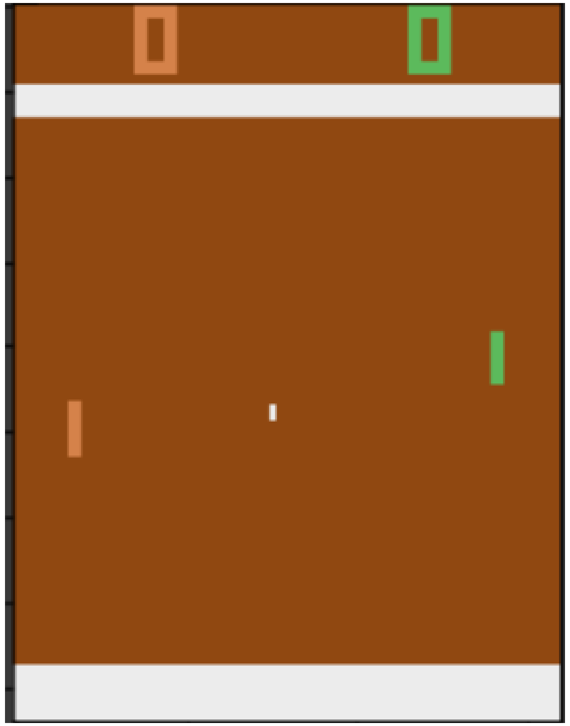
How should we solve a POMDP?

- Solution #1
  - Ignore the fact that it's a POMDP and just act like MDP
  - Policy just maps observations Z to actions A
  - Problems?



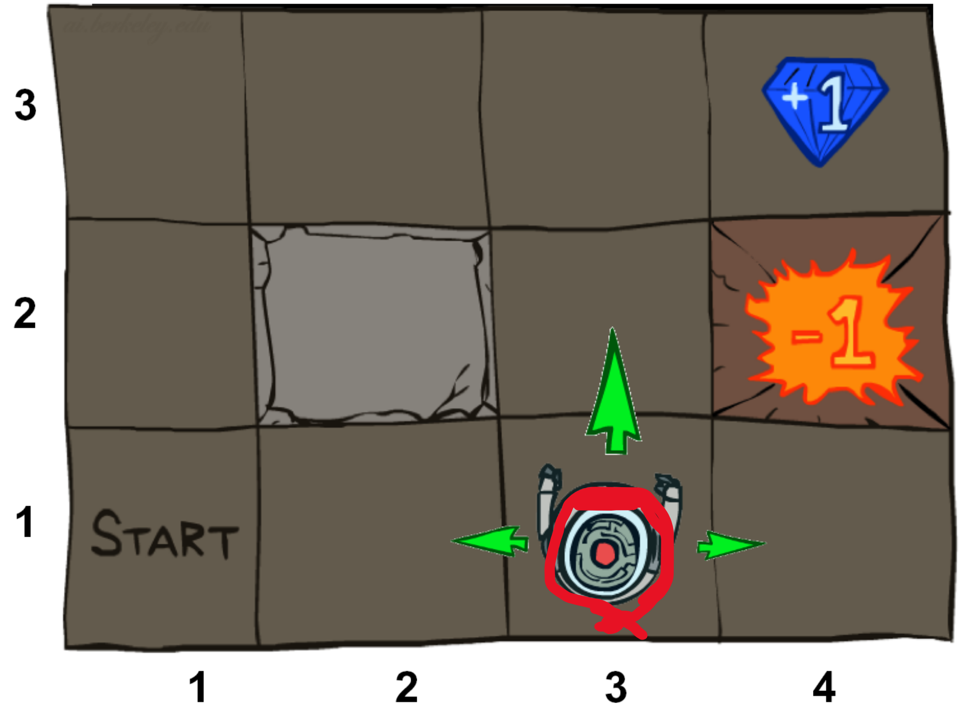
- Learn to play Pong

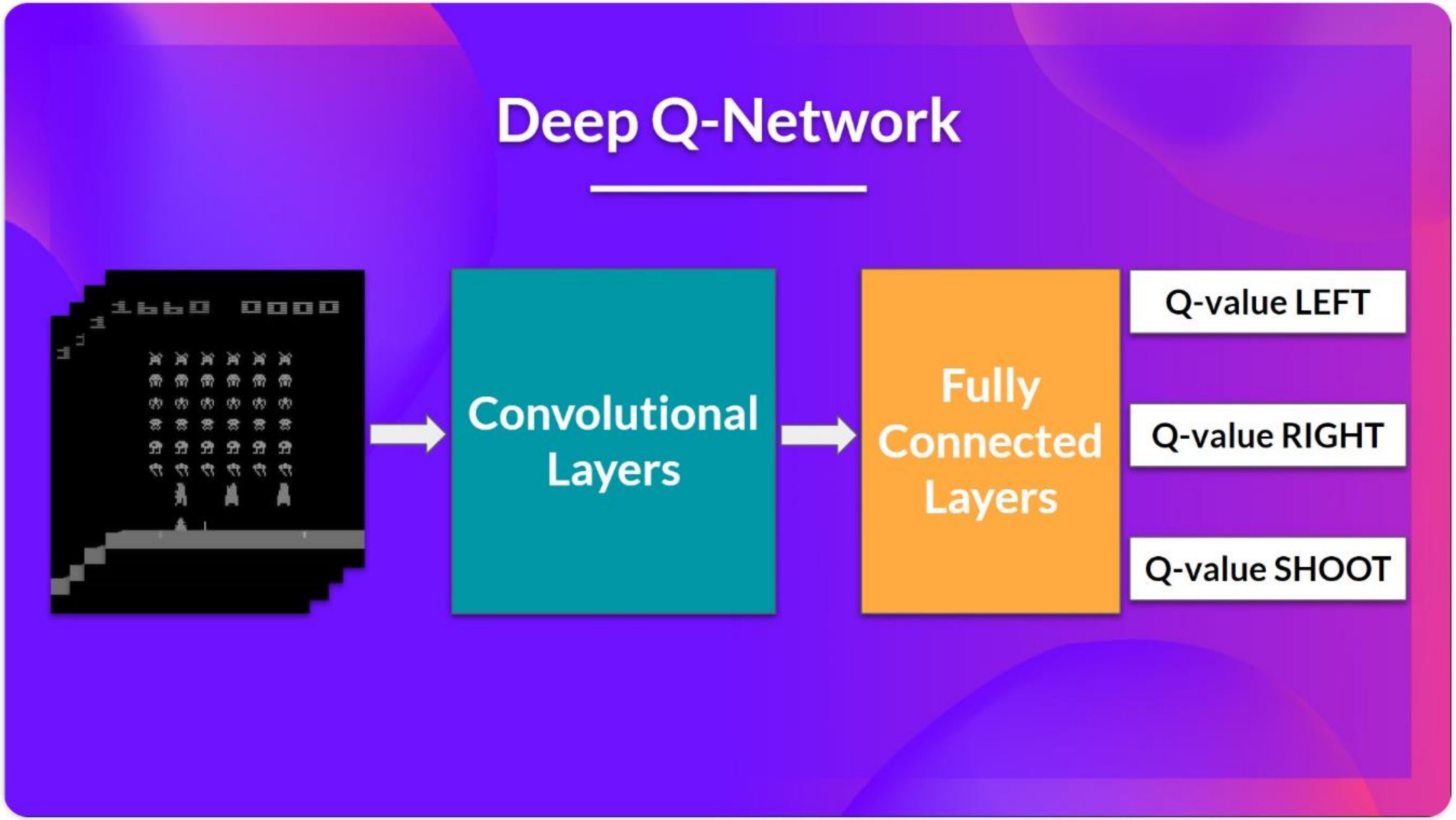




How should we solve a POMDP?

- Solution #2
  - Use the history to try to make the POMDP an MDP
  - Policy now maps observation histories $h_t = (z_1, a_1 \ldots, a_{t-1}, z_t)$ to actions A
  - Problems?

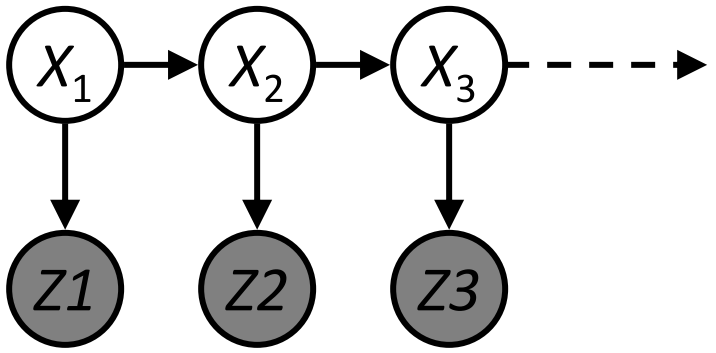



Deep Q-Network
Convolutional Layers
Fully Connected Layers
Q-value LEFT
Q-value RIGHT
Q-value SHOOT

How should we solve a POMDP?

- Solution #3
  - Use belief states: $b(s) = P(s|h)$ (probability we are in state s)
  - Policy now maps belief state vectors b to actions A
  - Goal: Turn POMDP into a Belief State MDP
    - We need to model transitions $b, a, z \rightarrow b'$

$$b'(s') = P(s'|b, a, z)$$



The Forward Algorithm

- We are given evidence at each time and want to know

$$B_t(X) = P(X_t | e_{1:t})$$

- We can derive the following recursive update

$$P(x_t|e_{1:t}) = P(x_t|e_{1:t-1}, e_t)$$ Divide up evidence

$$\propto P(e_t|x_t, e_{1:t-1})P(x_t|e_{1:t-1})$$ Bayes' rule

$$= P(e_t|x_t)P(x_t|e_{1:t-1})$$ Sensor Markov assumption

$$= P(e_t|x_t) \sum_{x_{t-1}} P(x_t, x_{t-1}|e_{1:t-1})$$ Reverse marginalization

$$= P(e_t|x_t) \sum_{x_{t-1}} P(x_t|e_{1:t-1}, x_{t-1})P(x_{t-1}|e_{1:t-1})$$ Product rule

$$= P(e_t|x_t) \sum_{x_{t-1}} P(x_t|x_{t-1})P(x_{t-1}|e_{1:t-1})$$ Markov assumption

This is variable elimination with ordering $X_1, X_2, \ldots$

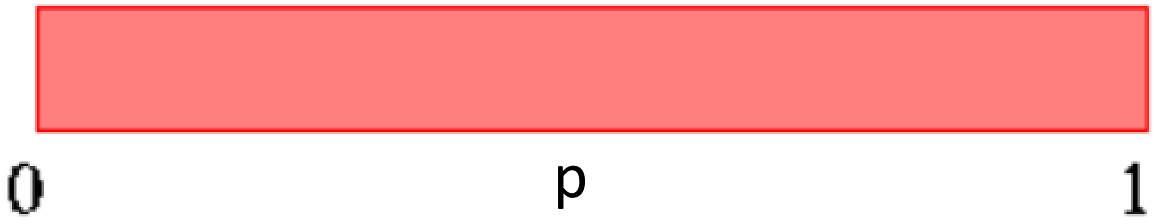
How should we solve a POMDP?

- Solution #3
  - Use belief states: $b(s) = P(s|h)$ (probability we are in state s)
  - Policy now maps belief state vectors b to actions A

Policies in POMDPs

- Need to map from belief states (probability distributions over states) to actions

- Toy example: 2-state MDP
  - P(s1) = p, P(s2) = 1-p

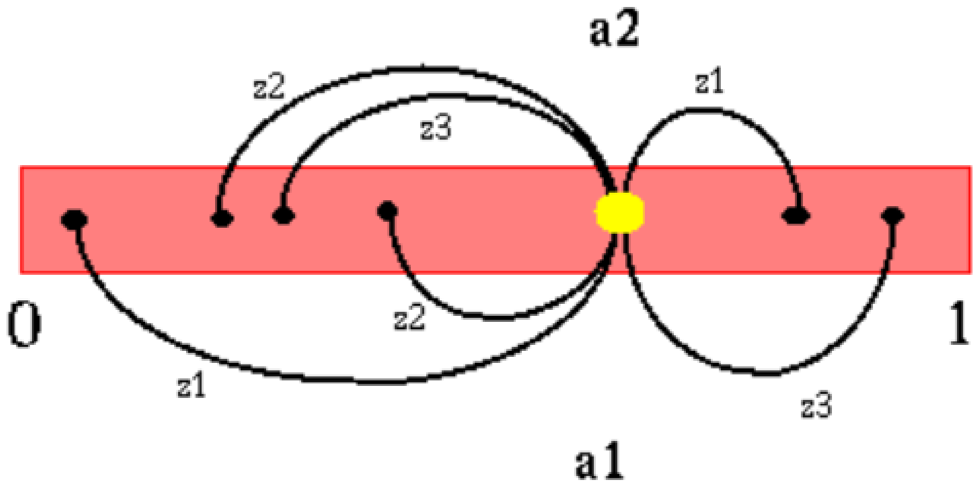


1D belief space for a 2 state POMDP

https://www.pomdp.org/tutorial/pomdp-solving.html

State Estimation

- If we start with a particular belief state b and take action a, then we will receive observation z
- If finite actions and observations, then finite number of possible next belief states, but we don't know ahead of time what z will be.



1D belief space for a 2 state POMDP

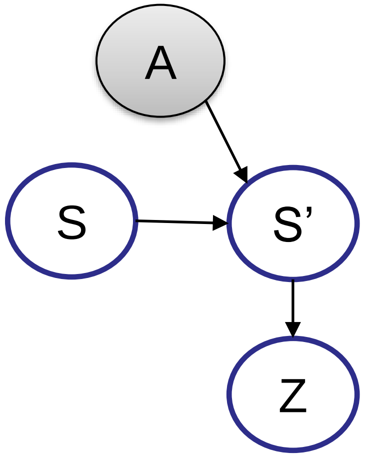
How should we solve a POMDP?

- Solution #3
  - Use belief states: $b(s) = P(s|h)$ (probability we are in state s)
  - Policy now maps belief state vectors b to actions A
  - Goal: Turn POMDP into a Belief State MDP
    - We need to model transitions $b, a, z \rightarrow b'$

$$b'(s') \propto P(z|s') \sum_{s} P(s'|s,a) b(s)$$

This is just state estimation like HMMs!

Belief State MDP



- State space: B
- Action space: A
- Transition Function: P(b'|b,a)

$$P(b'|b,a) = \sum_e P(b',z|b,a) = \sum_e P(b'|b,a,z)P(z|b,a)$$

0 or 1 depending on state estimation

Variable elimination in order S, S'

$$\sum_{s'} P(z|s') \sum_s P(s'|s,a)b(s)$$

- Reward function:

$$R(b,a) = \sum_s b(s)r(s,a)$$

- Problems?

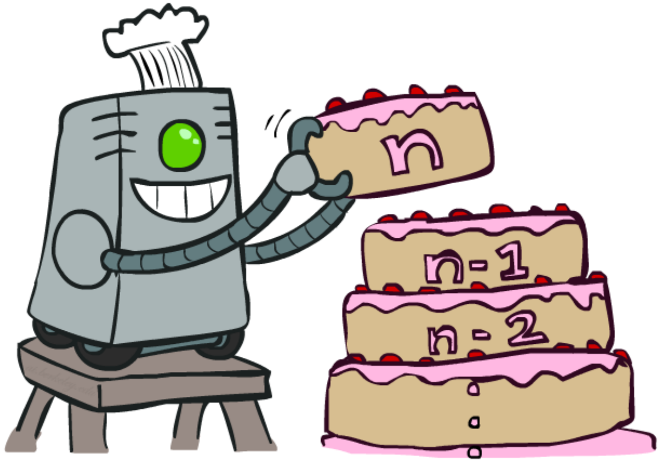
Value Iteration in MDPs

- Bellman Equation

$$V^*(s) = \max_a R(s,a) + \gamma \sum_{s'} T(s,a,s')V^*(s')$$

- Iterate until convergence:

$$V_{k+1}(s) = \max_a R(s,a) + \gamma \sum_{s'} T(s,a,s')V_k(s')$$



Value Iteration in POMDPs?

- Bellman Equation

$$V^*(b) = \max_a R(b,a) + \gamma \sum_{b\prime} T(b,a,b')V^*(b')$$

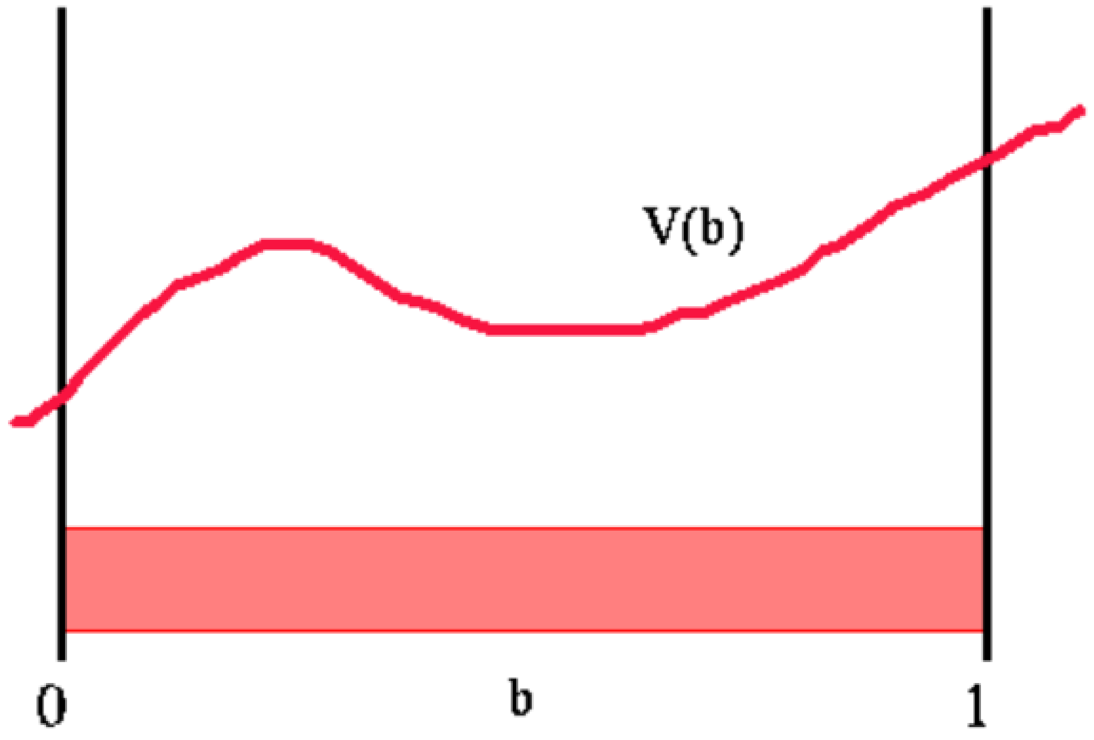
$$= \max_a R(b,a) + \gamma \textstyle\sum_{b\prime} P(b'|b,a)V^*(b')$$

$$= \max_a R(b,a) + \gamma \sum_{b\prime} \sum_{e} P(b'|b,a,z)P(z|b,a)\, V^*(b')$$

$$= \max_a R(b,a) + \gamma \sum_{z} P(z|b,a)V^*(SE(b,a,z))$$

How do we deal with the continuous state space?

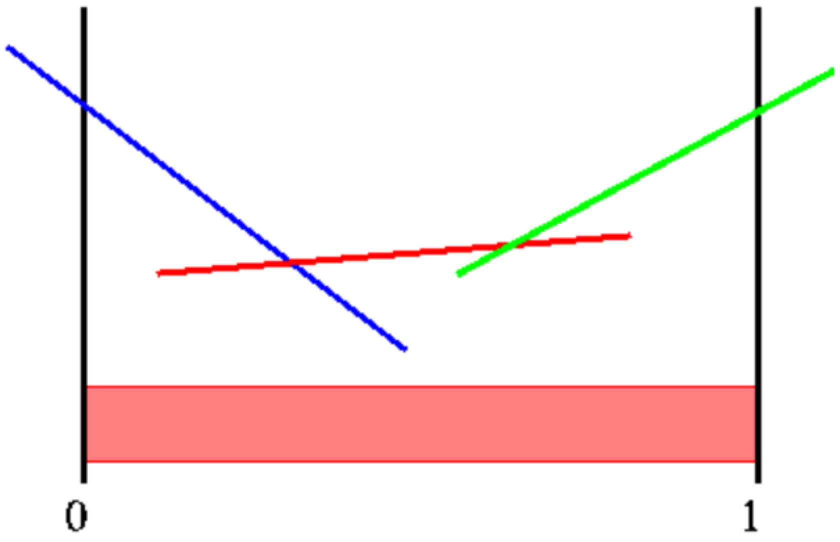
- We can't simply keep a table of values any more…



Value function over belief space

Value Functions for POMDPs

- For a fixed horizon, the value function is **piecewise linear and convex!**
  - Each iteration of value iteration only requires a finite number of linear segments.



Sample PWLC value function

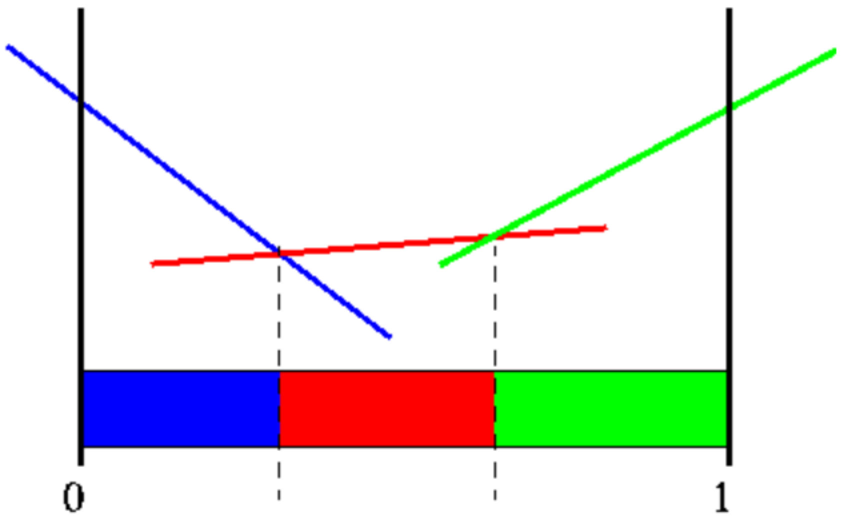
Let the utility of a conditional plan that starts in state s be $\alpha_p(s)$ Then the expected utility is linear in b:

$$V(p) = \max_p \sum_s b(s)\alpha_p(s)$$

- Value function for each horizon can be represented as a set of vectors Γ_t
- $V_t(b) = \max_{\alpha \in \Gamma_t} \alpha \cdot b$

Value Functions for POMDPs

- For a fixed horizon, the value function is **piecewise linear and convex!**
  - Each iteration of value iteration only requires a finite number of linear segments.



Partitioning belief space!

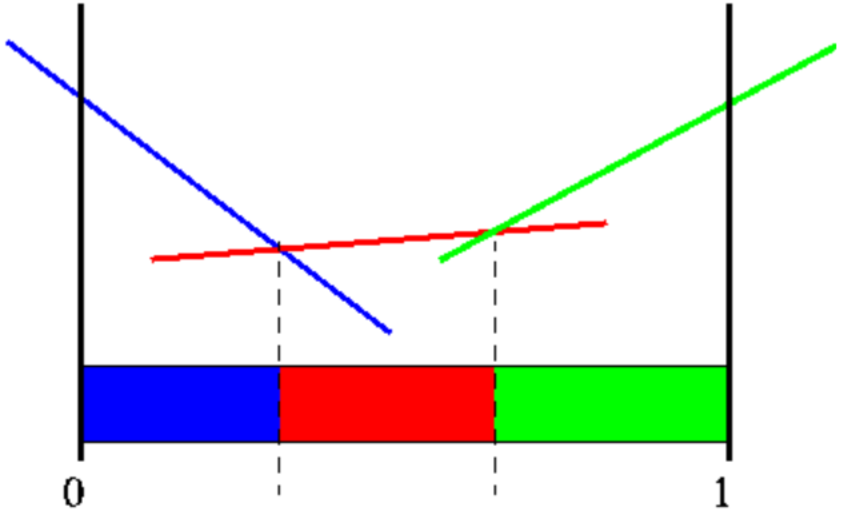
Let the utility of a conditional plan that starts in state s be $\alpha_p(s)$ Then the expected utility is linear in b:

$$V(p) = \max_p \sum_s b(s)\alpha_p(s)$$

- Value function for each horizon can be represented as a set of vectors Γ_t
- $V_t(b) = \max_{\alpha \in \Gamma_t} \alpha \cdot b$

Value Functions for POMDPs

- For a fixed horizon, the value function is **piecewise linear and convex!**
  - Each iteration of value iteration only requires a finite number of linear segments.



Partitioning belief space!

- Value function for each horizon can be represented as a set of vectors Γ_t
- $V_t(b) = \max_{\alpha \in \Gamma_t} \alpha \cdot b$

Good News: Don't have to worry about infinite states to represent V

Bad News: We still don't know how to go from V_t to V_{t+1}

Example

- Two states (s1, s2), Two actions (a1,a2), three observations (z1,z2,z3), R(s1,a1) = 0, R(s1,a2)=1.5, R(s2,a1) = 1, R(s2,a2) = 0
- Consider first horizon. Best we can do if we only take one action:

$$V_1(b) = \max_a R(b,a) + \gamma \sum_z P(z|b,a) V^*(SE(b,a,z))$$

$$= \max_a \sum_s b(s)\, R(s,a)$$

Quiz: What is the value of taking action a1 or a2 if b= [0.75, 0.25]?

V(a1) = 0.75 * 0 + 0.25 * 1 = 0.25

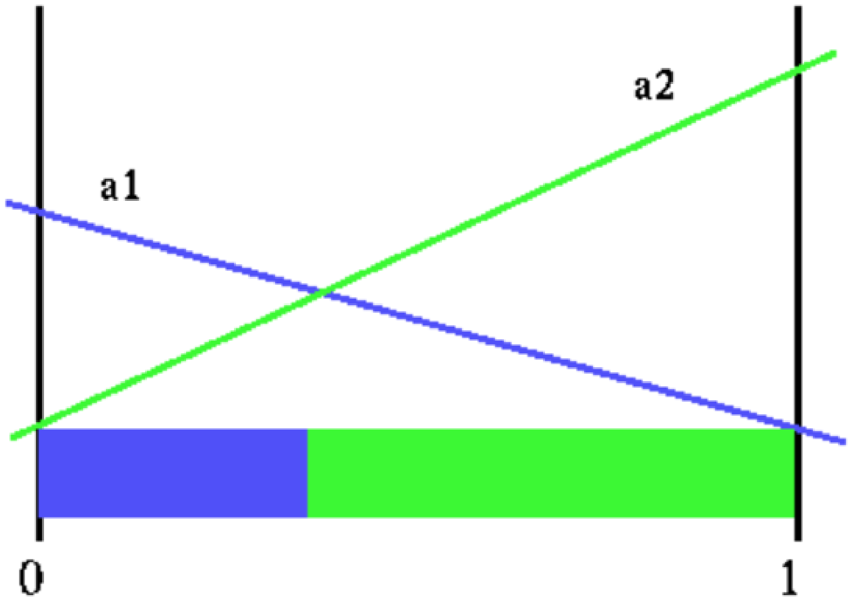
V(a2) = 0.75 * 1.5 + 0.25 * 0 = 1.125

Example

- Two states (s1, s2), Two actions (a1,a2), three observations (z1,z2,z3), R(s1,a1) = 0, R(s1,a2)=1.5, R(s2,a1) = 1, R(s2,a2) = 0
- Consider first horizon. Best we can do if we only take one action:

$$V_1(b) = \max_a R(b,a) = \max_a \sum_s b(s)\, R(s,a)$$

Just a linear function over belief space!
Partitions show where a1 or a2 are optimal.



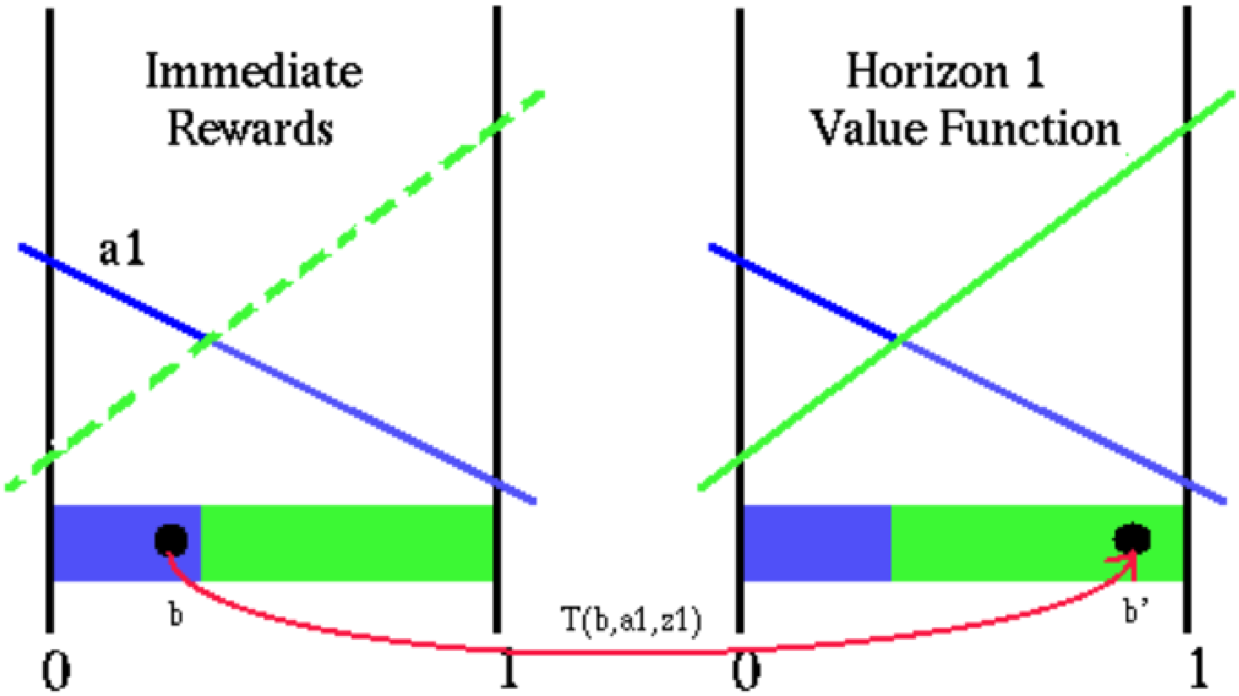
Horizon 1 value function

Horizon 2

1. We will first show how to compute the value of a single belief state for a given action and observation.
2. Then we show how to compute the value for every belief state for a given action and observation, in a finite amount of time.
3. Then we will show how to compute the value of a belief state given only an action.
4. Finally, we will show how to compute the actual value for a belief state.

Computing Belief State Value from an Action and Observation

- Given belief state b, what is the value of doing a1, if after the action we receive observation z1?

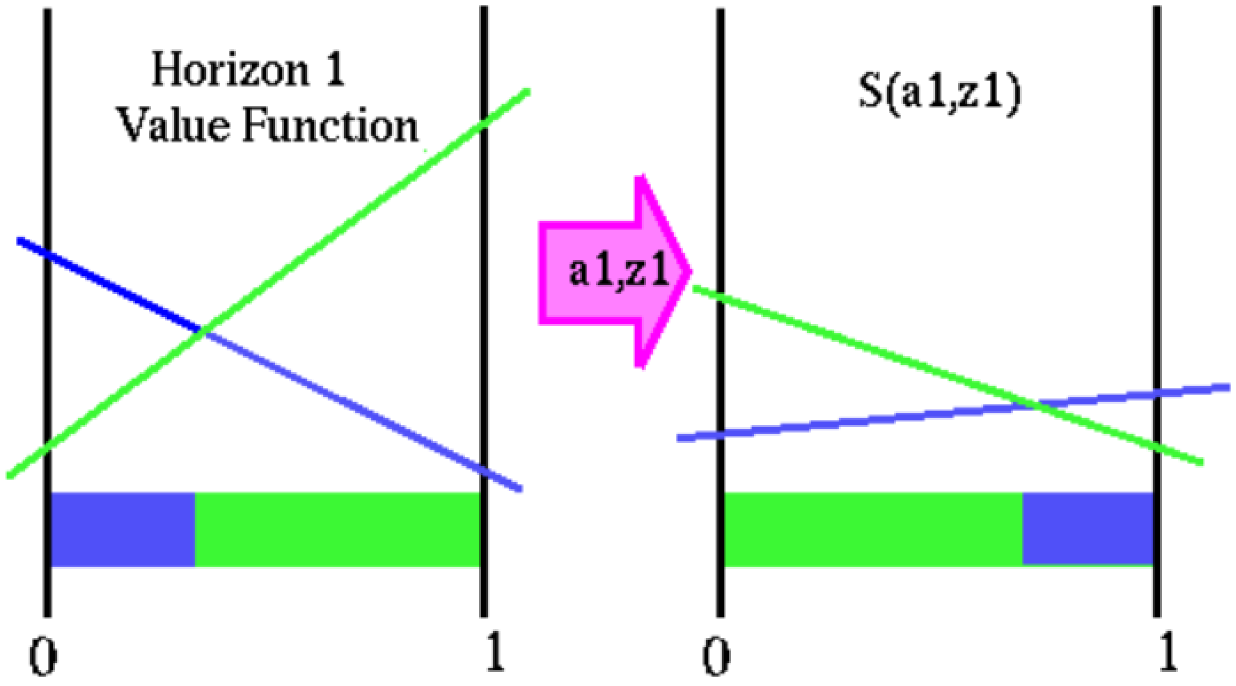


Value of a fixed action and observation

- We know how to go from b to b' given a and z!
- Horizon 1 Value function tells us best values for every belief state when just one action left to take.

Computing All Belief State Values for an Action and Observation

- Transform belief state b given a and z into b'
- Then we add the immediate rewards to the transformed function.

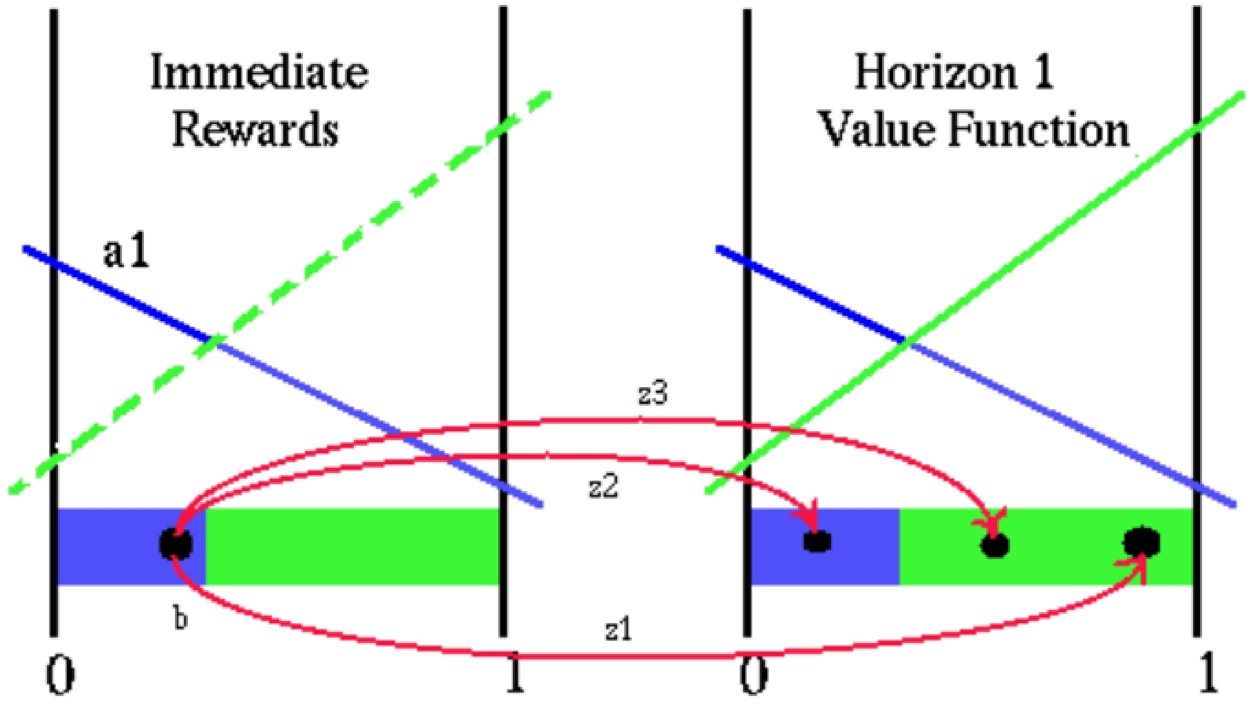


Transformed value function

Tells us the value of each belief state after action a1 is taken and observation z1 is seen.

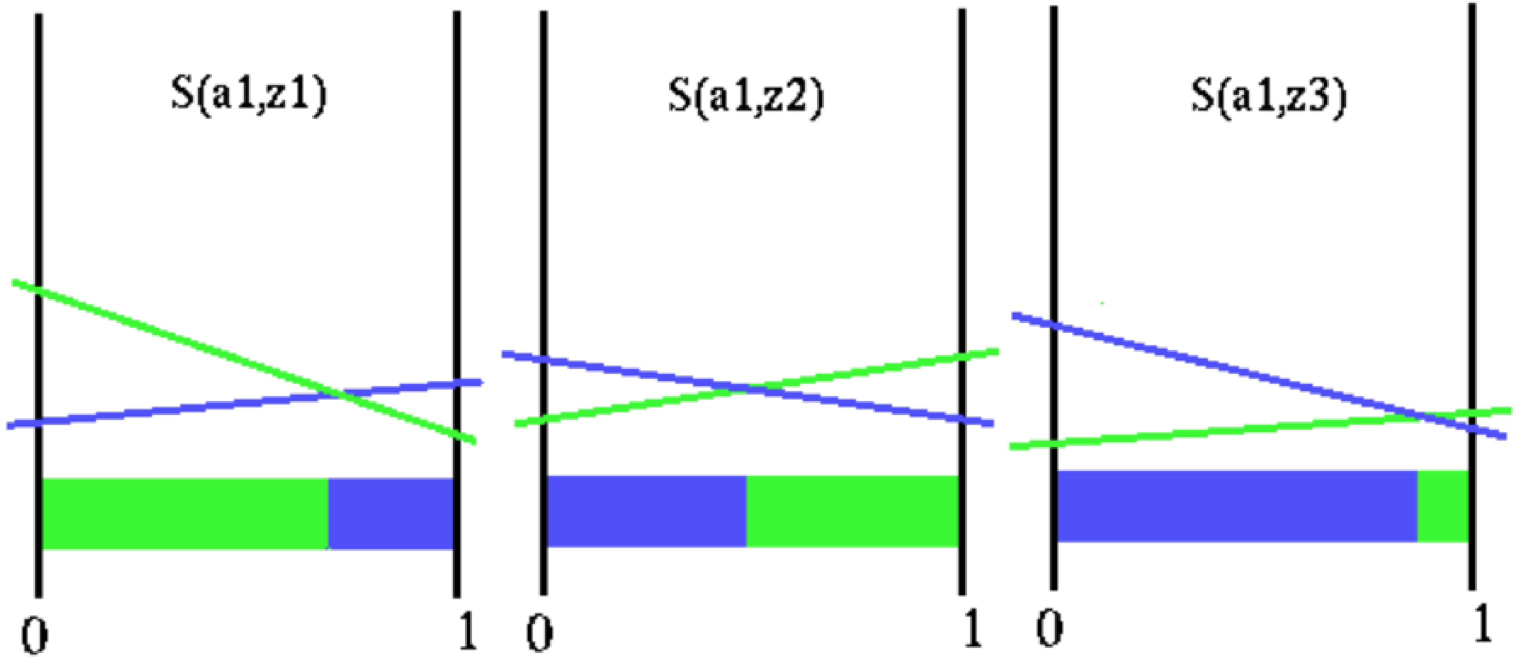
Computing a Belief State Value for a Single Action

- Before we computed conditional value given observation z.
- Now we don't know what observation we'll get

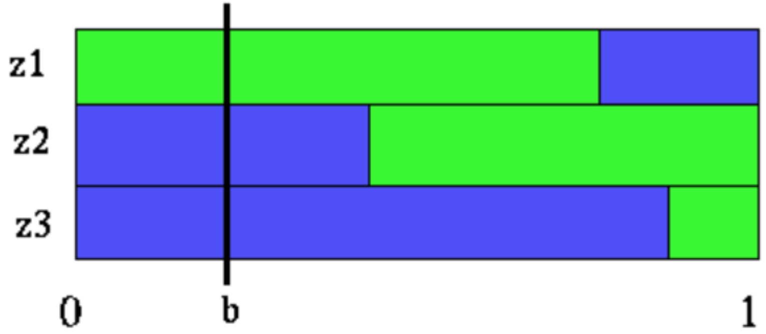


Transformed value function

- Best value of belief state b given first action is a1



Transformed value function for all observations

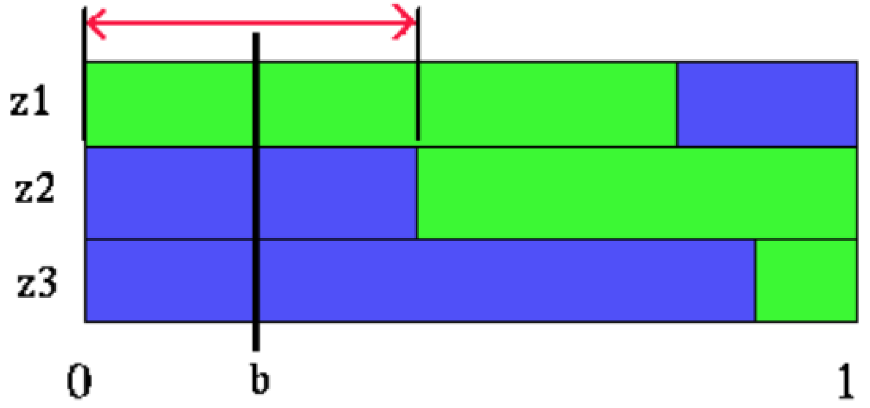


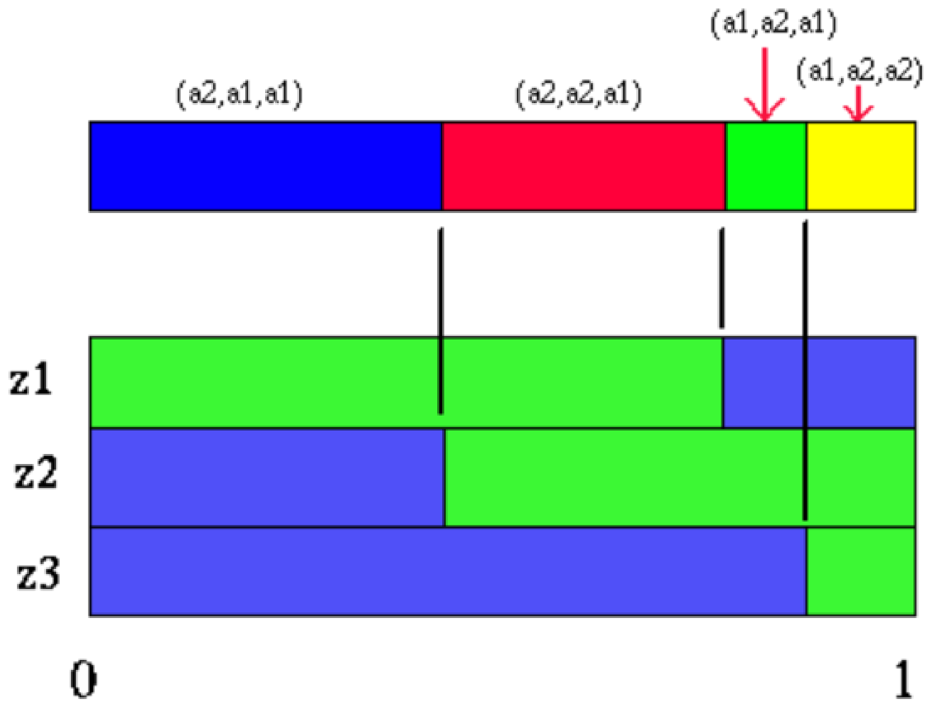
Future strategy

(z1:a2, z2:a1, z3:a1)

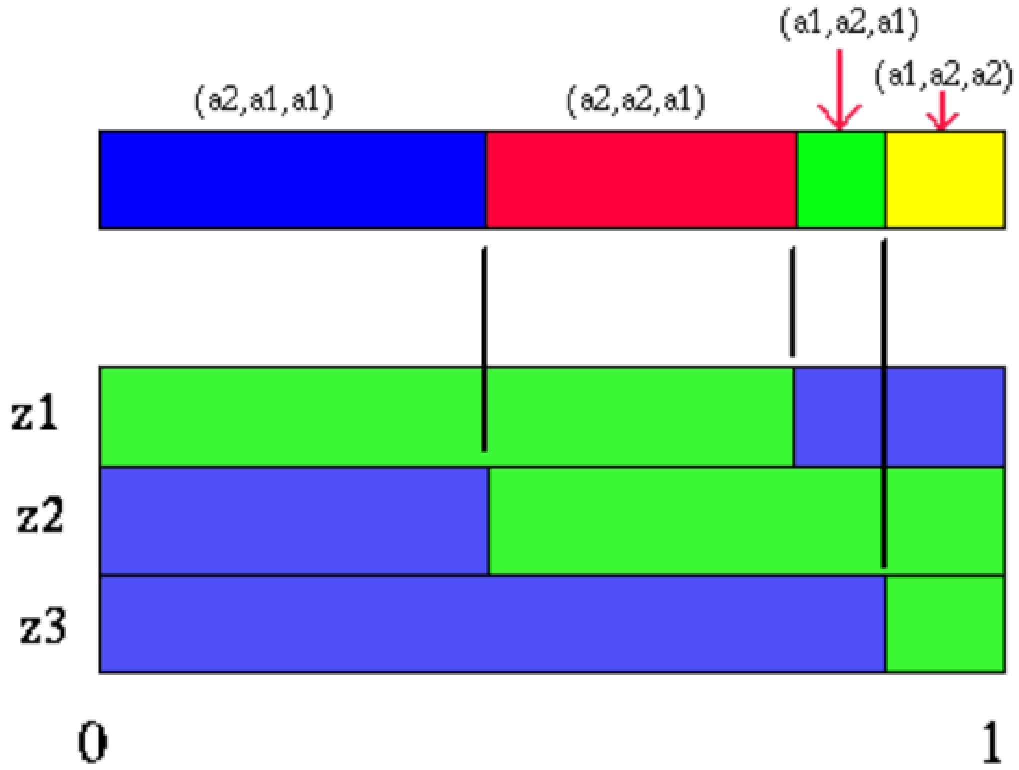
Computing Final Belief State Value

- If we fix first action to be a1 and follow strategy (z1:a2, z2:a1, z3:a1) as above then we can compute value for every single belief point.
- Add line segment for immediate reward of a1 and line segments S() for future strategy. Adding lines gives us a line.
- But when is this strategy good?

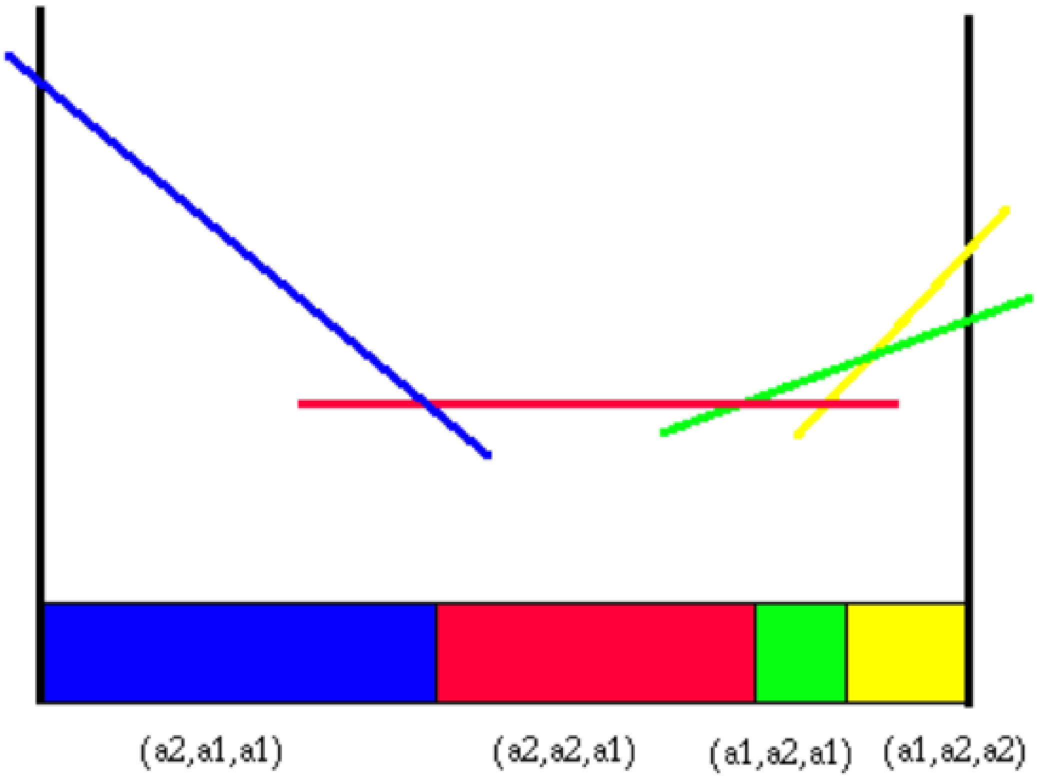




Computing Final Belief State Value



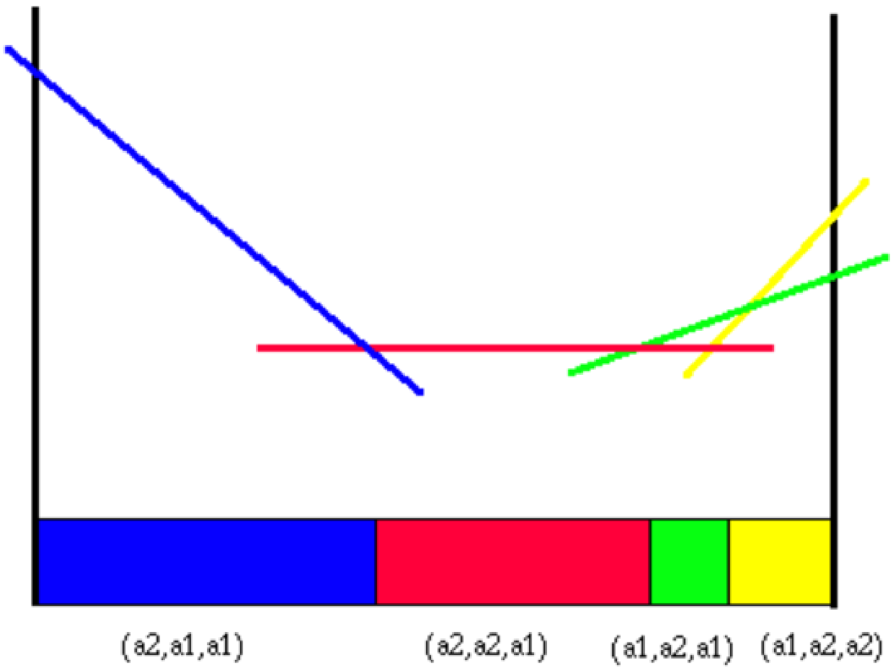
Partition for action a1



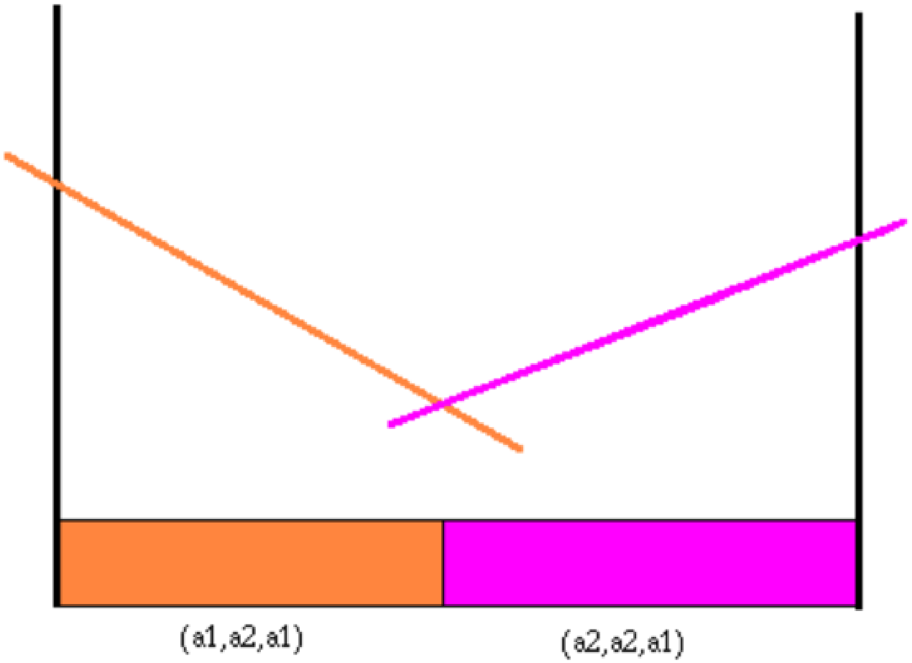
Value function and partition for action a1

Computing Final Belief State Value

- We can do the same thing for each action



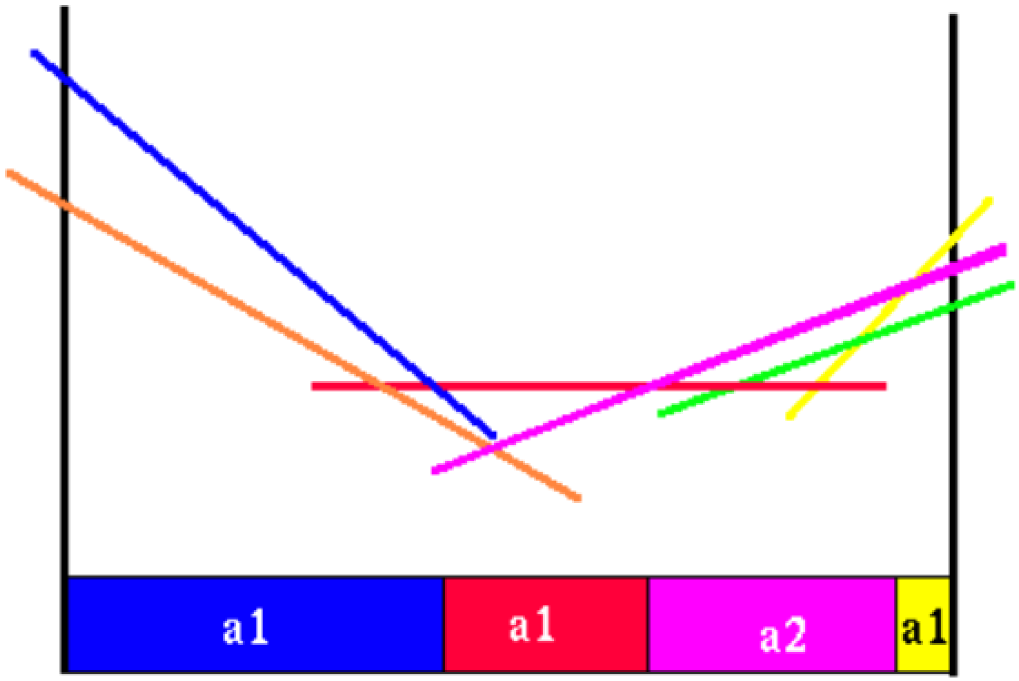
Value function and partition for action a1



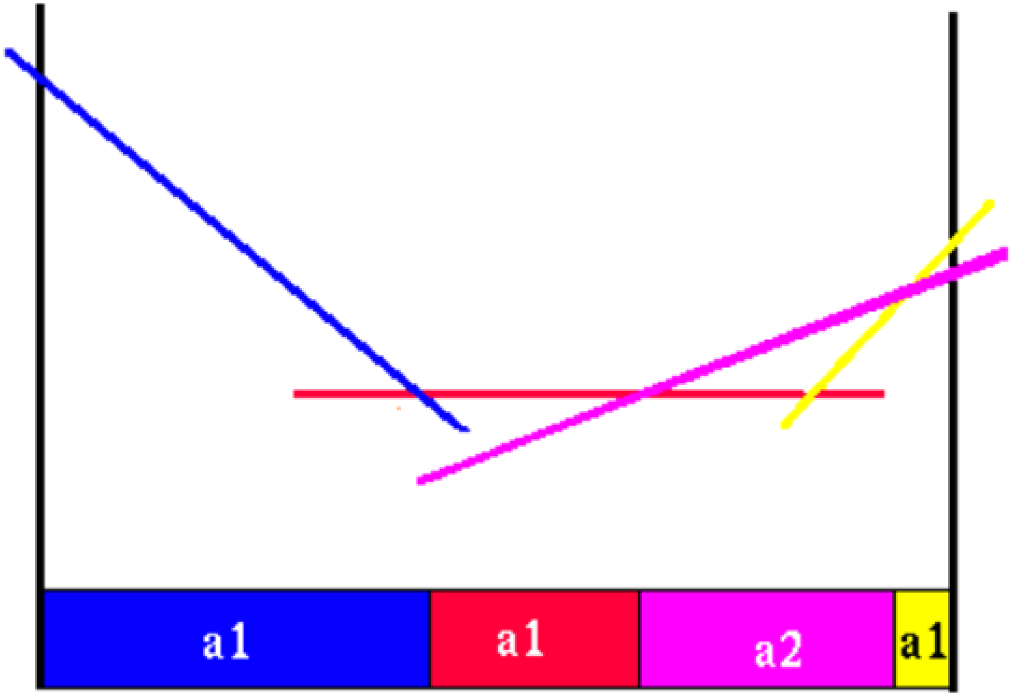
Value function and partition for action a2

Computing Final Belief State Value

- Combine to see where each action gives the highest value



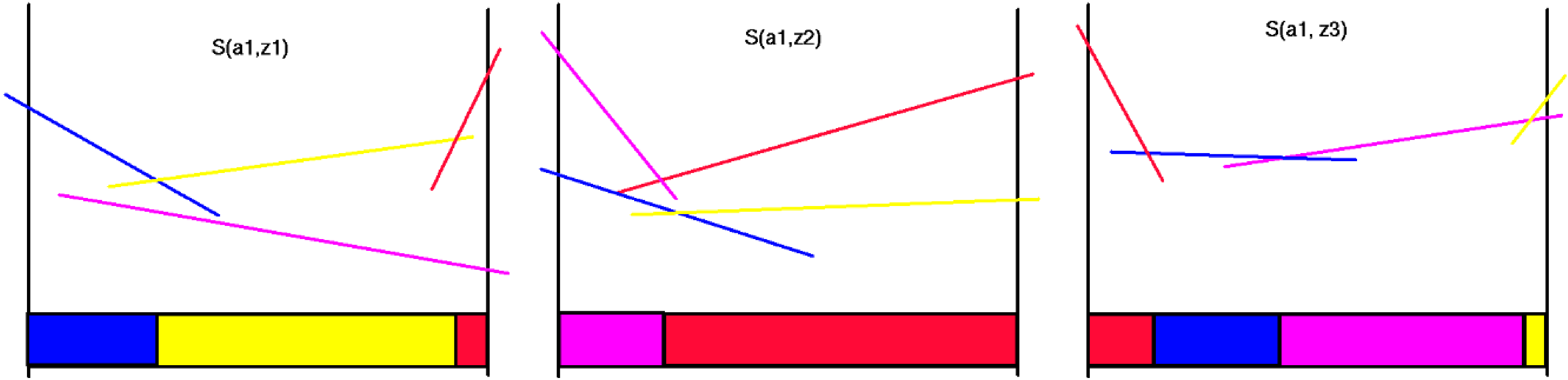
Combined a1 and a2 value functions



Value function for horizon 2

Horizon 3

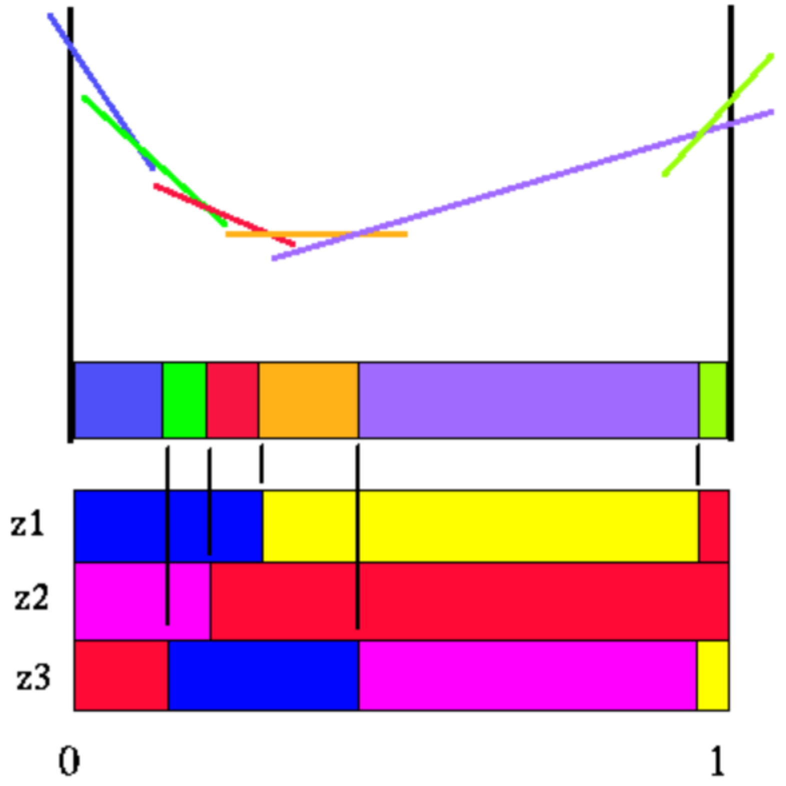
- Transform horizon 2 value function for action a1 all observations



Transformed Horizon 2 Value Functions for Action a1

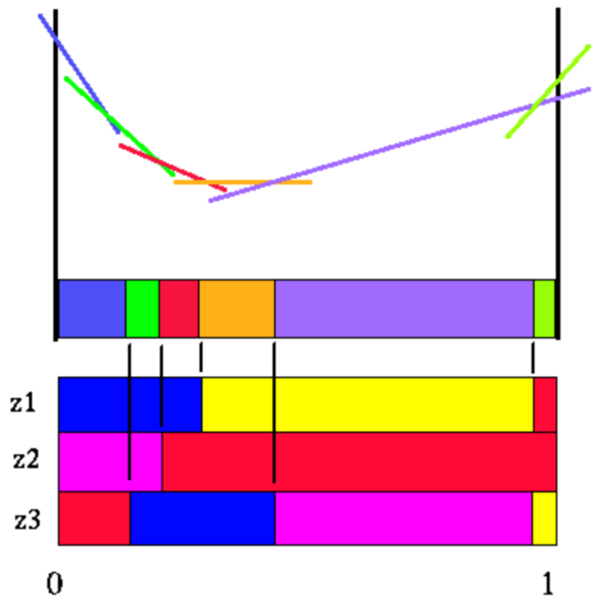
- Each color represents a complete future strategy

- Find value function by adding immediate rewards and the S() functions for each useful strategy
- Only 6 useful strategies!

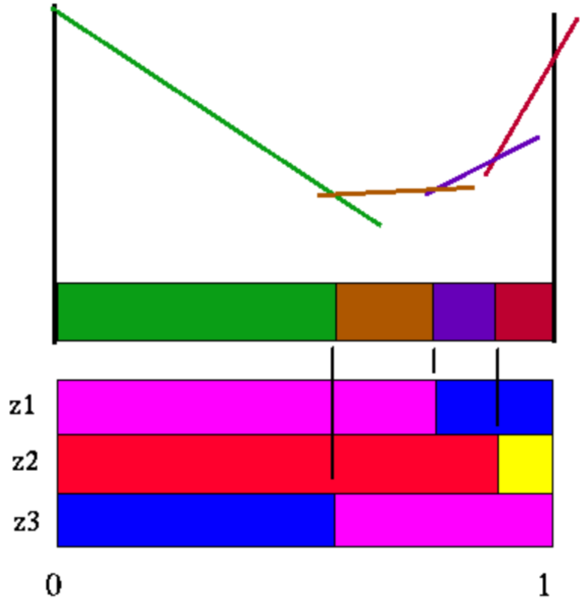


Value function for action a1 and horizon 3

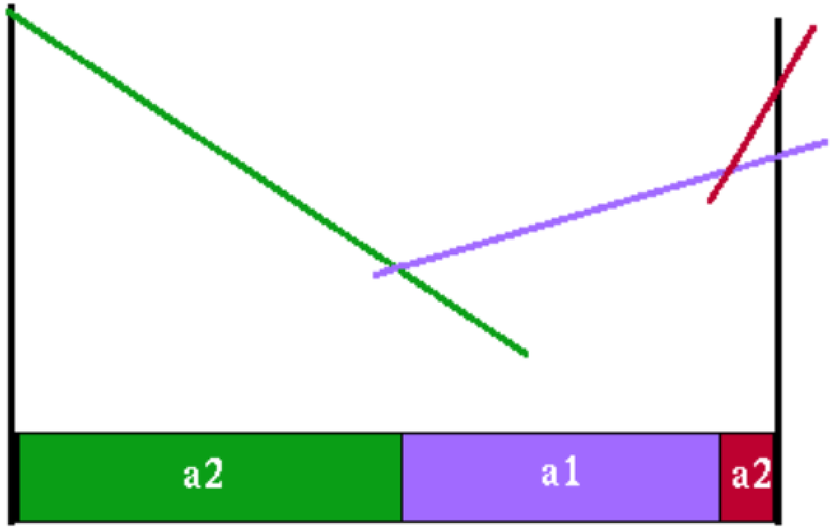
- Find value function by adding immediate rewards and the S() functions for each useful strategy



Value function for action a1 and horizon 3



Value function for action a2 and horizon 3



Value function for horizon 3

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.
- POMDP exact solutions only work with very small state spaces with small numbers of possible observations and actions.
- Lots of current research on approximations and faster solvers!

Next Time: Imitation Learning