07-280 Notes
Pre-reading: Markov Decision Processes

Carnegie Mellon University
School of Computer Science

Contents

1 Motivation: The Racing Car **1**

2 From Expectimax to MDPs **2**

2.1 Values and Q-Values 3

3 Markov Decision Processes **3**

3.1 Example: Gridworld 4

4 Policies **5**

5 Value Iteration **5**

5.1 Simple Gridworld Example 5

6 Optimal Values and Q-Values **7**

1 Motivation: The Racing Car

A robot car drives along a track and must choose at every step whether to go *Slow* or *Fast*. It can be in one of three states:

Cool **Warm** **Overheated**

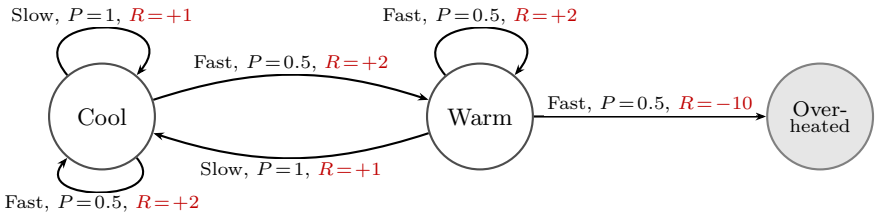
Illustrations: Ketrina Yim / UC Berkeley AI

The car earns **reward** for driving: +1 for a Slow move, +2 for a Fast move, but −10 if it overheats.

- From **Cool**: Slow → Cool (P=1, $R = +1$). Fast → Cool or Warm (P=0.5 each, $R = +2$).

- From **Warm**: Slow → Cool (P=1, $R = +1$). Fast → Warm or Overheated (P=0.5 each, $R = +2$ or $R = -10$).
- **Overheated**: game over.

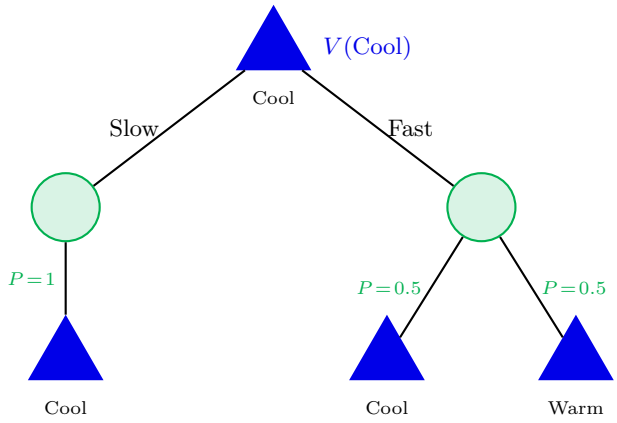
The full transition structure is shown below as an MDP **state diagram**: circles are states, arrows are transitions labeled with the action taken, transition probability, and reward collected.



The goal is to find a *strategy* — one action per state — that maximizes the total expected reward over many driving steps. What should the car do from Cool? From Warm? This is the kind of question **Markov Decision Processes** are built to answer.

2 From Expectimax to MDPs

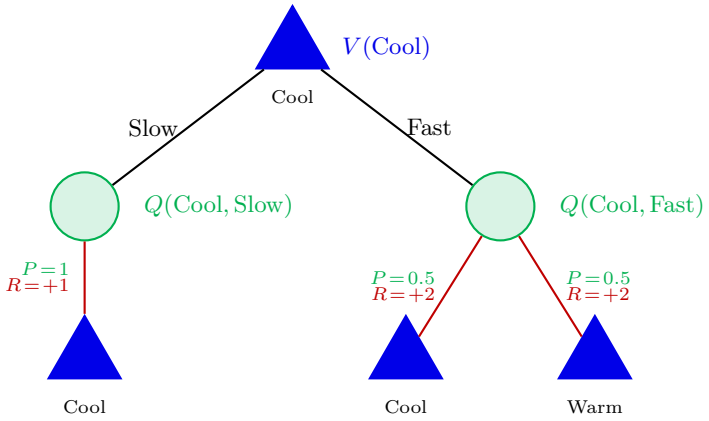
The expectimax framework already handles stochastic actions. A **max node** (blue triangle) chooses the best action; a **expectation node** (green circle) averages over outcomes. For the racing car starting from Cool, looking one step ahead:



But standard expectimax has three problems in this setting:

1. **Infinite tree.** The car never stops; the tree has no leaves.
2. **No terminal utilities.** Leaf nodes in game trees carry fixed utility values. Here there are none — the car just keeps driving.
3. **Rewards accumulate at every step.** The car earns +2 each time it drives Fast successfully, not a single utility at the end.

The fix is to fold the reward into each transition and replace terminal utilities with recursive calls to the same V function. The red edges below signal that a reward is collected at each transition:



The next-level triangles are not terminals — they are the same car in a new state, and the recursion continues. The tree goes on forever; we solve it algebraically instead of by explicit search.

2.1 Values and Q-Values

The two quantities in the tree each have a name:

- The **value** $V(s)$ is the expected total reward from state s when acting optimally from now on. It is the quantity at each **max node**:

$$V(s) = \max_a Q(s, a)$$

- The **Q-value** $Q(s, a)$ is the expected total reward when taking action a from s and then acting optimally. It is the quantity at each **expectation node**:

$$Q(s, a) = \sum_{s'} P(s' \mid s, a) \left[R(s, a, s') + V(s') \right]$$

These two equations are mutually recursive. Each Q-value depends on future V-values, and each V-value is the max over Q-values. Together they define how to compute the best possible behavior without ever explicitly constructing an infinite tree.

3 Markov Decision Processes

The structure above defines a **Markov Decision Process (MDP)**: a formal model for sequential decision-making under uncertainty. An MDP is defined by five components:

- **State space** S: the set of all possible states.
- **Action space** A: the set of actions available to the agent.
- **Transition function** $P(s' \mid s, a)$: the probability of reaching s' when taking action a from s. Also written $T(s, a, s')$.
- **Reward function** $R(s, a, s')$: the reward received on the transition from s to s' via a. Often simplified to $R(s, a)$ or $R(s)$ when reward does not depend on the next state.
- **Start state** $s_0 \in S$.

The Markov property. "Markov" means the transition depends only on the *current* state and action, not on the history of how the agent got there. $P(s' \mid s, a)$ is everything needed — no memory required.

3.1 Example: Gridworld

The canonical MDP example is the **gridworld**: an agent moves on a 4×3 grid, trying to reach the +1 exit while avoiding the −1 pit.

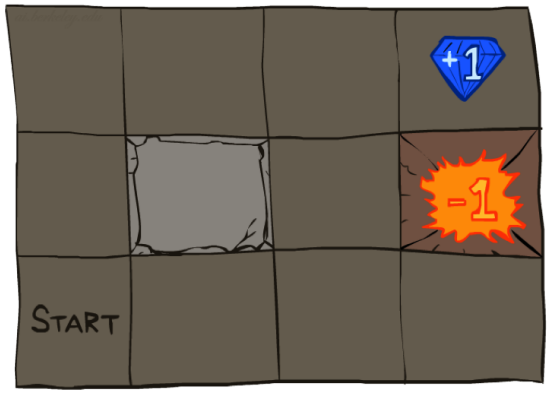
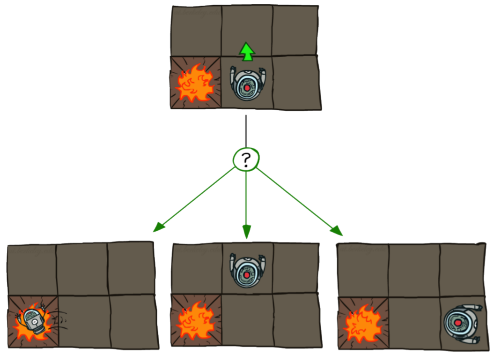


Illustration: Ketrina Yim / UC Berkeley AI

Actions are N, S, E, W, but the environment is stochastic with **noise = 0.2**: when the agent tries to move North, it succeeds with probability 0.8 but slips East or West with probability 0.1 each. The figure below shows the three possible outcomes of trying to move North from one cell:



So the transition function $P(s' \mid s, \text{North})$ assigns probability 0.8 to the cell directly North, 0.1 to the cell to the East, and 0.1 to the cell to the West (with wall-bumps keeping the agent in place if a wall blocks the slip).

The +1 and −1 squares are *pre-terminal* states: their only available action is *Exit*, which leads deterministically to an absorbing Terminal state and collects the displayed reward. The living reward — a small penalty applied at every non-exit transition — defaults to −0.1 in this example; it is a configurable setting of the gridworld environment, like noise.

4 Policies

In deterministic search, a solution is a fixed sequence of actions from start to goal. In a stochastic environment this breaks down: an unexpected transition leaves the agent with no instruction for what to do next.

A **policy** π solves this. A policy is a complete function $\pi : S \to A$ that specifies an action for *every* state the agent might encounter. Rather than following a script, the agent looks up its current state and acts.

A policy is naturally visualized as a map of arrows on the gridworld — one arrow per cell showing which direction the policy prescribes from that state:

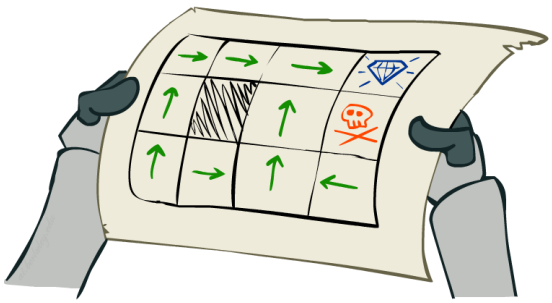


Illustration: Ketrina Yim / UC Berkeley AI

An **optimal policy** π^* is one that maximizes the agent's expected total reward from every starting state.

5 Value Iteration

To find the optimal policy we need the optimal values $V^*(s)$ for all states. The problem: the V/Q equations are circular. $V(s)$ depends on $V(s')$, which depends on $V(s'')$, and in a graph with cycles no ordering resolves all dependencies at once.

Value iteration treats this as a fixed-point computation (simplified here without discounting — we will add the discount factor γ in lecture):

1. Initialize $V_0(s) = 0$ for all states s.
2. Repeat for each non-terminal state:

$$V_{k+1}(s) = \max_a \sum_{s'} P(s' \mid s, a) \left[R(s, a, s') + V_k(s') \right]$$

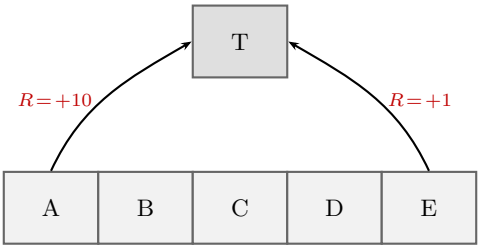
3. Stop when values converge.

Each iteration adds one more ply of lookahead — the same as deepening an expectimax tree by one level. After k iterations, $V_k(s)$ holds the best expected reward reachable within k steps.

5.1 Simple Gridworld Example

States A–E are arranged in a row. From the endpoints A and E, the only action is *Exit* (leading to terminal state T). Interior states B, C, D can go East or West (deterministic, reward 0). Exit from

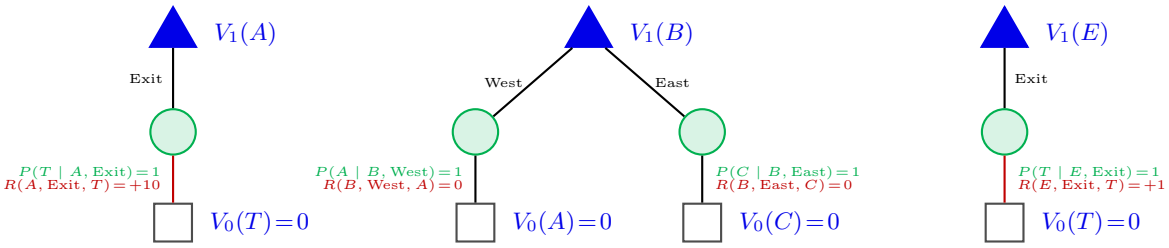
A gives reward +10; exit from E gives +1.



$V(T) = 0$: the game ends at T and no further reward is collected. B, C, D can go East or West (deterministic, $R = 0$); A and E can only Exit.

Iteration 0: $V_0(s) = 0$ for all states.

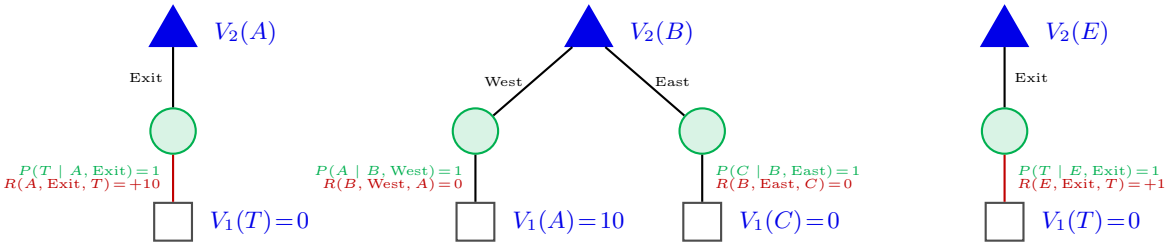
Iteration 1. Small trees show the one-step lookahead for states A, B, and E:



All five updates in order:

$$
\begin{aligned}
V_1(A) &= 10 + V_0(T) = 10 \\
V_1(B) &= \max\Big(\underbrace{0 + V_0(A)}_{\text{West: } 0},\ \underbrace{0 + V_0(C)}_{\text{East: } 0}\Big) = 0 \\
V_1(C) &= \max\Big(\underbrace{0 + V_0(B)}_{\text{West: } 0},\ \underbrace{0 + V_0(D)}_{\text{East: } 0}\Big) = 0 \\
V_1(D) &= \max\Big(\underbrace{0 + V_0(C)}_{\text{West: } 0},\ \underbrace{0 + V_0(E)}_{\text{East: } 0}\Big) = 0 \\
V_1(E) &= 1 + V_0(T) = 1
\end{aligned}
$$

Iteration 2. Now A's value of 10 is visible one step away from B:



All five updates in order:

$$V_2(A) = 10 + V_1(T) = 10$$
$$V_2(B) = \max\big(\underbrace{0 + V_1(A)}_{\text{West: } 10},\ \underbrace{0 + V_1(C)}_{\text{East: } 0}\big) = 10$$
$$V_2(C) = \max\big(\underbrace{0 + V_1(B)}_{\text{West: } 0},\ \underbrace{0 + V_1(D)}_{\text{East: } 0}\big) = 0$$
$$V_2(D) = \max\big(\underbrace{0 + V_1(C)}_{\text{West: } 0},\ \underbrace{0 + V_1(E)}_{\text{East: } 1}\big) = 1$$
$$V_2(E) = 1 + V_1(T) = 1$$

After four iterations the values converge:

k	$V_k(A)$	$V_k(B)$	$V_k(C)$	$V_k(D)$	$V_k(E)$
0	0	0	0	0	0
1	10	0	0	0	1
2	10	10	0	1	1
3	10	10	10	1	1
4	10	10	10	10	1

Each iteration, A's reward of +10 propagates one step further. By iteration 4, every state has converged to the best expected reward achievable.

From values to a policy. Extracting the optimal policy requires recomputing the Q-values from the converged V values and selecting the argmax action — we will cover this step in lecture.

What comes next. Lecture will introduce *discounting* for infinite-horizon agents, and *policy iteration* as an alternative algorithm.

6 Optimal Values and Q-Values

Value iteration converges to the **optimal values** $V^*(s)$: the best expected total reward from state s acting optimally forever. The **optimal Q-values** $Q^*(s,a)$ are the corresponding expectation-node values under optimal future behavior:

$$Q^*(s,a) = \sum_{s'} P(s' \mid s,a)\ \big[R(s,a,s') + V^*(s')\big] \tag{1}$$

$$V^*(s) = \max_a Q^*(s,a) \tag{2}$$

Given Q^*, the optimal policy picks the best action at each state:

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

Policy extraction requires Q^*, not just V^* — V^* alone does not encode which action produced that value.

v1.0