L17 – Decision-making under uncertainty

AIMA4e: Chapter 4.3; Chapter 16.1–3

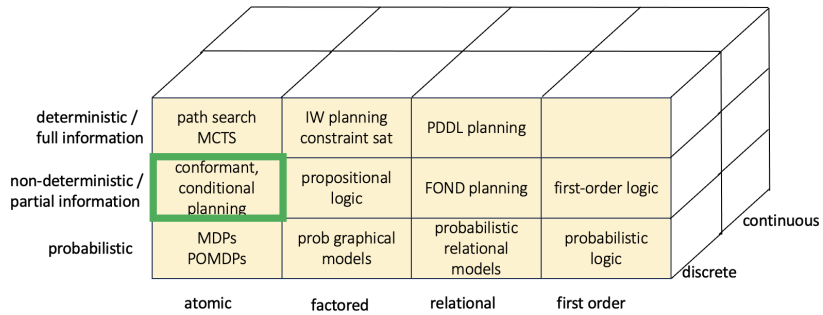
What you should know after this lecture

Decision-making when you are uncertain about the effects of your actions

- What if there is a set of possible outcomes?
- Sequential planning with set-based outcome uncertainty
- Decision-making under probabilistic models of uncertainty: maximizing expected utility.

Path-search problems with future-state uncertainty

Atomic states
Know an initial state
Transitions may only specify a resulting state set



Vacuum world doesn't (always) suck!

Example environment from AIMA:

- Robot in grid world has a vacuum cleaner
- Each cell in grid has dirt or not
- Robot can suck (with vacuum) or move left, right, up, or down
- Want the room to be completely clean

If completely observable and deterministic, then we can solve easily using min-cost-path search. But what if:

- The vacuum or the robot does not always work as expected?

Non-Deterministic Actions

This is sometimes known as the FOND (fully-observable non-deterministic) planning setting.

- Assume that the robot can always correctly observe the current world state
- But, when it takes an action, there is a set of possible outcomes.
- After each action, it can observe the result and decide what to do

Problem formulation:

- Given $(\mathcal{S}, \mathcal{A}, s_0, \tilde{T}, G, C)$ where $\tilde{T} : \mathcal{S} \times \mathcal{A} \to \text{Powerset}(\mathcal{S})$
- Find a contingent (conditional) plan in the form of a decision tree.
- Measuring the cost of a solution is tricky: could be maximum or average over the costs of the possible paths.

And/Or search

Search tree with alternating layers of node types:

- Or nodes : like our traditional search-tree nodes, where we get to pick an action
- And nodes : there are several possible resulting states and we have to find a plan for all of them.

Resulting plan is a tree with

- Internal nodes labeled with actions
- Branches labeled with states (that could possibly occur as a result of the action)
- Terminal leaf nodes indicating plan success.

There are optimal AO search methods (e.g. AO^*) but we will just look in detail at a simple one.

Depth-first And/Or search

AO-DFS$(s_0, (\mathcal{A}, \tilde{T}, G))$

1 **return** OR-SEARCH$(s_0, [\,], (\mathcal{A}, \tilde{T}, G))$

OR-SEARCH$(s, path, (\mathcal{A}, \tilde{T}, G))$

1 **if** $s \in G$: **return** SUCCESS-LEAF()
2 **if** $s \in path$: **return None** // Found a cycle
3 **for** $a \in \mathcal{A}$:
4 $\quad plan_dict = $ AND-SEARCH$(\tilde{T}(s, a), path + [s], (\mathcal{A}, \tilde{T}, G))$
5 $\quad$ **if** $plan\_dict \neq$ **None**: **return** TREE-NODE$(a, plan_dict)$
6 **return None**

AND-SEARCH$(states, path, (\mathcal{A}, \tilde{T}, G))$

1 $plan\_dict = \{\,\}$
2 **for** $s \in states$:
3 $\quad plan = $ OR-SEARCH$(s, path, (\mathcal{A}, \tilde{T}, G))$
4 $\quad$ **if** $plan =$ **None**: **return None**
5 $\quad plan\_dict[s] = plan$
6 **return** $plan\_dict$

What about cycles?

Sometimes, you just need cycles! Throw balls at target until you hit it!

- In line 2 of OR-SEARCH, return a special CYCLE-LEAF(s) node
- In execution, if you hit a CYCLE-LEAF(s), trace up your path to find state s and begin executing from there.

Handling probabilistic uncertainty over outcomes

One-step decision making:

- You have a finite set of possible actions $a_1, \ldots, a_k$
- For each one, there is a distribution P_a over a finite set of outcomes $o_1, \ldots, o_n$
- **You** assign a utility to each outcome $U(o_i) \in R$
- Which action should you select?

See very cool result due to Von Neumann and Morgenstern: under some assumptions about your preferences over uncertain actions (e.g., given two actions with the same two possible outcomes, you'd prefer the one that assigns higher probability to the preferred outcome) then

- There's a real-valued function U defined over actions, and
- Your uncertainty values p have to satisfy the usual axioms of probability theory.

If not, then I can construct a set of bets (a "dutch book") that you will be willing to take, but which guarantee that I profit in expectation.

Maximize expected utilty

In this setting, then, it's rational to selection the action with the maximum expected utility: that is, choose

$$\underset{a}{\text{argmax}} \sum_{o} P_a(o)U(o)$$

Do a lottery

Utility of money

- You don't have to assign utility linearly to amounts of money
- Would you value \$2M twice as much as \$4M?
- Would you value a lottery with 1/2 chance at \$4M and 1/2 chance at nothing the same as a sure \$2M?
- If you prefer the sure bet, then you are risk averse. That means that your utility value, as a function of monetary value, is concave, so that

$$\frac{1}{2}U(4M) + \frac{1}{2}U(0) < U(2M)$$

- Most humans are (roughly) risk averse in the domain of gains and risk seeking in the domain of losses.
- Maximizing expected utility is a good framing but there are lots of example (e.g., Allais' paradox) of ways in which it fails to model actual human preferences.

Utility of money

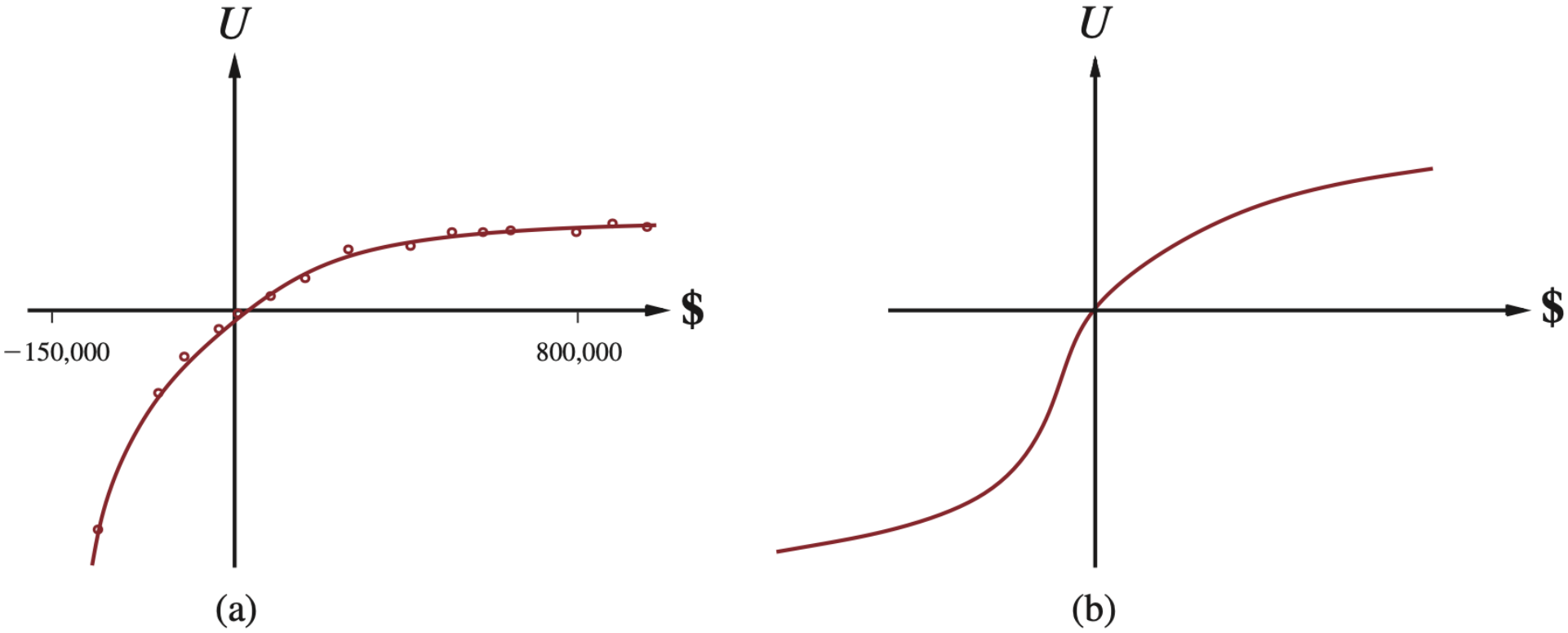


Figure 16.2 The utility of money. (a) Empirical data for Mr. Beard over a limited range. (b) A typical curve for the full range.

Sequential decision making: Markov decision process

- Sequential decision making: agent takes multiple steps
- Reward function maps states (and actions) to real numbers
- Next state depends probabilistically on the state and action
- The current state is exactly known
- Although we don't know, right now, what the next state will be, we will be able to observe it exactly, correctly, when it arises
- Reward function and transition distribution are fixed and known
- **Assume** : Utility of a sequence of states is the sum of the rewards

MDPs: formal definition

- $\mathcal{S}$: Set of possible world states (possibly infinite)
- $\mathcal{A}$: Set of possible actions (usually finite)
- $T : \mathcal{S} \times \mathcal{A} \to \mathcal{P}(\mathcal{S})$: Transition model maps state and action into probability distribution over next states
- $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$: Reward function defined most generally as a function of state, action, next state

Sometimes also

- $s_0 \in \mathcal{S}$: initial state
- $\gamma \in [0, 1]$: discount factor (defined in next slide)

MDPs: utility, reward, and action

We want to find a way of selecting actions in an MDP that is optimal in some sense. Maximize expected utility!
Pick the action now that will get us the state sequence with the best utility, in expectation, assuming we pick all future actions optimally as well!

$$a^* = \operatorname*{argmax}_a \mathop{\mathbb{E}}_{s_1,\ldots,s_H} [U(s_1,\ldots,s_H) \mid a_0 = a, \quad \text{optimal future actions}]$$

Assuming that our utility over sequences is additive (big assumption!) and stationary (big assumption!)[1], then

$$a^* = \operatorname*{argmax}_a \mathop{\mathbb{E}}_{s_1,\ldots,s_H} \left[\sum_{t=0}^{H} R(s_t, a_t, s_{t+1}) \mid a_0 = a, \quad \text{optimal future actions} \right]$$

[1] If your reward is non-stationary, then how much you value something depends on the time. You could potentially address this by putting t in the state.

MDPs: horizon

Different ways to think about how long the agent will be acting

- Finite horizon: H is finite
  - non-stationary policy may select different actions depending on number of steps remaining
- Receding horizon
  - On every step, act as if you have finite H steps remaining
- Indefinite horizon
  - Let $H = \infty$
  - But assume (at least) that the optimal policy has finite $\mathbb{E}_{s_1,\ldots,s_H}\left[\sum_{t=0}^{\infty} r_t\right]$
  - Usually in domains where there are terminal states, and the optimal strategy eventually terminates after a number of steps that is finite in expectation.

Finite-horizon value functions and policies

Utility of being in state s with H steps left to go:[2]

$$V^*_H(s) = \max_a \mathop{\mathbb{E}}_{s_1,\ldots,s_H} \left[\sum_{t=0}^{H} r_t \mid a_0 = a, \quad \text{optimal future actions} \right]$$

$$V^*_0(s) = 0$$

$$V^*_1(s) = \max_a \sum_{s'} P(s' \mid s, a) R(s, a, s')$$

$$V^*_H(s) = \max_a \sum_{s'} P(s' \mid s, a) \left[R(s, a, s') + V^*_{H-1}(s') \right]$$

Note that the optimal policy depends on the number of steps left to go!

$$\pi^*_H(s) = \operatorname*{argmax}_a \sum_{s'} P(s' \mid s, a) \left[R(s, a, s') + V^*_{H-1}(s') \right]$$

[2] AIMA and KAlg use U for this but every single other text uses V. Well...in operations research they often use costs c instead of rewards, try to minimize the expected sum of costs, and use J instead of V.

Q and V

Sometimes it's easier to define values in terms of Q instead of V. No new ideas, really, just bookkeeeping!
Value of starting in state s, taking action a and then continuing optimally:

- Finite horizon:

$$Q^*_H(s,a) = \sum_{s'} P(s' \mid s,a)\left[R(s,a,s') + V^*_{H-1}(s')\right]$$

$$V^*_H(s) = \max_a Q^*_H(s,a) \qquad \pi^*_H(s) = \operatorname*{argmax}_a Q^*_H(s,a)$$

Useful because if you know Q, you can derive π without knowing T or R.

Solution strategies for MDPs

Two main categories

- Online action selection given current state s_0 via some form of search
- Offline solution to derive a complete policy π that can be executed online very efficiently (next lecture)

Expectimax

Searching to a finite depth:

- Use remaining horizon H for finite and receding horizon problems; set $\gamma = 1$
- For infinite horizon problems, if you search to depth H, the error in your value estimate is bounded by $\gamma^H R_{max}/(1-\gamma)$. Note. [3]

EXPECTIMAX$(s, \mathcal{A}, T, R, \gamma, H)$

```
1  def Q(s, a, H):
2        if H = 0 return 0
3        return Σ_s' T(s, a, s') (R(s, a, s') + γ max_a' Q(s', a', H − 1))
4  return argmax_a Q(s, a, H)
```

[3] Why? Because at any point, $R_{max}/(1-\gamma)$ is the value you'd get if you got reward R_{max} on every step; but this big pile of value would be discounted by γ^d if you get it d steps in the future.

Less stupid expectimax

- Cache all (s, a, H) values computed and re-use when possible
- Don't expand zero-probability branches!
- sparse sampling algorithm: To reduce branching factor, sample $m << |S|$ elements s' from $P(s' \mid s, a)$ and average their values (will focus on the most likely outcomes) instead of doing full expectation over s'.
- Monte-Carlo tree search

See also (optionally!) RTDP (real-time dynamic programming)

MCTS for MDPs

Okay to ignore details — just here for reference.
Main differences from deterministic version we looked at earlier:

- *children* is dictionary from chance nodes to action
- We count how many times each action has been tried in each state
- A chance node has a set of children that are resulting state-nodes

MCTS$(s_0, (\mathcal{A}, \mathsf{T}, \mathsf{R}, \gamma, \mathsf{H}), \mathit{iters})$

1 $\mathit{root} = \textsc{StateNode}(s_0, \mathit{horizon} = \mathsf{H}, \mathit{parent} = \textbf{None}, \mathit{children} = \{\}, \mathsf{N} = 0)$
2 **for** $\mathit{iter} \in \{1, \ldots, \mathit{iters}\}$:
3 $\quad \mathit{leaf} = \textsc{select}(\mathit{root})$
4 $\quad \mathit{child} = \textsc{expand}(\mathit{leaf}, \mathcal{A}, \mathsf{T})$
5 $\quad \mathit{value} = \textsc{simulate}(\mathit{leaf}, \mathcal{A}, \mathsf{T}, \mathsf{R})$
6 $\quad \textsc{backup}(\mathit{leaf}, \mathit{value})$
7 $\mathit{max\_child} = \mathsf{max}(\mathit{root.children}, \mathit{key} = \lambda n.\ n.\mathsf{N})$
8 **return** $\mathit{root.children}[\mathit{max\_child}].\mathit{action}$

Monte-Carlo Tree Search (Cont)

EXPAND$(n, \mathcal{A}, T)$

```
    // Unless remaining horizon is 0, add child chance and state nodes and return one
 1  if n.horizon = 0:
 2      return n
 3  for a ∈ 𝒜:
 4      c' = ChanceNode(parent = n, children = { }, U = 0, N = 0)
 5      n.children[c'] = a
 6  c, a = random_choice(n.children)
 7  s' = T(n.s, a)
 8  n' = StateNode(s', n.horizon − 1, parent = c, children = { }, N = 0)
 9  c.children[s'] = n'
10  return
```

SIMULATE$(n, \mathcal{A}, T, R)$

```
    // Randomly finish path and return cumulative reward
1   s = n.state; total_reward = 0
2   for h ∈ (n.horizon, ..., 1):
3       a = random_choice(𝒜) // or use some rollout_policy
4       s' ~ P(s' | s, a)
5       total_reward += R(s, a, s')
6       s = s'
7   return total_reward
```

Monte-Carlo Tree Search (Cont)

```
SELECT(n)
1  c, a = max(n.children, key = λc.UCB(n.N, c.N, c.U))
2  s' ~ P(s' | n.state, c, a)
3  if not s' in c.children:
4        c.children[s'] = StateNode(s', H − 1, parent = c, children = {}, N = 0)
5        return c.children[s']
6  return SELECT(c.children[s'])

BACKUP(n, v_below)
   // Add value v to n's statistics and pass it up
1  n.N += 1
2  if n.parent:
3        a = n.parent.action                                  // Action that led to n
4        v = γ · v_below + R(n.parent.parent.s, a, n.s)  // Value of executing a in parent
5        n.parent.U += v
6        BACKUP(n.parent.parent, v)
```

Monte-Carlo Tree Search (Cont)

There are many strategies for doing backups. Could instead: directly update the U at leaf using value from simulate. But then work back up the path letting

$$U(s,a) += \sum_{s'} R(s,a,s') + \gamma \max_{a'} U(s',a')/N(s',a')$$

BACKUP(n, v_below)

```
    // Add value v to n's statistics and pass it up
1   n.N += 1
2   if n.parent:
3       a = n.parent.action                                  // Action that led to n
4       v = γ · v_below + R(n.parent.parent.s, a, n.s)       // Value of executing a in parent
5       n.parent.U += v
6       BACKUP(n.parent.parent, MAX(n.parent.parent.children, λc. c.U/c.N))
```

Stochastic shortest-paths problems

Min-cost path problems : stochastic shortest-path problems :: reward-maximization problems : MDPs

- $\mathcal{S}$: Set of possible world states (possibly infinite)
- $\mathcal{A}$: Set of possible actions (usually finite)
- $T : \mathcal{S} \times \mathcal{A} \to \mathcal{P}(\mathcal{S})$: Transition model maps state and action into probability distribution over next states
- $G \subset \mathcal{S}$: set of goal states

Sometimes also

- $s_0 \in \mathcal{S}$: initial state
- $C : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$: cost function (treat as always = 1 if missing)

Objective is to find a policy that minimizes the expected cost to reach some state in G.

Convert to MDP with: $\gamma = 1$; $R(s, a, s') = 0$ if $s \in G$;
$R(s, a, s') = -C(s, a)$ if $s \notin G$; $T(s, a, s) = 1$ if $s \in G$.

Deterministic, open-loop approximations

More efficient (but also more approximate) strategies. Risk ignoring low probability but very bad outcomes.

- Rely on replanning (as in receding horizon control)
- Build a deterministic search problem and solve it
  - General MDP $\rightarrow$ reward-maximization problem
  - Stochastic shortest-paths problem $\rightarrow$ min-cost path problem
- Two general strategies:
  - Make T deterministic by assuming the most likely outcome will occur
  - Make T deterministic by allowing the search to choose among the outcomes, but assigning an additional cost of $-\log T(s, a, s')$ to the selected transition.

Cool result: if you have a stochastic shortest-paths problem, and you determinize and assign costs $-\log T(s, a, s')$ to the transitions, then the least cost path to a goal state is the open-loop action sequence that is most likely to reach a goal.