AuSRoS
Australian School of Robotic Systems

B3 Planning under Uncertainty

Hanna Kurniawati

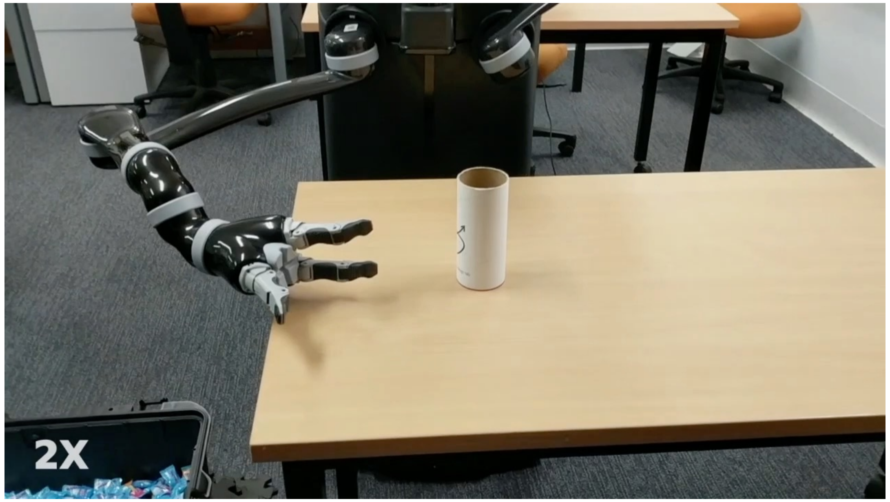
Australian National University | School of Computing

ROBOTICS @ ANU COMPUTING

This session

- Uncertainty is ubiquitous in robotics
- Frameworks for planning (sequential decision-making) under uncertainty
  - Markov Decision Processes (MDPs)
  - Partially Observable MDPs (POMDPs)
  - Reinforcement Learning
- Offline Solving: Value Iteration
  - Value Iteration for MDP
  - Point-based Value Iteration for POMDP
- Online Solving: MCTS-based
  - MDP
  - POMDP

Uncertainty is ubiquitous in Robotics



https://www.pinterest.com.au/pin/205054589256148973/?d=t&mt=login

Of course …

- Not just in robot manipulation

https://metro.tempo.co/read/1207214/semrawut-di-tanah-abang-parkir-liar-dan-pedagang-di-trotoar/full&view=ok

Self-driving cars?

The ubiquity of uncertainty …

- Deterministic planning assumes knowledge about:
  - What will happen after an action is performed
  - The current state of the system

- In this session, we'll discuss how to account for uncertainty during planning
  - Ideally, we want general purpose methods: The method to compute the strategy can be used for many different types of robots in many different scenarios

This session

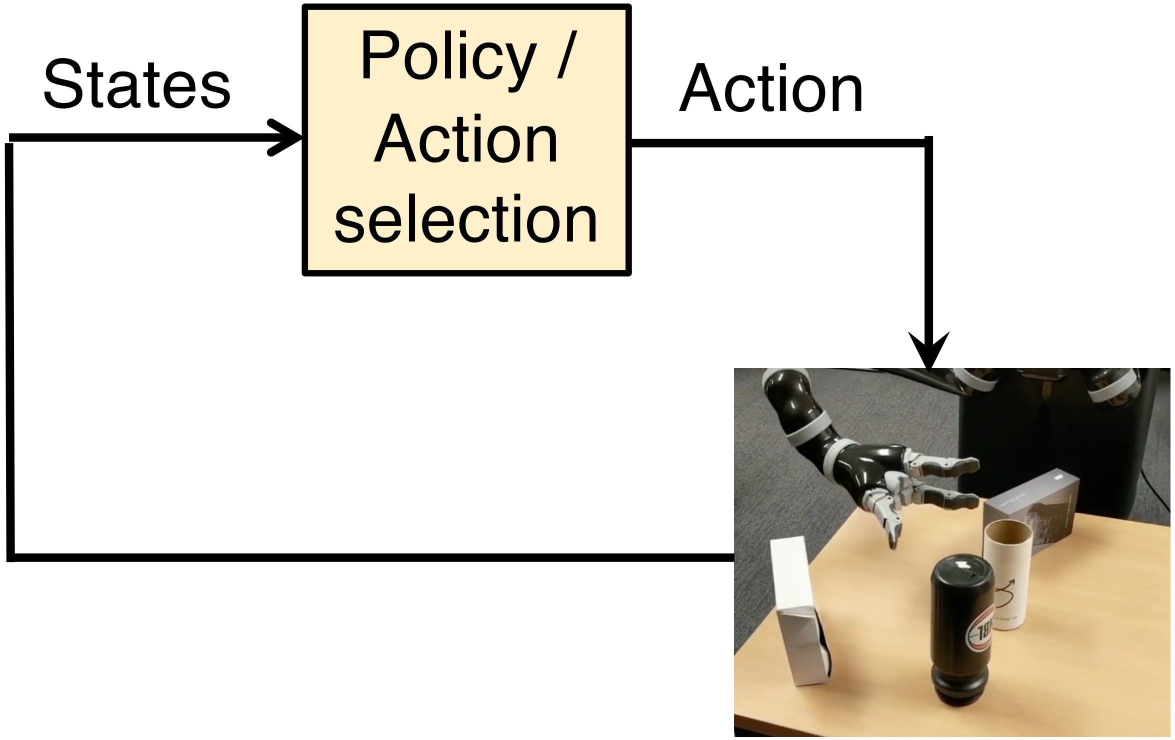
✓Uncertainty is ubiquitous in robotics

- Frameworks for planning (sequential decision-making) under uncertainty
  - Markov Decision Processes (MDPs)
  - Partially Observable MDPs (POMDPs)
  - Reinforcement Learning
- Offline Solving: Value Iteration
  - Value Iteration for MDP
  - Point-based Value Iteration for POMDP
- Online Solving: MCTS-based
  - MDP
  - POMDP

Markov Decision Processes (MDPs)

- Discrete time step
- At any given time step, the agent's state is known exactly (fully observed)
- But, the effects of actions are not known exactly before the action is executed (non-deterministic action effects)
  - This means, a solution in the form of a path / trajectory is no longer sufficient
  - Instead, we need to compute a mapping from states to actions (namely, a policy)

Markov Decision Processes (MDPs)

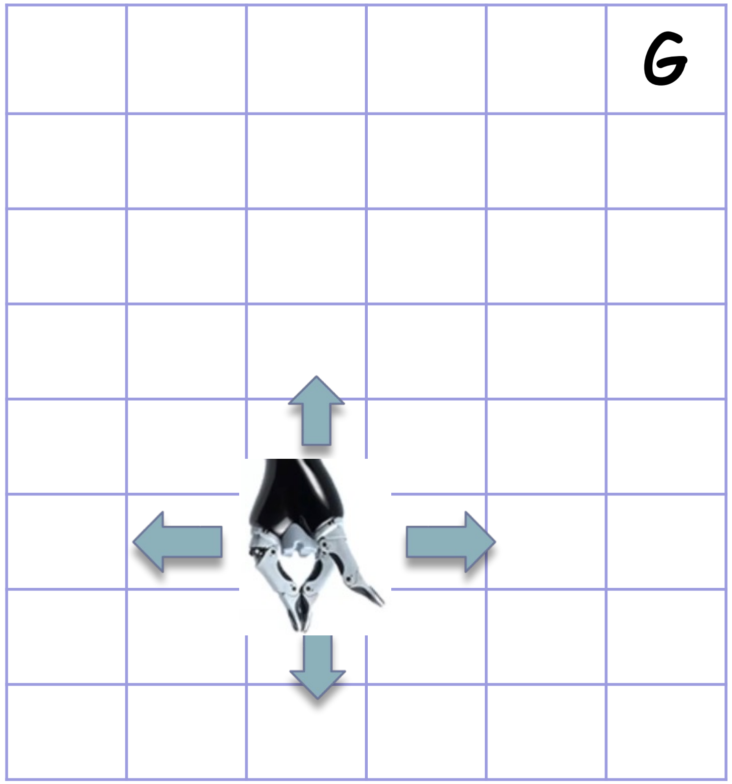


T(s, a, s’): Transition function, cond. prob. func P(s’ I s, a)
s, s’: States, a: action

R: Reward function

Defining an MDP Problem

- Formally defined as a 4-tuples (S, A, T, R):
  - S: State space.
  - A: Action space.
  - T: transition function.
    $T(s, a, s') = P(S_{t+1} = s' \mid S_t = s, A_t = a)$.
  - R: Reward function.
    $R(s)$ or $R(s, a)$ or $R(s, a, s')$



Markov Decision Processes (MDPs)

- Solving an MDP = finding the best policy π*
- What does "best" means?
- Many objectives have been proposed
- Some commonly used are the policy that maximizes
  - Myopic: $E[R_t(s,a)|\pi, s]$
  - Finite horizon: $E\left[\sum_{t=0}^{k} R_t(s,a)\,\middle|\pi, s_0\right]$
  - Infinite horizon: $E\left[\sum_{t=0}^{inf} \gamma^t R_t(s,a)\,\middle|\pi, s_0\right]$

s_0: initial state
s: state
a: action
$R_t(s,a)$: immediate reward
γ: discount factor with $0 < \gamma < 1$

The optimal policy is stationary

Value of a policy

- The expected total reward the agent will gather if it executes a policy π is called the value function and is denoted by V_π.

$$V_\pi(s) = \underbrace{R(s,a)}_{\text{Immediate reward}} + \gamma \underbrace{\sum_{s'} T(s,\pi(s),s')V_\pi(s')}_{\text{Expected total future reward}}$$

Optimal Value function

- Each policy has a corresponding value function.
- An optimal policy π^* is a policy whose corresponding value function is the maximum.
  - Meaning at each state, $V_{\pi^*}(s)$ is higher than (or at least the same as) V_π for any other policy π.
  - V_{π^*} is often simplified as V^*
  - In other words,

Q(s, a)

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

Bellman equation

A note

- There is a unique optimal value function V*
- But, there might be more than one optimal policy
  - If we know V*, the optimal policy can be generated easily

$$\pi^*(s) = \operatorname*{argmax}_a \left(R(s,a) + \gamma \sum_{s'} T(s,a,s')V^*(s') \right)$$

Markov Decision Processes (MDPs)

- Discrete time step
- At any given time, the agent's state is known exactly (fully observed)
- But, the effects of actions are not known exactly before the action is executed (non-deterministic action effects)
  - This means, a solution in the form of a path / trajectory is no longer sufficient
  - Instead, we need to compute a mapping from states to actions (namely, a policy)

This session

✓Uncertainty is ubiquitous in robotics

- Frameworks for planning (sequential decision-making) under uncertainty
  - ✓Markov Decision Processes (MDPs)
  - Partially Observable MDPs (POMDPs)
  - Reinforcement Learning
- Offline Solving: Value Iteration
  - Value Iteration for MDP
  - Point-based Value Iteration for POMDP
- Online Solving: MCTS-based
  - MDP
  - POMDP

Partially Observable Markov Decision Processes (POMDPs)

- Discrete time step
- The effects of actions are not known exactly before the action is executed (non-deterministic action effects)
- In addition, at any given time, the agent's state is NOT known exactly (partially observed)

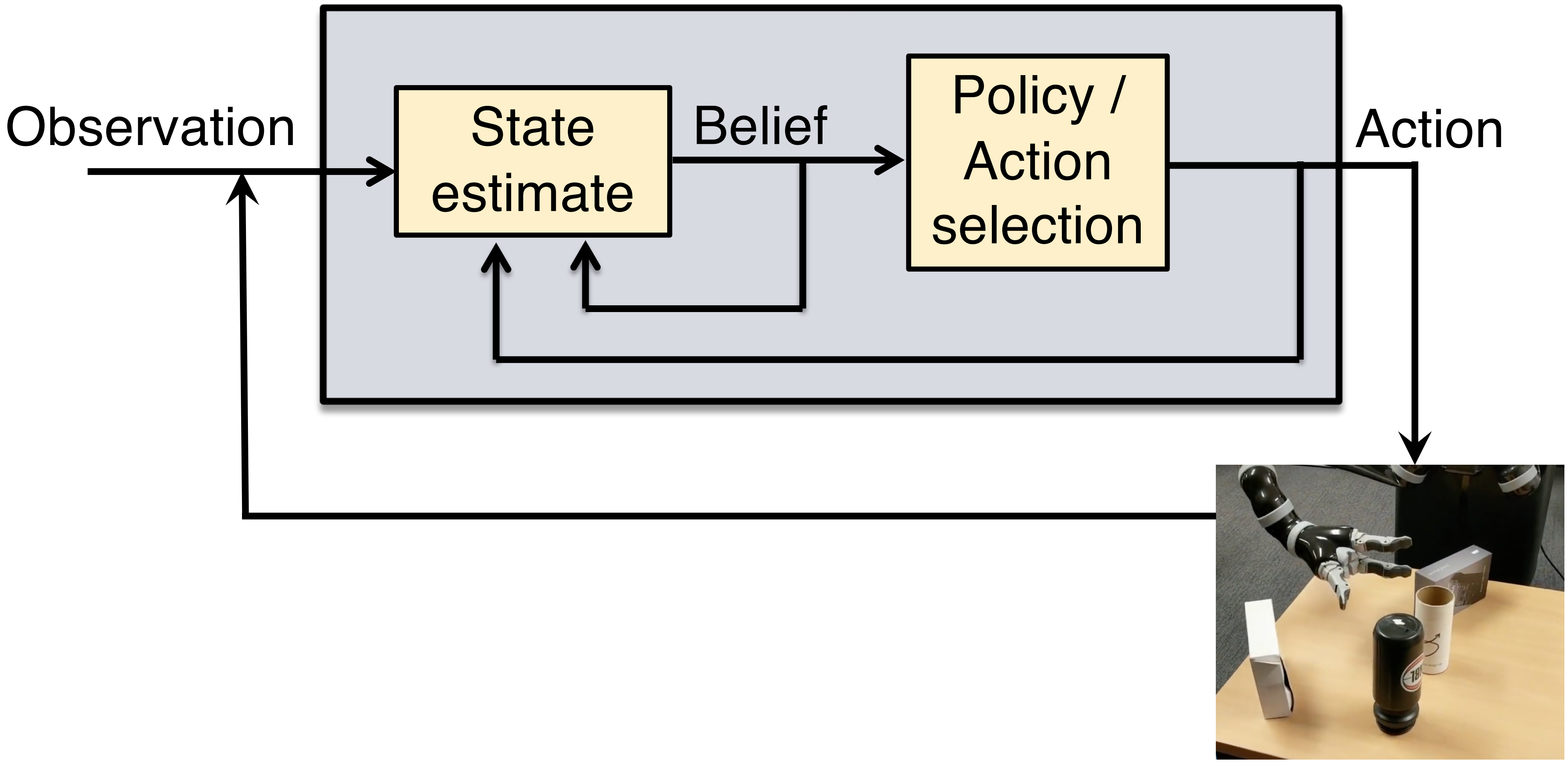
Partially Observable Markov Decision Processes (POMDPs)

Z(s’, a, o): Observation function P(o I s’, a) ; o ∈ O
s’: States, a: action, o: Observation

T(s, a, s’): Transition function P(s’ I s, a)
s, s’: States, a: action

R: Reward function

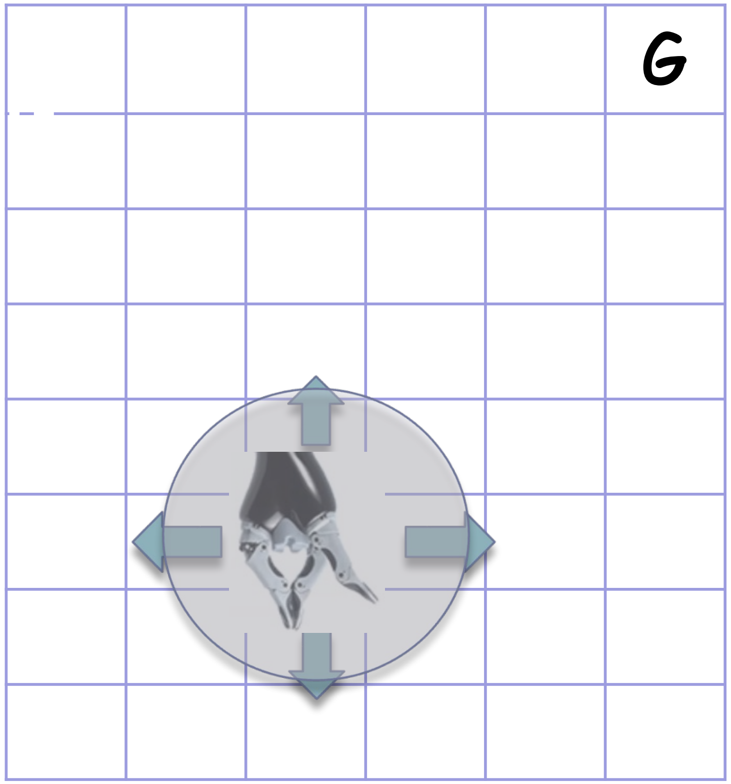
Partially Observable Markov Decision Processes (POMDPs)



Belief: Distribution over states
Solving a POMDP: Computing the best policy –maps beliefs to the best action

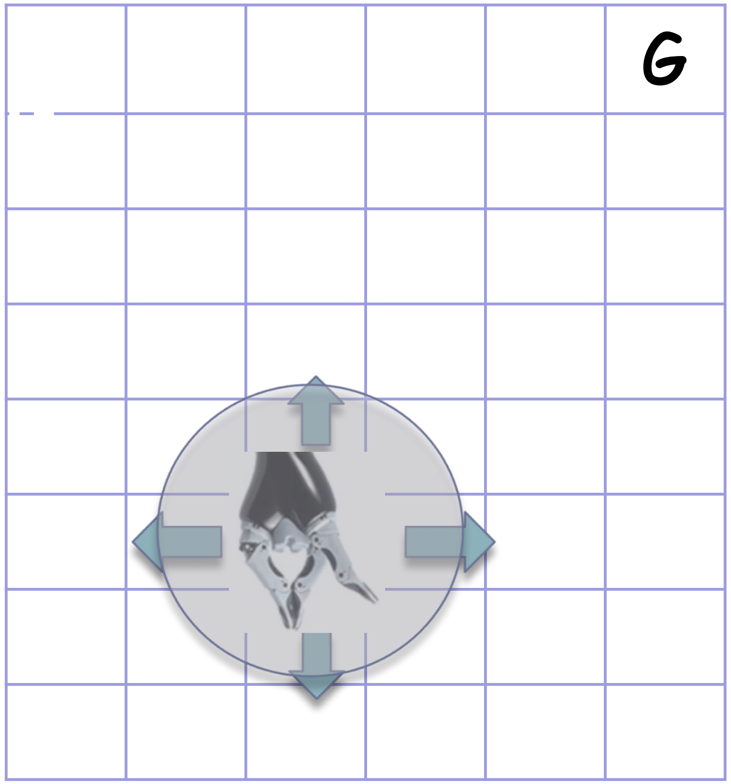
POMDP Model

- Main components:
  - State space (S)
  - Action space (A)
  - Observation space (O)



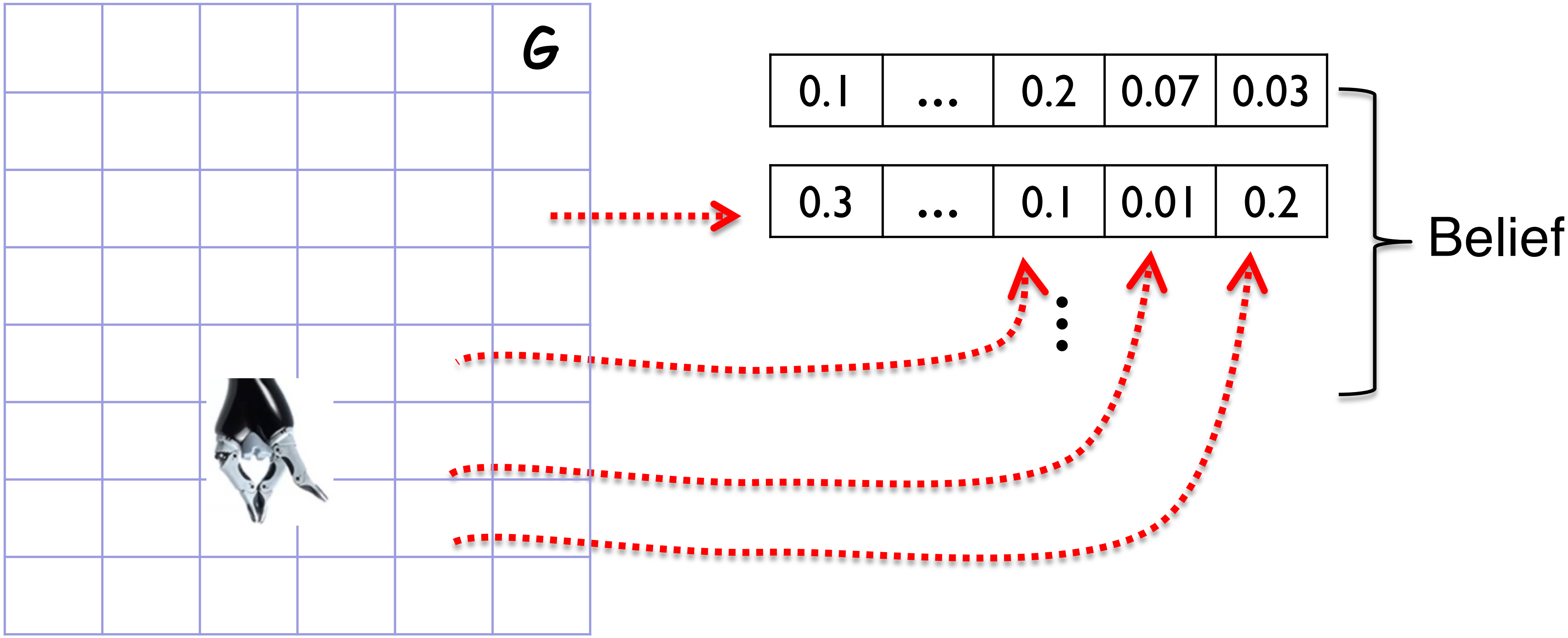
POMDP Model

- Main components:
  - State space (S) Not known
  - Action space (A)
  - Observation space (O)
  - Transition function (T)
    - T = P(s' l s, a)
  - Observation function (Z)
    - Z = P(o l s', a)
  - Reward function (R)
    - R(s, a)



$s, s' \in S, a \in A, o \in O$

Key to POMDPs: Belief



- Belief: distribution over the state space
  - The belief space is a simplex with dim = $|S|-1$
- Policy: mapping from beliefs to actions.

Key to POMDPs: Belief

- A belief is a sufficient statistics of the entire history of actions performed and observations perceived
- Hence, the 1st order Markov in POMDP is only in its model (transition and observation function)
- POMDPs policy is computed wrt to the entire history because it is computed wrt beliefs

Best policy

- Maps each belief to an action that maximizes the following objective function

$$V^*(b) = \max_{a \in A}\left(\underbrace{\sum_{s \in S} R(s,a)b(s)}_{\text{Expected immediate reward}} + \gamma \underbrace{\sum_{o \in O} P(o|b,a)V^*(b')}_{\text{Expected total future reward}}\right)$$

b': next belief after the system at belief b performs action a and observes o

γ: discount factor (0,1)

Computationally intractable [Papadimitriou & Tsikilis'87, Madani, et.al.'99].

Relation between MDPs & POMDPs

- A POMDP can be viewed as an MDP in the belief space

$$V^*(b) = \max_{a \in A} \left(\sum_{s \in S} R(s,a) b(s) + \gamma \sum_{o \in O} P(o|b,a) V^*(b') \right)$$

Immediate reward in the belief space: R(b,a)

Total future reward in the belief space: $\sum_{o \in O}$ P(b' l b, a, o)V(b')

- MDP can be viewed as as a degenerate POMDP

This session

✓Uncertainty is ubiquitous in robotics

- Frameworks for planning (sequential decision-making) under uncertainty
  - ✓Markov Decision Processes (MDPs)
  - ✓Partially Observable MDPs (POMDPs)
  - Reinforcement Learning
- Offline Solving: Value Iteration
  - Value Iteration for MDP
  - Point-based Value Iteration for POMDP
- Online Solving: MCTS-based
  - MDP
  - POMDP

Intuitively,

- Reinforcement learning (RL) is a machine learning approach that learns by doing
  - The agent is not provided with a model of how the robot's world works and/or which states are good
  - Rather, the agent learns by trying and evaluations states and actions

Formally,

- An RL agent is an MDP agent where the transition and/or reward functions are not initially known
  - S: State space.
  - A: Action space.
  - T: transition function.
    $T(s, a, s') = P(S_{t+1} = s' \mid S_t = s, A_t = a)$.
  - R: Reward function.
    $R(s, a)$.
- Need to try & explore

At least one of these is not known

POMDP for Reinforcement Learning (RL)
aka. Bayesian Reinforcement Learning

RL: MDP with missing components

- State space (S_{MDP})
- Action space (A_{MDP})
- Transition function (T_{MDP})
- Reward function (R_{MDP})

Not known

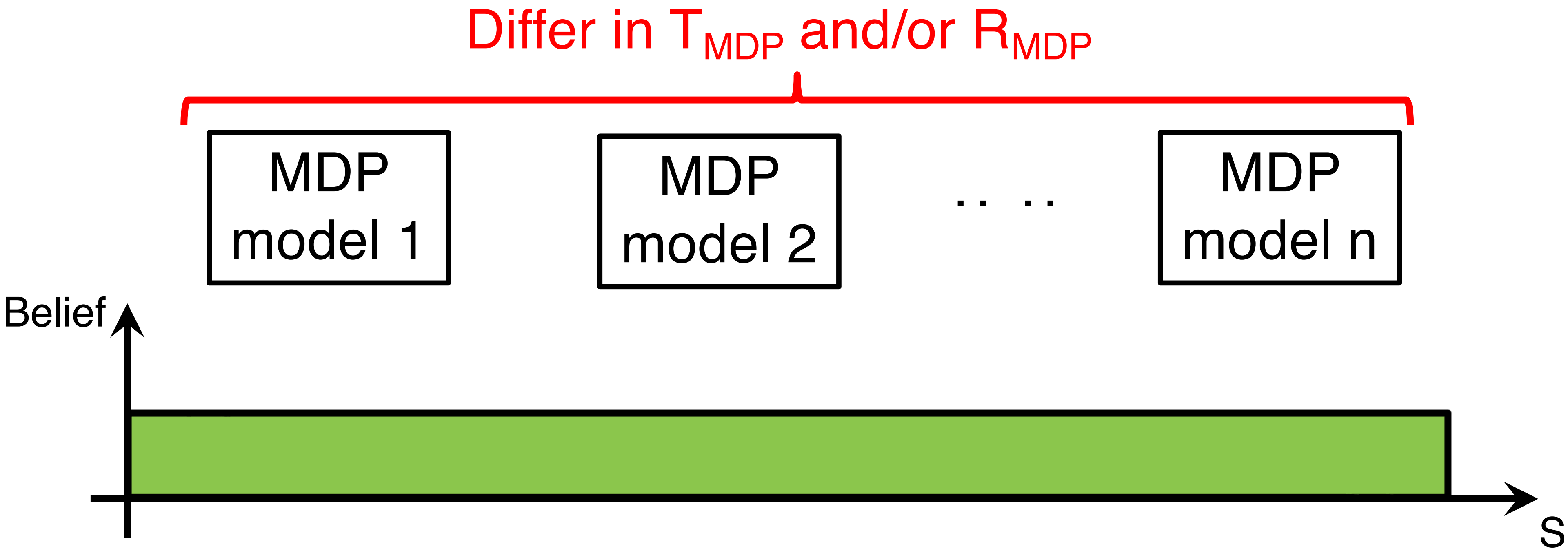
Bayesian RL

Construct a POMDP where

- The states are MDP states X parameters of the T_{MDP} & R_{MDP}
- Essentially, partial observability on which MDP model is the right model
- A, T, R follows from the particular MDP model
- O & Z are observations & observation function about which MDP model is correct

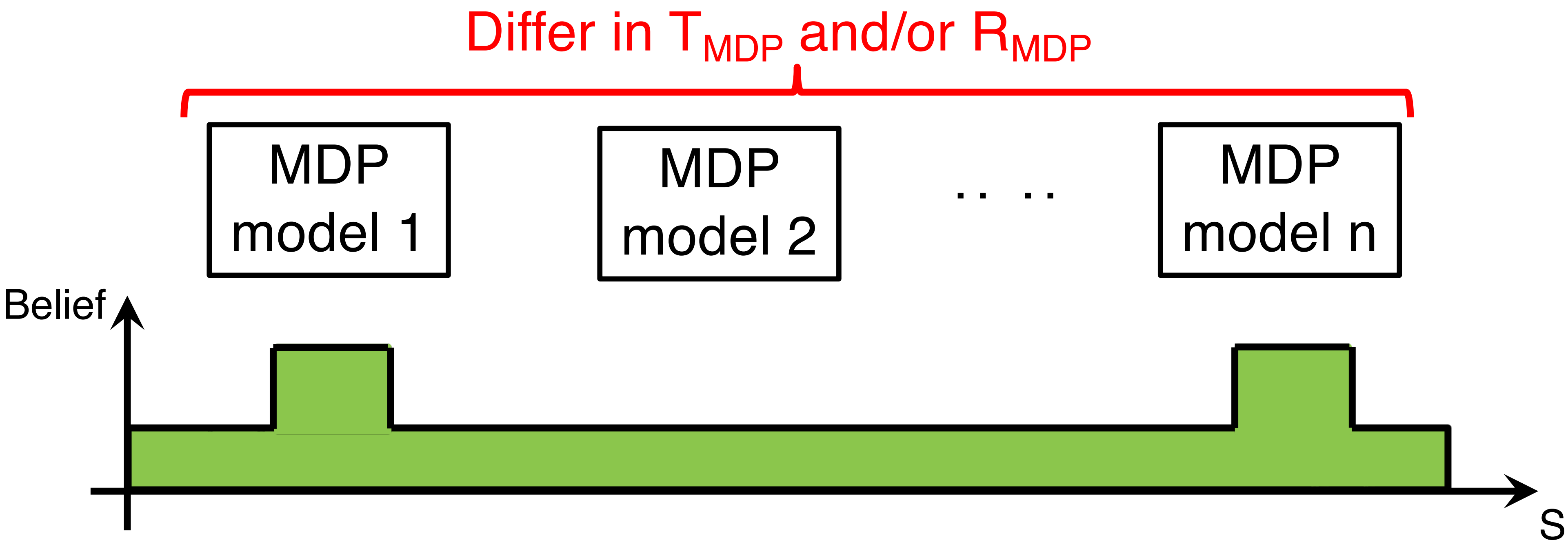
POMDP for Reinforcement Learning (RL)
aka. Bayesian Reinforcement Learning

- Beliefs in POMDP becomes distribution over the possible MDP models



POMDP for Reinforcement Learning (RL)
aka. Bayesian Reinforcement Learning

- Belief update means updates in the agent's understanding on which model is correct



POMDP automatically balances the trade-off between generating accurate model and to achieve the goal

In many cases, can achieve the goal without knowing the exact model

In a nutshell…

- MDP:
  - Non-deterministic action effects + fully observable
  - A degenerate POMDP
- POMDP:
  - Non-deterministic action effects + partially observable
  - MDP in the belief space
- RL:
  - MDP without transition and/or reward functions
  - Problem-wise, it’s essentially a POMDP, where partial observability is caused by incomplete information about the underlying MDP problem.

This session

- ✓ Uncertainty is ubiquitous in robotics
- ✓ Frameworks for planning (sequential decision-making) under uncertainty
  - ✓ Markov Decision Processes (MDPs)
  - ✓ Partially Observable MDPs (POMDPs)
  - ✓ Reinforcement Learning
- Offline Solving: Value Iteration
  - Value Iteration for MDP
  - Point-based Value Iteration for POMDP
- Online Solving: MCTS-based
  - MDP
  - POMDP

Solving: Offline vs Online

- Offline
  - Compute the policy before runtime
  - During runtime, only use the policy
    - Can think of the policy as a lookup table
- Online
  - At every step, the agent is given a short amount of time to compute the best action for the current step
- Combine offline and online

Value Iteration for MDP

- Offline method ; Iterate calculating the optimal value of a state until convergence.
- Algorithm:

Initialize $V^0(s) = R(s,a)$ for all s

Loop

For all s {

$$V^{t+1}(s) = \max_a \left(R(s,a) + \gamma \sum_{s'} T(s,a,s') V^t(s') \right)$$

}

t = t + 1

Until $V^{t+1}(s)=V^t(s)$ for all s (impl: $\max_s |V^{t+1}(s)-V^t(s)| < 1e-7$)

- Guarantee to converge to V^*

Often called value update or Bellman update or Bellman backup.

Value Iteration – Approximately Optimal

- No guarantee we'll reach optimal in finite time.
- However, it's guaranteed we can get close to optimal within finite time. In fact, polynomial in |S|, |A|, $1/(1-\gamma)$.
- If we want to ensure that we are ε close to optimal, then #iterations should be

$$\left\lceil \frac{\log\left({}^{2R_{max}}/_{\varepsilon(1-\gamma)}\right)}{\log\left({}^{1}/_{\gamma}\right)} \right\rceil$$

Value Iteration – Issues

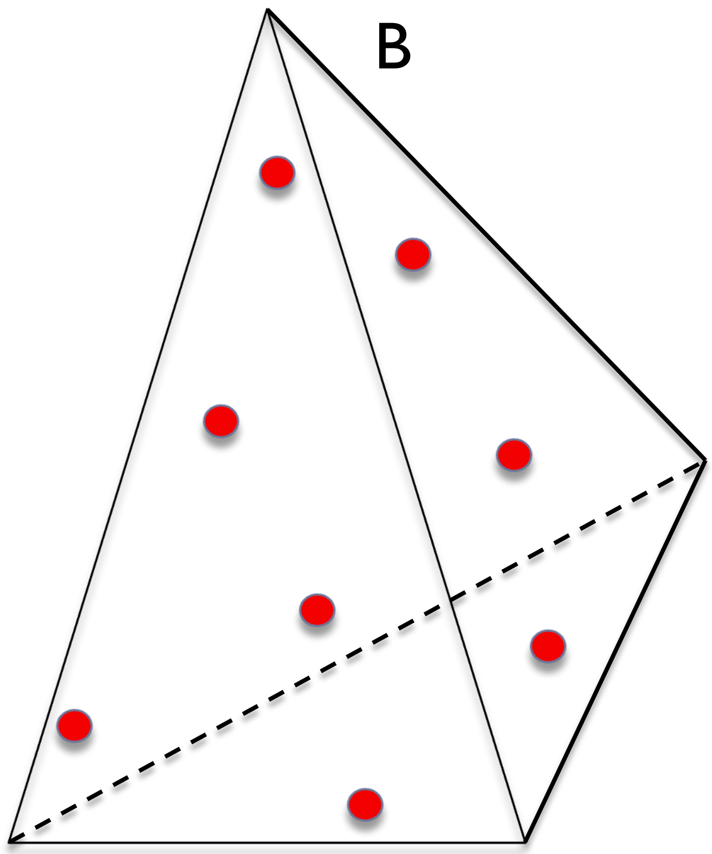
- Issues when the state space is large?
  - Computing Bellman update at every steps
  - Policy representation

- POMDP can be viewed as MDP in the belief space
  - Value iteration?

This session

- ✓ Uncertainty is ubiquitous in robotics
- ✓ Frameworks for planning (sequential decision-making) under uncertainty
  - ✓ Markov Decision Processes (MDPs)
  - ✓ Partially Observable MDPs (POMDPs)
  - ✓ Reinforcement Learning
- Offline Solving: Value Iteration
  - ✓ Value Iteration for MDP
  - Point-based Value Iteration for POMDP
- Online Solving: MCTS-based
  - MDP
  - POMDP

Point-based Value Iteration



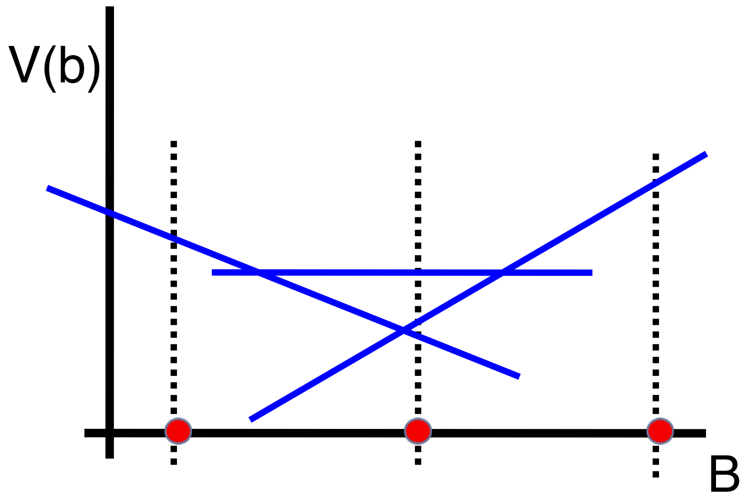
Idea:

- Represent the belief space with a set of small but representative sampled beliefs
- Iteratively apply Bellman update only on this set of sampled beliefs
- Interpolate to beliefs that have not been sampled

Joelle Pineau, Geoff Gordon, and Sebastian Thrun. "Point-based value iteration: An anytime algorithm for POMDPs.”. In IJCAI’03.

Policy representation: α-vector

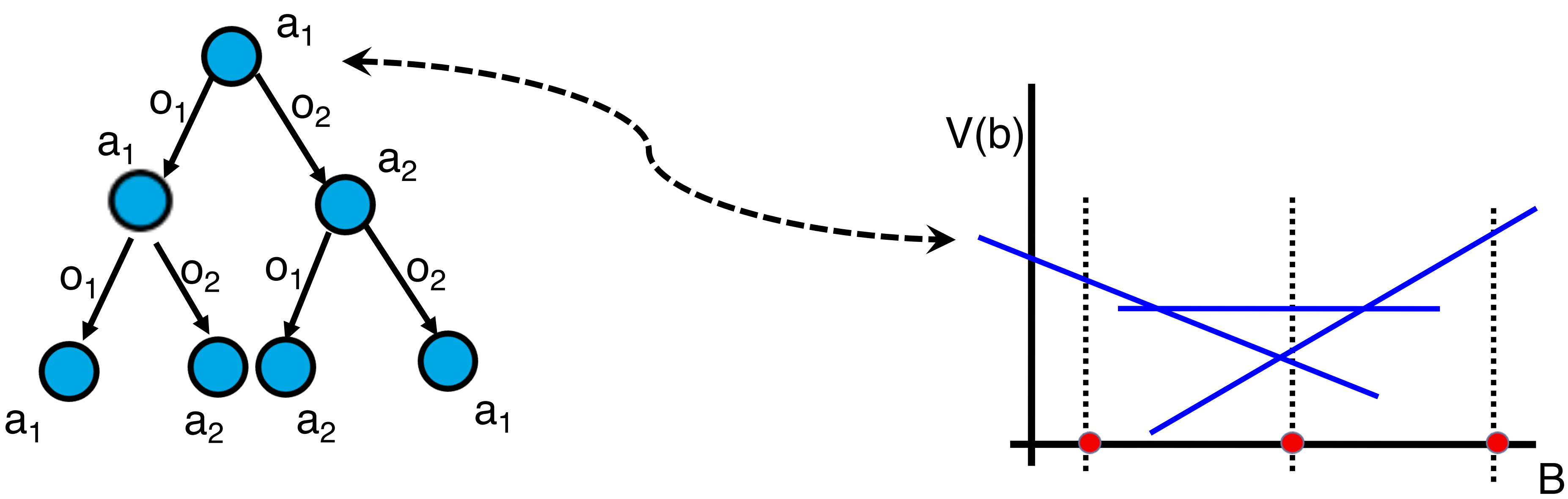
- This representation uses the piecewise linear convex property of POMDP's optimal value function
- The value function is represented by a set Γ of α-vectors, where ach α-vector is a gradient of a linear function
- $V^*(b) = \max_{\alpha \epsilon \Gamma} \alpha \, . \, b$



Leslie P. Kaelbling, Michael M. Littman, Anthony R. Cassandra. Planning & Acting in Partially Observable Domain. AIJ'98.

Policy representation: α-vector

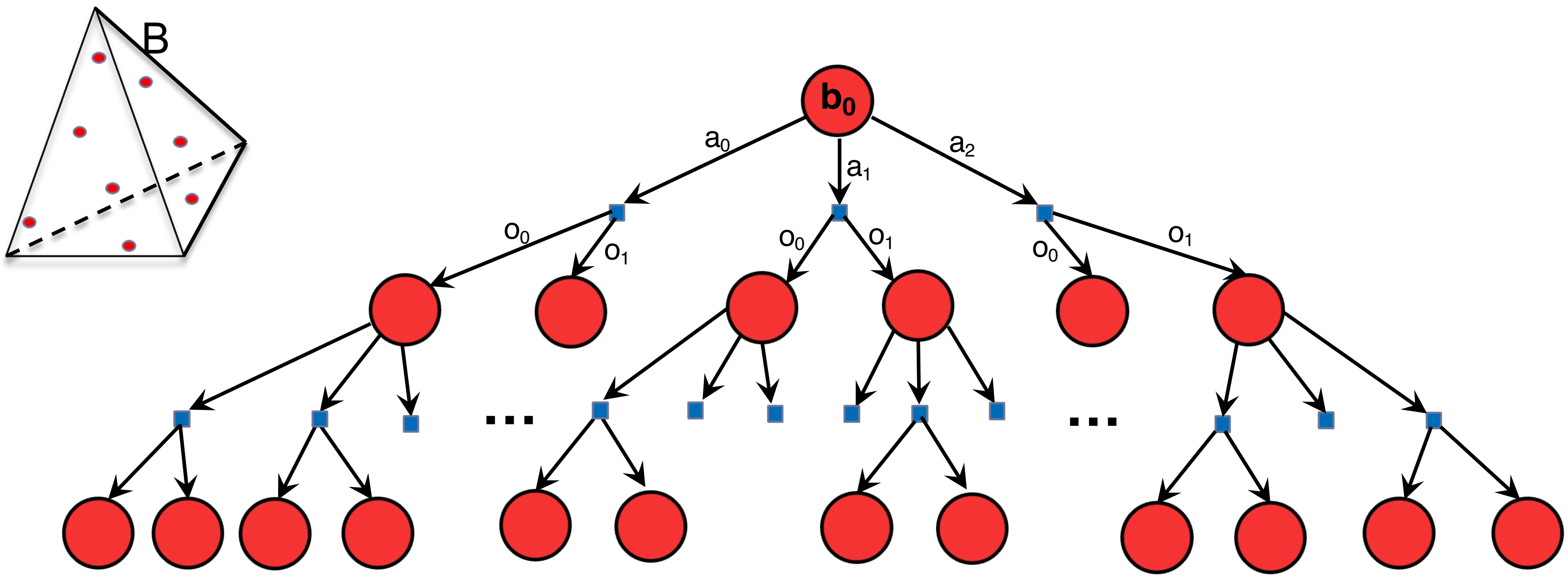
- Each α-vector also corresponds to an action, which is actually the root of a policy tree



The value if an agent having belief b executes a policy tree corresponds to a line.

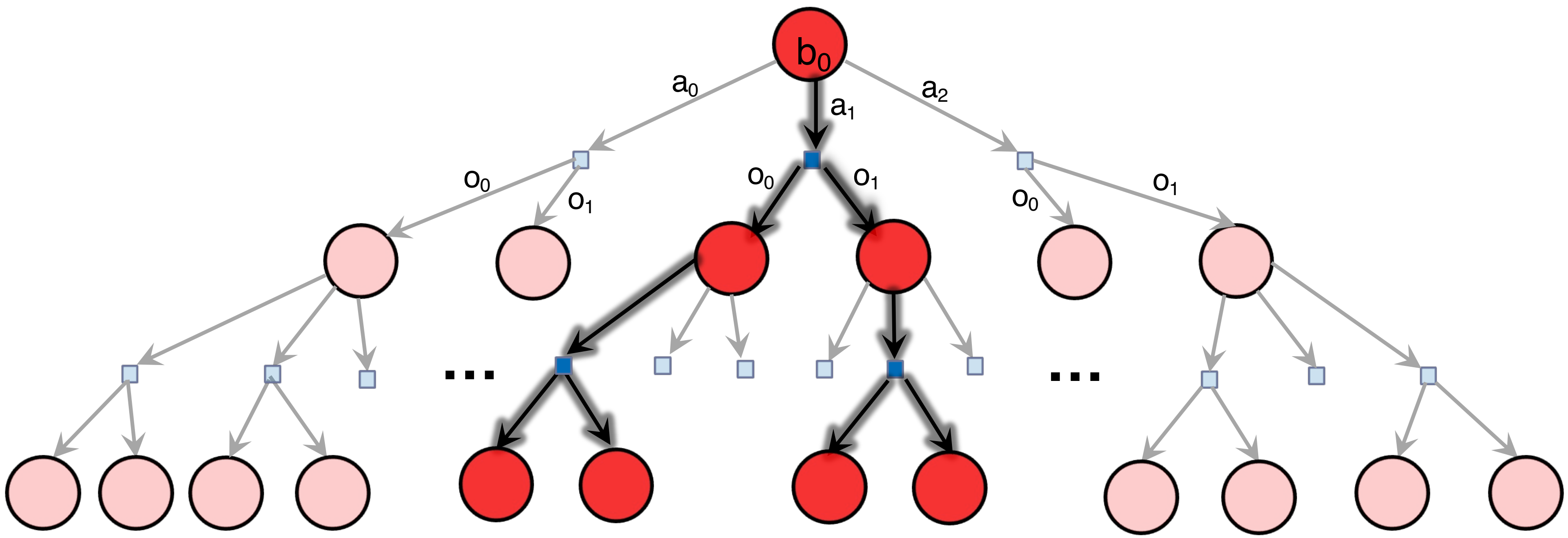
How to sample beliefs (distributions)?

- Assuming we start from an initial belief b_0
  - Sample an action & an observation
  - Sampling beliefs == expanding a belief tree rooted at b_0



To keep sampled beliefs small

- Sample from beliefs reachable under an optimal policy from the initial belief (b_0)

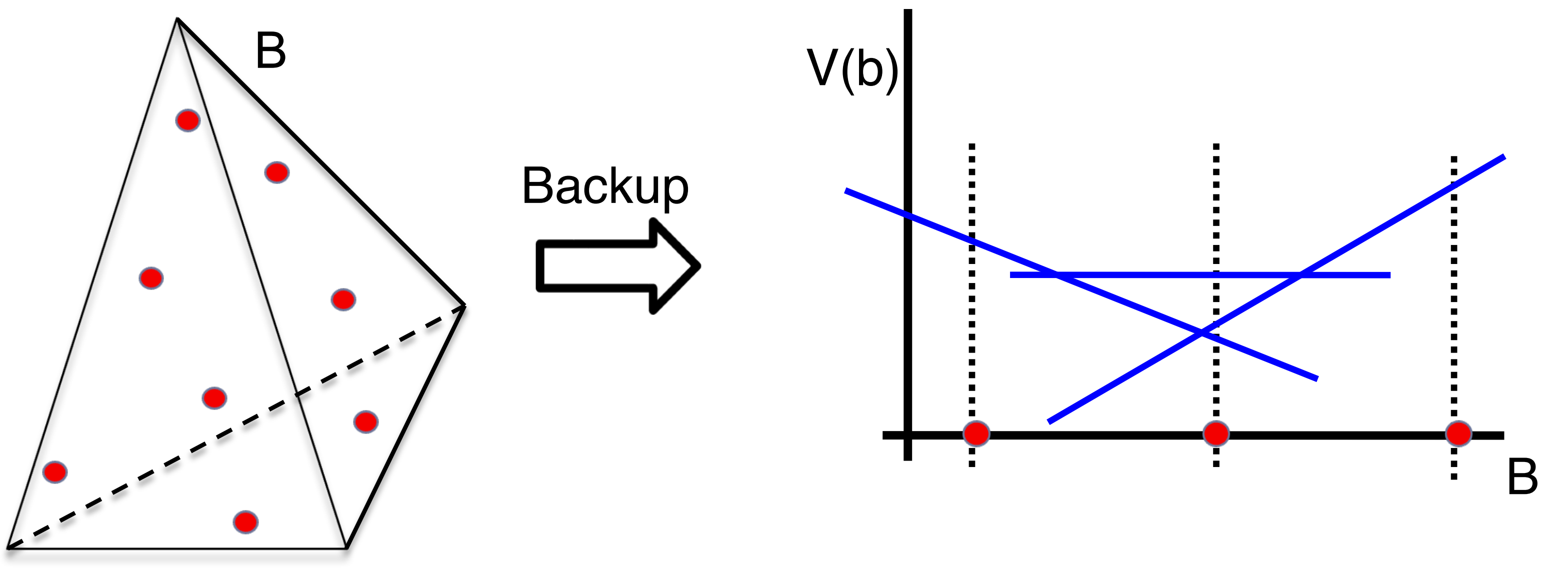


- Problem: We don't know the optimal policy…

H. Kurniawati and D. Hsu and W. S. Lee. SARSOP: Efficient point-based POMDP planning by approximating optimally reachable belief spaces. In RSS'08. RSS'21 Test of Time Award

Successive Approximations of the Reachable Space under Optimal Policies (SARSOP)

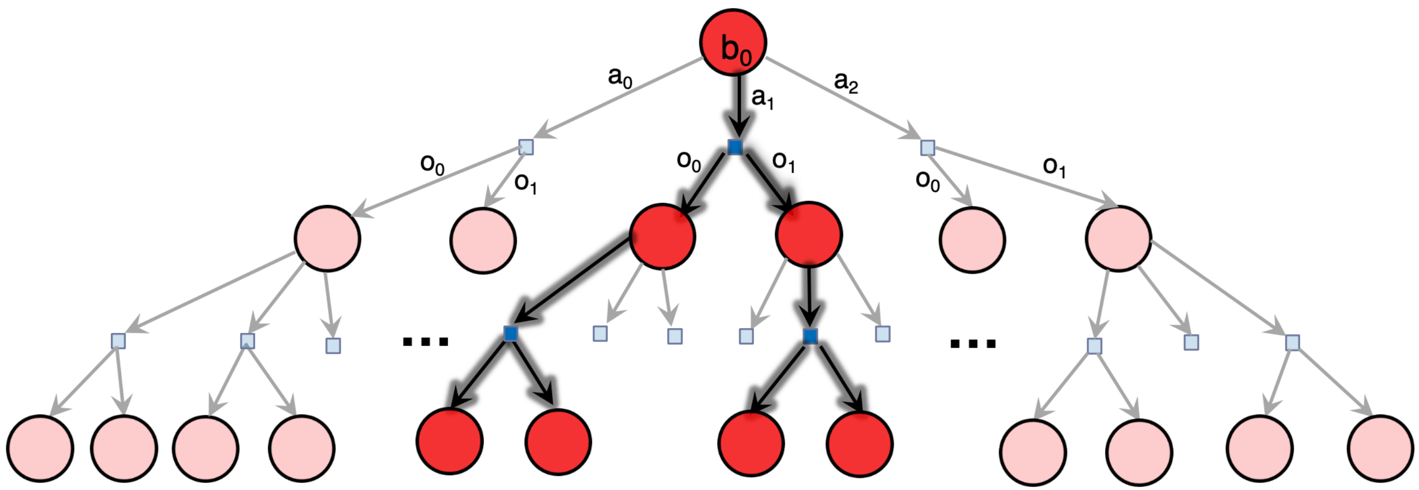
- Successively interleave sampling & value estimation



H. Kurniawati and D. Hsu and W. S. Lee. SARSOP: Efficient point-based POMDP planning by approximating optimally reachable belief spaces. In RSS'08. RSS'21 Test of Time Award

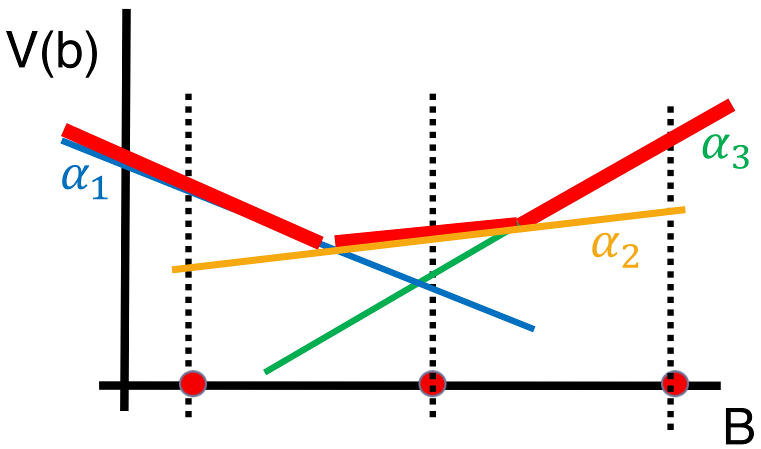
And …

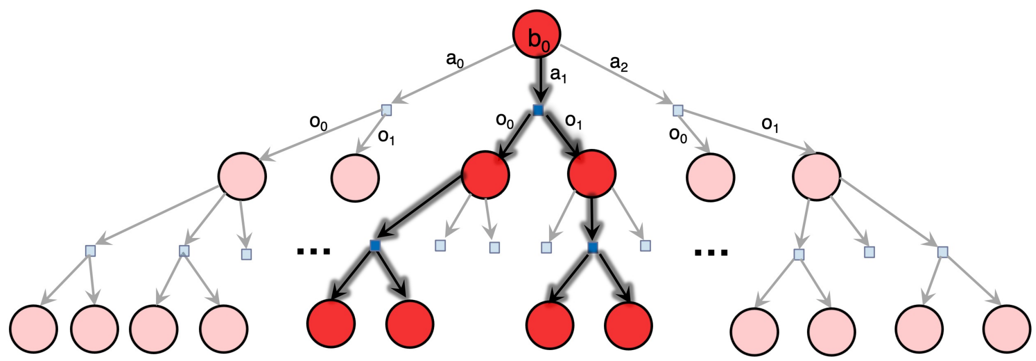
- Use simple machine learning method to predict the value function of a newly sampled belief b, $\widehat{V^*}(b)$
  - If the predicted value $\widehat{V^*}(b)$ is likely to improve $\widehat{V^*}(b_0)$, expand b
  - To expand b
    - Maintain upper & lower bound of $V^*(b)$
    - Select an action to expand b based on the upper bound
    - Select an observation based on the gap between upper & lower bounds



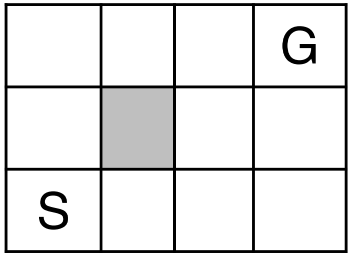
Once a belief is sampled

- Improve the value function estimated

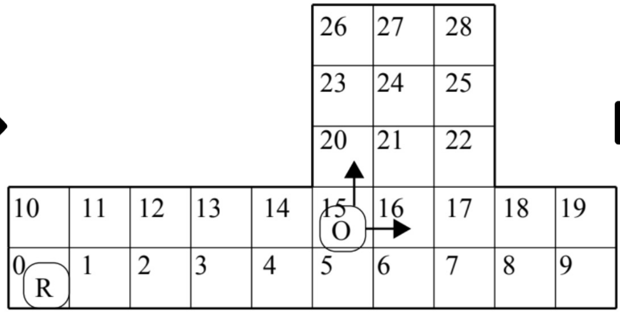




Progression with Offline Solver

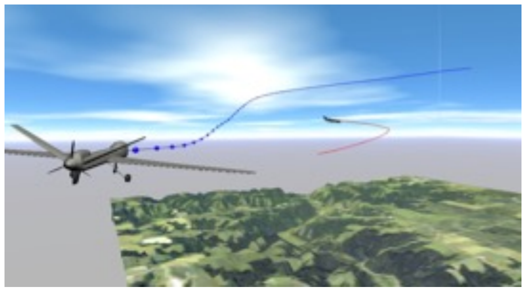


Before'03: ISI = 12, days

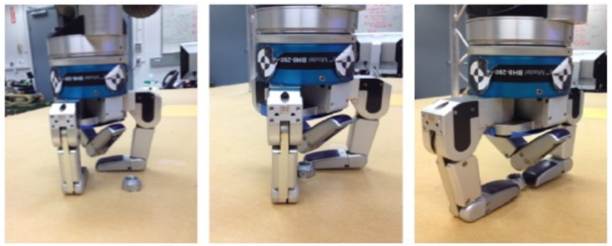


PBVI (Pineau'03): Tag ISI = 870, 50h

SARSOP (Kurniawati, et.al.'08, RSS'21 Test of Time Award)
Tag in ~ 6sec
Up to ISI ~ 100K, 2h



Temizer, et.al.
Study of next-gen TCAS (improve safety of TCAS by 20X)



Horowitz & Burdick

This session

- ✓ Uncertainty is ubiquitous in robotics
- ✓ Frameworks for planning (sequential decision-making) under uncertainty
  - ✓ Markov Decision Processes (MDPs)
  - ✓ Partially Observable MDPs (POMDPs)
  - ✓ Reinforcement Learning
- ✓ Offline Solving: Value Iteration
  - ✓ Value Iteration for MDP
  - ✓ Point-based Value Iteration for POMDP
- Online Solving: MCTS-based
  - MDP
  - POMDP

Changes from Offline Solving

- Does not need the exact transition and reward model
  - Use a generative model (can be a simulator)
  - Computes a good policy by interacting with the simulator
- Assume: Initial state is known

Monte Carlo Tree Search (MCTS)

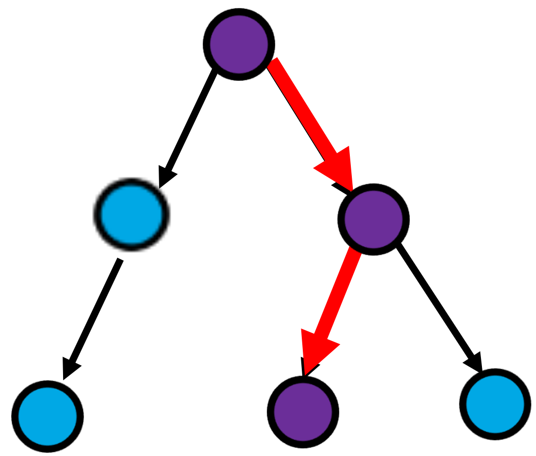
- Combines tree search & Monte Carlo
- Monte Carlo
  - A sampling-based method to approximate the value of complicated functions, i.e., the ones that are too difficult/time consuming to compute exactly.
  - In MDP, can be used to approximate the expected total future reward if the agent follows an optimal policy.

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

Kocsis, Levente, and Csaba Szepesvári. "Bandit based Monte-Carlo Planning." ECML'06

Monte Carlo Tree Search (MCTS)

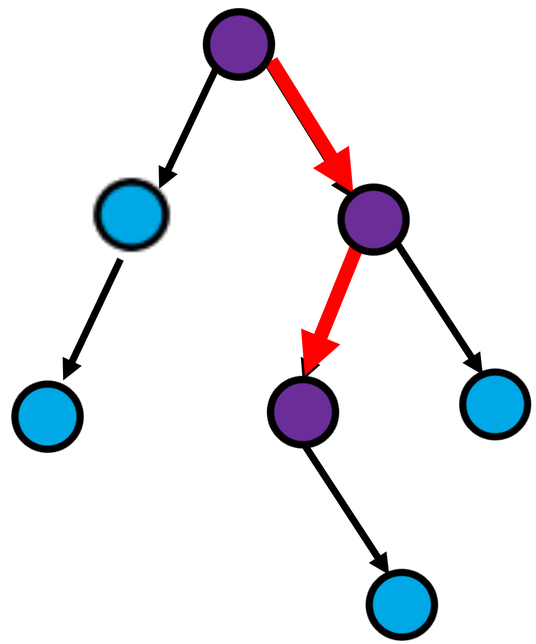
- Build the search tree based on the outcomes of the simulated plays.
- Iterate over 4 main components:
  - Selection: Choose the best path



A node: state & value

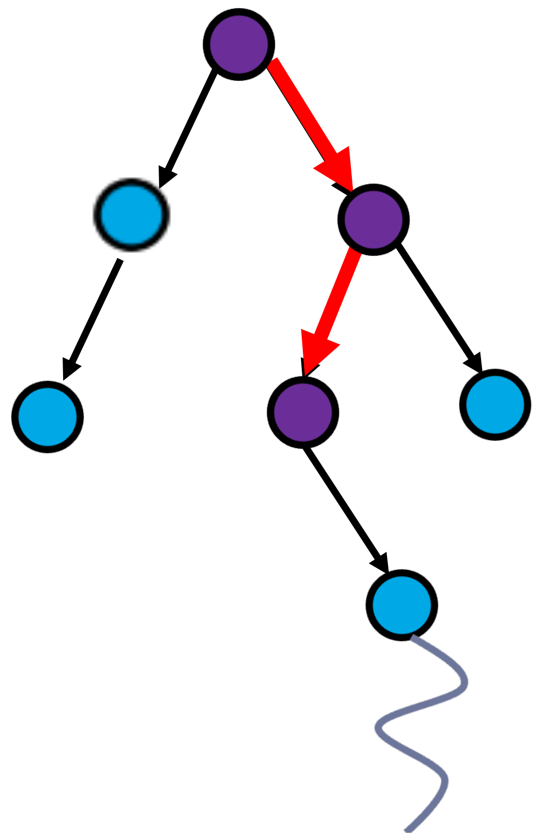
Monte Carlo Tree Search (MCTS)

- Build the search tree based on the outcomes of the simulated playouts.
- Iterate over 4 main components:
  - Selection: Choose the best path
  - Expansion: When a terminal node is reached, add a child node



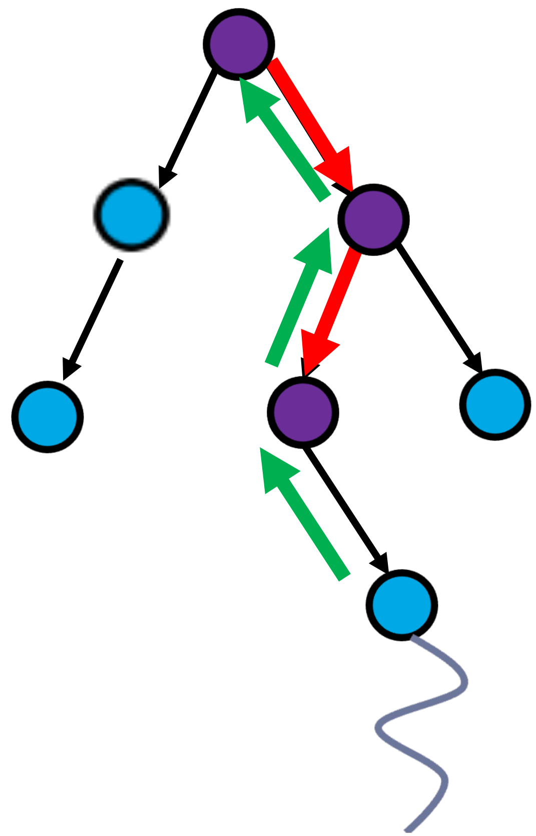
Monte Carlo Tree Search (MCTS)

- Build the search tree based on the outcomes of the simulated playouts.
- Iterate over 4 main components:
  - Selection: Choose the best path
  - Expansion: When a terminal node is reached, add a child node n
  - Simulation: Simulate from the newly added node n, to estimate its value



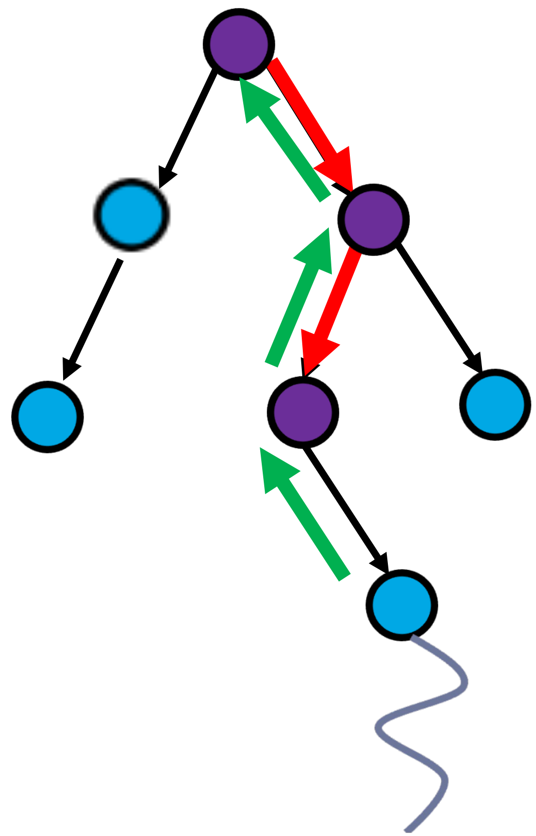
Monte Carlo Tree Search (MCTS)

- Build the search tree based on the outcomes of the simulated playouts.
- Iterate over 4 main components:
  - Selection: Choose the best path
  - Expansion: When a terminal node is reached, add a child node n
  - Simulation: Simulate from the newly added node n, to estimate its value
  - Backpropagation: Update the value of the nodes visited in this iteration
- Many variants



Commonly used MCTS for MDP

- Build the search tree based on the outcomes of the simulated playouts.
- Iterate over 4 main components:
  - Selection: Choose the best path
  - Expansion: When a terminal node is reached, add a child node n
  - Simulation: Simulate from the newly added node n, to estimate its value
  - Backpropagation: Update the value of the nodes visited in this iteration



Node Selection

- Multi-armed bandit to select which action to use
  - In general, use a method called Upper Confidence Bound (UCB).
  - Choose an action a to perform at s as:

$$\pi_{UCT}(s) = \arg\max_{a \in A} \underbrace{Q(s,a)}_{\text{Exploitation}} + \underbrace{c\sqrt{\frac{\ln(n(s))}{n(s,a)}}}_{\text{Exploration}}$$

c: A constant indicating how to balance exploration & exploitation, need to be decided by trial & error.

n(s): #times node s has been visited.

n(s,a): #times the out-edge of s with label a has been visited.

- MCTS + UCB is often called Upper Confidence Bound for Trees (UCT)

Simulation

- Often called rollout
- Essentially, a way to estimate the optimal value of the newly added state
- In practice,
  - Use heuristic, e.g., greedy, solution of deterministic case
  - Important for performance

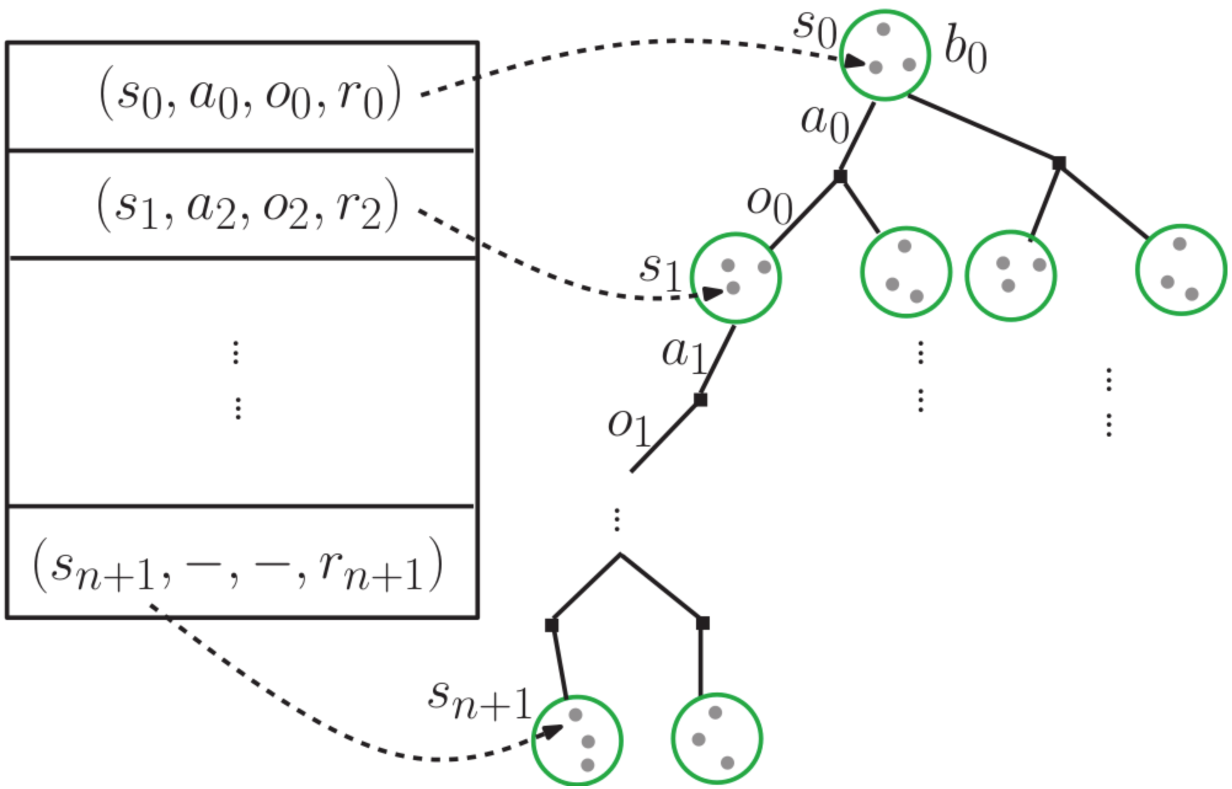
Backpropagation

- Essentially, updating the Q values
  - $Q(s, a) = (Q(s, a) \times N(s, a) + q) / (N(s, a) + 1)$
  - $N(s) = N(s) + 1$
  - $N(s, a) = N(s, a) + 1$

This session

- ✓ Uncertainty is ubiquitous in robotics
- ✓ Frameworks for planning (sequential decision-making) under uncertainty
  - ✓ Markov Decision Processes (MDPs)
  - ✓ Partially Observable MDPs (POMDPs)
  - ✓ Reinforcement Learning
- ✓ Offline Solving: Value Iteration
  - ✓ Value Iteration for MDP
  - ✓ Point-based Value Iteration for POMDP
- Online Solving: MCTS-based
  - ✓ MDP
  - POMDP

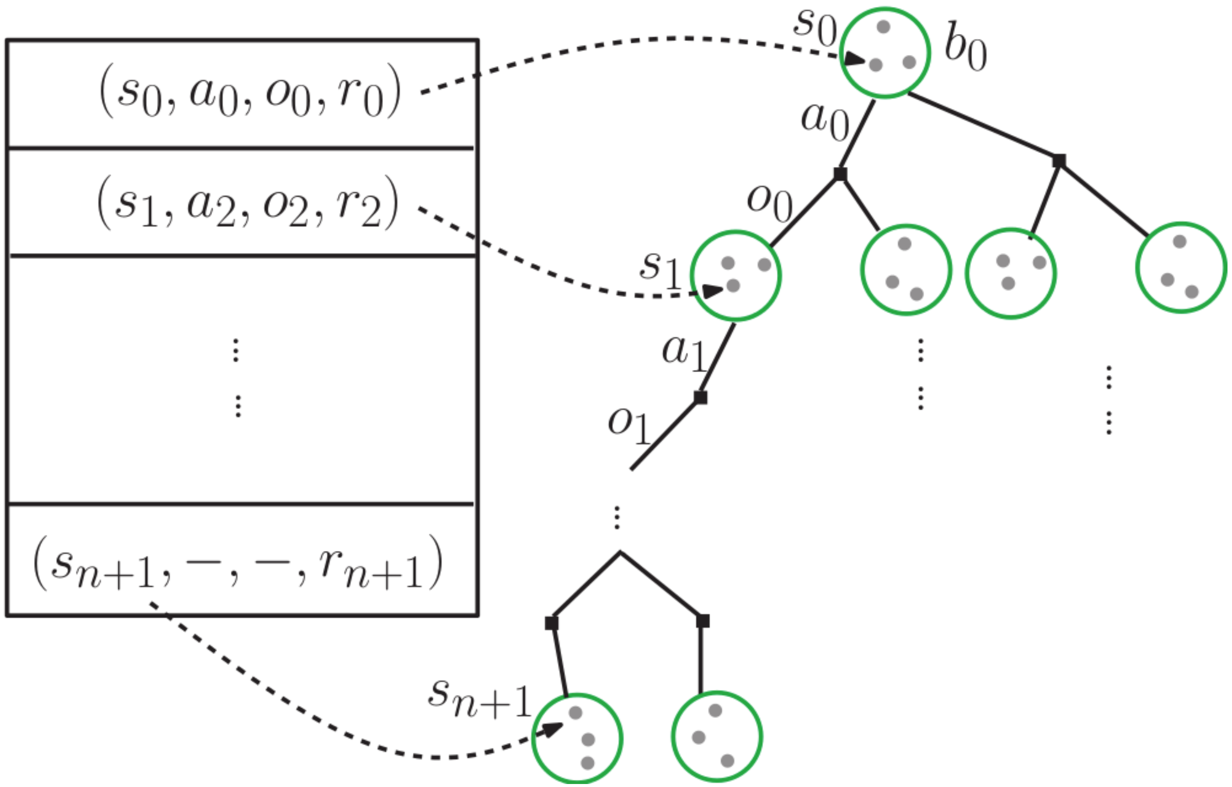
The Tree is now Belief Tree



- Nodes represent beliefs
- Edges represent pairs of action and observation

POMDP Solving: MCTS-based

- Sample: History in MCTS style
- Action selection: UCB1 wrt beliefs



David and Joel Veness. "Monte-Carlo planning in large POMDPs." NeurIPS'10

This session

- ✓Uncertainty is ubiquitous in robotics
- ✓Frameworks for planning (sequential decision-making) under uncertainty
  - ✓Markov Decision Processes (MDPs)
  - ✓Partially Observable MDPs (POMDPs)
  - ✓Reinforcement Learning
- ✓Offline Solving: Value Iteration
  - ✓Value Iteration for MDP
  - ✓Point-based Value Iteration for POMDP
- ✓Online Solving: MCTS-based
  - ✓MDP
  - ✓POMDP

What's Next?

- Other approaches for solving or approximately solving MDPs/POMDPs
  - Policy Iteration
  - LP
  - Deep Learning
  - Integrated Planning and Learning
- For robotics problems, approximate solvers for
  - Continuous high-dimensional state, action, and observation spaces
  - Long planning horizon
  - Complex dynamics

H. Kurniawati. Partially Observable Markov Decision Processes and Robotics. *Annual Review of Control, Robotics, and Autonomous Systems.* Vol. 5. No. 1. 2022.

Wait…

- How about RL solving?
  - RL can be framed as a POMDP, and so if we can solve POMDP, we can solve RL ☺
  - Next lecture is about RL