Solving POMDPs online through HTN Planning and Monte Carlo Tree Search (MCTS)

Robert P. Goldman
SIFT, LLC
rpgoldman@sift.net

TL;DR

- POMDPs are planning problems with partial observability, stochastic transitions, and rewards/costs.
- MCTS is a search method for large trees, well adapted to games. It balances *exploration* of new parts of the space with *exploitation* of parts of the space that look promising.
- SHOP3 is a forward-searching HTN Planner.

SHOP3 + Extensions + MCTS = SHOPPINGSPREE

POMDPs

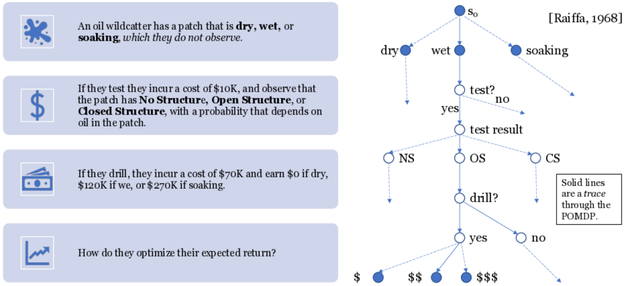
- "POMDP" = Partially Observable Markov Decision Process.
- Model for interaction of an agent with a partially observable environment that evolves stochastically.

A POMDP, $\mathcal{P} = (\mathcal{S}, s_0, \mathcal{A}, \mathcal{T}, \mathcal{R}, \Omega, \mathcal{O})$ where

1. $\mathcal{S}$ is a finite set of states, $s_0 \in S$ is the initial state,$\mathcal{A}$ is a finite set of actions;
2. $\mathcal{T} : \mathcal{S} \times \mathcal{A} \to \Pi(\mathcal{S})$ is the state-transition function, assigning a probability distribution over successor states when an action, a is executed in state s;
3. $\mathcal{R}$ is the reward function, which may be defined over $\mathcal{S} \times \mathcal{A}$, and defines the reward, a real number reward received when a is executed in s. The reward of a finite trace is the sum of the rewards at each step in the trace. Equivalently, we use a cost function, in the work described here.
4. Ω is a set of observations that the agent may make;
5. $\mathcal{O} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \Pi(\Omega)$ is the observation function, which gives a probability distribution over the set of observations that may be received when the agent takes action a in state s_k and the successor state (dictated by $\mathcal{T}$) is s_{k+1}: $\mathcal{O}(s_k, a, s_{k+1}) = P_{(o_t = o | s_{t+1} = s_j, a_t = a)}$

[adapted from Kaelbling, et al., 1998].

Oil Wildcatter Problem: Sample POMDP



Solving POMDPs

- "SOLVE" = Take a sequence of actions that will *maximize expected reward*.
- Often done by computing a *policy*. A policy is like:
  - A conditional plan
  - A closed-loop control program
  - An optimal strategy in a two player game.

Optimal strategy for wildcatter problem:

- Test
- If NS (0.41) stop: -10
- If OS (0.35) drill:
  0.43 * -70 + 0.34 * 50 + .23 * 200 - 10 = 22.9
- If CS (0.24) drill:
  0.21 * - 70 + 0.38 * 50 * 0.42 * 200 – 10 = 78.3

0.41 * -10 + 0.35 * 22.9 + 0.24 * 78.3 = 22.7

There are rounding errors.

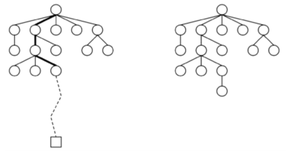
Solving POMDPs *Online*

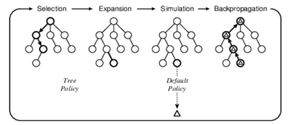
- Compute (approximately) optimal action as needed.
  - Don't compute full policy (often too large).
  - Like playing a game:
    - Often called "a game against nature".
    - Because opponent is nature, maximize *expected* utility, don't maximin.
- Search still needs projection.
- Suggests adopting techniques from game search.

- In the case of the oil wildcatter problem, first do the test, and then wait to see what happens.
- Of course, in this case, it's trivial to compute the full policy, because the state space is small, and we compute everything exactly, so to choose to test, we are computing what's best to do for each test outcome.

Monte Carlo Tree Search (MCTS)

- Developed for games with very large spaces (e.g., go). Very successful in practice (e.g., used in Alpha Go)
- Fusion of search and reinforcement learning.
- Choose search actions balancing *exploration* and *exploitation*.
- Use cheap default policy to extend the search to a terminal node.
- *Backpropagate* results of rollouts to estimate the value of actions in states.
- Make a large number of inexpensive rollouts to final state.
- *Online:* When resources are exhausted, make a move, then repeat.





Monte Carlo Tree Search (MCTS)

Figures from Browne, *et al.* 2012

MCTS and HTN planning: SHOPPINGSPREE

- Built on the SHOP3 HTN planner [Goldman & Kuter 2019; Nau,*et al.* 2003].
- For POMDPs:
  - Partial observability
  - Stochastic outcomes
- Monte Carlo Tree Search:
  - Modify SHOP search to implement the tree policy rule
  - Choices are made based on states that are equivalent wrt visible history
  - Run planner repeatedly until resources exhausted, take action, repeat

SHOP3: Distinctive Features

- Forward planner: at every state of the search we have a fully-specified state.
- Lifted planner: search chooses both methods and variable bindings.
- Expressive power: preconditions can invoke arbitrary code; axioms (Horn clauses).
- Supports PDDL and extended notations.

```
(:method
  ; task
  (delete-expected-weight ?v ?r ?w)
  ; preconditions
  ((expected ?v ?r ?value)
   ;; invoke external code
   (call > ?value ?w))
  ; task network
  (!!exp-weight-dec ?v ?r ?w))

(:action make-product
  :parameters (?p - product)
  :precondition
  (and (not (made ?p))
    (forall (?o - order)
      (imply (includes ?o ?p)
             (started ?o))))
  :effect (made ?p))
```

SHOP3 is available through GitHub: `github.com/shop-planner/shop3`

Representing POMDPs in HTNs

- Partial observability: `hidden` modality.
- Stochasticity:
  - Use `random` function to choose between methods for ***uncontrolled*** tasks.
  - Uncontrolled outcomes modeled by method choice.
- This enables a SHOP3 plan to capture *a single trace* through a POMDP.

Uncontrolled method

```
(:stochastic-method (init-test~)
  (and (assign ?r (random 1.0d0))
       (hidden (oil ?o))
       (test-outcome ?r ?o ?to))
  (:ordered (!init-test~ ?to)))
```

Uncontrolled action

```
(:op (!init-test~ ?x)
      :add ((hidden (test-result ?x)))
      :cost 0)
```

Horn clause axioms for probability tables:

```
(:- (test-outcome ?r ?oil ?obs)
    ((= ?oil dry)
     (dry-test-result ?r ?obs))
    ((= ?oil wet)
     (wet-test-result ?r ?obs))
    ((= ?oil soaking)
     (soaking-test-result ?r ?obs)))
```

MCTS

- For MCTS we run the Shop3 search algorithm repeatedly. Each iteration yields a single trace.
- The cost of each trace is backpropagated to update the MCTS tables.
- We use the UCT tree policy to choose methods and method bindings in the search.
- The search tracks both
  1. full state, to project what will happen and
  2. history + visible state to identify agent's knowledge.
- Tree policy makes choices based on history + visible state (belief equivalent states).

Related Work

- Wichlacz, *et al.*, 2020 – MCTS for classical HTN planning.
- Patra, *et al.*, 2021 – Using MCTS to evaluate PRS-style HTN choices.
- Bansod, *et al.*, HPLAN 2021 (!) – Reentrant planning key for integrating planning and action.
- Much work on solving POMDPs and Influence Diagrams.

Summary and Status

- Current implementation is quite rough:
  - Notation used is *ad hoc*.
  - Implementation is not efficient.
- Future work:
  - Improve the implementation.
  - Integrate plan *repair* algorithm [Goldman, *et al.* HPLAN 2020] with SHOPPINGSPREE.
  - Apply to experiment planning problems where utility is information gain.