Partially Observable Markov Decision Processes (POMDPs)

Geoff Hollinger

Graduate Artificial Intelligence

Fall, 2007

*Some media from Reid Simmons, Trey Smith, Tony Cassandra, Michael Littman, and Leslie Kaelbling

Outline for POMDP Lecture

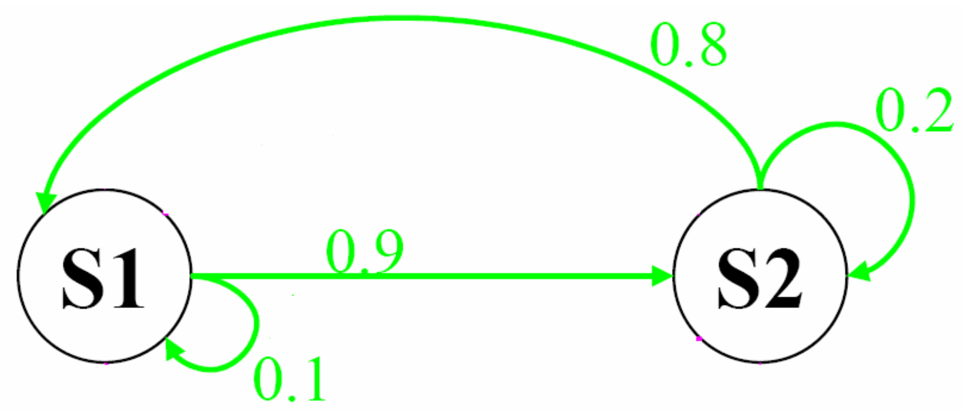
- Introduction
  - What is a POMDP anyway?
  - A simple example
- Solving POMDPs
  - Exact value iteration
  - Policy iteration
  - Witness algorithm, HSVI
  - Greedy solutions
- Applications and extensions
  - When am I ever going to use this (other than in homework five)?

So who is this Markov guy?

- Andrey Andreyevich Markov (1856-1922)
- Russian mathematician
- Known for his work in stochastic processes
  - Later known as Markov Chains

What is a Markov Chain?

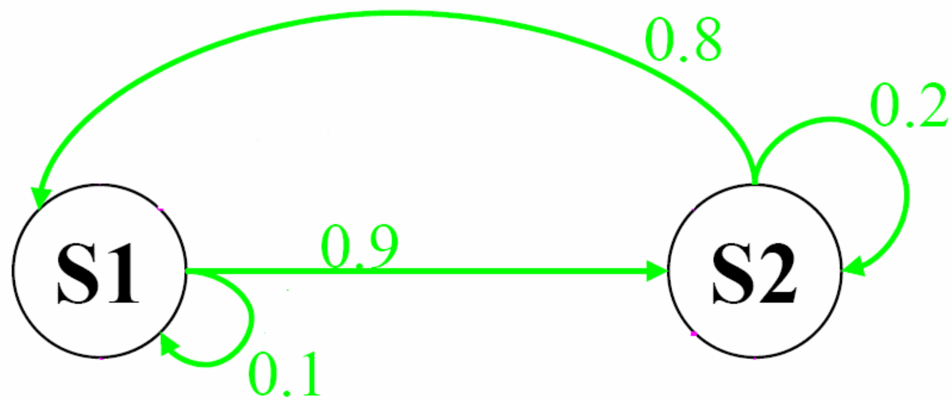
- Finite number of discrete states
- Probabilistic transitions between states
- Next state determined only by the current state
  - This is the Markov property



Rewards: S1 = 10, S2 = 0

What is a Hidden Markov Model?

- Finite number of discrete states
- Probabilistic transitions between states
- Next state determined only by the current state
- We're unsure which state we're in
  - The current states emits an observation

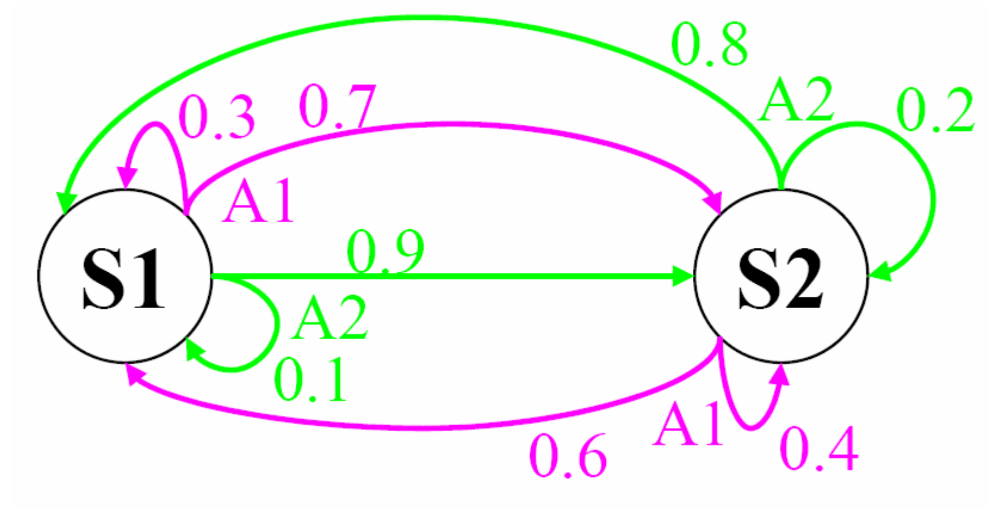


Rewards: S1 = 10, S2 = 0

Do not know state:
S1 emits O1 with prob 0.75
S2 emits O2 with prob 0.75

What is a Markov Decision Process?

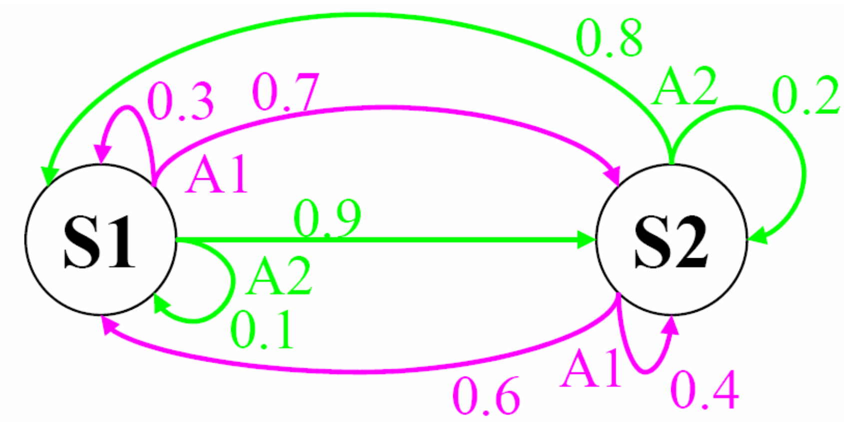
- Finite number of discrete states
- Probabilistic transitions between states ***and*** controllable actions in each state
- Next state determined only by the current state ***and*** current action
  - This is still the Markov property



Rewards: S1 = 10, S2 = 0

What is a Partially Observable Markov Decision Process?

- Finite number of discrete states
- Probabilistic transitions between states and controllable actions
- Next state determined only by the current state and current action
- We're unsure which state we're in
  - The current state emits observations



Rewards: S1 = 10, S2 = 0

Do not know state:
S1 emits O1 with prob 0.75
S2 emits O2 with prob 0.75

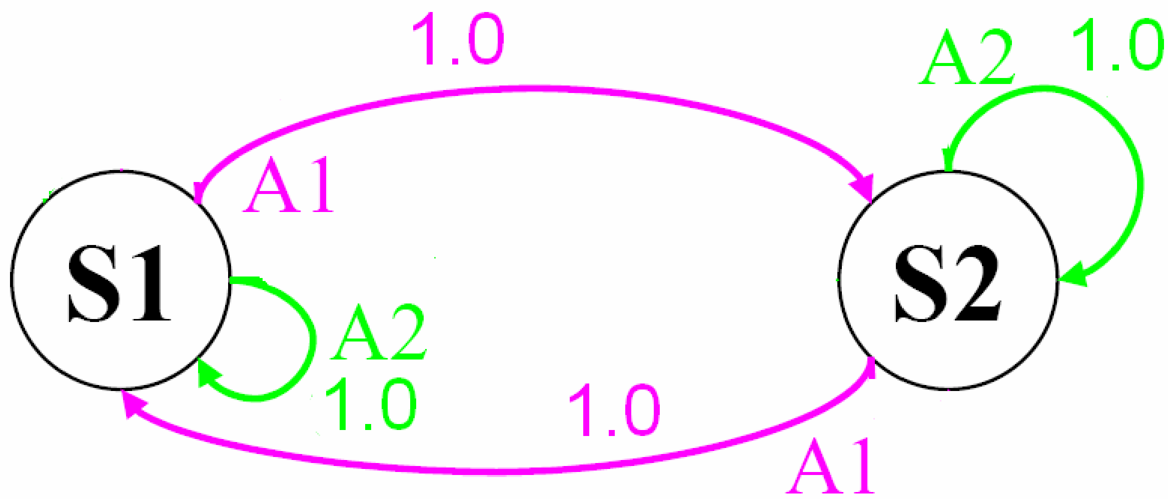
A Very Helpful Chart

Markov Models		Do we have control over the state transitons?	
		NO	YES
Are the states completely observable?	YES	**Markov Chain**	**MDP** Markov Decision Process
	NO	**HMM** Hidden Markov Model	**POMDP** Partially Observable Markov Decision Process

POMDP versus MDP

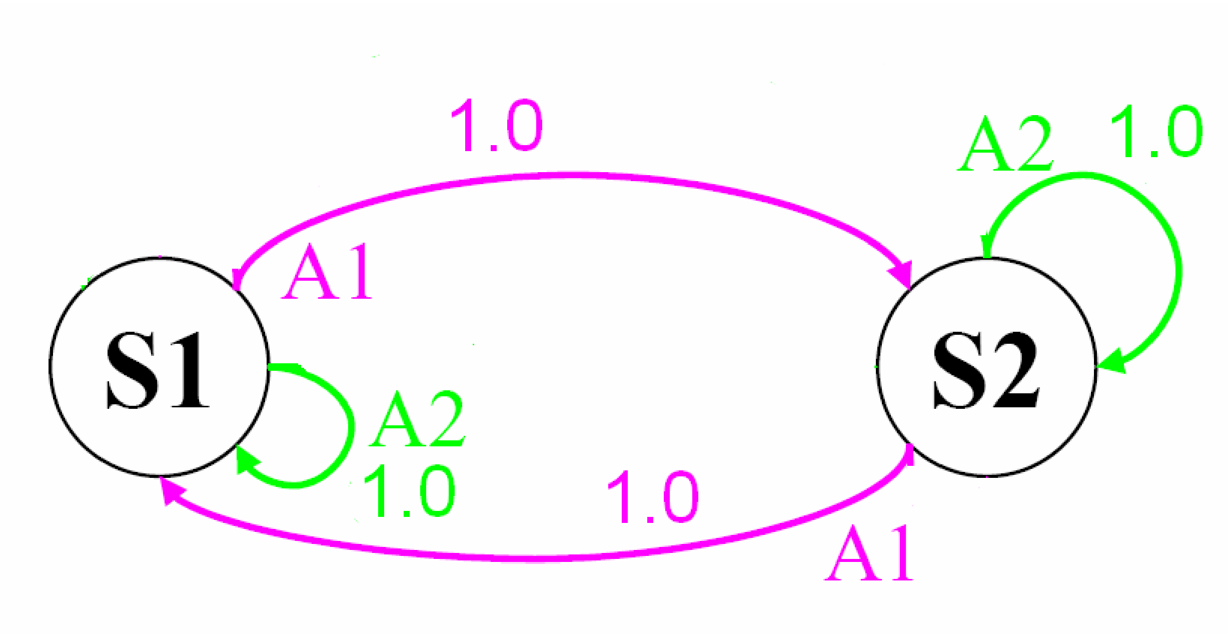
- MDP
  - +Tractable to solve
  - +Relatively easy to specify
  - -Assumes perfect knowledge of state
- POMDP
  - +Treats all sources of uncertainty uniformly
  - +Allows for information gathering actions
  - -Hugely intractable to solve optimally

Simple Example



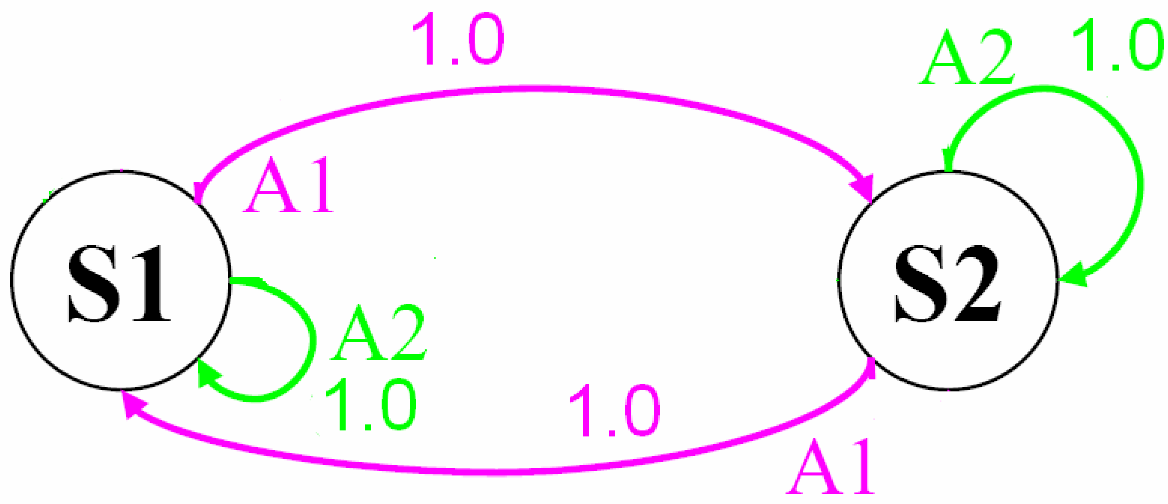
- Initial distribution: [0.1, 0.9]
- Discount factor: 0.5
- Reward: S1 = 10, S2 = 0
- Observations: S1 emits O1 with prob 1.0, S2 emits O2 with prob 1.0

Simple Example



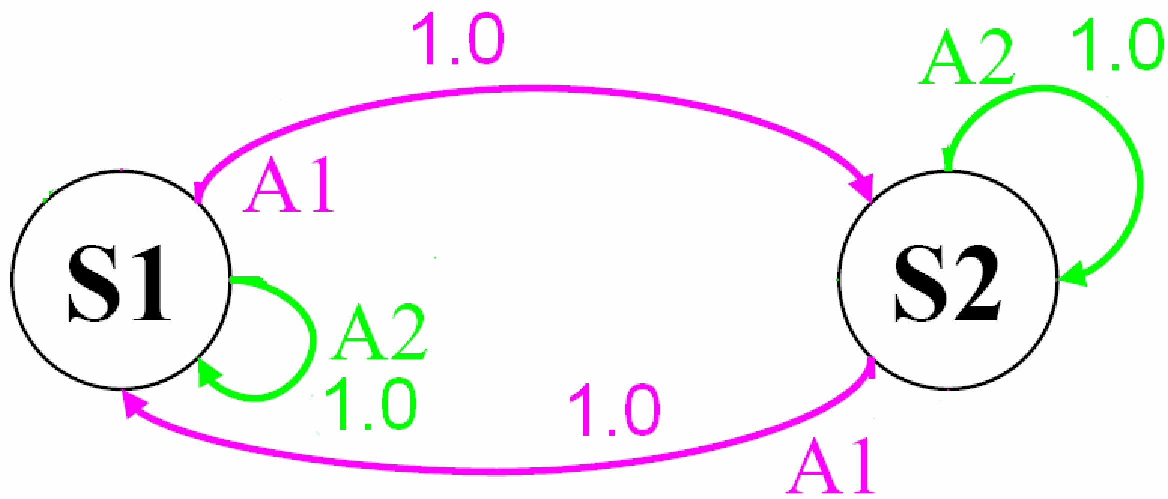
- Initial distribution: [0.9, 0.1]
- Discount factor: 0.5
- Reward: S1 = 10, S2 = 0
- Observations: S1 emits O1 with prob 1.0, S2 emits O2 with prob 1.0

Simple Example



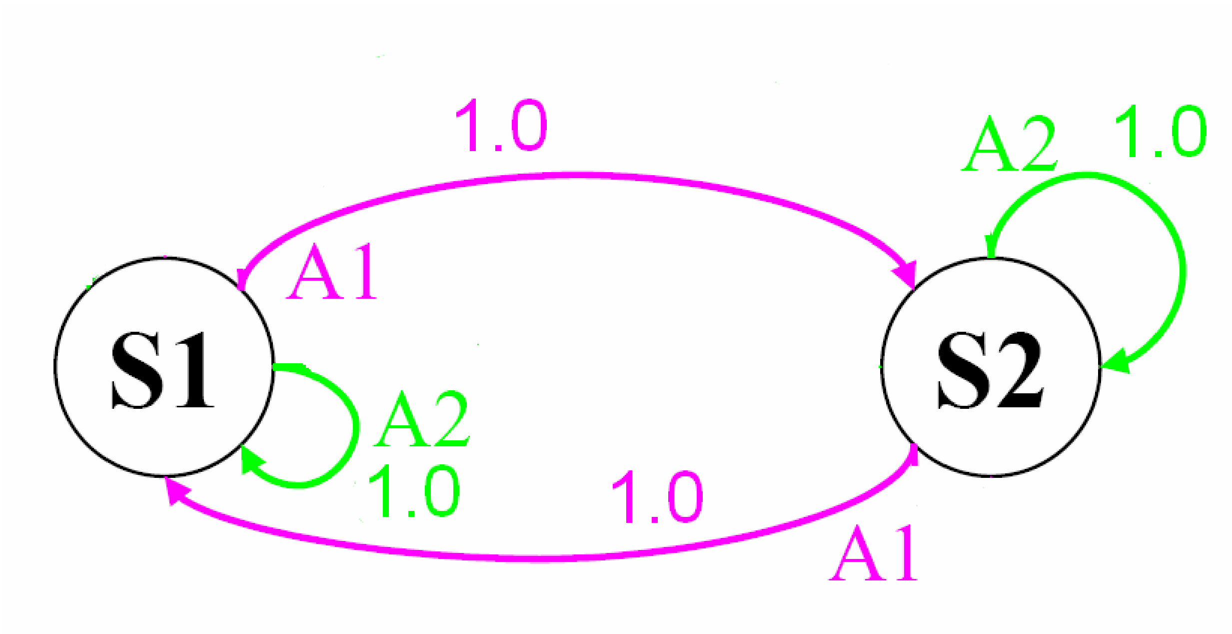
- Initial distribution: [0.1, 0.9]
- Discount factor: 0.5
- Reward: S1 = 10, S2 = 0
- Observations: S1 emits O1 with prob 0.75, S2 emits O2 with prob 0.75

Simple Example



- Initial distribution: [0.5, 0.5]
- Discount factor: 0.5
- Reward: S1 = 10, S2 = 0
- Observations: S1 emits O1 with prob 1.0, S2 emits O2 with prob 1.0

Simple Example



- Initial distribution: [0.5, 0.5]
- Discount factor: 0.5
- Reward: S1 = 10, S2 = 0
- Observations: S1 emits O1 with prob 0.5, S2 emits O2 with prob 0.5

Time for Some Formalism

- POMDP model
  - Finite set of states: $s_1, \ldots, s_n \in S$
  - Finite set of actions: $a_1, \ldots, a_m \in A$
  - Probabilistic state-action transitions: $p(s_i \mid a, s_j)$
  - Reward for each state/action pair*: $r(s, a)$
  - Conditional observation probabilities: $p(o \mid s)$
- Belief state
  - Probability distribution over world states: $b(s) = p(s)$
  - Action update rule: $b'(s) = \sum_{s' \in S} p(s \mid a, s') \cdot b(s')$
  - Observation update rule: $b'(s) = p(o \mid s) \cdot b(s)/k$

POMDP as Belief-State MDP

- Equivalent belief-state MDP
  - Each MDP state is a probability distribution (continuous belief state b) over the states of the original POMDP
  - State transitions are products of actions and observations

$$b'(s') = \mathrm{p}(s' \mid a, o, b) = \mathrm{p}(o \mid s', a, b) \cdot \mathrm{p}(s' \mid a, b) / \mathrm{p}(o \mid a, b)$$

$$\mathrm{p}(o \mid s', a, b) = \mathrm{p}(o \mid s')$$

$$\mathrm{p}(s' \mid a, b) = \sum_{s \in S} \mathrm{p}(s' \mid a, s) \cdot b(s)$$

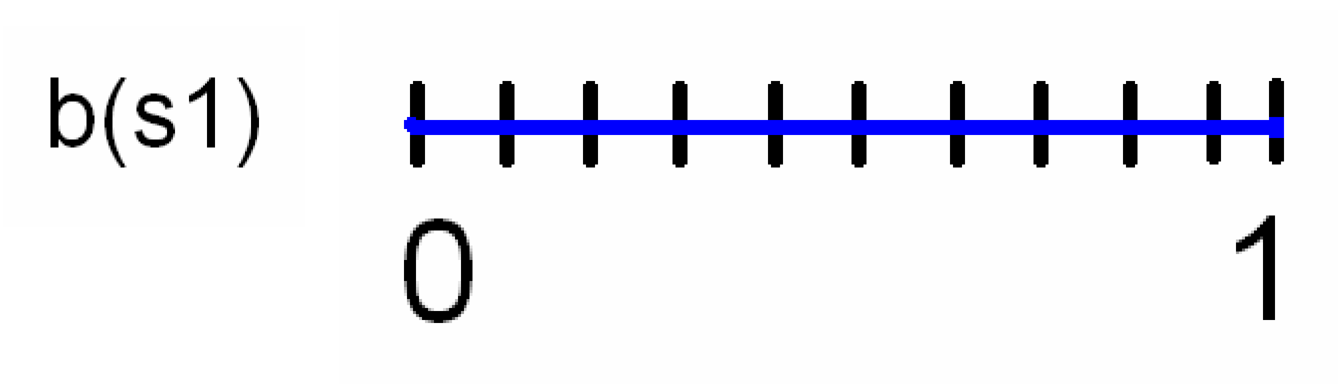
$$\mathrm{p}(o \mid a, b) = \sum_{s' \in S} \mathrm{p}(o \mid s') \cdot \mathrm{p}(s' \mid a, b)$$

  - Rewards are expected rewards of original POMDP

$$\mathrm{R}(a, b) = \sum_{s \in S} \mathrm{r}(a, s) \cdot b(s)$$

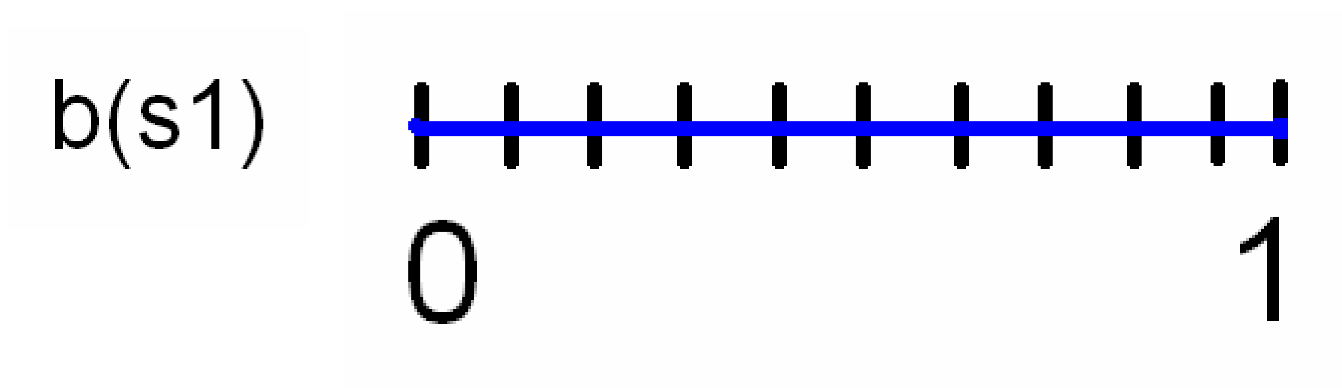
Our First POMDP Solving Algorithm

- Discretize the POMDP belief space
  - Solve the resulting belief-space MDP using
    - Value iteration
    - Policy iteration
    - Any MDP solving technique
- Why might this not work very well?



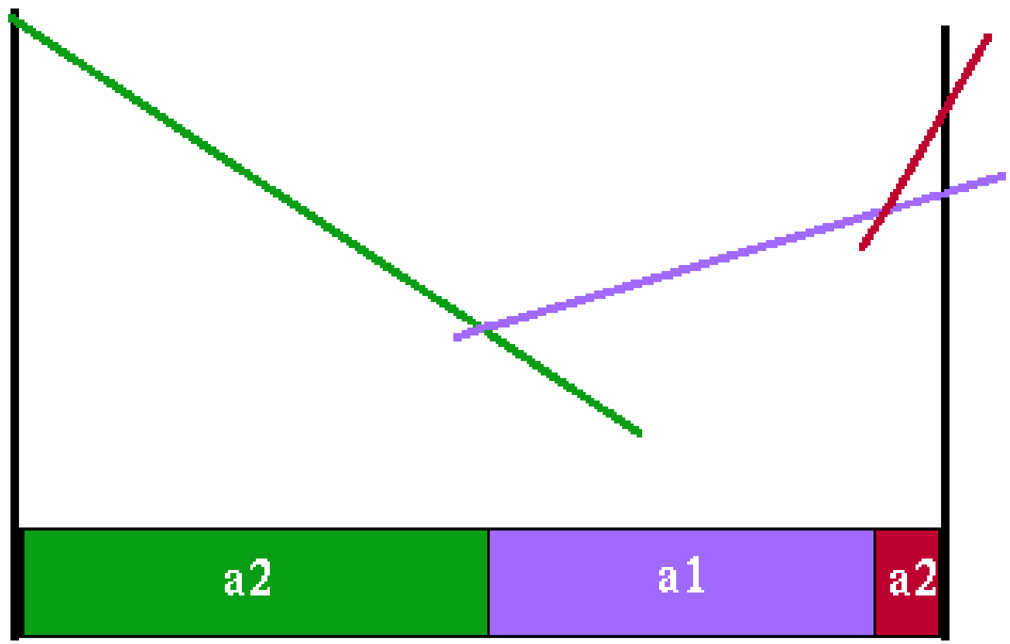
Our First POMDP Solving Algorithm

- Discretize the POMDP belief space
  - Solve the resulting belief-space MDP using
    - Value iteration
    - Policy iteration
    - Any MDP solving technique
- This was the best people could do for a while…



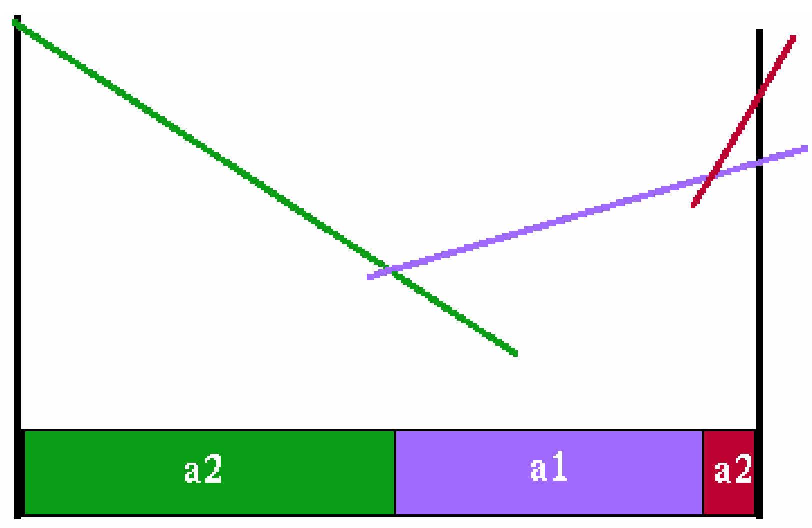
Value Iteration for POMDPs

- Until someone figured out
  - The value function of POMDPs can be represented as max of linear segments
    - Each vector typically called "alpha vector": $\alpha_i \cdot b$
    - This is piecewise-linear-convex (let's think about why)



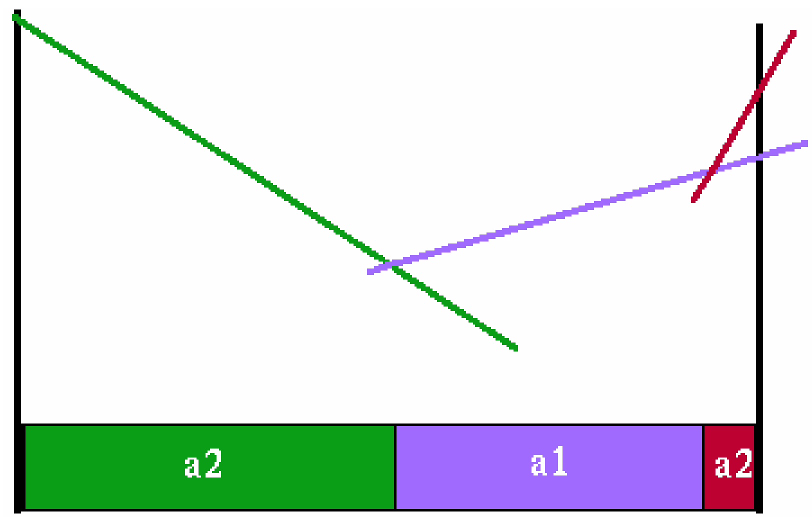
Value Iteration for POMDPs

- The value function of POMDPs can be represented as max of linear segments
  - This is piecewise-linear-convex (let's think about why)
    - Convexity
      - State is known at edges of belief space
      - Can always do better with more knowledge of state



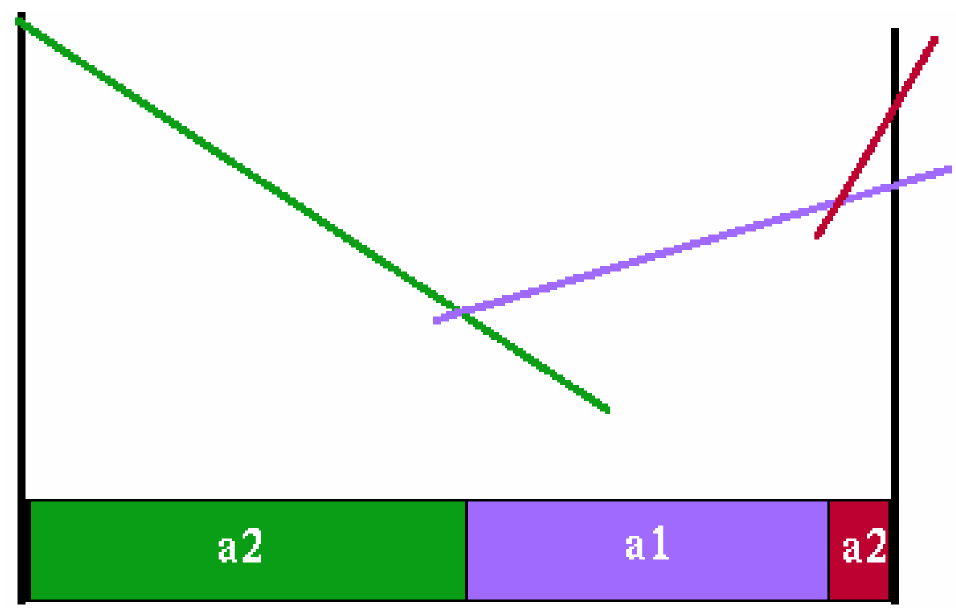
Value Iteration for POMDPs

- The value function of POMDPs can be represented as max of linear segments
  - This is piecewise-linear-convex (let's think about why)
    - Convexity
      - State is known at edges of belief space
      - Can always do better with more knowledge of state
    - Linear segments
      - Horizon 1 segments are linear (belief times reward)
      - Horizon n segments are linear combinations of Horizon n-1 segments (more later)



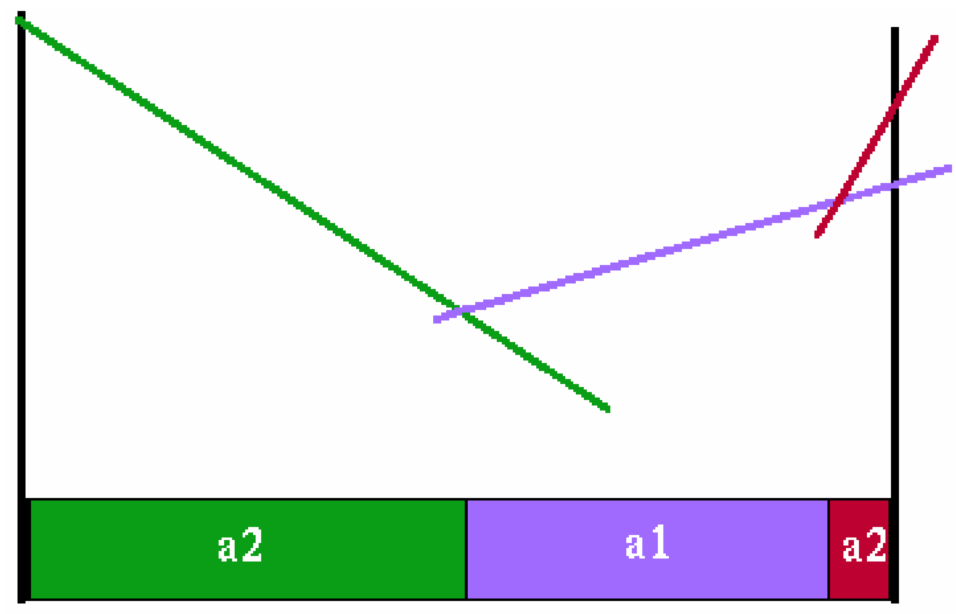
Value Iteration for POMDPs

- The value function of POMDPs can be represented as max of linear segments
  - This leads to a method of exact value iteration for POMDPs



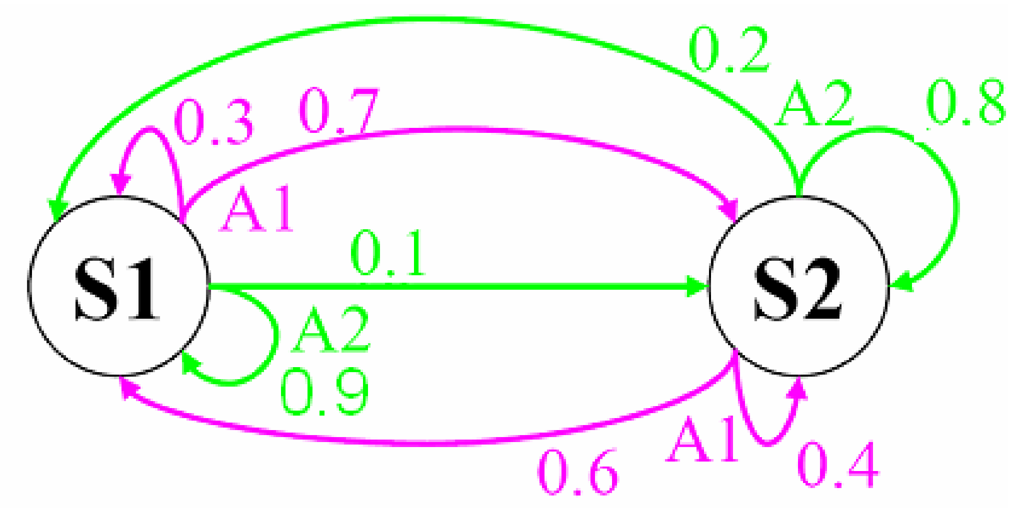
Value Iteration for POMDPs

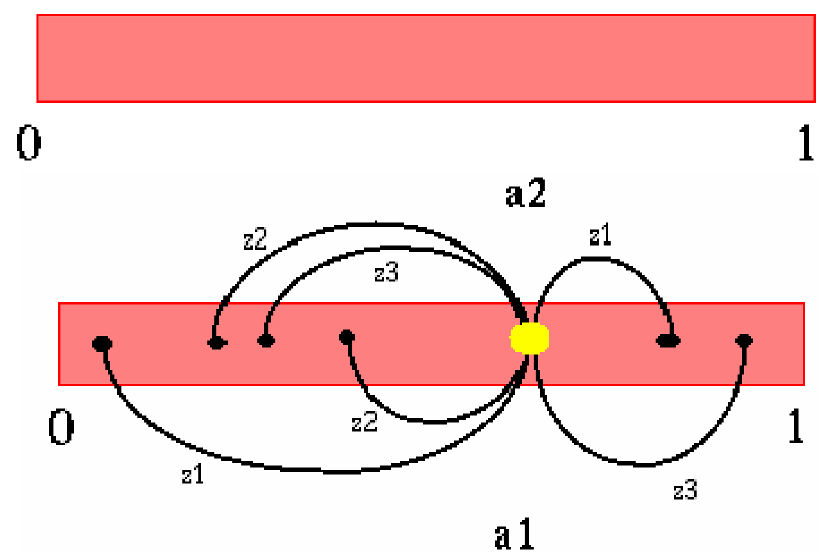
- Basic idea
  - Calculate value function vectors for each action (horizon 1 value function)
    - Keep in mind we need to account for observations
  - Continue looking forward (horizon 2, horizon 3)
  - Iterate until convergence
- Equations coming later



Value Iteration for POMDPs

- Example POMDP for value iteration
  - Two states: s1, s2
  - Two actions: a1, a2
  - Three observations: z1, z2, z3
  - Positive rewards in both states: R(s1) = 1.0, R(s2) = 1.5

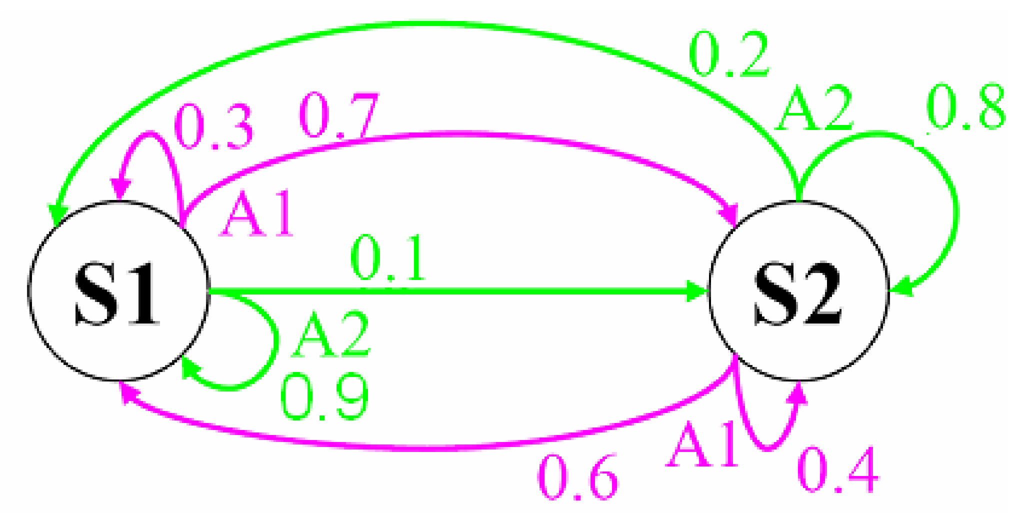




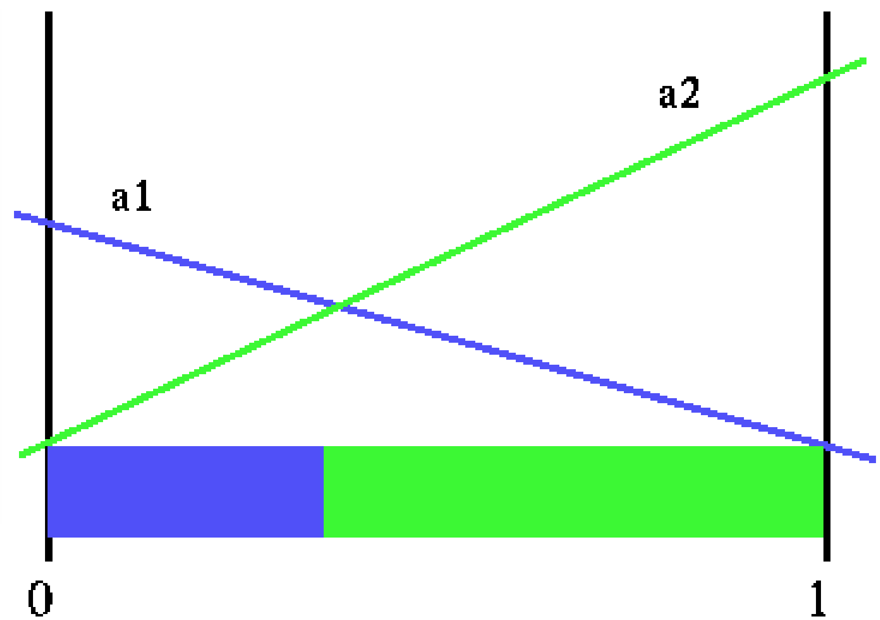
1D belief space for a 2 state POMDP

Value Iteration for POMDPs

- Horizon 1 value function
  - Calculate immediate rewards for each action in belief space



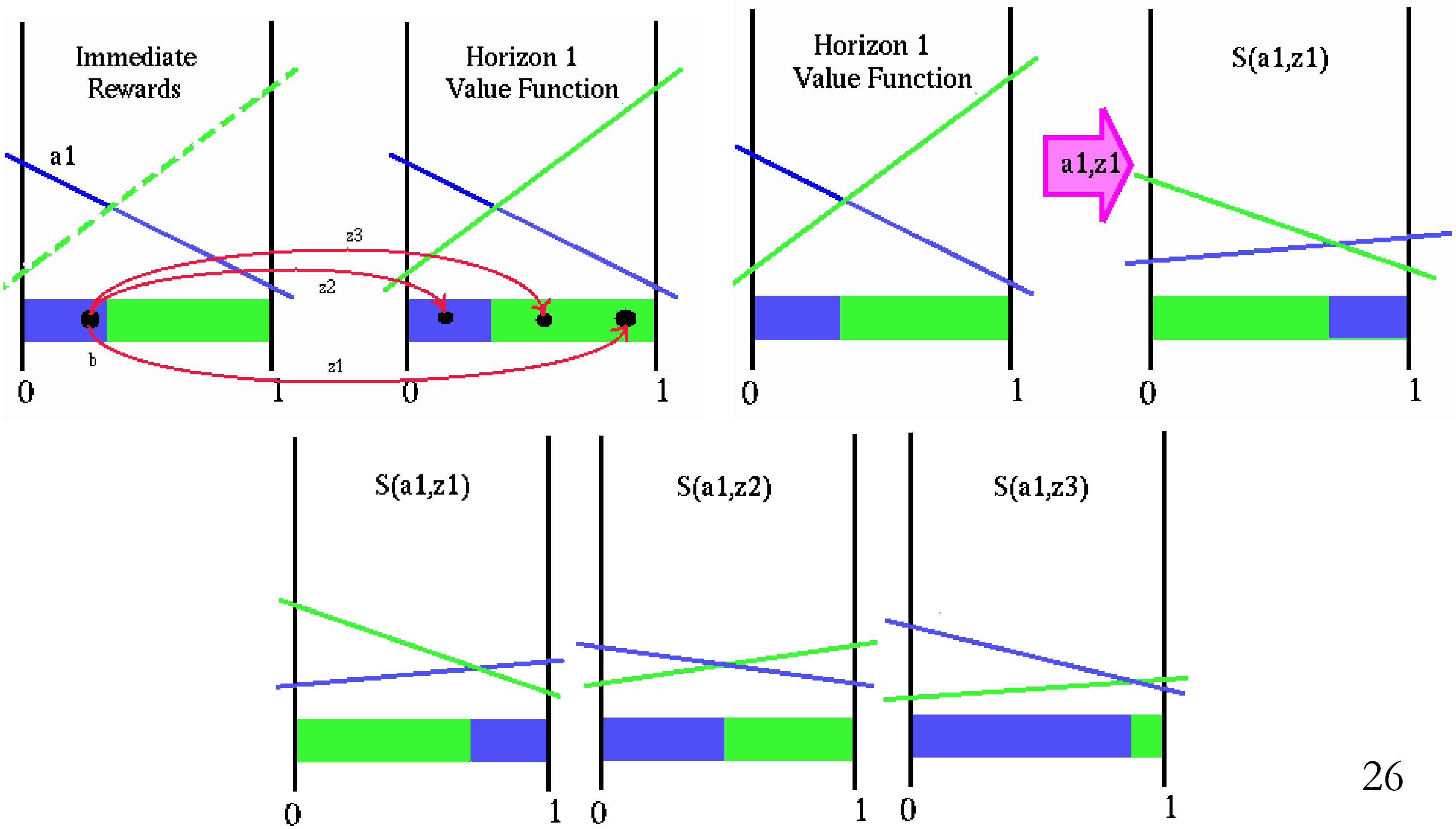
R(s1) = 1.0, R(s2) = 1.5



Horizon 1 value function

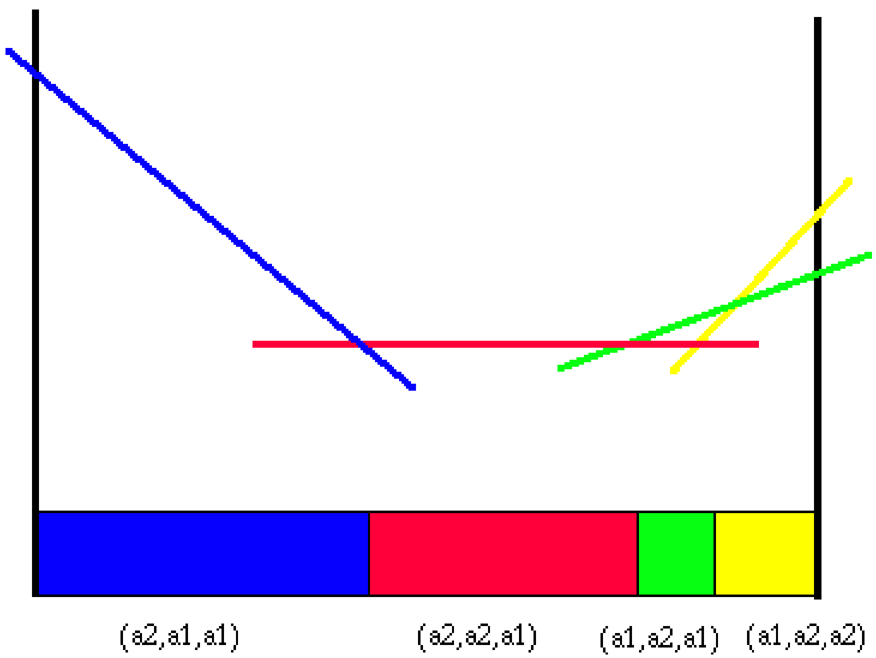
Value Iteration for POMDPs

- Need to transform value function with observations

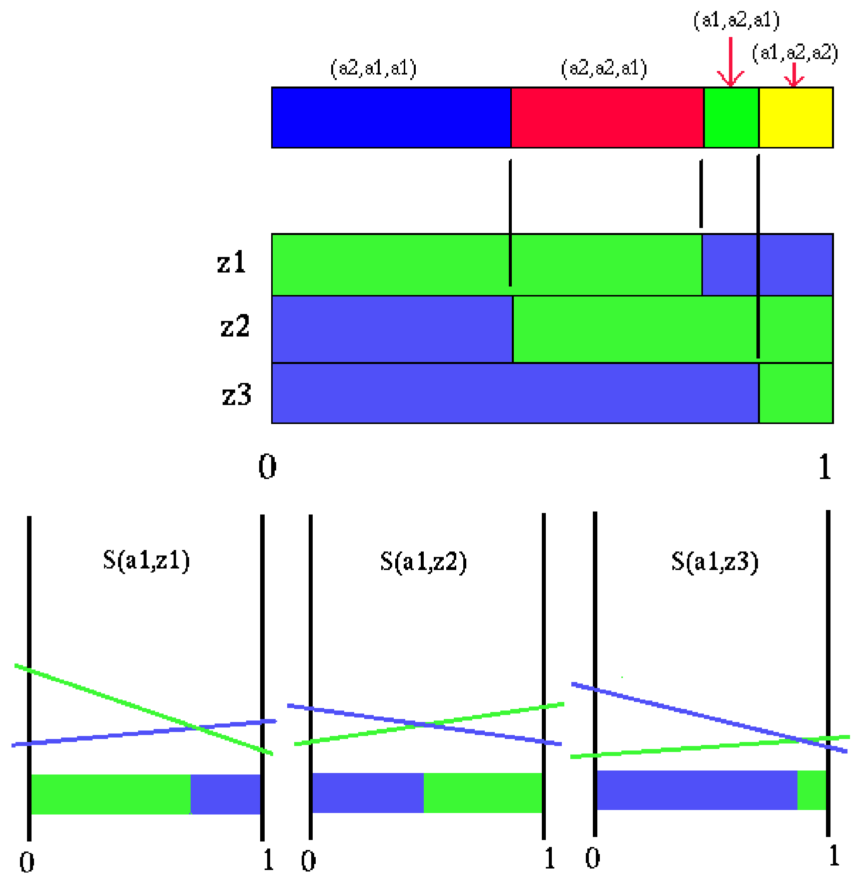


Value Iteration for POMDPs

- Each action from horizon 1 yields new vectors from the transformed space

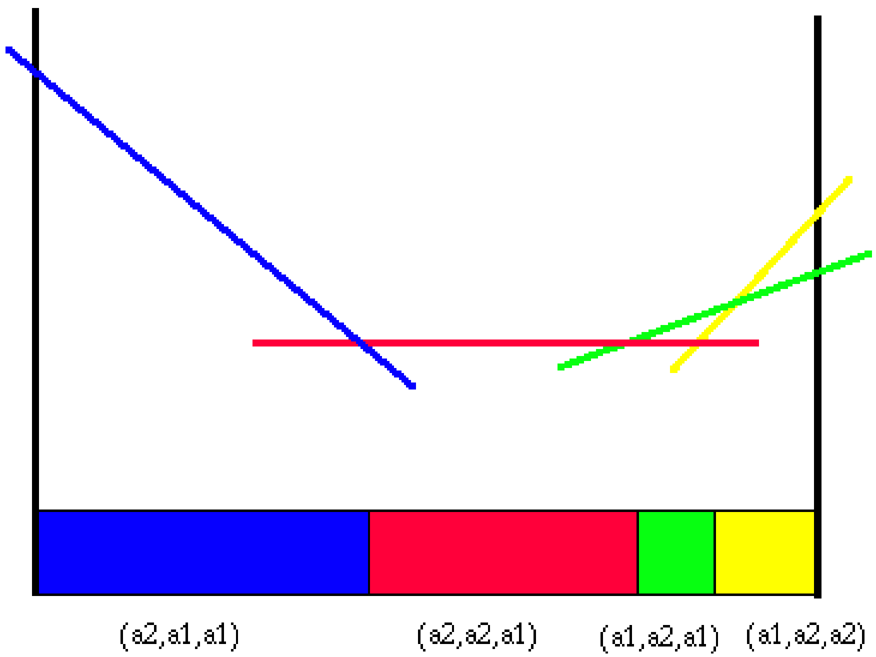


Value function and partition for taking action a1 in step 1

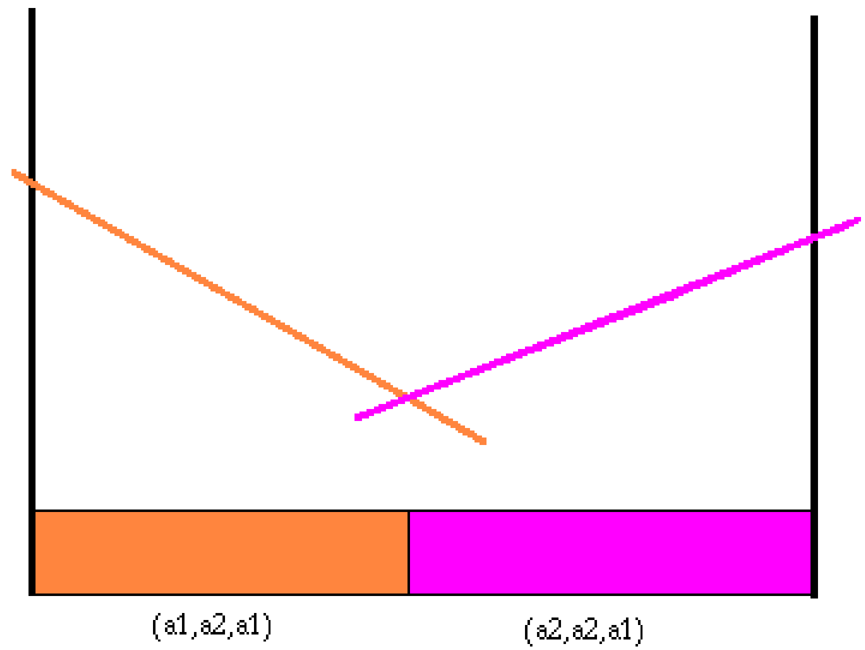


Value Iteration for POMDPs

- Each action from horizon 1 yields new vectors from the transformed space



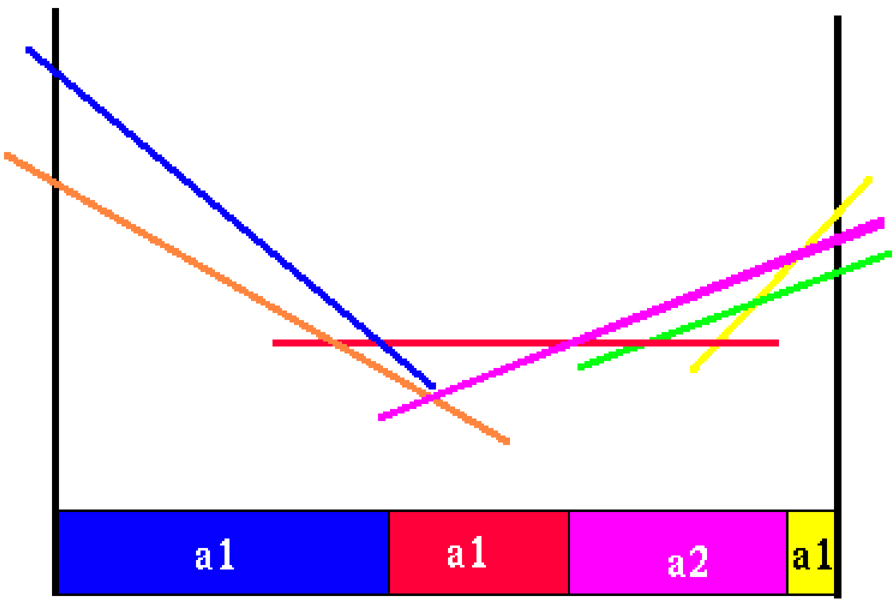
Value function and partition for taking action a1 in step 1



Value function and partition for taking action a2 in step 1

Value Iteration for POMDPs

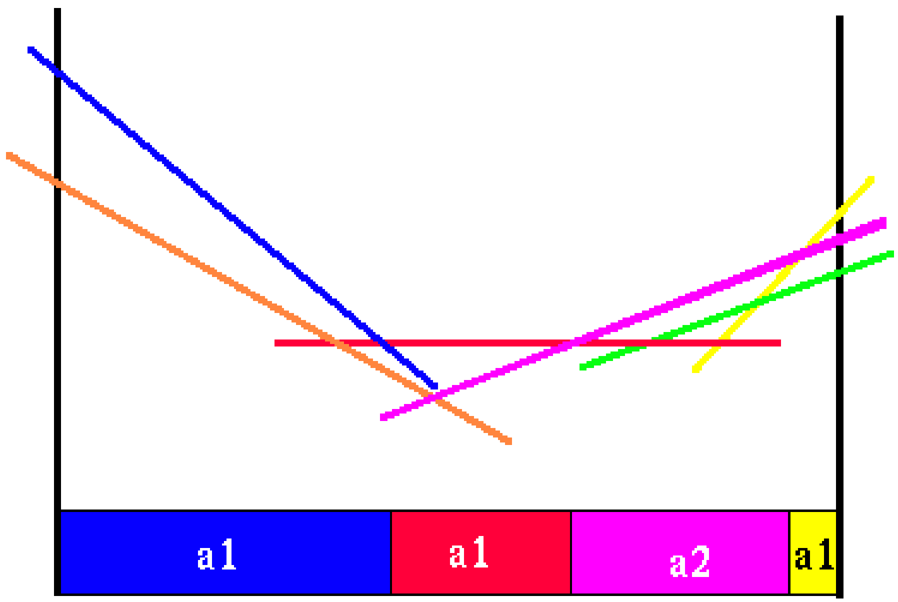
- Combine vectors to yield horizon 2 value function



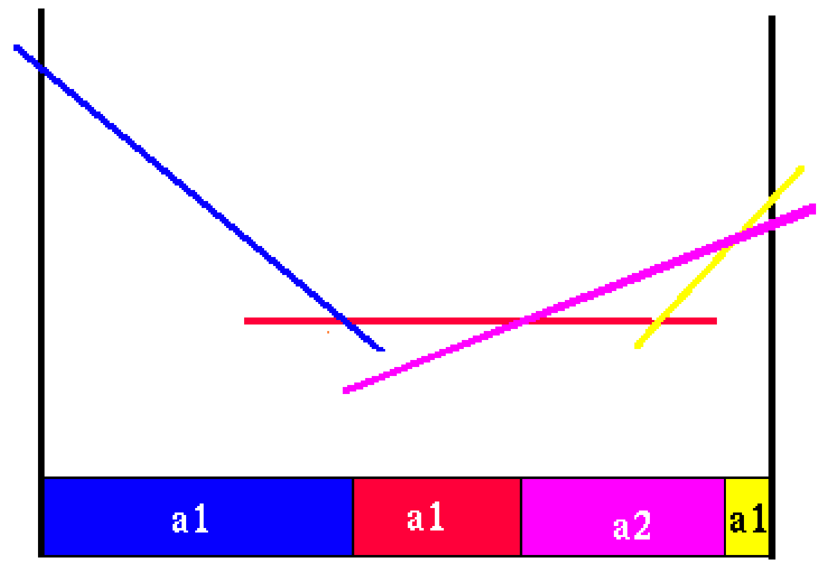
Combined a1 and a2 value functions

Value Iteration for POMDPs

- Combine vectors to yield horizon 2 value function (can also prune dominated vectors)



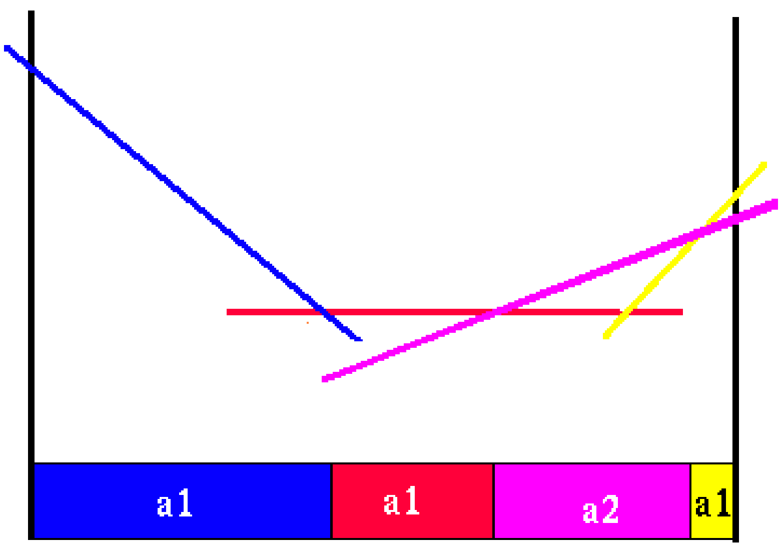
Combined a1 and a2 value functions



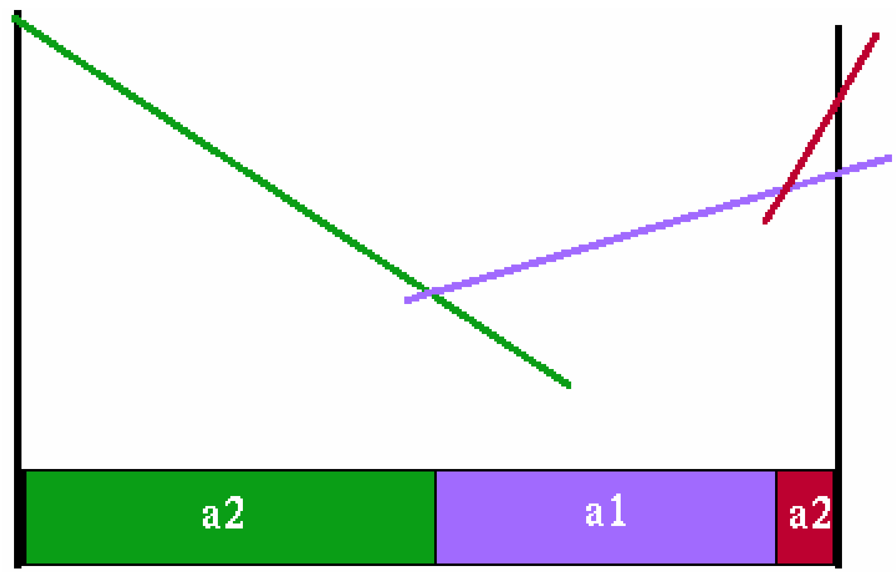
Horizon 2 value function with pruning

Value Iteration for POMDPs

- Iterate to convergence
  - This can sometimes take many steps
- Course reading also gives horizon 3 calculation
  - "POMDPs for Dummies" by Tony Cassandra



Horizon 2 value function with pruning



Horizon 3 value function with pruning

Value Iteration for POMDPs

- Equations for backup operator: V = HV'
- Step 1:
  - Generate intermediate sets for all actions and observations (non-linear terms cancel)

$$\Gamma^{a,*} \leftarrow \alpha^{a,*}(s) = R(s,a)$$

$$\Gamma^{a,o} \leftarrow \alpha_i^{a,o}(s) = \gamma \sum_{s' \in S} T(s,a,s')\Omega(o,s',a)\alpha_i'(s'), \forall \alpha_i' \in V'$$

- Step 2:
  - Take the cross-sum over all observation

$$\Gamma^a = \Gamma^{a,*} \oplus \Gamma^{a,o_1} \oplus \Gamma^{a,o_2} \oplus \ldots$$

- Step 3:
  - Take the union of resulting sets

$$V = \cup_{a \in A} \Gamma^a$$

Value Iteration for POMDPs

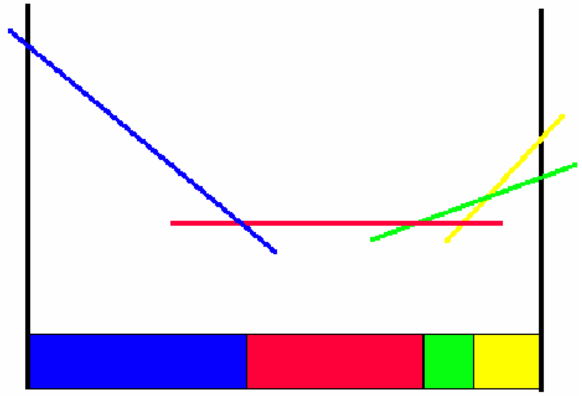
- After all that…
- The good news
  - Value iteration is an exact method for determining the value function of POMDPs
  - The optimal action can be read from the value function for any belief state
- The bad news
  - Guesses?

Value Iteration for POMDPs

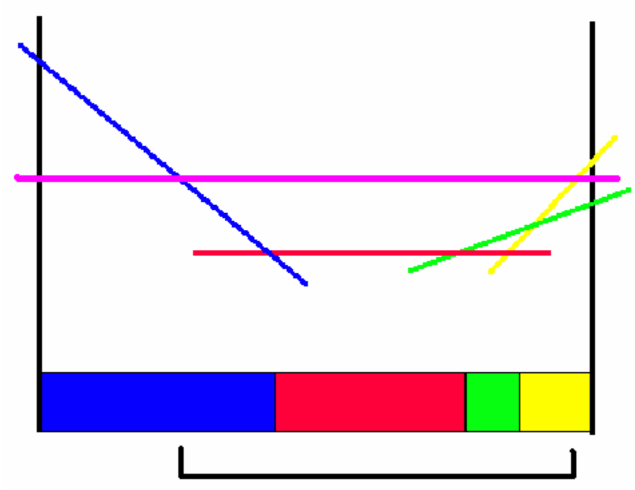
- After all that…
- The good news
  - Value iteration is an exact method for determining the value function of POMDPs
  - The optimal action can be read from the value function for any belief state
- The bad news
  - Time complexity of solving POMDP value iteration is exponential in:
    - Actions *and* observations
  - Dimensionality of the belief space grows with number of states

The Witness Algorithm (Littman)

- A ***Witness*** is a Counter-Example
  - Idea: Find places where the value function is suboptimal
  - Operates action-by-action and observation-by-observation to build up value (alpha) vectors
- **Algorithm**
  - Start with value vectors for known (“corner”) states
  - Define a linear program (based on Bellman’s equation) that finds a point in the belief space where the value of the function is incorrect
  - Add a new vector (a linear combination of the old value function)
  - Iterate



Current value function estimate



Witness region where value function is suboptimal

Policy Iteration for POMDPs

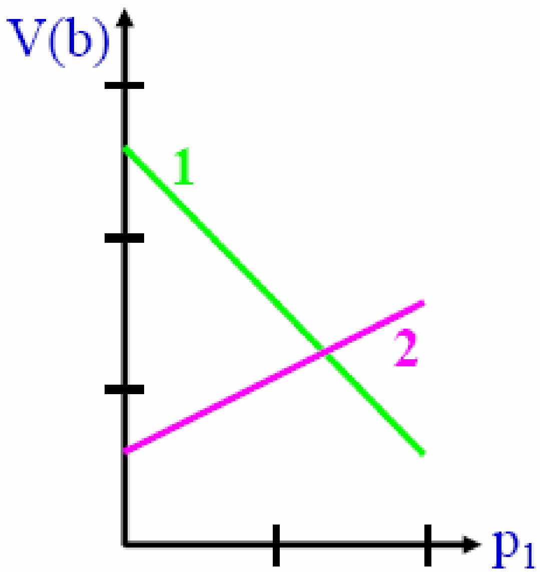
- ***Policy Iteration***
  - Choose a policy
  - Determine the value function, based on the current policy
  - Update the value function, based on Bellman's equation
  - Update the policy and iterate (if needed)

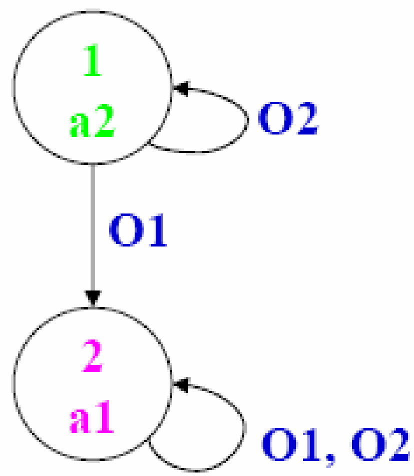
Policy Iteration for POMDPs

- ***Policy Iteration***
  - Choose a policy
  - Determine the value function, based on the current policy
  - Update the value function, based on Bellman's equation
  - Update the policy and iterate (if needed)
- ***Policy Iteration for POMDPs***
  - Original algorithm (Sondik) very inefficient and complex
  - Mainly due to evaluation of value function from policy!
  - Represent policy using finite-state controller (Hansen 1997):
    - Easy to evaluate
    - Easy to update

Policy Iteration for POMDPs (Hansen)

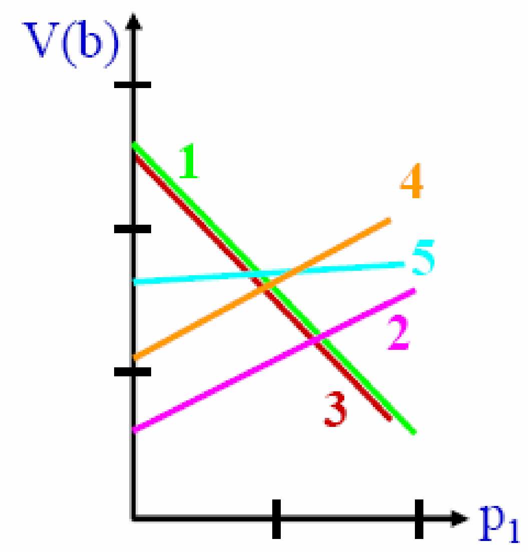
- Key Idea: Represent Policy as Finite-State Controller (***Policy Graph***)
  - Explicitly represents: "do action then continue with given policy"
  - Nodes correspond to vectors in value function
  - Edges correspond to transitions based on observations

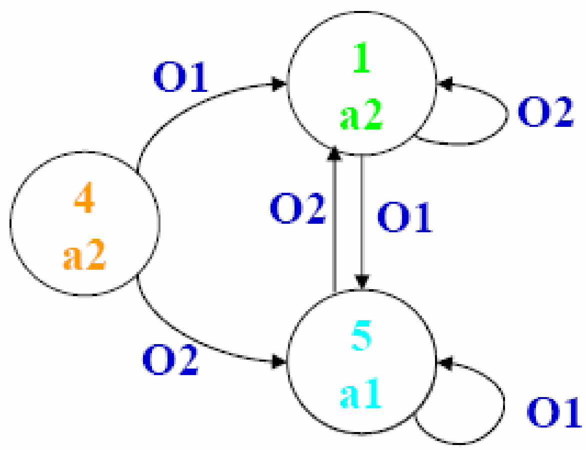




Policy Iteration for POMDPs (Hansen)

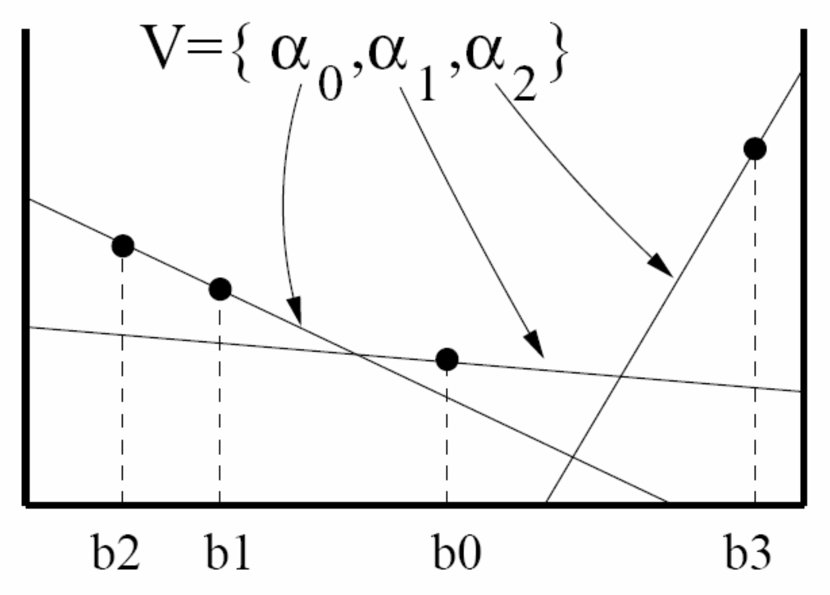
- Determine the value function, based on the current policy
  - Solve system of linear equations
- Update the value function, based on Bellman's equation
  - Can use any standard dynamic-programming method
- Update the policy
  - Ignore new vectors that are dominated by other vectors
  - Add new controller state otherwise





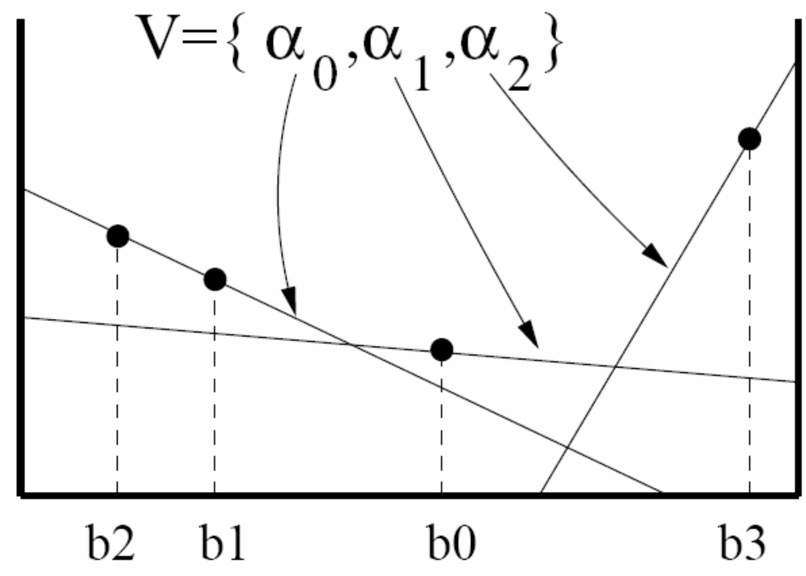
Point-Based Value Iteration (Pineau, Gordon, Thrun)

- Solve POMDP for finite set of belief points
  - Initialize linear segment for each belief point and iterate
- Occasionally add new belief points
  - Add point after a fixed horizon
  - Add points when improvements fall below a threshold



Point-Based Value Iteration (Pineau, Gordon, Thrun)

- Solve POMDP for finite set of belief points
  - Can do point updates in polytime
    - Modify belief update so that one vector is maintained per point
    - Simplified by finite number of belief points
  - Does not require pruning!
    - Only need to check for redundant vectors

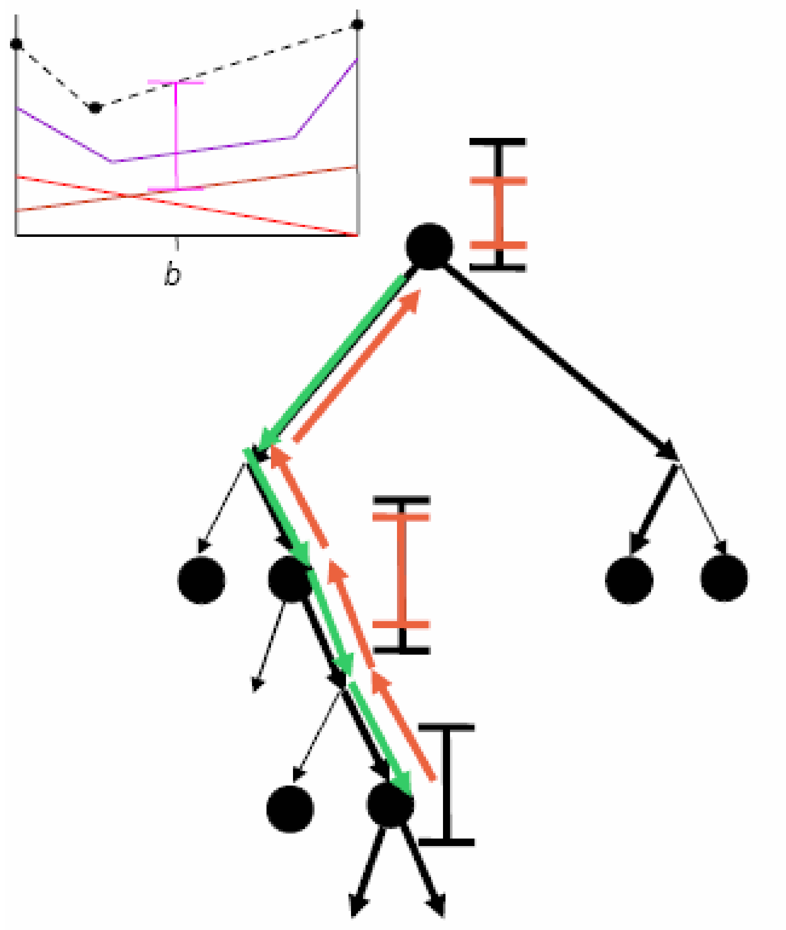


Heuristic Search Value Iteration (Smith and Simmons)

- ***Approximate Belief Space***
  - Deals with only a subset of the belief points
  - Focus on the most relevant beliefs (like point-based value iteration)
  - Focus on the most relevant actions and observations
- ***Main Idea***
  - Value iteration is the dynamic programming form of a tree search
  - Go back to the tree and use heuristics to speed things up
  - But still use the special structure of the value function and plane backups

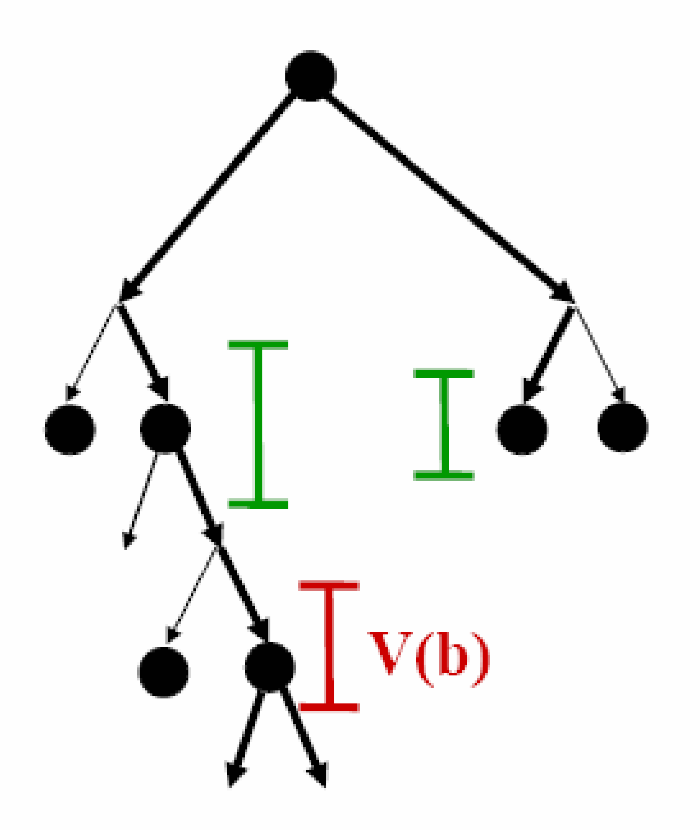
HSVI

- Constraints on Value of Beliefs
  - Lower and upper bounds
  - Initialize upper bound to QMDP;
  - Lower bound to “always a”
- Explore the “Horizon” Tree
  - Back up lower and upper bound to further constrain belief values
  - Lower bound is point-based value backups
  - Upper bound is set of points
    - Solve linear program to interpolate (can be expensive)
    - Or use approximate upper bound

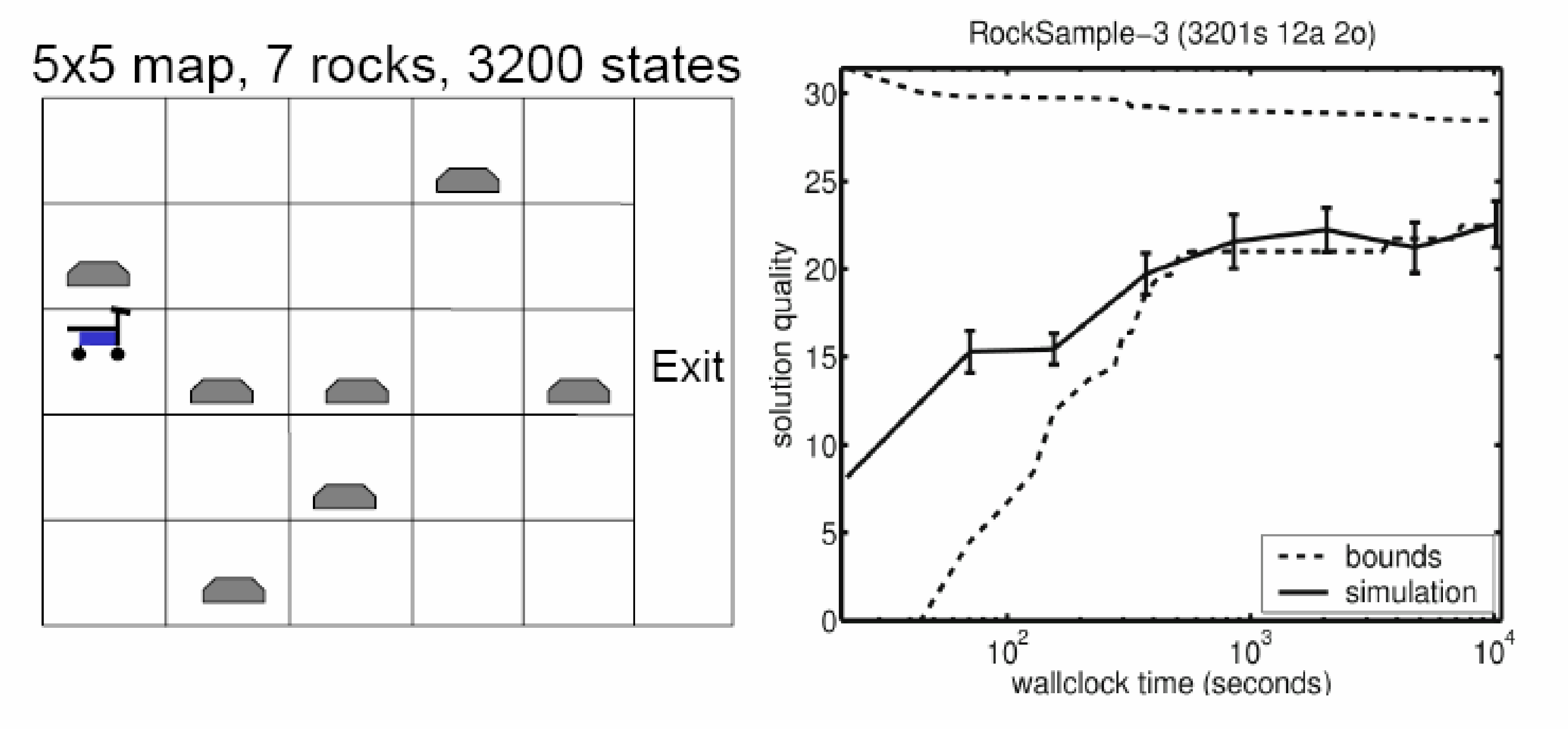


HSVI

- Need to decide:
  - When to terminate search?
    - Minimal gain
      - **width(V(b)) < $\varepsilon\gamma^{-t}$**
  - Which action to choose?
- Highest upper bound:
  - ***argmax$_a$ Q(b, a)***
- Which observation to choose?
  - Reduce excess uncertainty most
    - ***argmax$_o$*** **p(o | b, a)*(width(V(τ(b,a,o)) - $\varepsilon\gamma^{-t+1}$)**



HSVI Results



Greedy Approaches

- Solve Underlying MDP
  - $\pi_{MDP}: S \rightarrow A$; $Q_{MDP}: S \times A \rightarrow \mathfrak{R}$
- Choose Action Based on Current Belief State
  - **"most likely"** $\pi_{MDP}(\text{argmax}_s(b(s)))$
  - **"voting"** $\text{argmax}_a(\Sigma_{s \in S}\, b(s)\delta(a, \pi_{MDP}(s)))$ where
    $\delta(a, b) = (1 \text{ if } a{=}b;\ 0 \text{ otherwise})$
  - **"Q-MDP"** $\text{argmax}_a(\Sigma_{s \in S}\, b(s)\, Q_{MDP}(s, a))$
- Essentially, try to act optimally as if the POMDP were to become observable after the next action
  - ***Cannot plan to do actions just to gain information***

Greedy Approaches

- **"Dual-Mode Control"**
  - Extension to QMDP to allow Information-Gathering Actions
    - Compute entropy ***H(b)*** of belief state
    - If entropy is below a threshold, use a greedy method ***Z(a, b)*** for choosing action
    - If entropy is above a threshold, choose the action that reduces expected entropy the most

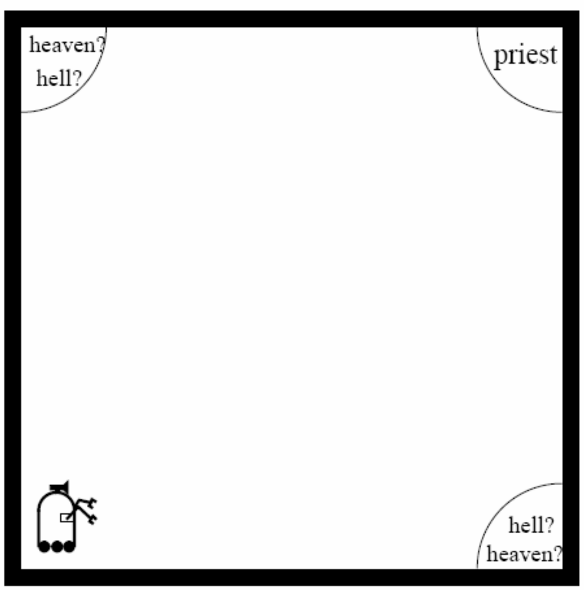
$$EE(a, b) = \sum_{b'} p(b' \mid a, b)\, H(b')$$

$$\pi(s) = \operatorname{argmax}_a Z(a, b) \text{ if } H(b) < t$$

$$\operatorname{argmin}_a EE(a, b) \text{ otherwise}$$

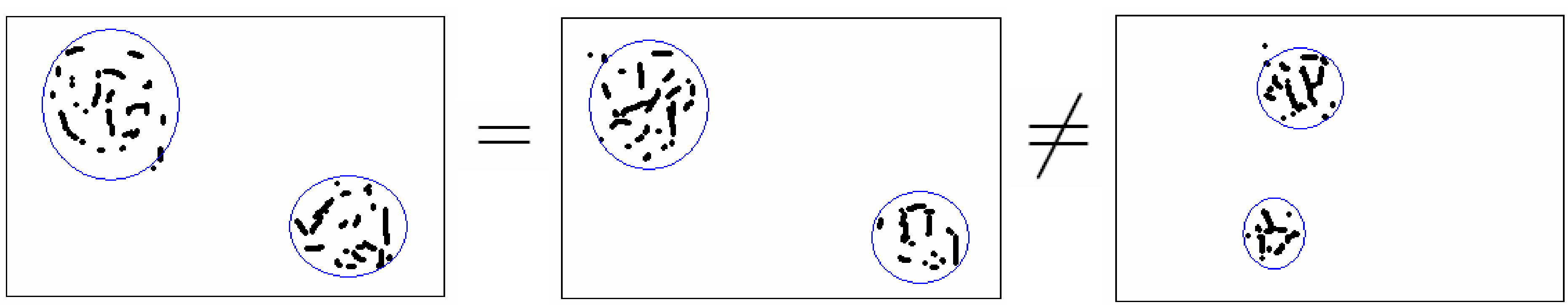
Extensions

- Monte Carlo POMDPs (Thrun)
  - Continuous state and action spaces
  - For example:
    - A holonomic robot traveling on the 2D plane
    - Controlling a robotic gripper
  - Requires approximating belief space and value function with Monte Carlo methods (particle filters)



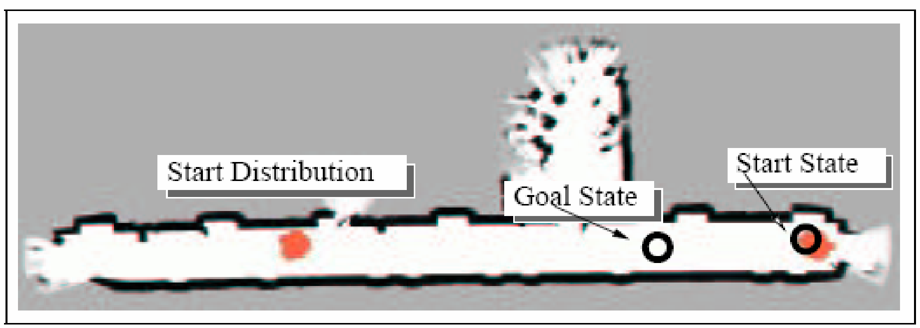
Extensions

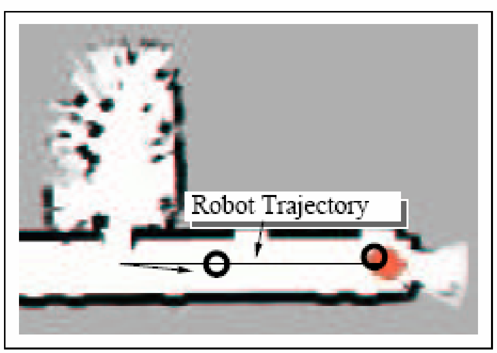
- Monte Carlo POMDPs (Thrun)
  - Continuous state space means infinite dimensional belief space!
  - How do we compare beliefs?
    - Nearest neighbor calculation
  - We can then do value function backups

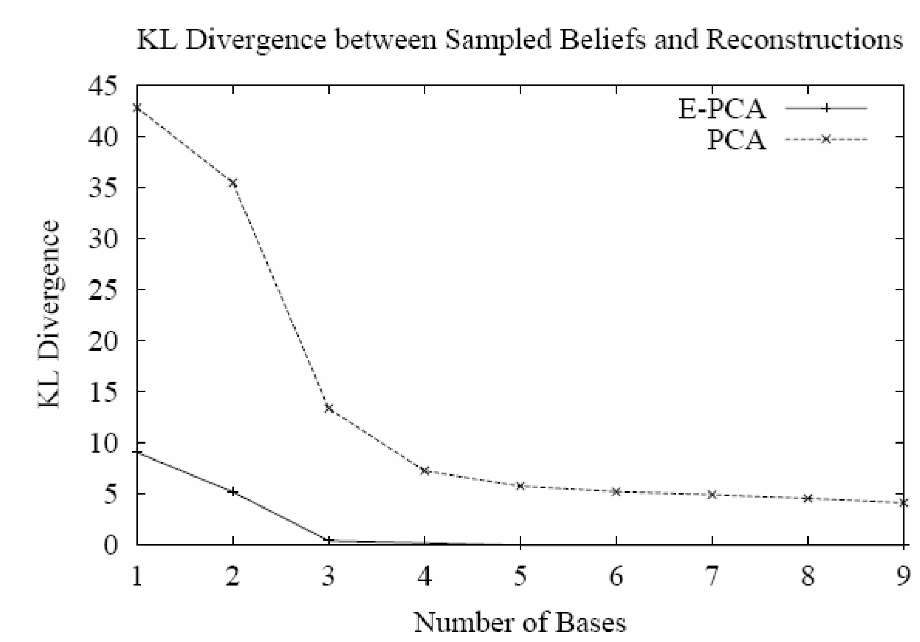


Extensions

- POMDPs with belief-state compression (Roy and Gordon)
  - Approximate belief space using exponential principal component analysis (E-PCA)
  - Reduces dimensionality of belief space
  - Applications to mobile robot navigation

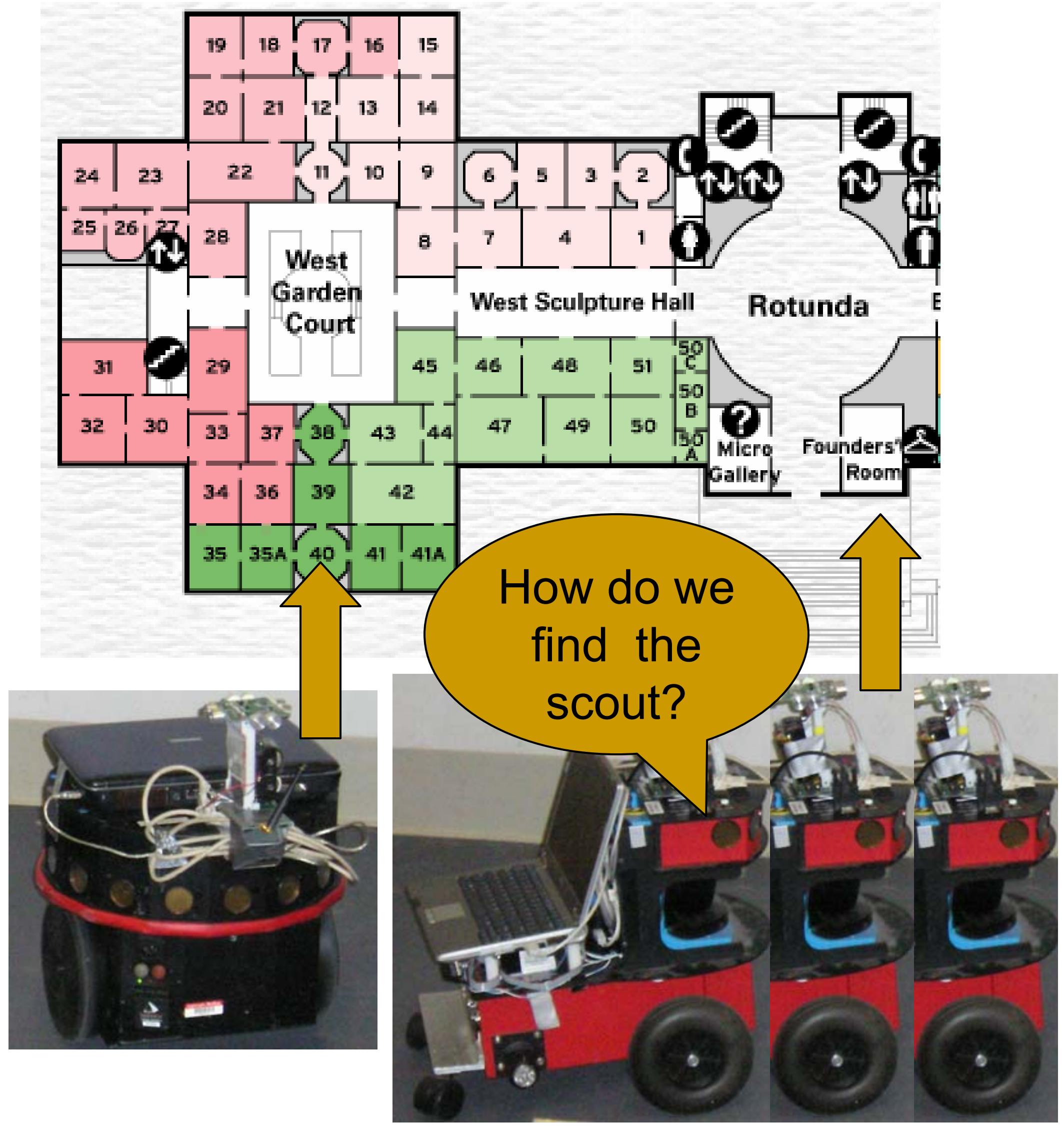






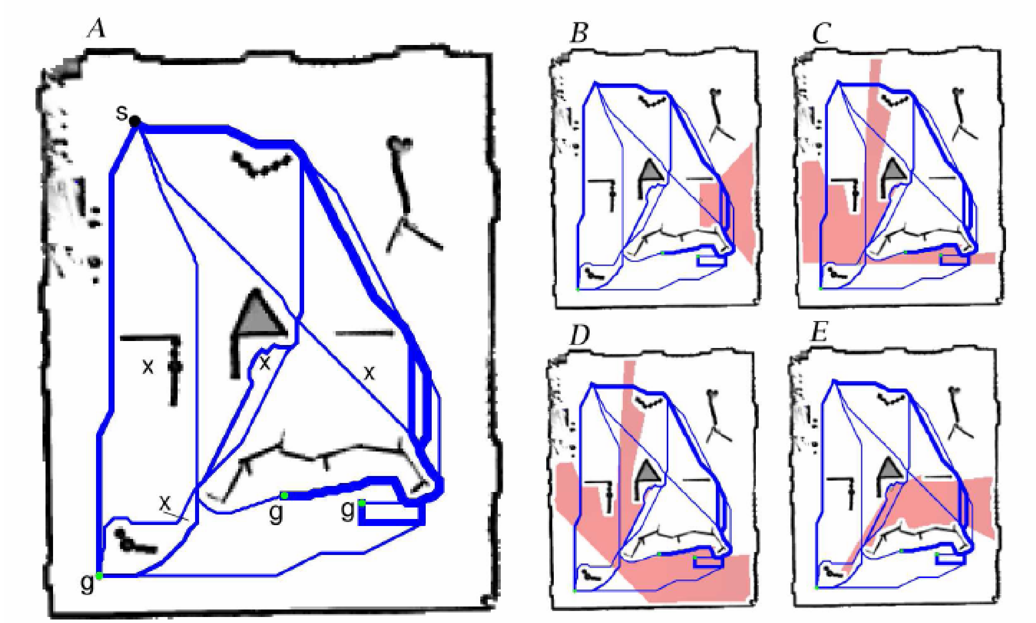
Applications

- Pursuit-Evasion
  - Evader’s state is partially observed
  - Pursuer’s state is known
  - Applied on
    - Graphs
    - Polygonal spaces
    - Indoor environments
  - Multi-agent search (Hollinger and Singh)
    - Sequential allocation
    - Finite-horizon search



Applications

- Sensor placement (McMahan, Gordon, Blum)
  - World is partially observed
  - Can place sensors in world
  - Construct a low-error representation of the world
  - Achieve some task
    - Find an intruder
    - Facilitate “stealthy” movement



Applications

- Games
  - Some games (like poker) have hidden states
  - POMDPs can compute a best response to a fixed opponent policy
  - Solving the full game is a Partially Observable Stochastic Game (POSG)
    - Even harder to solve than a POMDP

Applications

- In most (if not all) applications
  - Size of real-world problems are outside the scope of tractable ***exact*** solutions
  - This is why POMDPs are an active research area…