Sequential Decision Making and Reinforcement Learning

(SDMRL)

MCTS and Planning with Partial Information

Sarah Keren

The Taub Faculty of Computer Science
Technion - Israel Institute of Technology

Agenda

MCTS

Planning while accounting for partial observability

Monte-Carlo Tree Search

Monte Carlo Tree Search

WIKIPEDIA
The Free Encyclopedia

Main page
Contents
Current events
Random article
About Wikipedia
Contact us
Donate
Contribute

Article Talk Read Edit View hi

Monte Carlo tree search

From Wikipedia, the free encyclopedia

In computer science, **Monte Carlo tree search** (**MCTS**) is a heuristic search algorithm for some kinds of decision processes, most notably those employed in software that plays board games. In that context MCTS is used to solve the game tree.

MCTS was combined with neural networks in 2016 for computer Go.[1] It has been used in other board games like chess and shogi,[2] games with incomplete information such as bridge[3] and poker,[4] as well as in turn-based-strategy video games (such as Total War: Rome II's implementation in the high level campaign AI[5]). MCTS has also been used in self-driving cars, for example in Tesla's Autopilot software.[6]

GeeksforGeeks

paration Topic-wise Practice C++ Java Python Competitive Programming Machine Learning HTML

Difficulty Level : Hard • Last Updated : 19 Jan, 2022

Monte Carlo Tree Search (MCTS) is a search technique in the field of Artificial Intelligence (AI). It is a probabilistic and heuristic driven search algorithm that combines the classic tree search implementations alongside machine learning principles of reinforcement learning.
In tree search, there's always the possibility that the current best action is actually not the most optimal action. In such cases, MCTS algorithm becomes useful as it continues to evaluate other alternatives periodically during the learning phase by executing them, instead of the current perceived optimal strategy. This is known as the " ***exploration-exploitation trade-off***". It exploits the actions and strategies that is found to be the best till now but also must continue to explore the local space of alternative decisions and find out if they could replace the current best.

towardsdatascience.com/monte-carlo-tree-search-158a917a8baa

SAGAR SHARMA
Aug 1, 2018 · 6 min read · Listen

Monte Carlo Tree Search

MCTS For Every Data Science Enthusiast

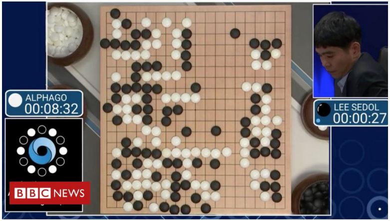
The Games like Tic-Tac-Toe, Rubik's Cube, Sudoku, Chess, Go and many others have common property that lead to exponential increase in the number of possible actions that can be played. These possible steps increase exponentially as the game goes forward. Ideally if you can predict every possible move and its result that may occur in the future. You can increase your chance of winning.

But since the moves increase exponentially — the computation power that is required to calculate the moves also goes through the roof.

Monte Carlo Tree Search is a method usually used in games to predict the path (moves) that should be taken by the policy to reach the final winning solution.

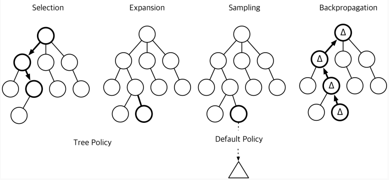
Monte Carlo Tree Search

- Combines exploration and exploitation to choose an action in a tree search setting
- Is relevant to classical planning, stochastic planning, and game playing
  - Revolutionized the world of computer Go
- Has many different variants
- Explosion in interest, applications far beyond games: Planning, motion planning, optimization, finance, energy management



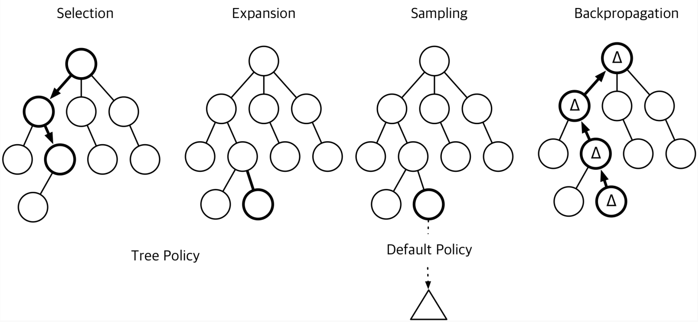
Key Idea: use **Monte Carlo simulation** to accumulate value estimates to guide towards highly rewarding trajectories in a **search tree**.

- Each simulation consists of two phases
  - **Tree policy:** pick actions to maximise values
  - **Default / roll-out policy:** pick actions randomly to simulate a trajectory.
- Repeat (each simulation)
  - **Evaluate** states $Q(S, A)$ by Monte-Carlo evaluation
  - **Improve** tree policy e.g. by ϵ-greedy.
- Converges on the optimal search tree $Q(S, A) \rightarrow q(S, A)$



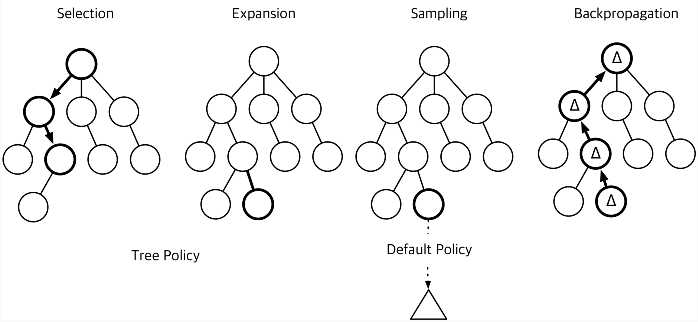
Monte Carlo Tree Search

function MONTE-CARLO-TREE-SEARCH(*state*) **returns** *an action*
  tree ← NODE(*state*)
  while IS-TIME-REMAINING() **do**
    leaf ← SELECT(*tree*)
    child ← EXPAND(*leaf*)
    result ← SIMULATE(*child*)
    BACK-PROPAGATE(*result*, *child*)
  return the move in ACTIONS(*state*) whose node has highest number of playouts

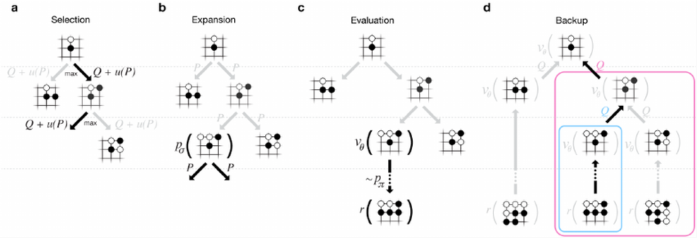


Monte Carlo Tree Search

- Selection
- Expansion
- Sampling / Simulation
- Back-propogation



Example: MCTS for GO



MCTS is not just for games

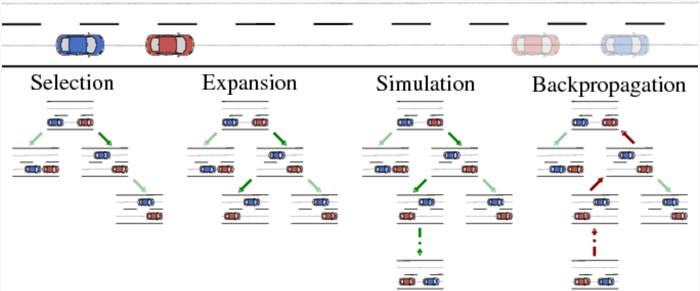


Fig. 1: Phases of Monte Carlo Tree Search for an overtaking maneuver; the

https://www.semanticscholar.org/paper/Decentralized-Cooperative-Planning-for-Automated-Kurzer-Zhou/585b73322365ba2d0afa7449691c81cb98777599

Selection Policy

- Use results of simulations to guide growth of the game tree
  - Exploitation: focus on promising moves
  - Exploration: focus on moves where uncertainty about evaluation is high
- Seems like two contradictory goals
  - Theory of bandits can help

Selection Policy: Multi-Armed Bandit Problem

- We can choose among several arms
- Each arm pull is independent of other pulls
- Each arm has fixed, unknown average payoff
- Which arm has the best average payoff?

Selection Policy: UCB1 [Auer et al 02]

- First, try each arm once
- Each arm pull is independent of other pulls
- Then, at each time step:
- Choose arm i that maximizes the UCB1 formula for the upper confidence

$$v_i + C \times \sqrt{\frac{ln(N)}{n_i}}$$

where

- v_i - current estimation of the value of bandit i
- C - tunable parmater to balance exploration / exploitation
- N total number of trials
- n_i - no. of trials for bandit

How is this relevant to search trees ?

Selection Policy: UCT (UCB applied to trees)

- UCB makes single decision
- What about sequences of decisions (e.g. planning games?)

Selection Policy: UCT (UCB applied to trees)

- UCB makes single decision
- What about sequences of decisions (e.g. planning games?)
- Answer: use a look-ahead tree
  - Bandit arm ≈ move in a game
  - Payoff ≈ quality of move
  - Regret ≈ difference to best move
- Apply UCB-like formula for node selection
  - Choose ”optimistically” where to expand next

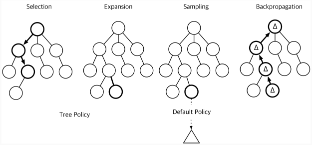
$$UCB(s) = \frac{U(s)}{N(s)} + C \times \sqrt{\frac{ln(N(s.parent))}{N(s)}}$$

where

- $U(s)$ - total utility of all rollouts that went through s
- N(s) - number of rollouts though s.

Simulation / Roll-out Policy

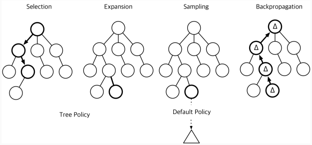
- Default roll-out policy is to make uniform random moves
- Goal is to find strong correlations between initial position and result of a simulation
- Domain independent techniques for games : Ideas ?



From Sarit Kraus: *https://u.cs.biu.ac.il/~krauss/advai2018/MCTS.pdf*

Simulation / Roll-out Policy

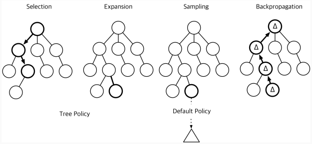
- Default roll-out policy is to make uniform random moves
- Goal is to find strong correlations between initial position and result of a simulation
- Domain independent techniques for games : Ideas ?
    - If there is an immediate win, take it
    - Avoid immediate losses
    - Avoid moves that give opponent immediate win
    - Last Good Reply
    - Using prior knowledge



From Sarit Kraus: *https://u.cs.biu.ac.il/~krauss/advai2018/MCTS.pdf*

Simulation / Roll-out Policy

- Last Good Reply (Drake 2009), Last Good Reply with Forgetting (Baier et al 2010)
- Machine-learned pattern values (Silver 2009)
- Simulation balancing (Silver and Tesauro 2009)
- Using prior knowledge



From Sarit Kraus: *https://u.cs.biu.ac.il/~krauss/advai2018/MCTS.pdf*

MCTS efficiency

MCTS efficiency

- The time to perform a rollout is linear in the depth of the tree
- This gives plenty of time to consider multiple rollouts
- For example
    - if:
        - Branching factor b=32
        - Average game length (tree depth) is d=100
        - We can compute 10^9 moves
    - Minimax can search 6 ply deep
    - AlphaBeta can search up to 12 ply deep
    - MCTS can do 10^7 rollouts
- Works great for games with
    - Large branching factor (then minimax can't search deep enough)
    - Poor evaluation function

MCTS for MDPs

A **Markov Decision Process**(MDP) is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ where

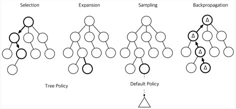
- $\mathcal{S}$ is a finite set of states
- $\mathcal{A}$ is a finite set of actions
- $\mathcal{P}$ is a state transition probability matrix $\mathcal{P}^a_{s,s'} = \mathcal{P}[S_{t+1} = s' | S_t = s, A_t = a]$
- $\mathcal{R}$ is a reward function, $\mathcal{R}^a_s = \mathbb{E}[R_{t+1} | S_t = s, A_t = a]$, and
- **optional:** γ is a discount factor

How to perform ?

- Selection
- Expansion
- Sampling / Simulation
- Back-propogation

MCTS: Summery

- UCB, UCT are very important algorithms in both theory and practice with well-founded convergence guarantees under relatively weak conditions
- Applicable to a variety of games and other applications, as it is domain independent
- Basis for extremely successful programs for games and many other applications
- Very general algorithm for decision making
  - Works with very little domain-specific knowledge
  - (But) needs a simulator of the domain
  - Can take advantage of knowledge when present
  - Anytime algorithm - can stop the algorithm and provide answer immediately, though improves answer with more time

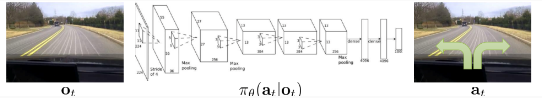


Planning With Partial Observability

- Can we use a **Markov Decision Process**(MDP) $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ to account for partially observable environments ?

Accounting for Partial Information

Sometimes, yes.



Sometimes, an MDP is not enough.

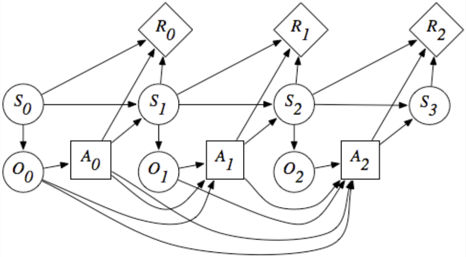
Remidner: Partially Observable Markov Decision Process (POMDP)

A **Partially Observable Markov Decision Process**(POMDP) is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma, \Omega, \mathcal{O}, \beta_0 \rangle$ where

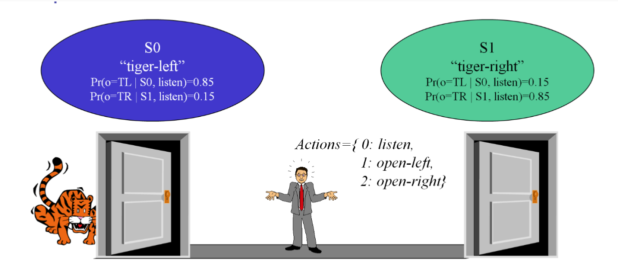
- $\mathcal{S}$, $\mathcal{A}$, $\mathcal{P}$, $\mathcal{R}$ and γ are as for an MDP.
- Ω is a set of observations (observation tokens),
- $\mathcal{O}$ is a sensor function specifying the conditional observation probabilities $\mathcal{O}^o_{s,a} = \mathcal{P}[O_{t+1} = o | S_t = s, A_t = a]$ of receiving observation token $o \in \Omega$ in state s after applying action a [1].
- β_0 the initial belief: a probability distribution over the states such that $\beta_0(s)$ stands for the probability of s being the true initial state.

[1] alternatively: $\mathcal{O}^o_s = \mathcal{P}[o_t = o | S_t = s]$

POMDP- graphical form

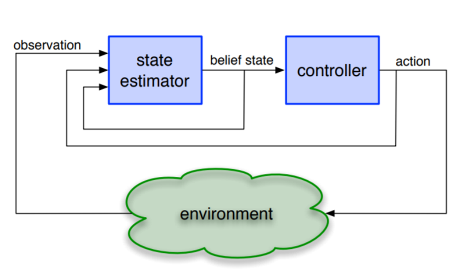


POMDP example



Planning in Belief Space

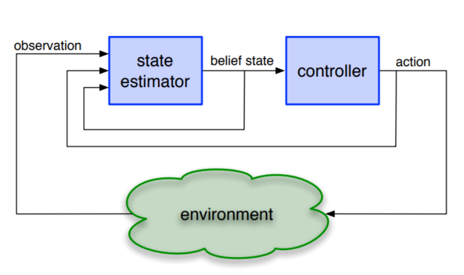
- A **belief** is a probability distribution over the possible world states such that $b(s)$ stands for the probability that s is the true world state.
- In partially observable domains, we may have a **sensor model / state estimator** represented as a mapping function from what is observed to the actual world state.



From Kaelbling, L. P., and T. Lozano-Perez. “Integrated Task and Motion Planning in Belief Space” 2013 *https://dspace.mit.edu/bitstream/handle/1721.1/87038/Kaelbling_Integrated%20task.pdf?sequence=1&isAllowed=y*

Two key challenges when planning in belief space:

- Belief tracking - what is the state of the world ?
- Policy computation - what is the best action to perform ?

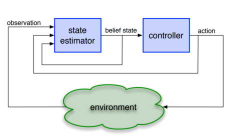


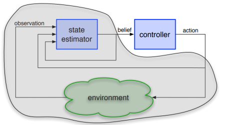
Pineau, Nicholas and Thrun. "A hierarchical approach to POMDP planning and execution." 2001. https://www.cs.mcgill.ca/ jpineau/files/jpineau-icml01.pdf

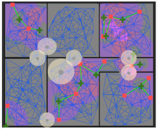
Planning in Belief Space: Solution Approaches

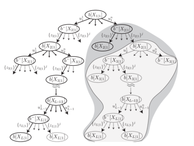
Combinations of different approaches:

- Planning in a **belief MDP**, an MDP with beliefs as states
- Sampling / discretization
- Approximations / relaxations









See work by Vadim Indelman from the Technion, e.g.,

https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8793548

Belief Update

When receiving an observation o, the agent updates its current belief β using its **belief update function** $\tau : \mathcal{B} \times \Omega \times \mathcal{X} \mapsto \mathcal{B}$ which maps belief $\beta \in \mathcal{B}$ to the new belief.

Commonly, a **Bayesian filter** is used:

$$\beta^{o,a} = \frac{\hat{P}\left(o|s,a\right)\beta\left(s\right)}{\int_{s' \in \mathcal{S}} \hat{P}\left(o|s',a\right)\beta\left(s'\right)ds'} \tag{1}$$

where $\beta\left(s\right)$ is the estimated probability that s is the actual world state when the new observation o is emitted.
Discrete version:

$$\beta^{o,a} = \frac{\hat{P}\left(o|s,a\right)\beta\left(s\right)}{\Sigma_{p' \in \mathcal{P}} \hat{P}\left(o|s',a\right)\beta\left(s'\right)} \tag{2}$$

Belief Update - Example

Planning with POMDPs

- A policy $\pi : \beta \mapsto \mathcal{A}$ of a POMDP maps the current belief into an action.
- The belief is assumed to be a sufficient statistic and an optimal policy is the solution of a continuous space “belief MDP”
- Some relevant links:
  - *https://people.csail.mit.edu/lpk/papers/aij98-pomdp.pdf*
  - *https://people.eecs.berkeley.edu/~pabbeel/cs287-fa13/slides/pomdps.pdf*
  - *https://cs.brown.edu/research/ai/pomdp/tutorial/pomdp-solving.html*
  - *https://www.youtube.com/watch?v=cTu7mvRE354*

Value iteration for POMDPs

Bellman optimality for MDPS:

$$\mathcal{V}^*(s) = \max_a \sum_{s'} \mathcal{P}(s'|s,a)\left[\mathcal{R}(s,a,s') + \gamma \mathcal{V}^*(s')\right]$$

Bellman optimality for POMDPs:

Value iteration for POMDPs

Bellman optimality for MDPS:

$$\mathcal{V}^*(s) = \max_a \sum_{s'} \mathcal{P}(s'|s,a)\left[\mathcal{R}(s,a,s') + \gamma \mathcal{V}^*(s')\right]$$

Bellman optimality for POMDPs:

$$\mathcal{V}^*(\beta) = \max_a \left[\mathcal{R}(\beta,a) + \gamma \sum_o \mathcal{P}(o|a,\beta)\mathcal{V}^*(\tau(\beta,a,o))\right]$$

Value iteration for POMDPs

Bellman optimality for MDPS:

$$\mathcal{V}^*(s) = \max_a \sum_{s'} \mathcal{P}(s'|s,a)\left[\mathcal{R}(s,a,s') + \gamma \mathcal{V}^*(s')\right]$$

Bellman optimality for POMDPs:

$$\mathcal{V}^*(\beta) = \max_a \left[\mathcal{R}(\beta,a) + \gamma \sum_{o} \mathcal{P}(o|a,\beta)\mathcal{V}^*(\tau(\beta,a,o))\right]$$

Update formula for MDPS:

$$\mathcal{V}_{k+1}(s) = \max_a \sum_{s'} \mathcal{P}(s'|s,a)\left[\mathcal{R}(s,a,s') + \gamma \mathcal{V}_k(s')\right]$$

Value iteration for POMDPs

Bellman optimality for MDPS:

$$\mathcal{V}^*(s) = \max_a \sum_{s'} \mathcal{P}(s'|s,a) \left[\mathcal{R}(s,a,s') + \gamma \mathcal{V}^*(s')\right]$$

Bellman optimality for POMDPs:

$$\mathcal{V}^*(\beta) = \max_a \left[\mathcal{R}(\beta,a) + \gamma \sum_{o} \mathcal{P}(o|a,\beta)\mathcal{V}^*(\tau(\beta,a,o))\right]$$

Update formula for MDPS:

$$\mathcal{V}_{k+1}(s) = \max_a \sum_{s'} \mathcal{P}(s'|s,a) \left[\mathcal{R}(s,a,s') + \gamma \mathcal{V}_k(s')\right]$$

Update formula for POMDPs:

$$\mathcal{V}_{k+1}(\beta) = \max_a \left[\mathcal{R}(\beta,a) + \gamma \sum_{o\in\Omega} \mathcal{O}(o|a,\beta)\mathcal{V}_k(\tau(\beta,a,o))\right]$$

Value iteration for POMDPs

$$\mathcal{V}_{k+1}(\beta) = \max_a \left[\mathcal{R}(\beta, a) + \gamma \sum_{o \in \Omega} \mathcal{O}(o|a, \beta) \mathcal{V}_k(\tau(\beta, a, o)) \right]$$

Problems?

Value iteration for POMDPs

$$\mathcal{V}_{k+1}(\beta) = \max_a \left[\mathcal{R}(\beta, a) + \gamma \sum_{o \in \Omega} \mathcal{O}(o|a, \beta) \mathcal{V}_k(\tau(\beta, a, o)) \right]$$

Problems?

- Reward is not a function of the belief, but of the state
- While states are discrete - beliefs are continuous (so the space is ∞)

Value iteration for POMDPs

Reward is a function of the state:

$$\mathcal{V}_{k+1}(\beta) = \max_a \left[\mathcal{R}(\beta, a) + \gamma \sum_{o \in \Omega} \mathcal{O}(o|a, \beta) \mathcal{V}_k(\tau(\beta, a, o)) \right]$$

Reward is a function of the state:

$$\mathcal{V}_{k+1}(\beta) = \max_a \left[\mathcal{R}(\beta, a) + \gamma \sum_{o \in \Omega} \mathcal{O}(o|a, \beta)\mathcal{V}_k(\tau(\beta, a, o))\right]$$

Beliefs are continuous:

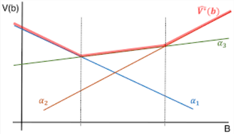
Reward is a function of the state:

$$\mathcal{V}_{k+1}(\beta) = \max_a \left[\mathcal{R}(\beta, a) + \gamma \sum_{o \in \Omega} \mathcal{O}(o|a, \beta) \mathcal{V}_k(\tau(\beta, a, o)) \right]$$

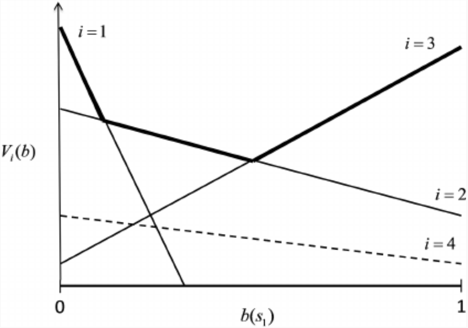
Beliefs are continuous:

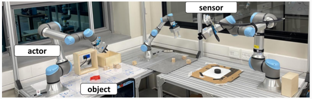
Instead of considering beliefs, we consider a finite set of α-vectors that represent beliefs.

$$\mathcal{V}_k(\beta) = max_{\alpha \in \Gamma_k} \alpha \cdot \beta = max_{\alpha \in \Gamma_k} \Sigma_s \alpha(s) \cdot \beta(s)$$



Value iteration for POMDPs





Value iteration for POMDPs

function POMDP-VALUE-ITERATION($pomdp, \epsilon$) **returns** a utility function
  inputs: $pomdp$, a POMDP with states S, actions $A(s)$, transition model $P(s' \mid s, a)$,
    sensor model $P(e \mid s)$, rewards $R(s)$, discount γ
    ϵ, the maximum error allowed in the utility of any state
  local variables: U, U', sets of plans p with associated utility vectors α_p

  $U' \leftarrow$ a set containing just the empty plan $[\,]$, with $\alpha_{[\,]}(s) = R(s)$
  repeat
    $U \leftarrow U'$
    $U' \leftarrow$ the set of all plans consisting of an action and, for each possible next percept,
      a plan in U with utility vectors
    $U' \leftarrow$ REMOVE-DOMINATED-PLANS(U')
  until MAX-DIFFERENCE(U, U') $\leq \epsilon(1-\gamma)/\gamma$
  return U

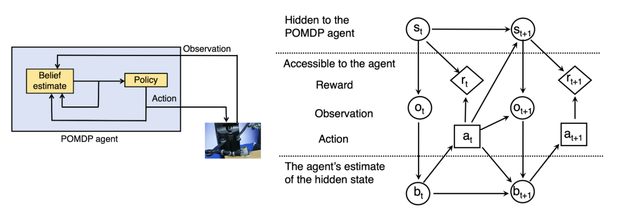
- A high-level sketch of the value iteration algorithm for POMDPs.
- The REMOVE-DOMINATED-PLANS step and MAX-DIFFERENCE test are typically implemented as linear programs.

- SARSOP: Efficient Point-Based POMDP Planning by Approximating Optimally Reachable Belief Spaces. Kurniawati et al. 2008 *https://bigbird.comp.nus.edu.sg/m2ap/wordpress/wp-content/uploads/2016/01/rss08.pdf*
- Efficient point-based POMDP planning by approximating optimally reachable belief spaces: Kurniawati (2021): *https://arxiv.org/pdf/2107.07599.pdf*

Planning with POMDPs

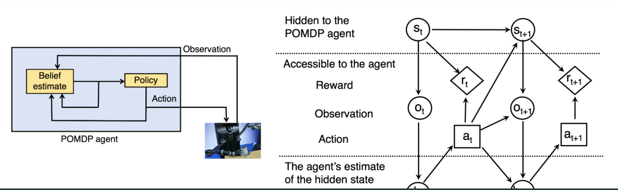
When a POMDP $\langle \mathcal{S}, \mathcal{A}, \mathbb{P}, \mathcal{R}, \gamma, \Omega, \mathcal{O}, \beta_0 \rangle$ is used to represent a robot's task

- the transition function is typically represented as a noisy dynamics function $s' = f(s, a, \eta)$, where $s, s' \in \mathcal{S}$ and $\eta \sim N$ is a noise vector sampled from noise distribution N, while $f(\cdot)$ denotes the system's dynamics.
- Similarly, $\mathcal{O}$ denotes the sensor/ observation function, representing errors and noise in measurement and perception.



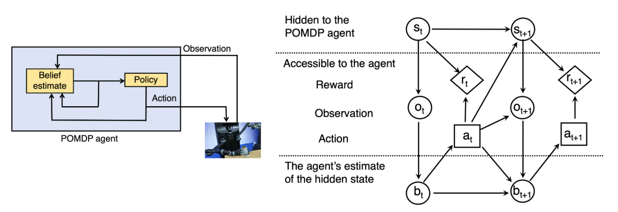
Planning with POMDPs

- POMDP is powerful in its quantification of the non-deterministic effects of actions and partial observability due to errors in sensor measurements and in perception
- The computed policy will balance information gathering and goal attainment.
- But precisely because of this, POMDP is notorious for its high computational complexity and deemed impractical for robotics.
- Until recently, most benchmark problems for POMDPs had less than 30 states and the best algorithms that could solve them took hours.



Planning with POMDPs

- In the past 2 decades, POMDPs solving capabilities have advanced tremendously, thanks to **sampling-based approximate solvers**.
- Although optimality is compromised, robustness and computational efficiency is improved: practical for many realistic robotics problems.



Algorithm 1 A typical program skeleton for sampling-based POMDP solvers

Initialize policy π and a set of sampled beliefs B
{Generally, B is initialised to contain only a single belief (e.g., the initial belief b_0)}
repeat
  Sample a (set of) beliefs {Some methods sample histories (a history is a sequence of action–observation tuples) rather than beliefs. In POMDPs, beliefs provide sufficient statistics of the entire history [25], and therefore the two provide equivalent information}
  Estimate the values of the sampled beliefs
  {Generally, via a combination of heuristics and update / backup operation}
  Update π {In most methods, this step is a byproduct of the previous step}
until Stopping criteria is satisfied

- Key idea: sample a set of representative beliefs and computes optimal policy only for them, thus substantially reducing complexity.
- Which set would be sufficiently representative ?
  - A variety of sampling strategies have been proposed to select the sample set and to estimate the values of the sampled beliefs.
  - Most sampling-based approximate POMDP solvers are **anytime**
  - Some methods compute upper and lower bound estimates of the value functions
  - Can be broadly divided into offline and online.

SARSOP: Efficient Point-Based POMDP Planning by Approximating Optimally Reachable Belief Spaces. Kurniawati et al. 2008 *https://bigbird.comp.nus.edu.sg/m2ap/wordpress/wp-content/uploads/2016/01/rss08.pdf*

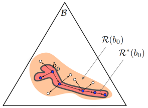


Fig. 1. Belief space $\mathcal{B}$, reachable space $\mathcal{R}(b_0)$, and optimally reachable space $\mathcal{R}^*(b_0)$. Note that $\mathcal{R}^*(b_0) \subseteq \mathcal{R}(b_0) \subseteq \mathcal{B}$.

- Some early POMDP algorithms sample the entire belief space B, using a uniform sampling distribution, such as a grid.
- More recent point-based algorithms sample only $\mathcal{R}(\beta_0)$, the subset of belief points reachable from a given initial point $\beta_0 \in \mathcal{B}$ under arbitrary sequences of actions.

SARSOP

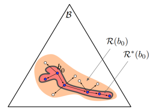


Fig. 1. Belief space $\mathcal{B}$, reachable space $\mathcal{R}(b_0)$, and optimally reachable space $\mathcal{R}^*(b_0)$. Note that $\mathcal{R}^*(b_0) \subseteq \mathcal{R}(b_0) \subseteq \mathcal{B}$.

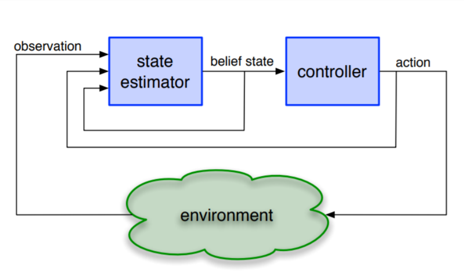
- SASOP pushes this direction further, by sampling near $\mathcal{R}^*(\beta_0)$, a subset of belief points reachable from β_0 under we**optimal sequences of actions** ($\mathcal{R}^*(\beta_0)$ is usually much smaller than $\mathcal{R}(\beta_0)$).
- Optimality not achievalbe, so approximations of $\mathcal{R}^*(\beta_0)$ are used.
  - Use successive approximations of $\mathcal{R}^*(\beta_0)$ and converge to it iteratively.
  - The algorithm relies on heuristic exploration to sample $\mathcal{R}(\beta_0)$ and improves sampling over time through a simple on-line learning technique.
  - Bounding technique are used to avoid sampling in regions that are unlikely to be optimal
  - This leads to substantial gain in computational efficiency

What if we don't have full access to the model of the environment ?

MCTS in POMDPs

Beliefs and Belief Tracking

- The agent maintains its belief via a **state estimator** - which we will refer to as the process of **Belief Tracking**.



From Kaelbling, L. P., and T. Lozano-Perez. "Integrated Task and Motion Planning in Belief Space" 2013 *https://dspace.mit.edu/bitstream/handle/1721.1/87038/Kaelbling_Integrated%20task.pdf?sequence=1&isAllowed=y*

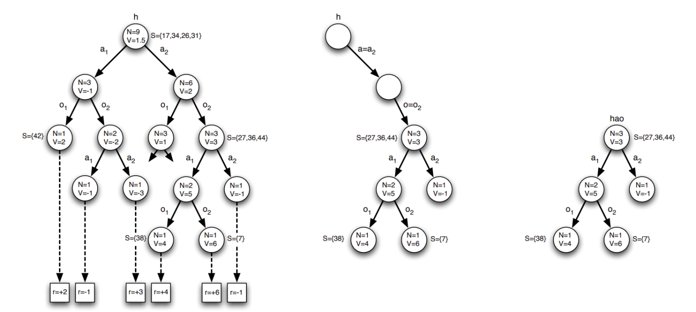
Partially Observable Markov Decision Process (POMDP)

A **Partially Observable Markov Decision Process**(POMDP) is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma, \Omega, \mathcal{O}, \beta_0 \rangle$ where

- $\mathcal{S}$, $\mathcal{A}$, $\mathcal{P}$, $\mathcal{R}$ and γ are as for an MDP.
- Ω is a set of observations (observation tokens),
- $\mathcal{O}$ is a sensor function specifying the conditional observation probabilities $\mathcal{O}^o_{s,a} = \mathcal{P}[O_{t+1} = o | S_t = s, A_t = a]$ of receiving observation token $o \in \Omega$ in state s after applying action a [2].
- β_0 the initial belief: a probability distribution over the states such that $\beta_0(s)$ stands for the probability of s being the true initial state.

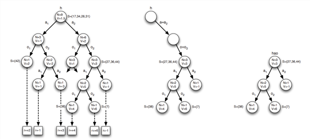
[2] alternatively: $\mathcal{O}^o_s = \mathcal{P}[o_t = o | S_t = s]$

POMCP


h
N=9 V=1.5
S={17,34,26,31}
a1
a2
N=3 V=-1
N=6 V=2
o1
o2
S={42}
N=1 V=2
N=2 V=-2
N=3 V=1
N=3 V=3
S={27,36,44}
N=1 V=-1
N=1 V=-3
N=2 V=5
N=1 V=-1
S={38}
N=1 V=4
N=1 V=6
S={7}
r=+2
r=-1
r=+3
r=+4
r=+6
r=-1
a=a2
o=o2
hao

POMCP

- Extending MCTS to POMDPs.
- In a problem with n states, value iteration reasons about an n-dimensional belief state.
- Furthermore, the number of histories that it must evaluate is exponential in the horizon.
- Basic idea: use Monte-Carlo in two ways:
  - sampling start states from the belief state
  - sampling histories using a black box simulator.
- The algorithm constructs online a search tree of histories.



Algorithm 1 Partially Observable Monte-Carlo Planning

procedure SEARCH(h)
  repeat
    if $h = empty$ **then**
      $s \sim \mathcal{I}$
    else
      $s \sim B(h)$
    end if
    SIMULATE($s, h, 0$)
  until TIMEOUT()
  return $\underset{b}{\operatorname{argmax}}\, V(hb)$
end procedure

procedure ROLLOUT($s, h, depth$)
  if $\gamma^{depth} < \epsilon$ **then**
    return 0
  end if
  $a \sim \pi_{rollout}(h, \cdot)$
  $(s', o, r) \sim \mathcal{G}(s, a)$
  return $r + \gamma.$ROLLOUT($s', hao, depth{+}1$)
end procedure

procedure SIMULATE($s, h, depth$)
  if $\gamma^{depth} < \epsilon$ **then**
    return 0
  end if
  if $h \notin T$ **then**
    for all $a \in \mathcal{A}$ **do**
      $T(ha) \leftarrow (N_{init}(ha), V_{init}(ha), \emptyset)$
    end for
    return ROLLOUT($s, h, depth$)
  end if
  $a \leftarrow \underset{b}{\operatorname{argmax}}\, V(hb) + c\sqrt{\frac{\log N(h)}{N(hb)}}$
  $(s', o, r) \sim \mathcal{G}(s, a)$
  $R \leftarrow r + \gamma.$SIMULATE($s', hao, depth + 1$)
  $B(h) \leftarrow B(h) \cup \{s\}$
  $N(h) \leftarrow N(h) + 1$
  $N(ha) \leftarrow N(ha) + 1$
  $V(ha) \leftarrow V(ha) + \frac{R - V(ha)}{N(ha)}$
  return R
end procedure

Recap and what next

- Spectrum of approaches to planning with
  - deterministic and stochastic actions
  - full and partial observability
  - next: what to do when we don't know the model ?

Do we still need to know about planning?

by Subbarao Kambhampati:
"Human, grant me the serenity to accept the things I cannot learn, the data to learn the things I can, and the wisdom to know the difference."[3]

[3] My addition: and the ability to know what to do about it