Making Complex Decisions
CSci 5512: Artificial Intelligence II
Sequential Decision Problems

Planning
- Decision-theoretic planning
  - uncertainty and utility

Search
- explicit actions and subgoals
  - uncertainty and utility

Markov decision problems (MDPs)
- explicit actions and subgoals
  - uncertain sensing

Partially observable MDPs (POMDPs)
  - (belief states)
Markov Decision Process

- States $s \in S$, actions $a \in A$
- Model $T(s, a, s') \equiv P(s'|s, a)$
- Reward function $R(s)$ (or $R(s, a)$, $R(s, a, s')$)

$$R(s) = \begin{cases} 
-0.04 & \text{(small penalty) for nonterminal states} \\
\pm 1 & \text{for terminal states}
\end{cases}$$
Solving MDPs

- In search problems, aim is to find an optimal sequence.
- In MDPs, aim is to find an optimal policy \( \pi(s) \):
  - Best action for every possible state \( s \)
  - Cannot predict where one will end up.
- Optimal policy maximizes expected sum of rewards.
- Optimal policy when state penalty \( R(s) \) is \(-0.04\):

![Diagram showing optimal policy decision areas]
Reward and Optimal Policy

\[ r = [-\infty : -1.6284] \]

\[ r = [-0.4278 : -0.0850] \]

\[ r = [-0.0480 : -0.0274] \]

\[ r = [-0.0218 : 0.0000] \]
Utility of State Sequences

- Need to understand preferences between \textit{sequences} of states
- Typically consider stationary preferences on reward sequences

\[ [r, r_0, r_1, r_2, \ldots] \succ [r, r'_0, r'_1, r'_2, \ldots] \iff [r_0, r_1, r_2, \ldots] \succ [r'_0, r'_1, r'_2, \ldots] \]

**Theorem:** Only two ways to combine rewards over time:

1) \textit{Additive} utility function:
   \[ U([s_0, s_1, s_2, \ldots]) = R(s_0) + R(s_1) + R(s_2) + \cdots \]

2) \textit{Discounted} utility function: For discount factor \( \gamma \)
   \[ U([s_0, s_1, s_2, \ldots]) = R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \cdots \]
Utility of States

- Utility $U(s)$ of a *state* (a.k.a. its *value*)
  - Expected (discounted) sum of rewards (until termination)
  - Assume optimal actions

Choosing the best action is just MEU
- Maximize the expected utility of the immediate successors
- Utilities of the states are given

\[
\begin{array}{cccc}
3 & 0.812 & 0.868 & 0.912 & +1 \\
2 & 0.762 & 0.660 & -1 \\
1 & 0.705 & 0.655 & 0.611 & 0.388 \\
1 &  &  &  & \\
2 &  &  &  & \\
3 &  &  &  & \\
\end{array}
\]
Problem: Infinite lifetimes $\implies$ Additive utilities are infinite

- Finite horizon:
  - Termination at a *fixed time* $T$
  - Non-stationary policy as $\pi(s)$ depends on time left

- Absorbing state(s):
  - With probability 1, agent eventually “dies” for any $\pi$
  - Expected utility of every state is finite

- Discounting:
  - Assuming $\gamma < 1$, $R(s) \leq R_{\text{max}}$,
    $$U([s_0, \ldots s_\infty]) = \sum_{t=0}^{\infty} \gamma^t R(s_t) \leq R_{\text{max}}/(1 - \gamma)$$
  - Average reward per time step
    - **Theorem:** $\pi^*(s)$ has constant gain after initial transient
The Optimal Policy

- Given a policy \( \pi \), the overall (discounted) utility

\[
U_\pi = \sum_{t=0}^{\infty} \gamma^t R(s_t)
\]

- \( U_\pi \) a random variable as \( s_t \) are random
- Optimal policy corresponds to the MEU

\[
\pi^* = \text{argmax}_\pi E[U_\pi] = \text{argmax}_\pi E \left[ \sum_{t=0}^{\infty} \gamma^t R(s_t) | \pi \right]
\]
Dynamic Programming: The Bellman Equation

- Simple relationship among utilities of neighboring states
- Expected sum of rewards = current reward
  + $\gamma \times$ Expected sum of rewards after taking best action
- Bellman equation (1957):

$$U(s) = R(s) + \gamma \max_{a} \sum_{s'} U(s') T(s, a, s')$$

- One equation per state $s$
  - $n$ nonlinear equations in $n$ unknowns
Value Iteration Algorithm

- **Main Idea**
  - Start with arbitrary utility values
  - Update to make them locally consistent
  - Everywhere locally consistent $\Rightarrow$ Global optimality

- **Repeat for every $s$ simultaneously until “no change”**

$$U(s) \leftarrow R(s) + \gamma \max_a \sum_{s'} U(s') T(s, a, s') \quad \forall s$$

![Utility estimates vs. Number of iterations graph](image)
Convergence of Value Iteration

- Define the max-norm \( ||U|| = \max_s |U(s)| \)
  - \( ||U - V|| = \) maximum difference between \( U \) and \( V \)

- Let \( U^t \) and \( U^{t+1} \) be successive approximations to the true \( U \)

**Theorem:** For any two approximations \( U^t \) and \( V^t \)

\[
||U^{t+1} - V^{t+1}|| \leq \gamma ||U^t - V^t||
\]

- Any distinct approximations must get closer to each other
  - Any approximation must get closer to the true \( U \)
  - VI converges to a unique, stable, optimal solution

**Theorem:** If \( ||U^{t+1} - U^t|| < \epsilon(1 - \gamma)/\gamma \), then
\[
||U^{t+1} - U|| < \epsilon
\]

- Once the change in \( U^t \) becomes small, we are almost done

- MEU policy using \( U^t \) may be optimal long before convergence
Policy Iteration

- Search for optimal policy and utility values simultaneously

Algorithm:

1. \( \pi \leftarrow \) an arbitrary initial policy
2. Repeat until no change in \( \pi \)
   1. Compute utilities given \( \pi \)
   2. Update \( \pi \) as if utilities were correct (i.e., local MEU)

To compute utilities given a fixed \( \pi \) (value determination):

\[
U(s) = R(s) + \gamma \sum_{s'} U(s') T(s, \pi(s), s') \quad \text{for all } s
\]

- \( n \) linear equations in \( n \) unknowns, solve in \( O(n^3) \)
Modified Policy Iteration

- Policy iteration often converges in few iterations
  - But each iteration is expensive

- Main Idea:
  - An approximate value determination step
  - Use a few steps of value iteration (with $\pi$ fixed)
  - Start from the value function produced in the last iteration

- Often converges much faster than pure VI or PI

- Leads to much more general algorithms
  - Value/Policy updates can be performed locally in any order

- Reinforcement learning algorithms use such updates
Partial Observability

- POMDP has an observation model
  - \( O(s, e) = P(\text{obtain evidence } e \text{ when in state } s) \)

- Agent does not know which state it is in
  - Makes no sense to talk about policy \( \pi(s) \)

- **Theorem** (Astrom, 1965): The optimal policy in a POMDP is a function \( \pi(b) \) where \( b \) is the belief state (probability distribution over states)

- Can convert a POMDP into an MDP in belief-state space

- \( T(b, a, b') \) is the probability that the new belief state is \( b' \) given that the current belief state is \( b \) and the agent does \( a \)
  - Essentially a filtering update step
Solutions automatically include information-gathering behavior

If there are $n$ states, $b$ is an $n$-dimensional real-valued vector

- Solving POMDPs is very (actually, PSPACE-) hard

The real world is a POMDP (with initially unknown $T$ and $O$)