Probabilistic Planning Background

Zongzhang Zhang

Multi-Agent Systems Labs., School of Computer Science
University of Science and Technology of China

August 13, 2009

Question

What's the relationship between Simulation 2D and POMDP?

- Simulation 2D's server can be considered as POMDP model.
- Partially observable Markov decision processes (POMDPs) provide a principled mathematical framework for planning under uncertainty in both action effects and state observability.
- Existing POMDP algorithms can't solve problems as complex as Simulation 2D.

Question

What's the relationship between Simulation 2D and POMDP?

- Simulation 2D's server can be considered as POMDP model.
- Partially observable Markov decision processes (POMDPs) provide a principled mathematical framework for planning under uncertainty in both action effects and state observability.
- Existing POMDP algorithms can't solve problems as complex as Simulation 2D.

Question

What's the relationship between Simulation 2D and POMDP?

- Simulation 2D's server can be considered as POMDP model.
- Partially observable Markov decision processes (POMDPs) provide a principled mathematical framework for planning under uncertainty in both action effects and state observability.
- Existing POMDP algorithms can't solve problems as complex as Simulation 2D.

Question

What's the relationship between Simulation 2D and POMDP?

- Simulation 2D's server can be considered as POMDP model.
- Partially observable Markov decision processes (POMDPs) provide a principled mathematical framework for planning under uncertainty in both action effects and state observability.
- Existing POMDP algorithms can't solve problems as complex as Simulation 2D.

Outline

1. Deterministic Planning
2. Uncertainty in State Transitions
3. Partial Observability
4. Belief MDP Value Function Structure
5. Prior Research on POMDPs
   - Exact value iteration
   - Point-based value iteration
   - Heuristic search
   - Decentralized POMDPs
6. References

1 Deterministic Planning
2 Uncertainty in State Transitions
3 Partial Observability
4 Belief MDP Value Function Structure
5 Prior Research on POMDPs
- Exact value iteration
- Point-based value iteration
- Heuristic search
- Decentralized POMDPs

6 References

Brief introduction to Probabilistic Planning

Introduction

- Probabilistic planning describes a set of techniques that an intelligent agent can use to choose actions in the face of uncertainty about its environment and the results of its actions.
- Probabilistic means that the techniques model uncertainty using probability theory and select actions in order to maximize expected utility.
- Planning means the techniques can reason about long-term goals that require multiple actions to accomplish, often taking advantage of feedback from the environment to reduce uncertainty and help select actions on the fly.

Brief introduction to Probabilistic Planning

Introduction

- Probabilistic planning describes a set of techniques that an intelligent agent can use to choose actions in the face of uncertainty about its environment and the results of its actions.
- Probabilistic means that the techniques model uncertainty using probability theory and select actions in order to maximize expected utility.
- Planning means the techniques can reason about long-term goals that require multiple actions to accomplish, often taking advantage of feedback from the environment to reduce uncertainty and help select actions on the fly.

Brief introduction to Probabilistic Planning

Introduction

- Probabilistic planning describes a set of techniques that an intelligent agent can use to choose actions in the face of uncertainty about its environment and the results of its actions.
- Probabilistic means that the techniques model uncertainty using probability theory and select actions in order to maximize expected utility.
- Planning means the techniques can reason about long-term goals that require multiple actions to accomplish, often taking advantage of feedback from the environment to reduce uncertainty and help select actions on the fly.

Deterministic planning problem

Indoor navigation task

Consider an indoor navigation task in which a robot must navigate to a goal position at the end of a hallway without colliding with the walls or other obstacles. To start with, assume that the robot has a complete map of the hallway and that it knows at all times both its own position and the position of the goal. Furthermore, assume that to a first approximation we can model the robots position as taking on discrete values in a map grid.

Example 1

Figure 1 shows an example map for such a task. The planning problem is to choose a sequence of one-step motion actions that will cause the robot at the west side of the map to reach the goal at the east side without colliding with walls or obstacles. One such sequence is east, east, north, east, east, south. This is a classical discrete deterministic planning problem.

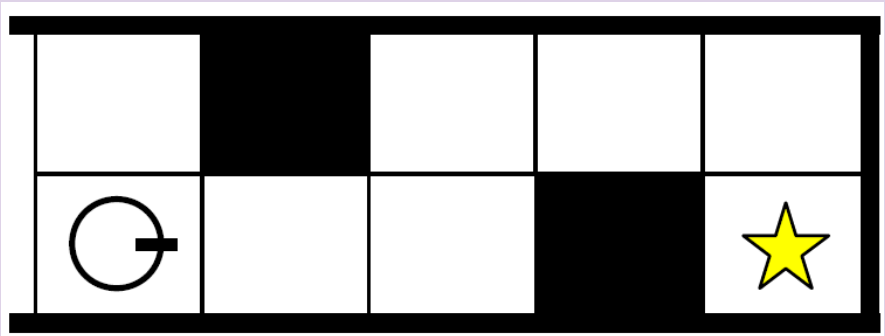

Figure 1: A simple indoor navigation problem

General form of deterministic planning problems

- In general, this class of problem has world states drawn from a finite set $\mathcal{S}$ and actions available to the agent drawn from a finite set $\mathcal{A}$. State changes occur in discrete steps according to a transition function $\mathcal{T} : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$. The world starts in some known state $s_0 \in \mathcal{S}$.
- Under a deterministic world model it is natural to represent the agents policy as a sequence of actions $\pi = a_0, a_1, a_2, \ldots$. The corresponding world states the agent will reach can be calculated using the transition function: $s_1 = \mathcal{T}(s_0, a_0), s_2 = \mathcal{T}(s_1, a_1)$, etc.

General form of deterministic planning problems

- In general, this class of problem has world states drawn from a finite set $\mathcal{S}$ and actions available to the agent drawn from a finite set $\mathcal{A}$. State changes occur in discrete steps according to a transition function $\mathcal{T} : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$. The world starts in some known state $s_0 \in \mathcal{S}$.
- Under a deterministic world model it is natural to represent the agents policy as a sequence of actions $\pi = a_0, a_1, a_2, \ldots$. The corresponding world states the agent will reach can be calculated using the transition function: $s_1 = \mathcal{T}(s_0, a_0), s_2 = \mathcal{T}(s_1, a_1)$, etc.

Definition: finite horizon

Sometimes there is a pre-specified maximum policy length; this is called a finite horizon problem, and the maximum number of actions is called the horizon length, denoted h.

Definition: infinite horizon

Infinite-horizon problems have no pre-specified maximum sequence length, in which case we write $h = \infty$.

Definition: goal states

There may also be a set $\mathcal{G} \subseteq \mathcal{S}$ of absorbing goal states, also called terminal states, such that the action sequence ends when a terminal state is reached, regardless of the horizon.

Definition: finite horizon

Sometimes there is a pre-specified maximum policy length; this is called a finite horizon problem, and the maximum number of actions is called the horizon length, denoted h.

Definition: infinite horizon

Infinite-horizon problems have no pre-specified maximum sequence length, in which case we write $h = \infty$.

Definition: goal states

There may also be a set $\mathcal{G} \subseteq \mathcal{S}$ of absorbing goal states, also called terminal states, such that the action sequence ends when a terminal state is reached, regardless of the horizon.

Definition: finite horizon

Sometimes there is a pre-specified maximum policy length; this is called a finite horizon problem, and the maximum number of actions is called the horizon length, denoted h.

Definition: infinite horizon

Infinite-horizon problems have no pre-specified maximum sequence length, in which case we write $h = \infty$.

Definition: goal states

There may also be a set $\mathcal{G} \subseteq \mathcal{S}$ of absorbing goal states, also called terminal states, such that the action sequence ends when a terminal state is reached, regardless of the horizon.

Definition: preference level

There are many ways to formulate the planning problem. One might wish to find any action sequence that achieves a goal state, or the shortest such action sequence, or the lowest cost sequence (if actions have different costs). More generally, any of these preferences can be captured by specifying a utility function U that maps the sequence of actions and corresponding states to a numeric score, such that larger scores indicate preferred plans.

$$\textit{preference level} = U(h, \pi, s_0) = U(s_0, a_0, s_1, a_1, \ldots, s_{h-1}, a_{h-1}) \quad (1)$$

Then optimal control means selecting a policy π that maximizes $U(h, \pi, s_0)$.

Weighted sum of immediate reward values

We focus on a restricted class of problems for which the utility function can be expressed as a weighted sum of immediate reward values. Immediate rewards are specified by a function $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ and

$$U(h, \pi, s_0) := \sum_{t=0}^{h-1} \gamma^t R(s_t, a_t), \tag{2}$$

where $\gamma \in (0, 1]$ is called the discount factor. Multiplying the reward at step t by γ^t serves to place more weight on the earlier rewards in the sequence. If $\gamma < 1$ we say the problem is discounted; if $\gamma = 1$ all time steps have equal weight and the problem is undiscounted. Discounting is sometimes motivated by real world effects such as interest rates, but most often it is used as a convenient way of ensuring that the reward sum converges in infinite-horizon problems.

1 Deterministic Planning

2 Uncertainty in State Transitions

3 Partial Observability

4 Belief MDP Value Function Structure

5 Prior Research on POMDPs
- Exact value iteration
- Point-based value iteration
- Heuristic search
- Decentralized POMDPs

6 References

Example 2

A robots actions frequently have unexpected outcomes. For example, a robot attempting to move using dead reckoning will often suffer from position errors due to slippage. We can add an (exaggerated) version of this error to the original navigation domain by assuming that each motion action has only a 50% chance of achieving its desired effect, and leaves the robot in the same cell the other 50% of the time. If the robots move is blocked by a wall or other obstacle, it fails 100% of the time.

Example 2 (cont'd)

Figure 2 shows this kind of uncertainty graphically. The result of applying the east action can no longer be predicted with certainty; instead, there are two possible outcomes with associated probabilities. However, for the time being we assume that after the action is completed the robot learns with certainty which outcome actually occurred. (Perhaps it periodically receives accurate position information from radio beacons in the hallway.)

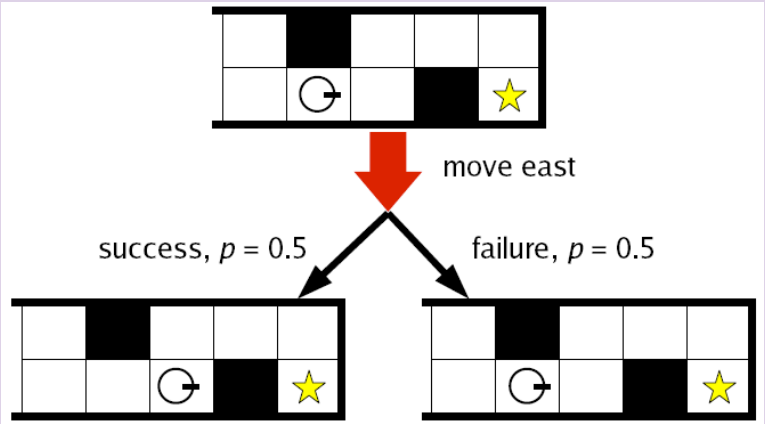

Figure 2: Navigation problem with uncertainty in state transitions

General form of Uncertainty in State Transitions

In general, this model requires a new type of transition function $T : \mathcal{S} \times \mathcal{A} \rightarrow \prod(\mathcal{S})$ that maps a state/action pair to a probability distribution of possible next states. $T(s, a, s_0)$ denotes the probability of transitioning from s to s_0 when action a is applied. Note that when T is specified in this form the system is Markovian, meaning that its future behavior is conditionally independent of the history of states and actions given the current state s_t. For this reason the model is called a Markov decision process (MDP).

Graphical model for an MDP

Figure 3 shows the graphical model for an MDP. Square nodes in a graphical model represent variables that are controlled by the agent. Circular nodes represent uncontrolled dependent variables. Directed edges in the graph are used to indicate dependence relationships between variables. Note the chain structure of the MDP graphical model, which derives from the Markovian structure.

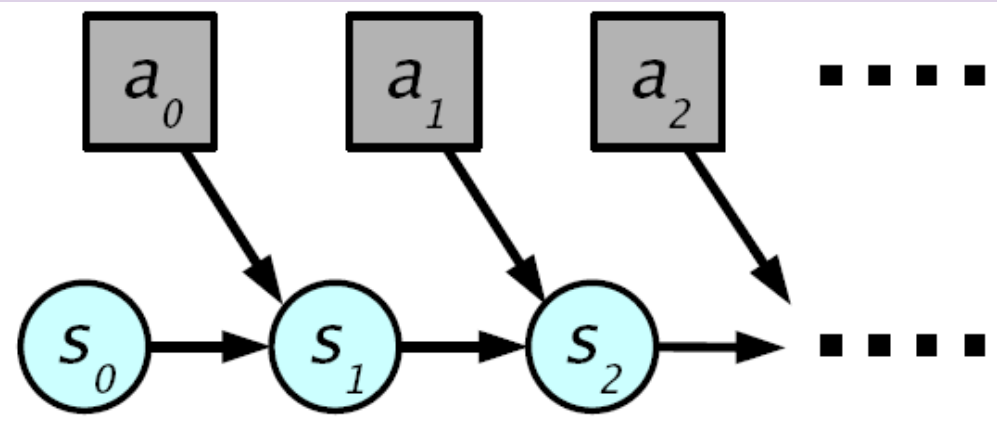

Figure 3: Graphical model for an MDP

Preference level

According to the decision-theoretic definition of rationality, in the presence of uncertainty the agent should choose the policy that maximizes expected utility

$$\textit{preference level} = E[U(h, \pi, s_0)] = E[\sum_{t=0}^{h-1} \gamma^t R(s_t, a_t)]. \quad (3)$$

1 Deterministic Planning
2 Uncertainty in State Transitions
3 Partial Observability
4 Belief MDP Value Function Structure
5 Prior Research on POMDPs
- Exact value iteration
- Point-based value iteration
- Heuristic search
- Decentralized POMDPs

6 References

Introduction

In the last section we rather unrealistically assumed that the navigating robot received perfectly accurate position information after every time step. In this section, we relax that assumption. In particular, suppose the robot cannot tell how far along the hallway it has traveled. Instead, it has a noisy obstacle sensor that nominally returns an obstacle reading when the map cell to the east of the robot is blocked and a clear reading otherwise, but gives an incorrect reading 10% of the time.

Example 3

As figure 4 shows, now we have the additional complication of the noisy observation. Taken together there are four possible results, labeled condition A (success, obstacle observation), B (success, clear observation), C (failure, clear observation), and D (failure, obstacle observation).

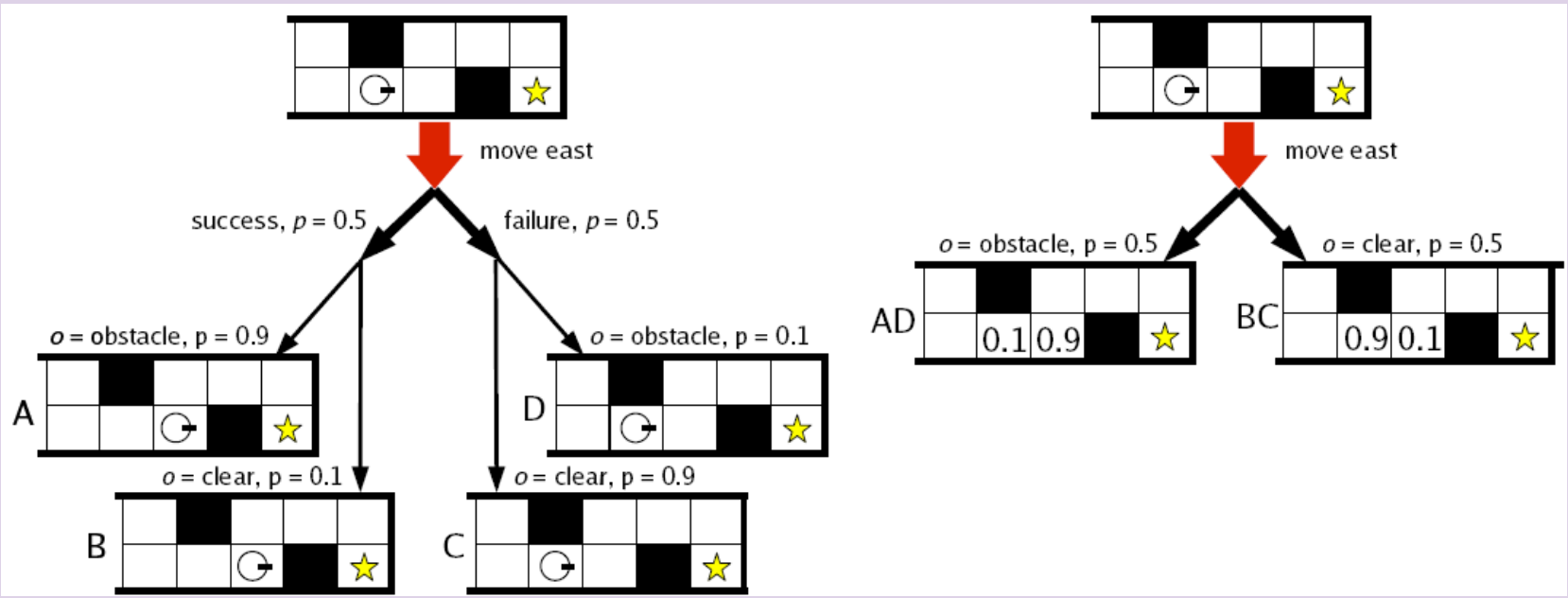

Figure 4: (left) Fully expanded view of state transition.
(right) Information available to the robot.

Example 3 (cont'd)

- Unfortunately, because the robot receives only the observation, it cannot distinguish between conditions A and D, nor between conditions B and C. As a result, its available information is better represented by the transition diagram on the right. Conditions A and D have been combined into condition AD, and B and C have been combined into BC.
- Looking more closely at condition AD, we see that the robot cannot infer its position with certainty, since condition A and condition D have the robot in different positions. Instead, the likelihoods of possible positions of the rover are marked on the map; the values correspond to the relative likelihood of condition A and condition D.

Example 3 (cont'd)

- Unfortunately, because the robot receives only the observation, it cannot distinguish between conditions A and D, nor between conditions B and C. As a result, its available information is better represented by the transition diagram on the right. Conditions A and D have been combined into condition AD, and B and C have been combined into BC.
- Looking more closely at condition AD, we see that the robot cannot infer its position with certainty, since condition A and condition D have the robot in different positions. Instead, the likelihoods of possible positions of the rover are marked on the map; the values correspond to the relative likelihood of condition A and condition D.

partially observable Markov decision process

- This type of model is called a partially observable Markov decision process (POMDP). In general, a POMDP is an MDP extended to include an observation model. Rather than receiving complete state information s_t after each step of execution, in a POMDP model the agent is assumed to receive a noisy observation o_t.
- POMDPs model observations probabilistically. The set of possible observations is a discrete set $\mathcal{O}$, and each observation carries information about the preceding action and current state according to the noisy observation function $O : \mathcal{A} \times \mathcal{S} \rightarrow \prod(\mathcal{O})$, defined such that

$$O(a, s', o) := Pr(O_{t+1} = o | a_t = a, s_{t+1} = s') \quad (4)$$

partially observable Markov decision process

- This type of model is called a partially observable Markov decision process (POMDP). In general, a POMDP is an MDP extended to include an observation model. Rather than receiving complete state information s_t after each step of execution, in a POMDP model the agent is assumed to receive a noisy observation o_t.
- POMDPs model observations probabilistically. The set of possible observations is a discrete set $\mathcal{O}$, and each observation carries information about the preceding action and current state according to the noisy observation function $O : \mathcal{A} \times \mathcal{S} \rightarrow \prod(\mathcal{O})$, defined such that

$$O(a, s', o) := Pr(O_{t+1} = o | a_t = a, s_{t+1} = s') \qquad (4)$$

Belief

The agent is also assumed to have a probability distribution $b_0 \in \prod(\mathcal{S})$ describing the initial state of the system, such that $b_0(s) = Pr(s_0 = s)$. In general, we describe a probability distribution over $\mathcal{S}$ as a belief, and denote the space of beliefs with $\mathcal{B} = \prod(\mathcal{S})$. Beliefs can be thought of as length-$|\mathcal{S}|$ vectors, and because the entries of a belief vector must sum to 1, $\mathcal{B}$ is a simplex of dimension $|\mathcal{S}|$-1 embedded in $\mathbb{R}^{|\mathcal{S}|}$.

Graphical model for a POMDP

Figure 4 shows the graphical model for a POMDP. The chain structure of the state sequence from the MDP is retained, but the agent no longer has direct access to the state information at each time step. Instead, it can infer only uncertain state information from the history of observations.

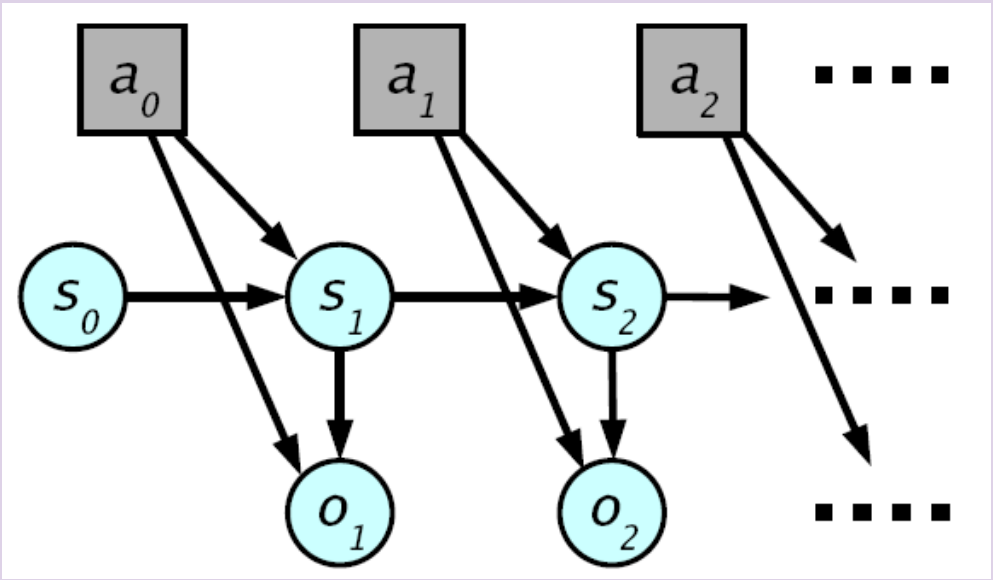

Figure 5: Graphical model for a POMDP

Bayesian updating

- At each time step, the agent can use Bayesian reasoning to generate an updated belief. Let b^{ao} denote the agent's updated belief at the next time step after taking action a and receiving observation o. That is,

$$b^{ao}(s') := Pr(s_{t+1} = s' | b_t = b, a_t = a, o_{t+1} = o) \tag{5}$$

- The updated belief can be calculated as follows:

$$\begin{aligned} b^{ao}(s') = Pr(s'|b,a,o) &= \frac{Pr(s',o|b,a)}{Pr(o|b,a)} \\ &= \frac{Pr(o|a,s')\sum_s Pr(s'|s,a)Pr(s|b)}{\sum_{\bar{s}} Pr(o|a,\bar{s})\sum_s Pr(\bar{s}|s,a)Pr(s|b)} \\ &= \frac{O(a,s',o)\sum_s T(s,a,s')b(s)}{\sum_{\bar{s}} O(a,\bar{s},o)\sum_s T(s,a,\bar{s})b(s)} \end{aligned} \tag{6}$$

Bayesian updating

- At each time step, the agent can use Bayesian reasoning to generate an updated belief. Let b^{ao} denote the agent's updated belief at the next time step after taking action a and receiving observation o. That is,

$$b^{ao}(s') := Pr(s_{t+1} = s' | b_t = b, a_t = a, o_{t+1} = o) \tag{5}$$

- The updated belief can be calculated as follows:

$$\begin{aligned} b^{ao}(s') = Pr(s'|b,a,o) &= \frac{Pr(s',o|b,a)}{Pr(o|b,a)} \\ &= \frac{Pr(o|a,s')\sum_s Pr(s'|s,a)Pr(s|b)}{\sum_{\bar{s}} Pr(o|a,\bar{s})\sum_s Pr(\bar{s}|s,a)Pr(s|b)} \\ &= \frac{O(a,s',o)\sum_s T(s,a,s')b(s)}{\sum_{\bar{s}} O(a,\bar{s},o)\sum_s T(s,a,\bar{s})b(s)} \end{aligned} \tag{6}$$

Policy

- When planning in an MDP model, the agent's policy could be conditioned on a history of states and actions. In a POMDP model, the agent has knowledge only of actions and observations, so a deterministic policy with horizon h has the form

$$a_t = \pi(h, a_0, o_1, a_1, o_2, a_2, \cdot, a_{t-1}, o_t). \tag{7}$$

- With an MDP, the history could be safely discarded given the current state because the system was Markovian. With a POMDP, the agent does not have access to the current state, but it can use the current belief as a sufficient statistic for the history. Thus, given the current belief, the agent gains no advantage from conditioning on history, and it can restrict its reasoning to policies in the form

$$a_t = \pi(h, b_t). \tag{8}$$

Policy

- When planning in an MDP model, the agent's policy could be conditioned on a history of states and actions. In a POMDP model, the agent has knowledge only of actions and observations, so a deterministic policy with horizon h has the form

$$a_t = \pi(h, a_0, o_1, a_1, o_2, a_2, \cdot, a_{t-1}, o_t). \tag{7}$$

- With an MDP, the history could be safely discarded given the current state because the system was Markovian. With a POMDP, the agent does not have access to the current state, but it can use the current belief as a sufficient statistic for the history. Thus, given the current belief, the agent gains no advantage from conditioning on history, and it can restrict its reasoning to policies in the form

$$a_t = \pi(h, b_t). \tag{8}$$

Transform the POMDP into a belief MDP

This change of variables suggests transforming the POMDP into a belief MDP, as follows:

- The POMDP belief simplex $\mathcal{B}$ plays the role of the MDP state set.
- The action set, horizon, and discount factor of the POMDP are used unchanged for the belief MDP.
- The transition function $\widetilde{T}$ of the belief MDP gives the probability of transitioning from one belief to another according to the observation model and belief update rule

$$\begin{aligned}\widetilde{T}(b,a,b') &:= Pr(b'|b,a) = \sum_o Pr(b'|b,a,o)Pr(o|b,a) \\ &= \sum_o \delta(b', b^{ao}) \sum_{s,s'} O(a,s',o)T(s,a,s')b(s) \qquad (9)\end{aligned}$$

where $\delta(x,y) = 1$ if $x = y$ and 0 otherwise.

Transform the POMDP into a belief MDP

This change of variables suggests transforming the POMDP into a belief MDP, as follows:

- The POMDP belief simplex $\mathcal{B}$ plays the role of the MDP state set.
- The action set, horizon, and discount factor of the POMDP are used unchanged for the belief MDP.
- The transition function $\widetilde{T}$ of the belief MDP gives the probability of transitioning from one belief to another according to the observation model and belief update rule

$$\begin{aligned}\widetilde{T}(b,a,b') &:= Pr(b'|b,a) = \sum_o Pr(b'|b,a,o)Pr(o|b,a) \\ &= \sum_o \delta(b', b^{ao}) \sum_{s,s'} O(a,s',o)T(s,a,s')b(s) \qquad (9)\end{aligned}$$

where $\delta(x,y) = 1$ if $x = y$ and 0 otherwise.

Transform the POMDP into a belief MDP

This change of variables suggests transforming the POMDP into a belief MDP, as follows:

- The POMDP belief simplex $\mathcal{B}$ plays the role of the MDP state set.
- The action set, horizon, and discount factor of the POMDP are used unchanged for the belief MDP.
- The transition function $\widetilde{T}$ of the belief MDP gives the probability of transitioning from one belief to another according to the observation model and belief update rule

$$\begin{aligned}\widetilde{T}(b,a,b') &:= Pr(b'|b,a) = \sum_o Pr(b'|b,a,o)Pr(o|b,a) \\ &= \sum_o \delta(b', b^{ao}) \sum_{s,s'} O(a,s',o)T(s,a,s')b(s) \qquad (9)\end{aligned}$$

where $\delta(x,y) = 1$ if $x = y$ and 0 otherwise.

Transform the POMDP into a belief MDP (cont'd)

- The reward function $\widetilde{R}$ gives the expected immediate reward from applying an action in a given belief

$$\widetilde{R}(b,a) := E_{b_t=b}[R(s_t,a)] = \sum_s R(s,a)b(s) \tag{10}$$

Belief MDP

Note that the state set of the belief MDP is the POMDP belief simplex, which is uncountably infinite. Although our earlier discussion of MDPs assumed that the state space was finite, the same theoretical results go through with an infinite state set.

Transform the POMDP into a belief MDP (cont'd)

- The reward function $\widetilde{R}$ gives the expected immediate reward from applying an action in a given belief

$$\widetilde{R}(b,a) := E_{b_t=b}[R(s_t,a)] = \sum_s R(s,a)b(s) \tag{10}$$

Belief MDP

Note that the state set of the belief MDP is the POMDP belief simplex, which is uncountably infinite. Although our earlier discussion of MDPs assumed that the state space was finite, the same theoretical results go through with an infinite state set.

1 Deterministic Planning

2 Uncertainty in State Transitions

3 Partial Observability

4 Belief MDP Value Function Structure

5 Prior Research on POMDPs
- Exact value iteration
- Point-based value iteration
- Heuristic search
- Decentralized POMDPs

6 References

Piecewise-linear and convex

- The value functions of belief MDP policies have a special structure that makes it possible to generalize value iteration. For any finite horizon h, the optimal value function V_h^* is piecewise-linear and convex (PWLC).
- Formally, let π be any policy whose actions are not conditioned on the initial belief (such as a policy tree) and let α be a length-$|\mathcal{S}|$ vector such that $\alpha(s)$ is the expected value of following π starting from state s:

$$\alpha(s) = E_{\pi, s_0=s}[\sum_{t=0}^{h-1} \gamma^t R(s_t, a_t)] \tag{11}$$

Piecewise-linear and convex

- The value functions of belief MDP policies have a special structure that makes it possible to generalize value iteration. For any finite horizon h, the optimal value function V_h^* is piecewise-linear and convex (PWLC).
- Formally, let π be any policy whose actions are not conditioned on the initial belief (such as a policy tree) and let α be a length-$|\mathcal{S}|$ vector such that $\alpha(s)$ is the expected value of following π starting from state s:

$$\alpha(s) = E_{\pi, s_0=s}[\sum_{t=0}^{h-1} \gamma^t R(s_t, a_t)] \tag{11}$$

Piecewise-linear and convex (cont'd)

- Then the value of executing π starting from a belief b is

$$J\pi(b) = E_{\pi, b_0=b}[\sum_{t=0}^{h-1} \gamma^t R(s_t, a_t)]$$

$$= \sum_s b(s) E_{\pi, s_0=s}[\sum_{t=0}^{h-1} \gamma^t R(s_t, a_t)]$$

$$= \sum_s b(s)\alpha(s) = \alpha^T b \quad (12)$$

- Abusing notation, we also write

$$\alpha(b) = \alpha^T b \quad (13)$$

Piecewise-linear and convex (cont'd)

- Then the value of executing π starting from a belief b is

$$J\pi(b) = E_{\pi, b_0=b}[\sum_{t=0}^{h-1} \gamma^t R(s_t, a_t)]$$

$$= \sum_s b(s) E_{\pi, s_0=s}[\sum_{t=0}^{h-1} \gamma^t R(s_t, a_t)]$$

$$= \sum_s b(s)\alpha(s) = \alpha^T b \quad (12)$$

- Abusing notation, we also write

$$\alpha(b) = \alpha^T b \quad (13)$$

Notation

- When the max operator is applied to a set of α vectors, each vector is interpreted in this second sense, as a function over the belief simplex. In other words, if

$$V = max\{\alpha_1, \cdots, \alpha_n\}, \tag{14}$$

then

$$V(b) := \max_{\alpha_i} \alpha_i^T b. \tag{15}$$

Example 4: TIGER problem

In the TIGER problem, you stand before two doors. Behind one door is a tiger and behind the other is a pot of gold, but you do not know which is which. Thus there are two equally likely statesłthe tiger is located either behind the left or right door (the tiger-left or tiger-right state). You may try to learn more using the listen action, which provides information about which door the tiger is lurking behind, either the noise-left or noise-right observationłthe observation has an 85% chance of accurately indicating the location of the tiger. Alternately, you may choose to open one of the two doors using the open-left or open-right actions, at which point the game ends and you will either happily receive the gold (reward +10) or be eaten (reward -100). The parameters of the POMDP are summarized in Table 1; the discount factor $\gamma = 0.95$.

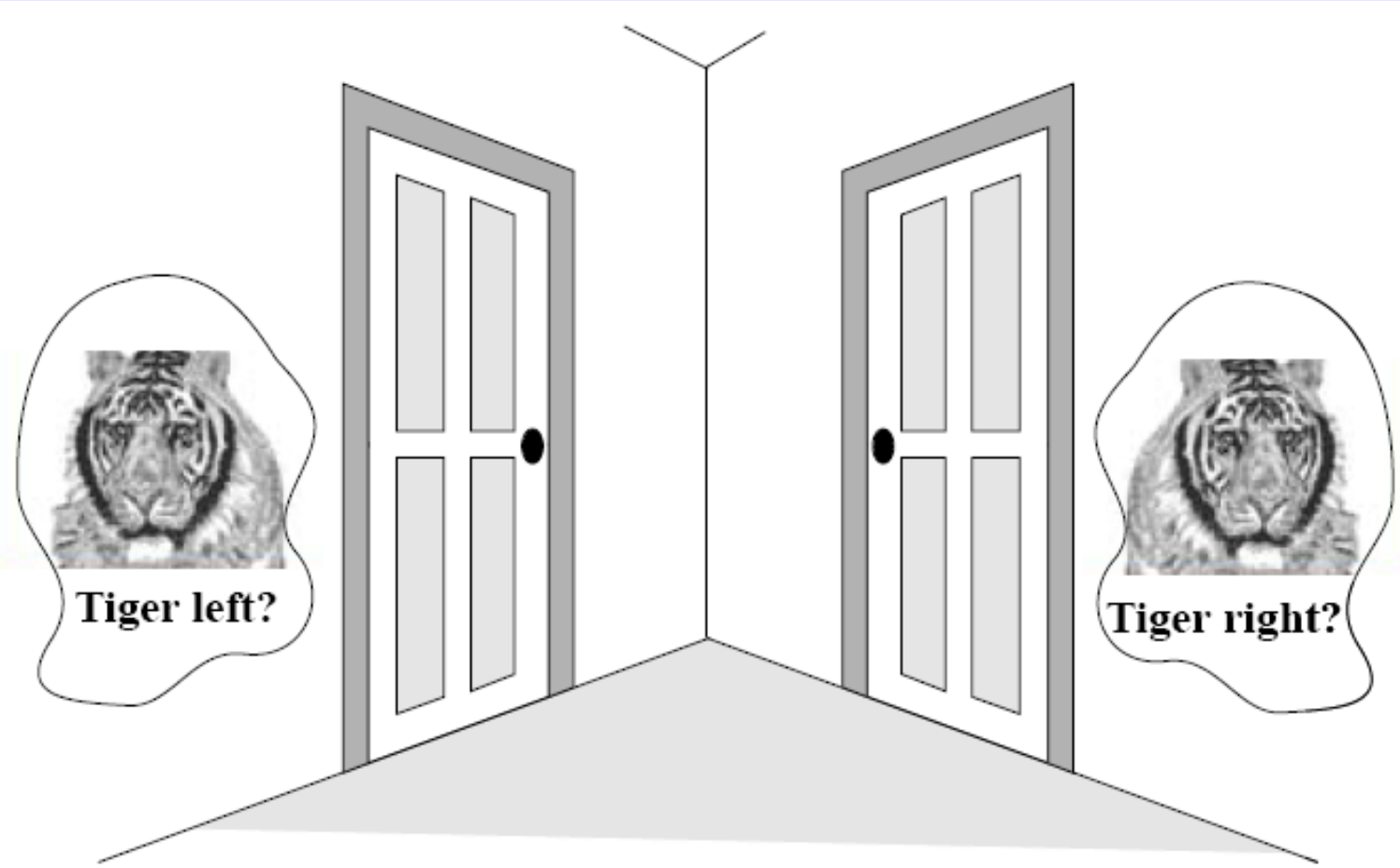

Figure 6: the TIGER Problem

TIGER problem (cont'd)

	s =tiger-left	s =tiger-right
$b_0(s)$	0.5	0.5
$R(s, open-left)$	-100	10
$R(s, open-right)$	10	-100
$R(s, listen)$	-1	-1
$O(listen, s, noise-left)$	0.85	0.15
$O(listen, s, noise-right)$	0.15	0.85

Table 1: TIGER problem parameters

Policy trees

Figure 6 shows some example two-step policy trees for the TIGER POMDP and the corresponding α vectors. The belief *b* varies along the x axis of the plot, from 100% certainty of tiger-left at the extreme left to 100% certainty of tiger-right at the extreme right. Each line in the plot relates to one of the policy trees as follows:

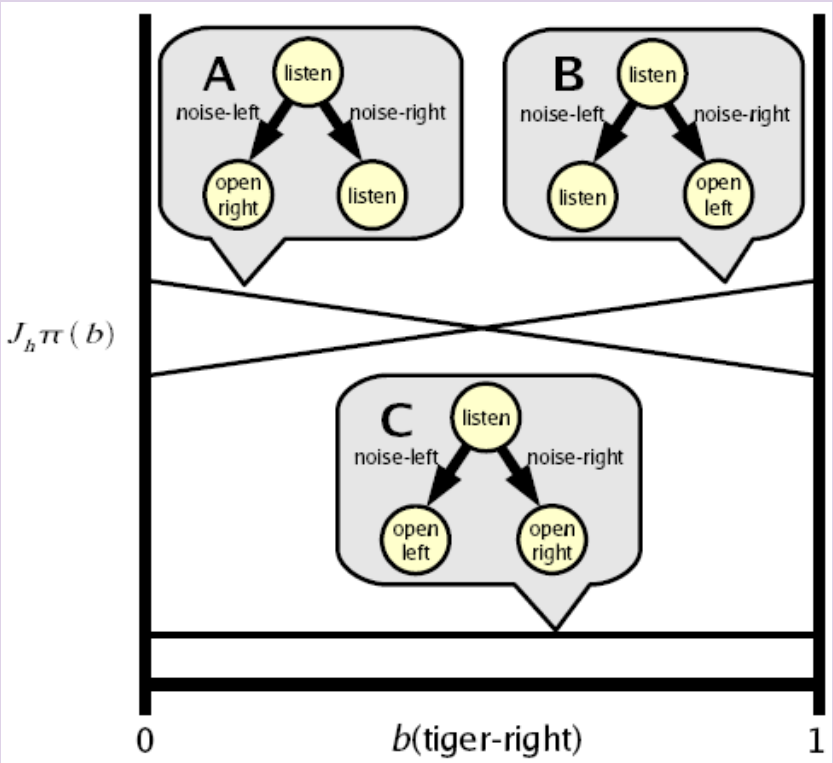

Figure 7: Example policy trees and corresponding α vectors for the TIGER POMDP

Policy A

Policy A: You select listen, then select open-right if noise-left is heard (and otherwise listen again). This is a good policy in the tiger-left state, since the most likely course of events is that you will hear noise-left and then perform the open-right action, receiving the pot of gold. Policy A performs poorly in the tiger-right state, it usually causes you to receive a small penalty for listening twice, and in the worst case you get a false noise-left reading and are eaten by the tiger. This is reflected in the policy A value function, which is at its highest in the tiger-left state and slopes down as the probability of being in the tiger-right state increases.

Policy B

Policy B: This policy is symmetrical with policy A, it works best when there is a high probability of being in the tiger-right state.

Policy C

Policy C: You listen, then open whichever door you hear a noise behind. This policy is clearly a bad idea regardless of what the you believe, since it preferentially opens the door where the tiger is located! This is reflected in its uniformly poor value function.

Max-planes representation

Let $\mathcal{T}_h = \{\pi_1, ..., \pi_n\}$ be the set of all policy trees of depth h as before, and let $\Gamma_h = \{\alpha_1, \cdots, \alpha_n\}$ be the corresponding set of α vectors. The optimal value for any particular belief b is the largest value assigned to b by any policy tree, meaning

$$V_h^*(b) = \max_{\pi \in J_h} J_h\pi(b) = \max \Gamma_h := \max_{\alpha \in \Gamma_h} \alpha^T b \tag{16}$$

In other words, for any finite h, the function V_h^* can be written as the maximum of a finite number of linear functions. We call this the max-planes representation. The existence of this representation means that V_h^* is a PWLC function.

Optimal value functions

Figure 7 shows optimal value functions V_h^* for a typical two-state POMDP, for several values of h. For each value of h, the thin lines are individual α vectors, each corresponding to a depth h policy tree, and V_h^* is the upper surface of all these α vectors, represented with a thicker line. The number of α vectors needed to represent V_h^* tends to increase with h, and in the limit V_∞^* may contain a countably infinite number of α vectors, in which case it is still convex but no longer piecewise-linear.

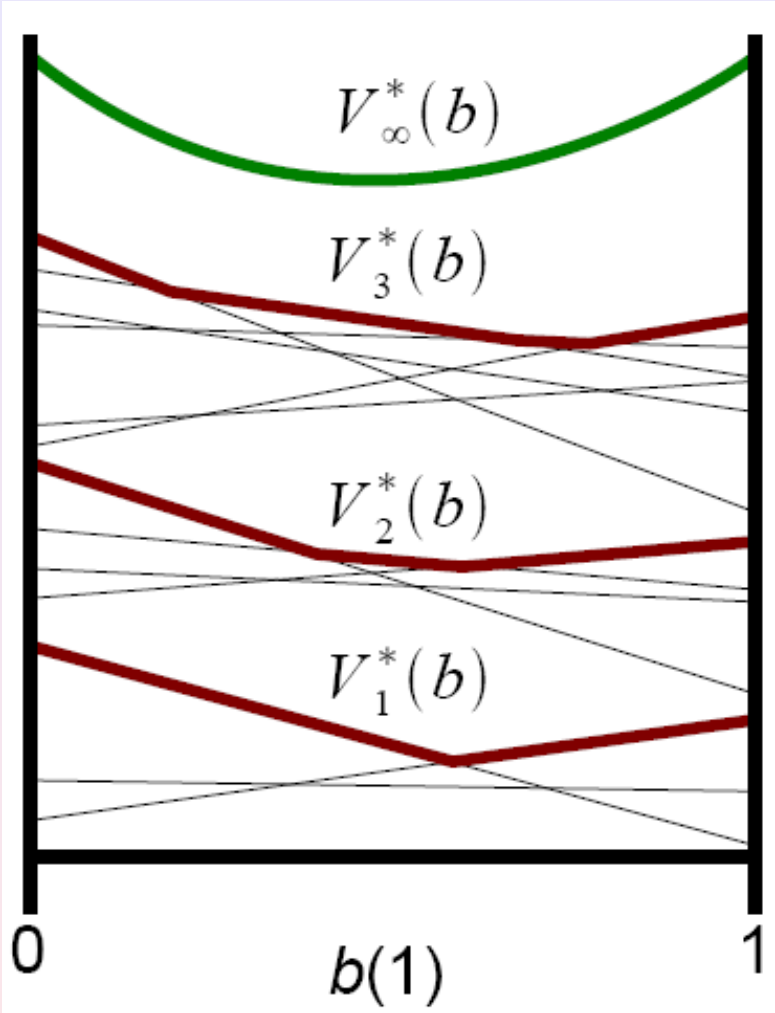

Figure 8: Optimal value functions at different time horizons for a typical two state POMDP

Pruning

Note that the value function for policy C in Figure 6 is everywhere dominated by policies A and B. It turns out that in a typical POMDP most policy trees from $\mathcal{T}_h$ are, like policy C, sub-optimal for every belief. This means that the corresponding α vectors are not part of the upper surface of V_h^*. These dominated α vectors can be pruned from the set Γ_h without affecting the value function. Pruning dominated vectors can exponentially reduce the size of the value function representation; thus it plays an important role in practical POMDP value iteration algorithms.

V^* for an example three-state POMDP

So far we have graphed value functions for POMDPs with just two states. POMDPs with more states have the same value function structure, but since the value function is definen harder to visualize. Figure 8 provides some geometrical intuition by showing the upper surface V_h^* for an example three-state POMDP. The hatched area below the surface represents the domain over which V_h^* is defined.

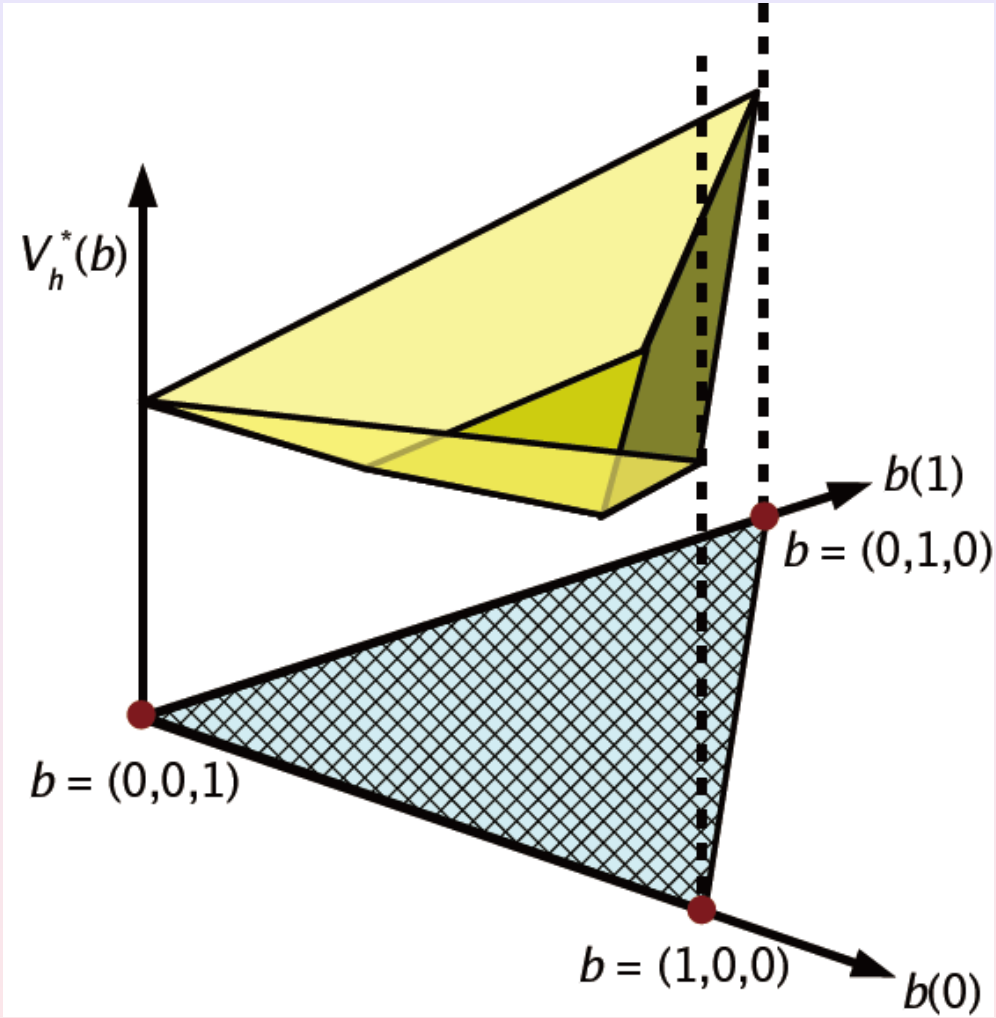

Figure 9: V^* for an example three-state POMDP

1 Deterministic Planning
2 Uncertainty in State Transitions
3 Partial Observability
4 Belief MDP Value Function Structure
5 Prior Research on POMDPs
- Exact value iteration
- Point-based value iteration
- Heuristic search
- Decentralized POMDPs

6 References

Exact value iteration

Many early POMDP solution algorithms performed exact value iteration using piecewise linear convex value function representations (Sondik, 1971; Monahan, 1982; Cheng, 1988; White, 1991). These algorithms had high computational complexity both in theory and in practice; they were largely impractical given the computing hardware available at the time.

Exact value iteration (cont'd)

The process of computing the Bellman update H used by value iteration can be broken up into (1) choosing a set of important beliefs, and (2) at each selected belief, performing a local "point-based" update that calculates the optimal α vector for that belief from *HV* (Cheng, 1988). The Witness algorithm (Littman, 1996) uses this idea to compute an exact Bellman update using a series of local updates at points selected using linear programs. Witness is more tractable than earlier exact VI techniques because it generates far fewer dominated α vectors.

Exact value iteration (cont'd)

The Incremental Pruning algorithm (Cassandra et al., 1997) achieved similar performance improvement. Incremental Pruning breaks up calculation of the Bellman update into a series of batch operations and interleaves these operations with pruning steps so that dominated α vectors are pruned earlier in the calculation.

Larger POMDPs require approximation techniques discussed below, including more compact approximate value function representations and point-based value iteration techniques, which break up the global Bellman update into smaller focused updates.

Exact value iteration (cont'd)

Websites about POMDP exact value iteration algorithms:

1. `http://www.pomdp.org/pomdp/index.shtml`

Point-based value iteration

The Witness algorithm computes an exact Bellman update using a series of pointbased updates at carefully selected points. On the other hand, if we are willing to accept some approximation error in the Bellman update, we can get away with updating fewer beliefs and selecting them less carefully. This is the key idea behind point-based value iteration algorithms.

Point-based value iteration (cont'd)

The first algorithms in this class, Cheng's Point-Based Dynamic Programming (Cheng, 1988) and Zhang's Point-Based Value Iteration (Zhang and Zhang, 2001), maintain their value function in the form of a set of α vectors and, associated with each vector, a witness belief where the vector dominates all other vectors. They interleave many (cheap) point-based updates at the witness points with occasional (very expensive) exact Bellman updates that generate new α vectors and witness points. The algorithms differ mainly in their process for selecting witness points.

Point-based value iteration (cont'd)

Pineau's Point-Based Value Iteration (PBVI) (Pineau et al., 2003b, 2006) is perhaps the most widely used point-based algorithm. PBVI selects a finite set $\widetilde{\mathcal{B}}$ of beliefs and performs point-based updates on all points of $\widetilde{\mathcal{B}}$ in synchronous batches until a convergence condition is reached, then uses a heuristic to expand $\widetilde{\mathcal{B}}$ and continues the process. They show that each batch update approximates the Bellman update and repeated batch updates converge to an approximation of V^* whose error is related to the sample spacing of $\widetilde{\mathcal{B}}$. In extensions of this work, they improved the heuristic for selecting new beliefs (Pineau and Gordon, 2005) and improved update efficiency by storing witness points in a metric tree (Pineau et al., 2003a).

Point-based value iteration (cont'd)

The PERSEUS algorithm (Spaan and Vlassis, 2005) uses batch updates that are slightly weaker but considerably faster than those of PBVI. During each update, PERSEUS backs up randomly sampled points from $\widetilde{\mathcal{B}}$, stopping when all points of $\widetilde{\mathcal{B}}$ have higher values than they did according to the previous value function. Since a single α vector often improves the value of several points, the batch usually requires many fewer than $|\widetilde{\mathcal{B}}|$ point-based updates.

Point-based value iteration (cont'd)

Websites about POMDP point-based value iteration algorithms:

1. The Symbolic Perseus POMDP Solver (MATLAB/Java source code), written by Pascal Poupart:
   `http://www.cs.uwaterloo.ca/~ppoupart/index.html`
2. The Perseus POMDP Solver (MATLAB source code), written by Matthijs Spaan:
   `http://staff.science.uva.nl/~mtjspaan/software/approx/`

Heuristic search

Heuristic search played a role in many of the POMDP solution algorithms already discussed; this section is specifically concerned with heuristic search techniques for selecting POMDP beliefs to update.

The first example of this technique is the RTDP-BEL algorithm (Geffner and Bonet, 1998), based on the Real-Time Dynamic Programming (RTDP) algorithm for MDPs (Barto et al., 1995). RTDP-BEL is a straightforward extension of RTDP to POMDPs represented as belief-MDPs, using a grid-based value function approximation.

Heuristic search (cont'd)

The same authors later developed improved algorithms to address some of RTDP's limitations. Labeled RTDP labels states once it knows from tracking residuals that they have approximately optimal values; labeled states can be skipped in later trials (Bonet and Geffner, 2003b). The Heuristic Dynamic Programming algorithm also labels finished states, but often more efficiently, and it fits better into a broad framework of find-and-revise algorithms that applies across many types of problems (Bonet and Geffner, 2003a). Bonet and Geffner applied these algorithms only to MDPs, but they can be generalized to POMDPs in the same fashion as RTDP.

Heuristic search (cont'd)

The Bounded RTDP (BRTDP) algorithm (McMahan et al., 2005) is another MDP heuristic search algorithm based on RTDP. More recently, Shani et al. have modified a number of point-based value iteration algorithms to use prioritized asynchronous updates similar to HSVI algorithm (Smith and Simmons, 2004), and have developed several novel heuristics for selecting good belief points (Shani et al., 2006, 2007; Virin et al., 2007).

Heuristic search (cont'd)

Websites about POMDP heuristic search value iteration algorithms:

1. ZMDP Software for POMDP and MDP Planning, written by Trey Smith: `http://www.cs.cmu.edu/~trey/zmdp/`

Decentralized POMDPs

Recently, there has been growing interest in multi-agent systems. The partially observable stochastic game (POSG) framework models planning for multiple agents with different information resources; the agents may or not cooperate with each other (Hansen et al., 2004). A decentralized POMDP (DEC-POMDP) is a special type of POSG in which the agents are assumed to be purely cooperative, but still have limited ability to share information (Bernstein et al., 2002).

Decentralized POMDPs (cont'd)

Unlike single-agent POMDPs, DEC-POMDPs can be used to decide when agents should communicate (Roth et al., 2006). Solving general DEC-POMDPs is known to be intractable, but approximate POMDP solution techniques such as BPI and PERSEUS have been adapted to the DEC-POMDP framework (Bernstein et al., 2005; Spaan et al., 2006).

1 Deterministic Planning
2 Uncertainty in State Transitions
3 Partial Observability
4 Belief MDP Value Function Structure
5 Prior Research on POMDPs
- Exact value iteration
- Point-based value iteration
- Heuristic search
- Decentralized POMDPs

6 References

References


Trey Smith
Probabilistic Planning for Robotic Exploration.
PhD thesis, The Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, July 2007.