Example: Insurance

Consider the lottery \([0.5,$1000; 0.5, 0]\)

- What is its expected monetary value (EMV)? \((500)\)
- What is its certainty equivalent?
  - Monetary value acceptable in lieu of lottery
  - $400 for most people
- Difference of $100 is the insurance premium
  - There’s an insurance industry because people will pay to reduce their risk
  - If everyone were risk-neutral, no insurance needed!

Example: Insurance

Because people ascribe different utilities to different amounts of money, insurance agreements can increase both parties’ expected utility.

You own a car. Your lottery:

\[L_Y = [0.8, 0; 0.2, -200]\]

i.e., 20% chance of crashing

You do not want -$200!

\[U_Y(-200) = 0.2 U_Y(-$200) = -200\]

\[U_Y(-50) = -150\]

Insurance company buys risk:

\[L_I = [0.8, 50; 0.2, -150]\]

i.e., $50 revenue + your \(L_Y\)

Insurer is risk-neutral:

\[U(L_I) = U(EMV(L))\]

\[U(L_I) = U(0.8\times50 + 0.2\times(-150)) = U($10) > U($0)\]

Example: Human Rationality?

Famous example of Allais (1953)

- A: \([0.8, 4k; 0.2, 0]\)
- B: \([1.0, 3k; 0.0, 0]\)
- C: \([0.2, 4k; 0.8, 0]\)
- D: \([0.25, 3k; 0.75, 0]\)

Most people prefer B > A, C > D

But if \(U(0) = 0\), then

\[B > A \Rightarrow U(3k) > 0.8 U(4k)\]

\[C > D \Rightarrow 0.2 U(4k) > 0.25 U(3k)\]

equivalently: \(0.8 U(4k) > U(3k)\)
**Reinforcement Learning**

- **Basic idea:**
  - Receive feedback in the form of rewards
  - Agent's utility is defined by the reward function
  - Must learn to act so as to maximize expected rewards

**Grid World**

- The agent lives in a grid
- Walls block the agent’s path
- The agent’s actions do not always go as planned:
  - 80% of the time, the action North takes the agent North
  - (if there is no wall there)
  - 10% of the time, North takes the agent West; 10% East
- If there is a wall in the direction the agent would have been taken, the agent stays put
- Small “living” reward each step
- Big rewards come at the end
- Goal: maximize sum of rewards

**Markov Decision Processes**

- An MDP is defined by:
  - A set of states \( s \in S \)
  - A set of actions \( a \in A \)
  - A transition function \( T(s, a, s') \)
  - Prob that \( a \) from \( s \) leads to \( s' \)
  - Also called the model
  - A reward function \( R(s, a, s') \)
    - Sometimes just \( R(s) \) or \( R(s') \)
  - A start state (or distribution)
  - Maybe a terminal state

- MDPs are a family of non-deterministic search problems
  - Reinforcement learning: MDPs where we don’t know the transition or reward functions

**What is Markov about MDPs?**

- Andrey Markov (1856-1922)
- “Markov” generally means that given the present state, the future and the past are independent
- For Markov decision processes, “Markov” means:

\[
P(S_{t+1} = s' | S_t = s, A_t = a_t, S_{t-1} = s_{t-1}, A_{t-1} = s_{t-1}, \ldots S_0 = s_0)
\]

\[
= P(S_{t+1} = s' | S_t = s, A_t = a_t)
\]

**Solving MDPs**

- In deterministic single-agent search problems, want an optimal plan, or sequence of actions, from start to a goal
- In an MDP, we want an optimal policy \( \pi^*: S \rightarrow A \)
  - A policy \( \pi \) gives an action for each state
  - An optimal policy maximizes expected utility if followed
  - Defines a reflex agent

**Example Optimal Policies**

- Reward at states
  - \( R(s) = 0.01 \)
  - \( R(s) = 0.03 \)
  - \( R(s) = 0.4 \)
  - \( R(s) = -2.0 \)
Example: High-Low

- Three card types: 2, 3, 4
- Infinite deck, twice as many 2's
- Start with 3 showing
- After each card, you say "high" or "low"
- New card is flipped
- If you're right, you win the points shown on the new card
- Ties are no-ops
- If you're wrong, game ends

Differences from expectimax:
- #1: get rewards as you go
- #2: you might play forever!

High-Low as an MDP

- States: 2, 3, 4, done
- Actions: High, Low
- Model: \( T(s, a, s') \):
  - \( P(s'=4 \mid 4, \text{Low}) = 1/4 \)
  - \( P(s'=3 \mid 4, \text{Low}) = 1/4 \)
  - \( P(s'=2 \mid 4, \text{Low}) = 1/2 \)
  - \( P(s'=\text{done} \mid 4, \text{Low}) = 0 \)
  - \( P(s'=4 \mid 4, \text{High}) = 1/4 \)
  - \( P(s'=3 \mid 4, \text{High}) = 0 \)
  - \( P(s'=2 \mid 4, \text{High}) = 0 \)
  - \( P(s'=\text{done} \mid 4, \text{High}) = 3/4 \)

- Rewards: \( R(s, a, s') \):
  - Number shown on \( s' \) if \( s \neq s' \) and \( a \) is "correct"
  - 0 otherwise
  - Start: 3

Example: High-Low

<table>
<thead>
<tr>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>High</td>
<td>Low</td>
<td>High</td>
</tr>
</tbody>
</table>

MDP Search Trees

- Each MDP state gives an expectimax-like search tree

Utilities of Sequences

- In order to formalize optimality of a policy, need to understand utilities of sequences of rewards
- Typically consider stationary preferences:
  \[
  [r, r_0, r_1, r_2, \ldots] > [r, r'_0, r'_1, r'_2, \ldots]
  \]
- Theorem: only two ways to define stationary utilities
  - Additive utility:
    \[
    U([r_0, r_1, r_2, \ldots]) = r_0 + r_1 + r_2 + \ldots
    \]
  - Discounted utility:
    \[
    U([r_0, r_1, r_2, \ldots]) = r_0 + \gamma r_1 + \gamma^2 r_2 + \ldots
    \]

Infinite Utilities?!

- Problem: infinite state sequences have infinite rewards
- Solutions:
  - Finite horizon:
    - Terminate episodes after a fixed \( T \) steps (e.g. life)
    - Gives nonstationary policies (\( \pi \) depends on time left)
  - Absorbing state: guarantee that for every policy, a terminal state will eventually be reached (like "done" for High-Low)
  - Discounting: for \( 0 < \gamma < 1 \)
    \[
    U([r_0, \ldots, r_x]) = \sum_{i=0}^{\infty} \gamma^i r_i \leq R_{\text{max}}/(1 - \gamma)
    \]
    - Smaller \( \gamma \) means smaller "horizon" – shorter term focus
Discounting

- Typically discount rewards by $\gamma < 1$ each time step
- Sooner rewards have higher utility than later rewards
- Also helps the algorithms converge

Recap: Defining MDPs

- Markov decision processes:
  - States $S$
  - Start state $s_0$
  - Actions $A$
  - Transitions $P(s'|s,a)$ (or $T(s,a,s')$)
  - Rewards $R(s,a,s')$ (and discount $\gamma$)

- MDP quantities so far:
  - Policy = Choice of action for each state
  - Utility (or return) = sum of discounted rewards

Optimal Utilities

- Fundamental operation: compute the values (optimal expectimax utilities) of states $s$
- Why? Optimal values define optimal policies!
- Define the value of a state $s$:
  $V^*(s) = \text{expected utility starting in } s \text{ and acting optimally}$
- Define the value of a q-state $(s,a)$:
  $Q^*(s,a) = \text{expected utility starting in } s, \text{ taking action } a \text{ and thereafter acting optimally}$
- Define the optimal policy:
  $\pi^*(s) = \text{optimal action from state } s$

The Bellman Equations

- Definition of “optimal utility” leads to a simple one-step lookahead relationship amongst optimal utility values:
  - Optimal rewards = maximize over first action and then follow optimal policy
- Formally:
  $V^*(s) = \max_a Q^*(s,a)$
  $Q^*(s,a) = \sum_{s'} T(s,a,s') \left[ R(s,a,s') + \gamma V^*(s') \right]$
  $V^*(s) = \max_a \sum_{s'} T(s,a,s') \left[ R(s,a,s') + \gamma V^*(s') \right]$

Solving MDPs

- We want to find the optimal policy $\pi^*$
- Proposal 1: modified expectimax search, starting from each state $s$:
  $\pi^*(s) = \arg\max_a Q^*(s,a)$
  $Q^*(s,a) = \sum_{s'} T(s,a,s') \left[ R(s,a,s') + \gamma V^*(s') \right]$
  $V^*(s) = \max_a Q^*(s,a)$

Why Not Search Trees?

- Why not solve with expectimax?
- Why not search trees?
- Problems:
  - This tree is usually infinite (why?)
  - Same states appear over and over (why?)
  - We would search once per state (why?)
- Idea: Value iteration
  - Compute optimal values for all states all at once using successive approximations
  - Will be a bottom-up dynamic program similar in cost to memoization
  - Do all planning offline, no replanning needed!
Value Estimates

- Calculate estimates $V_k(s)$
  - Not the optimal value of $s$!
  - The optimal value considering only next $k$ time steps ($k$ rewards)
  - As $k \to \infty$, it approaches the optimal value
  - Why:
    - If discounting, distant rewards become negligible
    - If terminal states reachable from everywhere, fraction of episodes not ending becomes negligible
    - Otherwise, can get infinite expected utility and then this approach actually won’t work