Finite Markov Decision Processes

Markov Property
- "The future is independent of the past given the present"
- The state captures all relevant information of the history
- Previous states can be thrown away
- The state is a sufficient statistic of the future

Markov Process (or Markov Chain)
- Memoryless random process with the Markov property
- Markov Process is a tuple (S, P)
  - S is a (finite) set of states
  - P is a state transition probability matrix

State Transition Matrix
- The probability of transitioning from state **s** to any **s'**

Markov Reward Process (MRP)
- A Markov chain with values
- MRP is a tuple (S, P, R, γ)
  - R is a reward function
  - γ is a discount factor, $0 \le \gamma \le 1$

Markov Decision Process (MDP)
- A MRP with decisions
- An environment in which all states are Markov
- MDP is a tuple (S, A, P, R, γ)
  - A is a finite set of action

Notes
- Bandits are MDPs with one state
- In a finite MDP, the sets of states, actions, and rewards (S, A, and R) all have a finite number of elements

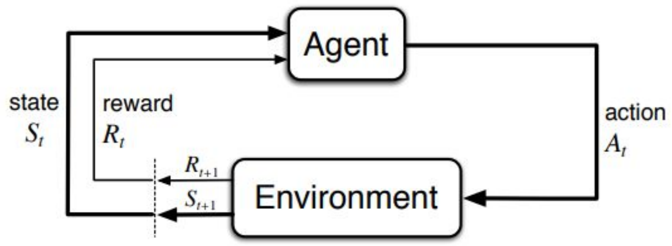
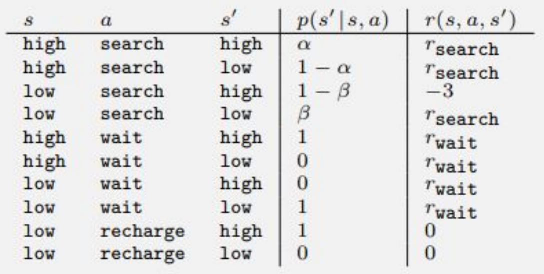
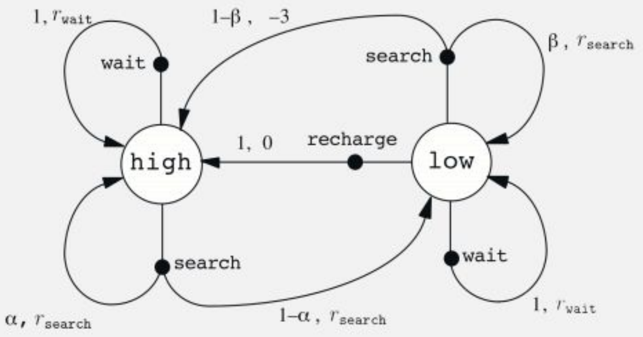


Figure 3.1: The agent–environment interaction in a Markov decision process.

Example Markov Reward Process



s	a	s'	$p(s' \mid s,a)$	$r(s,a,s')$
high	search	high	α	r_{search}
high	search	low	$1-\alpha$	r_{search}
low	search	high	$1-\beta$	-3
low	search	low	β	r_{search}
high	wait	high	1	r_{wait}
high	wait	low	0	r_{wait}
low	wait	high	0	r_{wait}
low	wait	low	1	r_{wait}
low	recharge	high	1	0
low	recharge	low	0	0



Rewards

- The agent always learns to maximize its reward. The reward signal is not the place to impart to the agent prior knowledge about how to achieve what we want it to do.
- If you reward subgoals, agent may learn to exploit them without solving problem
- The reward signal is your way of communicating to the robot what you want it to achieve, not how you want it achieved.

Episodes

- Episode (Episodic tasks)
  - A natural subsequence of agent-environment interaction
  - EX: A game of chess, or playing a single song
- Each episode ends in a state called the terminal state, followed by a reset to a standard starting state, or to a sample from a standard distribution of states
  - Episodes can end with different rewards (winning or losing)
- Each episode begins independently of how the previous one ended

Expected Return

The goal of an agent is to maximize total expected return

$$G_t \doteq R_{t+1} + R_{t+2} + R_{t+3} + \cdots + R_T,$$

G_t is a function of the reward sequence
T is the final time step

Discounting

- Agent tries to maximize the total expected discounted return

Why Discount?

- Most MDPs discount
- Value judgements on future actions are harder to make - discount makes these judgements less important
- Hard to maximize total expected reward for continuing tasks since the final time step is $T = \infty$
- Animal/Human behaviour shows preference for immediate reward

Discounted Return

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1},$$

$$\begin{aligned} G_t &\doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \cdots \\ &= R_{t+1} + \gamma\left(R_{t+2} + \gamma R_{t+3} + \gamma^2 R_{t+4} + \cdots\right) \\ &= R_{t+1} + \gamma G_{t+1} \end{aligned}$$

- Where γ is a parameter, $0 \le \gamma \le 1$, called the discount rate
  - Discount rate determines the present value of future rewards: a reward received k time steps in the future is worth only γ^k times what it would be worth if it were received immediately.
- If γ close to 0, the agent is 'myopic', only concerned with maximizing immediate rewards
- As γ approaches 1, the agent takes future rewards into account more strongly; the agent becomes farsighted
- Although discounted return is a sum of an infinite number of terms, it is finite if $\gamma < 1$

Policy Functions

- A mapping from states to probabilities of selecting each possible action
- **π(a|s)** gives the probability that action **a** will be chosen in state **s** following policy **π**
- MDP policies depend on the current state
- RL methods specify how the agent's policy is changed as a result of experience

Q Action-Value Functions

- **q_π(s,a)** → The expected value of taking action **a**, when in state **s**, and under the policy **π**

Value Functions - v_π(s)

- The value of a state **s** under the policy **π**
- The expected return (long-term value) of state **s** and following **π** thereafter

Monte Carlo Methods

- Estimation method for **q_π(s,a)**, averaging over many random samples of actual returns

Bellman Equation

$$= \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)\Big[r + \gamma v_\pi(s')\Big], \quad \text{for all } s \in \mathcal{S},$$

- The value function can be decomposed into two parts:
  - Immediate reward Rt+1
  - Discounted value of successor state **$\gamma v_\pi(S_{t+1})$**
- Bellman Equation Explained
  - For each possible action **a** in state **s**, multiply the following by **a**'s probability of being chosen
    - Probability of a state-reward pair occuring [**p(s',r|s,a)**]
      - Only matters if an action has a random result, otherwise result is 1
    - Multiplied by the state's reward, plus the discounted Bellman's Equation for the following state [**(r + γv_π(s'))**]

Solving the Bellman Equation

$$v = R + \gamma P v$$
$$(1 - \gamma P)v = R$$
$$v = (1 - \gamma P)^{-1} R$$

- Computational complexity is O(n^3) for n states
- Direct solution only possible for small MRPs
- There are many iterative methods for large MRPs, e.g.
  - Dynamic programming
  - Monte-Carlo evaluation
  - Temporal-Difference learning

Optimal State-Value Function

- The maximum value function over all policies
- **$v_*(s) = \max_\pi v_\pi(s)$**

Optimal Action-Value Function

- The maximum action-value function over all policies
- **$q_*(s, a) = \max_\pi q_\pi(s, a)$**

Optimal Policy Function

- A policy π is defined to be better than or equal to policy π' if its expected return is greater than or equal to that of π' for all states
- There is always at least one policy that is better than or equal to all other policies
- π_* indicates an optimal policy
- If we know $q_*(s, a)$, then we have the optimal policy

Greedy Search

- A greedy policy (choose the **a** with greatest value), AKA one step search, is the optimal policy if it is following v_*, since v_* already takes into account the reward consequences of all future behaviour
- With q_*, one-step search isn't even needed - Agent picks action that maximizes $q_*(s,a)$

Solving the Bellman Optimality Equation

- Bellman Optimality Equation is non-linear
- No closed form solution (in general)
- Many iterative solution methods
  - Value Iteration
  - Policy Iteration
  - Q-learning
  - Sarsa
- Solving Bellman's Optimality Equation requires that
  - The dynamics of the environment are known
  - Enough computational resources are available to compute the solution
  - The environment is Markov

Approximate Solutions

- For many problems, solving Bellman's Optimality Equation would require too much computing power.
  - Instead, approximate solutions are often 'good-enough'
    - Decision tree is expanded to a certain depth, than a value heuristic gives an estimate of that state's value
- For many problems, there are too many states, memory limits would be exceeded
  - There are many possible game states in Go then there are atoms in the universe
- Many 'approximate' agents only have to solve value functions for 'decent+' actions
  - There's no point in exhaustively computing all the ways you can lose