Sequential Decisions: Solving Markov Decision Processes

Nathan Ratliff

Feb 6, 2015

Abstract

Sequential decisions are the building blocks of intelligence. Intelligent agents must reason about future consequences of every action and optimize their behavior accordingly to discover high performing policies. This document introduces Markov Decision Processes (MDPs), a tractable mathematical formulation of the sequential decision problem, and three algorithms for solving them: Value Iteration, Policy Iteration, and Linear Programming. It additionally presents a form of Soft Value Iteration that enables tractable manipulation of Gibbs distributions over trajectories in these settings. These tools are some of the most basic constructs constituting the heavily researched field of Decision Theory. Throughout, we see numerous connections to core areas of mathematics such as Linear Algebra and Optimization, from a linear system whose solution reveals the cumulative accrued reward of policies acting over time, to an instance of the Power Method derived as an algorithm for calculating stochastic policy distributional normalizers. This exposition is necessarily incomplete, but these ideas are the catalyst for many successful areas of Robotics and Machine Learning dedicated to the design and implementation of intelligent machines.

1 Sequential Decisions

"Think about the consequences of your actions!" Your mom is right. Intelligence revolves around reaching beyond simple reactive action choices to the complex but powerful tools designed to reason about future consequences of actions and sequential decisions. We can't answer the question, "What action should I take now?" without considering what that action does and what might come next. The efficacy of actions taken ten seconds from now depend critically on actions performed now. And in choosing actions now, we must account for what possibilities that opens up for the future and how we plan to capitalize on those opportunities. Intelligent behavior must reason about how entire sequences of actions change the state of the world and whether that cumulative outcome is good or bad.

This document formalizes the elements of sequential decision making by introducing the Markov Decision Process (MDP) (Section 2) and a collection of classical methods for solving them (Section 3). We then show connections to a more probabilistic model of behavior (Section 4) and discuss some of the limitations of the tools mentioned here (Section 5). Many more advanced fields in Robotics and Machine Learning, such as Optimal Control and Reinforcement Learning, build from the ideas presented here.

2 Defining Markov Decision Processes

To discuss sequential decisions we must first define mathematically what our world is, how actions change the world, and what it means to perform well or poorly. Abstractly, we say the world is just a collection of *states* defined as a mathematical set $\mathcal{S}$. For simplicity in this document we take this set to be finite and discrete so that $|\mathcal{S}| \in \mathbb{N}$, although generalizations to continuous spaces are common (especially in Optimal Control Theory). From every state $s \in \mathcal{S}$ the agent can execute any of a discrete collection of actions $\mathcal{A}(s)$. Generally, the set of actions may be different state-to-state, but again for simplicity here, we assume the action set is a single fixed set $\mathcal{A}$ so that $\mathcal{A}(s) = \mathcal{A}$ for all $s \in \mathcal{S}$. This assumption models a fixed set of capabilities for the agent. For instance, a robot might be able to turn, move up, down, left, right, or diagonally, and hammer a nail into whatever's in front of it. If the state is it's two-dimensional (x, y) location in the world and its orientation θ, then no matter what state (x, y, θ) the robot is in, it can still always perform any those actions. Again, here we're considering the action set to be discrete and finite, but generalizations to continuous action spaces are also common.

Now, given the set of states $\mathcal{S}$ and actions $\mathcal{A}$ that together describe the world and the capabilities of the robot, we need to somehow express mathematically how taking a particular action a in a state s changes that state. The most straightforward model of this state transition is simply a probability distribution $p(s'|s, a)$ that defines the probability of transitioning to any other state s' given that the agent took action a from state s.

When this transition probability distribution is a Kronecker Delta

$$p(s'|s, a) = \delta_{s_{\text{next}}}(s') = \begin{cases} 1 & \text{for } s' = s_{\text{next}} \\ 0 & \text{otherwise} \end{cases}$$

we say the transitions are *deterministic*, and the resulting model can be represented as a directed graph (see Figure 1).

Finally, we need a way to score the success, failure, or general "goodness" or "badness" of an action taken from a particular state. One way to represent this intuitive notion is by a function $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ that assigns a real number $r(s, a)$ to every state-action pair. This function is known as the **reward function** because we interpret the real valued output as the *reward* received upon taking action a from state s. Other reward representations exist, such as reward functions of the form $r(s)$ assigning reward to just states and reward functions

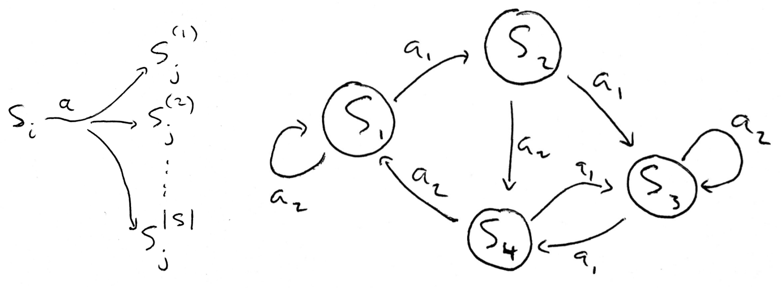

Figure 1: **Left:** Depiction of probabilistic transitions to multiple possible next states under a single action. **Right:** Graph representation of deterministic transitions.

of the form $r(s, a, s')$ depending additionally on the state the agent arrives in upon taking the action, but it can be shown that each of these representations are theoretically equivalent. The particular choice of representation for a given problem is a matter of convenience.

To summarize, a **Markov Decision Processes (MDP)** consists of the following five elements:

1. A discrete finite set of **states** $\mathcal{S}$.
2. A discrete finite set of **actions** $\mathcal{A}$.
3. A **transition function** defined as $p(s'|s, a)$.
4. A **reward function** $r(s, a)$.

Any given instance of the model additionally specifies an **initial distribution** $p(s_0)$ defining a distribution over states where the agent might start.

2.1 What do these words mean?

Much of the formalism discussed in Section 2 develops intuitively, but then we tag it with the weird name Markov Decision Process which seems awkward and foreign. The qualifying adjectives constituting the name, though, suggest that this model is an interesting mathematical object with specific structure. Let's take a moment to examine the name to give it context.

First, the term **Process** refers to a probabilistic system that evolves over time. An example of a probabilistic process is the act of walking forward with your eyes closed. Suppose you start at one side of a room, close your eyes, and walk forward for ten steps. If you set up a measurement device, such as an overhead camera, to measure your progress after each step, you get a sequence of ten two-dimensional points defining how your location evolves over time. If you then repeat the experiment a hundred times, you get a collection of sampled state trajectories through the system. These samples are samples taken from the probabilistic process underlying the system.

Notice that in this simple example we can't fully represent the evolution of the system just by the two-dimensional location measurement mentioned above. We're all big lumbering creatures with significant momentum, so really the probability distribution over where you're going to be in the future depends not only on your two-dimensional location now, but also on where you've been in the past. In this sense, the two-dimensional locational information we've measured doesn't record everything we need to know about the system to predict the future. Such a system is called *non-Markovian.*

These observations bring us to the work **Markov**. As described above, for any fixed sequence of actions $a_0, \ldots, a_T$ (or fixed assignment of states to actions $\pi : \mathcal{S} \rightarrow \mathcal{A}$ (see the discussion on policies below in Section 2.2)), the evolution of the system is defined by a fixed set of probabilities $p(s_{t+1}|s_t, a_t)$. The probability of seeing a sequence of states, then is

$$p(s_0, \ldots, s_{T+1}) = p(s_0)p(s_1|s_0)p(s_2|s_1)\cdots p(s_{T+1}|s_T).$$

where here we denote the transitions as simply a conditional probability distribution of the previous state since the action sequence (or policy) is fixed. Thus, we can ask mathematically, what is probability distribution over future states $s_{t+1}, \ldots, s_{T+1}$ given all the previous states $s_0, \ldots, s_t$:

$$\begin{aligned}
&p(s_{t+1}, \ldots, s_{T+1}|s_0, \ldots, s_t) \\
&\quad = \frac{p(s_0, \ldots, s_{T+1})}{\sum_{s_{t+1},\ldots,s_{T+1}} p(s_0, \ldots, s_{T+1})} \\
&\quad = \frac{p(s_0)p(s_1|s_0)p(s_2|s_1)\cdots p(s_{T+1}|s_T)}{p(s_0)p(s_1|s_0)\cdots p(s_t|s_{t-1}) \sum_{s_{t+1}} p(s_{t+1}|s_t) \cdots \sum_{s_{T+1}} p(s_{T+1}|s_T).} \\
&\quad = p(s_{t+1}|s_t)p(s_{t+2}|s_{t+1})\cdots p(s_{T+1}|s_T),
\end{aligned}$$

where the last equality holds because all of the sums in the denominator of the second line, from right to left, evaluate to 1, and the rest of the terms in the denominator directly cancel the corresponding terms in the numerator. Notice that this final expression is just the conditional marginal $p(s_{t+1}, \ldots, s_{T+1}|s_t)$. In other words, given s_t, the distribution over all future states s_{t+1} through s_{T+1}, is independent of everything that happened in the past s_0 through s_{t-1} (and the associated actions). This model that we've defined, the MDP, assumes that the state is sufficiently descriptive to summarize *everything* that's happened in the past. This is the defining property of a *Markov* Process.

Returning to the above intuitive example of walking across the room, we see that the (x, y) location does not encode everything about the past needed to predict the future. This process is, therefore, non-Markovian as mentioned. Making a simplifying assumption that human beings are like bowling balls moving under the influence of some force field (a huge assumption, but not altogether terrible in practice) the new piece of information we'd need to add to the state to fully encode the past is simply the velocity. If we know both the (x, y) location and two-dimensional velocity (four numbers in total), then we're much better equipped to predict the future from just this larger state. That velocity encodes what we need to know about the momentum of the system (the tendency to keep moving in the same direction) and, while it's not a perfect model of the system, we can now much better predict the behavior of the system using this Markov assumption.

Finally, the word **Decision** denotes that the actual Markov Process is governed by the choice of actions. Choosing actions either as a function of state or a sequence fixed in advanced defines the transition probabilities and how the process evolves over time. And in turn, the process evolution defines the accumulated reward. MDPs are Markov Processes that mathematically model the outcomes of action choice and enable the definition and discovery of *optimal* strategies for acting in the environment.

2.2 Policies and Cumulative Reward

For an agent to act in an MDP, it needs to choose a **policy**. A policy is a formal model of action choice defined as a mapping from states to actions[1] $\pi : \mathcal{S} \to \mathcal{A}$. The policy dictates the action $a = \pi(s)$ an agent takes from every state s. This function is an abstract mathematical model of, for instance, a computer program calculating what a robot should do in a given situation. Here we denote the set of all policies by Π.

Any policy π defines a world dynamic by $p_\pi(s'|s) = p(s'|s, \pi(s))$. Starting from initial distribution $p(s_0)$, the distribution over trajectories $\xi = (s_0, s_1, \ldots, s_T)$ is given by

$$p_\pi(\xi) = p(s_0) \prod_{t=0}^{T-1} p_\pi(s_{t+1}|s_t).$$

A good model for the success of a given policy π is its cumulative discounted reward. This model adds up all rewards encountered along the trajectory weighing each by increasing powers of a discount factor of $\gamma \in [0, 1]$ to bias the score toward rewards encountered earlier. For a given trajectory through the state space $\xi = (s_0, s_1, \ldots, s_T)$, this cumulative reward takes the form

$$R_\pi(\xi) = \sum_{t=0}^{T-1} \gamma^t r(s_t, \phi(s_t)).$$

[1]Technically, such a mapping is a *stationary* policy in the sense that it's fixed and applied in the same way at every moment in time. A given state is always mapped to the same action. Non-stationary policies, which we don't consider here, may change their behavior over time. They're useful for modeling how behavior changes, for instance, based on time limits or game stages. As the end of a football game nears, the players might employ more risky behavior in hopes of achieving a last minute favorable outcome before the game ends. Such a time-dependent strategy is an example of a non-stationary policy.

We're now equipped to evaluate the *expected* cumulative reward of a policy:

$$\begin{aligned}
&E_{p_\pi(\xi)}[R_\pi(\xi)] && (1)\\
&\quad= \sum_{s_0,\ldots,s_T}\left(p(s_0)\prod_{t=0}^{T-1}p_\pi(s_{t+1}|s_t)\sum_{t=0}^{T-1}\gamma^t r(s_t,\pi(s_t))\right)\\
&\quad= \sum_{s_0}p(s_0)\left(r_0+\gamma\sum_{s_1}p_\pi(s_1|s_0)\left(r_1+\gamma\sum_{s_2}p_\pi(s_2|s_1)\left(r_2+\gamma\sum_{s_3}\cdots\right)\right)\right),
\end{aligned}$$

where we denote $r_i = r(s_i, \pi(s_i))$ for notational brevity. See Appendix A for a derivation of this result. Letting $T \to \infty$ (transitioning us to an *infinite-horizon* setting) and defining $v_\pi(s)$ to be the cumulative reward of starting specifically in state s and acting in accordance with policy π, we see that the recursive structure of this cumulative future reward takes the form

$$v_\pi(s) = r(s,\pi(s)) + \gamma\sum_{s'}p_\pi(s'|s)v_\pi(s'). \qquad (2)$$

This function $v_\pi(s)$ is known as the **Value Function** for policy π, and it denotes the policy's **infinite-horizon discounted cumulative reward**. Given the Value Function, our expression for the expected reward simplifies to

$$E_{p_\pi(\xi)}[R_\pi(\xi)] = E_{p(x_0)}[v_\pi(x_0)] = \sum_{x_0}p(x_0)v_\pi(x_0). \qquad (3)$$

This expected cumulative future discounted reward measures the quality of any given policy. Given such a quality measure, we can now rank policies from bad to better to best. Section 3 formalizes the notion of an "optimal" policy and presents three algorithms for finding such a policy.

2.3 Rewards vs Costs

This document models the benefits of taking various actions as "rewards", which is a common formalism in the Reinforcement Learning literature. But what about the *consequences* of doing things? Do we *reward* the agent for spilling the soda all over the floor when we ask for it? In the reward setting, we consider detrimental actions to accrue *negative* reward, reward that detracts from the total cumulative reward. We could have equally well modeled everything as "costs", all of which are bad, and instead desire to *minimize* the cumulative expected cost-to-go of a policy. Mathematically, costs are just negative rewards, but as a modeling construct sometime they can be more intuitive. Many planning systems are built around costs because the final goal (reaching a particular state) is usually clear and each action along the way costs time, energy, or likelihood of failure, and the robot desire in life is to simply achieve it's goal while avoiding as much cost from the harsh world as possible. It's especially common to view action consequences as costs in Optimal Control (Stengel, 1994), where bad actions force our attention as consequences include crashing an aircraft.

3 Solving Markov Decision Processes

Section 2 introduced the Markov Decision Process that models the environment and how actions affect state, and Section 2.2 calculated the cumulative reward of a policy and gave us a way to measure and compare the value of policies to one another. Here, we take the final step and define *optimal* policies and, in particular, derive an equation, called the Bellman Equation, that concisely expresses the optimality of a policy. We then use Bellman's Equation to derive three classical algorithms for discovering optimal policies in MDPs to *solve* these Markov Decision Problems[2]

3.1 The Bellman Equation and Value Iteration

The expected cumulative reward $E_{p(x_0)}[v_\pi(x_0)]$ measures the quality of any policy π and allows us to rank the set of all policies based on their performance. We can, therefore, define an optimal policy π^* as any policy for which

$$E_{p(x_0)}[v_{\pi^*}(x_0)] \geq E_{p(x_0)}[v_\pi(x_0)] \quad \forall \pi \in \Pi,\ p(x_0). \tag{4}$$

Notice that dominance of the expected cumulative reward for *any* initial distribution $p(x_0)$ implies state-by-state Value Function dominance of the form

$$v_{\pi^*}(s) \geq v_\pi(s) \quad \forall \pi \in \Pi, s \in \mathcal{S}. \tag{5}$$

For such an optimal policy, the recursive representation of Value Function given by Equation 2 in combination with the optimality criterion implies

$$\begin{aligned} v_{\pi^*}(s) &= r(s, \pi^*(s)) + \gamma \sum_{s'} p(s'|s, \pi^*(s))\, v_{\pi^*}(s') & (6)\\ &\geq r(s, a) + \gamma \sum_{s'} p(s'|s, a)\, v_{\pi^*}(s') \quad \forall a \in \mathcal{A}. & (7) \end{aligned}$$

In this setting, it can be shown that the optimal value function is unique even though there might not be a single unique optimal policy. For that reason, we typically denote the optimal value function as simply $v^*(s)$ rather than notationally referencing a particular policy. Since this inequality holds for all actions, it must also hold for the max over all actions:

$$v^*(s) = \max_{a \in \mathcal{A}} \left\{ r(s, a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s, a)\, v^*(s') \right\}. \tag{8}$$

This equation is called the **Bellman Equation**. It characterizes optimal policies in terms of an *optimal* Value Function. Notice that the equation holds

[2]There's a conflation in the literature between the terms Markov Decision Problem (Littman et al., 1995) and Markov Decision Process (Puterman, 1994). The latter refers to the probabilistic process modeling the evolution of a system under the action of a policy, while the former refers to the problem of solving those Decision Processes. This document uses the acronym MDP exclusively to abbreviate Markov Decision Process.

with equality since the optimal value function is created by a specific choice of action for every state, so one particular inequality in Equation 7 must hold with equality. It also says that an optimal policy can be characterized based on this optimal Value Function as

$$\pi^*(s) = \operatorname*{argmax}_{a\in\mathcal{A}} \left\{ r(s,a) + \gamma \sum_{s'\in\mathcal{S}} p(s'|s,a)\, v_{\pi^*}(s') \right\}. \tag{9}$$

Note that the argmax operator may not be unique when multiple actions produce equivalent infinite horizon discounted cumulative rewards.

This Bellman Equation forms the basis of numerous algorithms for solving MDPs. A simple example is **Value Iteration**. Value Iteration is simply a fixed point iteration algorithm on the Bellman Equation with updates

$$v_{i+1}(s) = \max_{a\in\mathcal{A}} \left\{ r(s,a) + \gamma \sum_{s'\in\mathcal{S}} p(s'|s,a)\, v_i(s') \right\}. \tag{10}$$

The Value Iteration algorithm starts with an arbitrary first value function approximation $v_1(s)$, usually a naïve approximation such as the zero function, and iterates Equation 10, sometimes known as a **Bellman Backup**, until convergence.

The theoretical analysis of this algorithm is based on the contraction of the value function toward the optimal function. We'll sketch it briefly here. Define the max norm as $\|v\| = \max_{s\in\mathcal{S}} |v(s)|$ and denote the application of Equation 10 to an arbitrary value function v as $F(v)$. It can be shown that for any two Value Function approximations u and v, applying Equation 10 brings them closer together in the sense $\|F(u) - F(v)\| \leq \gamma\|u - v\|$. Given that result, if we choose $u = v^*$ to be the optimal Value Function, since $F(v^*) = v^*$ (the Bellman Equation), we have $\|v^* - F(v)\| \leq \gamma\|v^* - v\|$. In other words, the Value Iteration algorithm converges linearly to the unique optimal Value Function.

3.2 Vector and Matrix Notation

It's common to denote the Bellman Equation and other related expressions as functions as we do above. But these functions are really functions over finite discrete sets, so they can all naturally be represented more compactly as vectors or matrices. This section introduces equivalent vector and matrix notation and rewrites some of the above expressions to emphasize linear relationships leveraged in the following two sections.

Since the set of states is countable, we can choose an ordering to the states $s^{(1)}, s^{(2)}, \ldots, s^{(|\mathcal{S}|)}$ and represent any function over states as a vector. For instance, we can write the Value Function $v_\pi(s)$ for any policy π and the optimal Value Function $v^*(s)$ as

$$v_\pi(s) \to \boldsymbol{v}_\pi \in \mathbb{R}^{|\mathcal{S}|} \quad \text{and} \quad v^*(s) \to \boldsymbol{v}^* \in \mathbb{R}^{|\mathcal{S}|}. \tag{11}$$

Additionally, for either any single choice of action $a \in \mathcal{A}$ to apply at every state, or a particular policy $\pi \in \Pi$ mapping from states to actions, the Reward Function becomes a fixed function of state and hence a vector of the form

$$r(s,a) \to \boldsymbol{r}_a \in \mathbb{R}^{|\mathcal{S}|} \quad \text{and} \quad r(s,\pi(s)) \to \boldsymbol{r}_\pi \in \mathbb{R}^{|\mathcal{S}|}. \tag{12}$$

Similarly, the under these conditions the transition dynamics of the environment, $p_a(s'|s) = p(s'|s,a)$ and $p_\pi(s'|s) = p(s'|s,\pi(s))$, give a fixed distribution over next states for every state. These distributions can be represented as rows of a matrix of the form

$$p_a(s'|s) \to \boldsymbol{T}_a = \begin{bmatrix} p_a(s^{(1)}|s^{(1)}) & p_a(s^{(2)}|s^{(1)}) & \dots & p_a(s^{(|\mathcal{S}|)}|s^{(1)}) \\ p_a(s^{(1)}|s^{(2)}) & p_a(s^{(2)}|s^{(2)}) & \dots & p_a(s^{(|\mathcal{S}|)}|s^{(2)}) \\ \vdots & \vdots & \ddots & \vdots \\ p_a(s^{(1)}|s^{(\mathcal{S})}) & p_a(s^{(2)}|s^{(\mathcal{S})}) & \dots & p_a(s^{(|\mathcal{S}|)}|s^{(\mathcal{S})}) \end{bmatrix} \in \mathbb{R}^{|\mathcal{S}|^2}. \tag{13}$$

and

$$p_\pi(s'|s) \to \boldsymbol{T}_\pi = \begin{bmatrix} p_\pi(s^{(1)}|s^{(1)}) & p_\pi(s^{(2)}|s^{(1)}) & \dots & p_\pi(s^{(|\mathcal{S}|)}|s^{(1)}) \\ p_\pi(s^{(1)}|s^{(2)}) & p_\pi(s^{(2)}|s^{(2)}) & \dots & p_\pi(s^{(|\mathcal{S}|)}|s^{(2)}) \\ \vdots & \vdots & \ddots & \vdots \\ p_\pi(s^{(1)}|s^{(\mathcal{S})}) & p_\pi(s^{(2)}|s^{(\mathcal{S})}) & \dots & p_\pi(s^{(|\mathcal{S}|)}|s^{(\mathcal{S})}) \end{bmatrix} \in \mathbb{R}^{|\mathcal{S}|^2}. \tag{14}$$

Notice that the rows of these matrices range over current states and the columns range over next states. This orientation means that multiplication of the form $\boldsymbol{T}_\pi \boldsymbol{v}_\pi$ treats the Value Function vector $\boldsymbol{v}_\pi$ as ranging over *next* states and, therefore, calculates the expected future discounted cumulative rewards of taking actions according to policy π. Using this notation, Equation 2, which depicts the recursive structure of the Value Function, takes the form

$$\boldsymbol{v}_\pi = \boldsymbol{r}_\pi + \gamma \boldsymbol{T}_\pi \boldsymbol{v}_\pi. \tag{15}$$

Similarly, we can rewrite the Bellman Equation given in Equation 8 as

$$\boldsymbol{v}^* = \max_{a \in \mathcal{A}} \left\{ \boldsymbol{r}_a + \gamma \boldsymbol{T}_a \boldsymbol{v}^* \right\}, \tag{16}$$

where this max operator is applied individually and independently to each element of the vector.

3.3 Policy Evaluation and Policy Iteration

Equation 15 shows an interesting linear relationship between the MDP's parameters and a policy's Value Function. Rearranging the equation gives

$$\begin{aligned} \left(\boldsymbol{I} - \gamma \boldsymbol{T}_\pi\right) \boldsymbol{v}_\pi &= \boldsymbol{r}_\pi \\ \Rightarrow \boldsymbol{v}_\pi &= \left(\boldsymbol{I} - \gamma \boldsymbol{T}_\pi\right)^{-1} \boldsymbol{r}_\pi. \end{aligned} \tag{17}$$

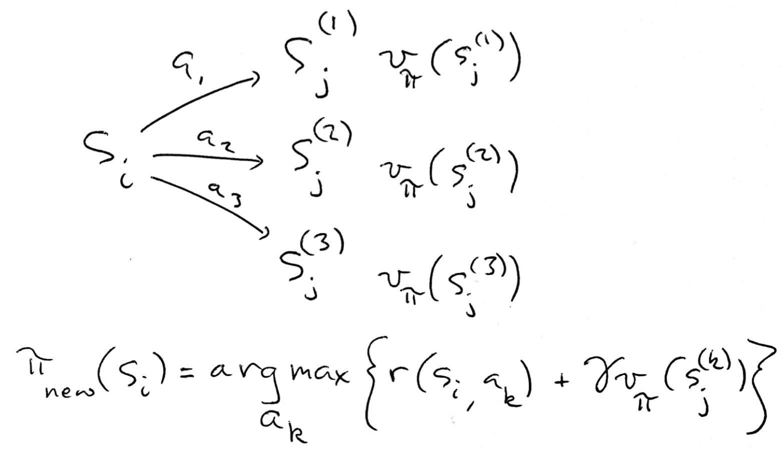
s_i
a_1
a_2
a_3
$s_j^{(1)}$ $v_\pi(s_j^{(1)})$
$s_j^{(2)}$ $v_\pi(s_j^{(2)})$
$s_j^{(3)}$ $v_\pi(s_j^{(3)})$
$\pi_{new}(s_i) = \operatorname{argmax}_{a_k} \{ r(s_i, a_k) + \gamma v_\pi(s_j^{(k)}) \}$

Figure 2: The one step lookahead procedure of Policy Iteration chooses the action that optimizes the immediate reward plus discounted cumulative reward of running the previous policy from the resulting state.

For any fixed policy $\pi \in \Pi$, we can find the corresponding Value Function simply by solving this linear system. This is a very powerful observation. The Value Iteration algorithm propagates information about values only one action step at a time. But Equation 17 says that for a given policy (or policy modification), value information can be propagated globally throughout the entire state space with the solving of a single linear system. This suggests that perhaps a better algorithm for finding optimal policies would be to start with some initial *policy* π_0, and iterate a procedure of finding the current policy's Value Function by directly solving the linear system, and then using that Value Function to improve the policy by performing a one step lookahead on the Value Function:

$$\begin{aligned}&\textbf{Until convergence, iterate:}\\ &\quad\textbf{Evaluate: } \boldsymbol{v}_{\pi_i} = \left(\boldsymbol{I} - \gamma \boldsymbol{T}_{\pi_i}\right)^{-1} \boldsymbol{r}_{\pi_i}\\ &\quad\textbf{Improve: } \pi_{i+1} = \operatorname*{argmax}_{a \in \mathcal{A}} \left\{ \boldsymbol{r}_a + \gamma \boldsymbol{T}_a \boldsymbol{v}_{\pi_i} \right\},\end{aligned} \tag{18}$$

where the argmax in the policy improvement step is performed element-wise. This policy improvement step is called a one-step lookahead because it finds the new policy's action for a given state by searching over all possible actions from that state and choosing the one has the best chance of landing the agent in a new state whose value under the previous policy is large. Notice that all values increase monotonically since one of the actions this one-step lookahead procedure optimizes over is the action previously chosen by the policy. Figure 2 depicts this one-step lookahead process. Note that it's also possible to define multi-step lookahead policy improvement algorithms. Longer lookahead horizons speed convergence iteration by iteration, but longer lookaheads add a computational burden to each iteration that may slow the overall convergence time.

In practice, since this algorithm leverages the linear relation in Equation 17, Policy Iteration converges faster than Value Iteration.

3.4 Linear Programming formulation

Another interesting solution technique, especially for its theoretical ramifications, is the Linear Programming reduction. Equation 7, suggests a set of inequality constraints on the optimal Value Function, which in vector form can be written as

$$\boldsymbol{v} \geq \boldsymbol{r}_a + \gamma \boldsymbol{T}_a \boldsymbol{v} \quad \forall a \in \mathcal{A}. \tag{19}$$

Note that these inequality constraints treat $\boldsymbol{v}$ as an arbitrary vector, and it's not immediately clear to what extent the constraints enforce the structure of a Value Function, let alone the optimal Value Function. However, again denoting the operation of a Bellman Backup as $F(\boldsymbol{v})$ as we did in Section 8, it's fairly straightforward to understand the implications of these inequalities.

Since the right hand side of Equation 19 holds for all $a \in \mathcal{A}$, we can maximize over it to get

$$\boldsymbol{v} \geq \max_{a\in\mathcal{A}} \left\{ \boldsymbol{r}_a + \gamma \boldsymbol{T}_a \boldsymbol{v} \right\} = F(\boldsymbol{v}).$$

Moreover, we know that if $\boldsymbol{u} \geq \boldsymbol{v}$ in the sense that all elements of $\boldsymbol{u}$ are at least as large as the corresponding element of $\boldsymbol{v}$, then since, for each dimension, the Bellman backup applied to $\boldsymbol{u}$ is maximizing over sets of values that dominate the corresponding values in the application of the Bellman backup to $\boldsymbol{v}$, it must be that $F(\boldsymbol{u}) \geq F(\boldsymbol{v})$. Explicitly, for each dimension,

$$\begin{aligned} F(u(s)) &= \max_{a\in\mathcal{A}} \left\{ r(s,a) + \sum_{s'\in\mathcal{S}} p(s'|s,a)u(s') \right\} \\ &\geq \max_{a\in\mathcal{A}} \left\{ r(s,a) + \sum_{s'\in\mathcal{S}} p(s'|s,a)v(s') \right\} \quad \Big(u(s') \text{ dominates } v(s') \Big) \\ &= F(v(s)). \end{aligned}$$

Combining these results, we see that

$$\begin{aligned} &\boldsymbol{v} \geq F(\boldsymbol{v}) \\ &\Rightarrow F(\boldsymbol{v}) \geq F(F(\boldsymbol{v})) \\ &\Rightarrow F(F(\boldsymbol{v})) \geq F(F(F(\boldsymbol{v}))) \\ &\Rightarrow \boldsymbol{v} \geq \lim_{i\to\infty} \boldsymbol{v}_i = \boldsymbol{v}^*, \end{aligned}$$

where $\boldsymbol{v}_i$ is the ith application of $F(\cdot)$ to $\boldsymbol{v}$. The final relation is true because each application of $F(\cdot)$ is one iteration of the Value Iteration algorithm the limit denotes the convergence of that algorithm to the optimal Value Function. Therefore, any vector $\boldsymbol{v}$ satisfying the system of linear inequalities in Equation 19, is an optimal Value Function. Moreover, since this class of problem has a *unique* optimal Value Function, there is only one feasible point, and it is *the* optimal Value Function.

Since these inequalities are linear, we can use them to construct a Linear Programming formulation of the Markov Decision Problem:

$$\begin{aligned} \min_{\boldsymbol{v}} \quad & \boldsymbol{\mu}^T \boldsymbol{v} \\ \text{s.t.} \quad & \boldsymbol{v} \geq \boldsymbol{r}_a + \gamma \boldsymbol{T}_a \boldsymbol{v} \quad \forall a \in \mathcal{A}, \end{aligned} \tag{20}$$

for any $\boldsymbol{\mu} \in \mathbb{R}_+^{|\mathcal{S}|}$ with strictly positive entries.

This reduction to Linear Programming is theoretically interesting because we know that to any $\epsilon > 0$ precision, we can solve Linear Programs in polynomial time (Boyd & Vandenberghe, 2004), which means that Markov Decision Problems are polynomial time solvable up to ϵ precision.

4 (Optional) Soft Value Iteration for Gibbs distributions

A common probabilistic representation of stochastic behavior that's becoming increasingly popular in modern planning and control algorithms is the Gibbs distribution over trajectories. This distribution takes the form

$$p(\xi) = \frac{1}{Z} e^{R(\xi)}, \tag{21}$$

where $\xi = (s_0, a_0, s_1, a_2, \ldots, s_T, a_T)$ is a trajectory of T state-action pairs, $R(\xi) = \sum_{t=0}^T r(s_t, a_t)$ (for simplicity we're assuming $\gamma = 1$) and $Z = \sum_{\xi \in \Xi} e^{R(\xi)}$ is the distribution's normalizer. The most difficult part of working with these distributions is calculating these normalizers Z. But if we can explicitly represent this distribution, and particularly its normalizer, we suddenly have access to any number of Bayesian queries of interest. For instance, given an observed partial trajectory recorded by a tracking system, and a prior distribution over goals that the agent might be seeking, we can calculate both a posterior distribution over the goals given this information and a distribution over where we might see the agent at any moment in time on route toward one of those goals.

This section shows that a slight modification to the Value Iteration algorithm results in a simple iterative algorithm that converges to the entire field of normalizers $Z_T(s) = \sum_{\xi \in \Xi_s^{(T)}} e^{R(\xi)}$ for to distributions

$$p(\xi) = \frac{1}{Z_T(s)} e^{R(\xi)}, \tag{22}$$

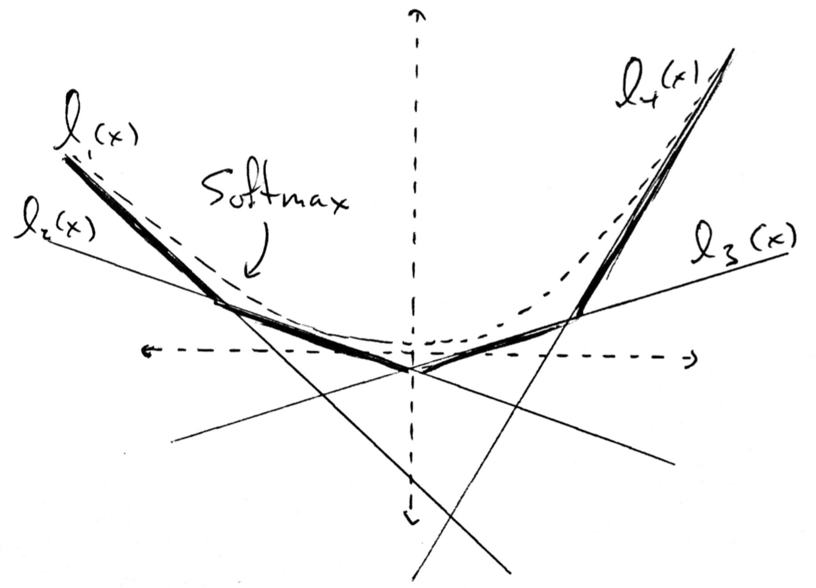

Figure 3: This figure depicts the max and soft max over four linear functions $l_1(x), \ldots, l_4(x)$. The max over the linear functions is shown as the bold upper envelope, and the soft max is shown as the smoother dashed curve upper bounding that hard max envelope. As $l_1(x)$ on the left and $l_4(x)$ on the right begin to increasingly dominate the other lines, the soft max and max start to converge.

over the set of all length T trajectories $\Xi_s^{(T)}$ starting in a given state s.

The slight modification is to turn the hard max in Equation 10 into a soft max. The soft max over a collection of items $\{\alpha_i\}_{i=1}^n$ is softmax$\{\alpha_i\}_{i=1}^n = \log \sum_{i=1}^n e^{\alpha_i}$. This soft max behaves increasingly like the hard max operator as the gap between the largest and second largest $\alpha_i' s$ increases. But as it decreases it starts to upper bound the max value. For instance, when all the values are the same $\alpha_i = \alpha$ for all i, the soft max is

$$\log \sum_{i=1}^n e^\alpha = \log\left(ne^\alpha\right) = \alpha + \log(n). \tag{23}$$

Figure 3 visualizes how the value of the soft max changes and relates to the hard max over the domain of a collection of linear functions.

For simplicity, we assume deterministic transitions and denote the deterministic transition from s to s' under action a by $s' = \phi_a(s)$. Under this setting, this Soft Value Iteration algorithm uses updates of the form

$$\begin{aligned} u_{i+1}(s) &= \operatorname*{softmax}_{a\in\mathcal{A}} \Big\{ \ r(s,a) + u_i(\phi_a(s)) \ \Big\} \\ &= \log \sum_{a\in\mathcal{A}} \exp \Big\{ \ r(s,a) + u_i(\phi_a(s)) \ \Big\}. \end{aligned} \tag{24}$$

Here we denote the "Soft" Value Function by u to emphasize that it's not the Value Function representing cumulative sum of future reward. Instead, we'll see that it's converges on the log of the desired field of normalizers.

The analysis of this algorithm centers around a simple recursive expansion of an exponentiated variant of Equation 24. In this expansion, we use subscripts on states and actions to denote the number of steps taken away from the initial state (i.e. the number of recursive expansions) so that $s_{t+1} = \phi_{a_t} \circ \phi_{a_{t-1}} \circ \cdots \circ \phi_{a_0}(s_0)$. Assuming $u_0(s) = 0$ for all s, the recursive expansion gives

$$\begin{aligned} e^{u_T(s_0)} &= \sum_{a_0\in\mathcal{A}} e^{r(s_0,a_0)+u_{T-1}(\phi_{a_0}(s_0))} = \sum_{a_0\in\mathcal{A}} e^{r(s_0,a_0)} e^{u_{T-1}(\phi_{a_0}(s_0))} \\ &= \sum_{a_0\in\mathcal{A}} e^{r(s_0,a_0)} \sum_{a_1\in\mathcal{A}} e^{r(s_1,a_1)+u_{T-2}(\phi_{a_1}(s_1))} \\ &= \sum_{a_0\in\mathcal{A}} e^{r(s_0,a_0)} \sum_{a_1\in\mathcal{A}} e^{r(s_1,a_1)} \sum_{a_2\in\mathcal{A}} e^{r(s_2,a_2)} \cdots \sum_{a_T\in\mathcal{A}} e^{r(s_T,a_T)} \\ &= \sum_{a_0\in\mathcal{A}} \sum_{a_1\in\mathcal{A}} \cdots \sum_{a_T\in\mathcal{A}} e^{r(s_0,a_0)} e^{r(s_1,a_1)} \cdots e^{r(s_T,a_T)} \\ &= \sum_{\xi\in\Xi_{s_0}} e^{R(\xi)} = Z_T(s_0). \end{aligned}$$

The last line follows because deterministic transitions mean summing over action sequences is equivalent to summing over trajectories. In other words, applying this Soft Value Iteration algorithm T times results in a field of normalizers for distribution over trajectories of length T.

Notice that defining

$$\boldsymbol{z}_i = \begin{bmatrix} e^{u_i(s^{(1)})} \\ e^{u_i(s^{(2)})} \\ \vdots \\ e^{u_i(s^{(|\mathcal{S}|)})} \end{bmatrix}$$

and

$$\boldsymbol{A} = \begin{bmatrix} a_{11} & \cdots & a_{1|\mathcal{S}|} \\ \vdots & \ddots & \vdots \\ a_{|\mathcal{S}|1} & \cdots & a_{|\mathcal{S}||\mathcal{S}|} \end{bmatrix} \quad \text{with} \quad a_{ij} = \begin{cases} e^{r(s^{(i)},a)} & \text{if } \phi_a(s^{(i)}) = s^{(j)} \\ 0 & \text{otherwise} \end{cases},$$

this algorithm can be summarized concisely as

$$\boldsymbol{z}_{i+1} = \boldsymbol{A}\boldsymbol{z}_i. \tag{25}$$

This algorithm is a Power Iteration on the matrix $\boldsymbol{A}$, and, therefore, converges over time to a vector proportional to the primary Eigenvector of $\boldsymbol{A}$. If $\sum_{a \in \mathcal{A}} e^{r(s,a)} = 1$ for all s, $\boldsymbol{A}$ is a stochastic matrix and must have a primary Eigenvector of 1. This algorithm alone will converge as $i \to \infty$. More generally for arbitrary rewards, since $\boldsymbol{z}_i$ is a vector of normalizers for trajectories of length i, we can even run a normalized Power Method method, which converges much more nicely to the primary Eigenvector, and as long as we keep track of the scaling factors associated with every i induced by the normalization, the resulting sequence of vectors are all still usable as trajectory distribution normalizers.

5 Limitations: Why don't MDPs solve everything?

The above discussion demonstrates that some decision problems can be formalized as discrete MDPs and solved using any of a collection of algorithms derived using the Bellman Equation. Those algorithms return *optimal* policies that dictate exactly how to best act in the modeled environment. So why aren't robot's entirely dominating the world in decision making? Everyone finds decisions hard, but here we've shown how to solve them optimally in polynomial time! Doesn't that solve everything?

These algorithms and their relatives are very powerful and have found numerous practical applications, but in many ways MDPs are just a very rough approximation of how the world works. These algorithms are built on discrete state and action spaces, which can be difficult to find in practice. Robots, for instance, have state spaces that are both high-dimensional and continuous describing their location, orientation, configuration of joints, and even manipulable surroundings. Discretizing these spaces explodes exponentially and isn't feasible, so we often need to resort to the local techniques of Optimal Control (Stengel, 1994) or approximate graph-based discretization methods of Motion Planning LaValle (2006). Moreover, it can be hard to even write down precisely what the reward function, itself, is. How much more does it cost walk across grass than pebbles? How does that change when it looks like it might rain? It can take some work to even find the right reward function to make an intelligent system behave in a desired way. Some algorithms have even gone so far as to leverage human intuition to specify how we'd *like* robots to behave and automatically figure out what reward function the system must optimize in order to produce that behavior (Abbeel & Ng, 2004; Ratliff et al., 2006).

But even more fundamentally, the main reason why we, as humans, usually find decision problems so difficult is because we often can't even fully observe our state. We can't predict the future outcome of actions without making observations along the way. You might join a rock band hopeful that you'll hit it big. But there's no way to predict that your lead singer's going to fall in love with a groupie from Switzerland, run off, get married, and leave you stranded right before your big world debut with nothing but a bunch of instruments, a drunk drummer, and some horse backup singers. The real world's only *partially observable*, especially since the state of other actors in our world is a function of the entire history of their own past experiences. The best we can do is make decision based on what we believe we'll probably observe in the future and how we believe that affects our state. Mathematically analyzing models of this partial observability leads to the theory of Partially Observable Markov Decision Processes (POMDPs) and related models. These models are fundamentally intractable, and decision making becomes a game of approximation and speculation. The everyday difficulties we experience when trying to make decisions are fundamental to the computational complexity of the problem, and that's a barrier that, as of yet, even computers haven't been able to overcome.

A Derivation of cumulative reward

The derivation of the result in Equation 1 just takes some algebra, but it's a little cumbersome, so we provide it here for completeness. Define

$$B_k(s_k) = \sum_{s_{k+1},\dots,s_T} \left(\prod_{t=k}^{T-1} p_\pi(s_{t+1}|s_t)\right)\left(\sum_{t=k}^{T-1} \gamma^{t-k} r(s_t,\pi(s_t))\right).$$

Then the sum in question is

$$\begin{aligned}
E_{p_\pi(\xi)}[R_\pi(\xi)] &= \sum_{s_0,\dots,s_T} p(s_0)\left(\prod_{t=0}^{T-1} p_\pi(s_{t+1}|s_t)\right)\left(\sum_{t=0}^{T-1}\gamma^t r(s_t,\pi(s_t))\right)\\
&= \sum_{s_0} p(s_0)\left[\sum_{s_1,\dots,s_T}\left(\prod_{t=0}^{T-1} p_\pi(s_{t+1}|s_t)\right)\left(\sum_{t=0}^{T-1}\gamma^t r(s_t,\pi(s_t))\right)\right]\\
&= \sum_{s_0} p(s_0)B_0(s_0). \qquad (26)
\end{aligned}$$

The basic result is the following recursive expansion:

$$\begin{aligned}
B_k(s_k) &= \sum_{s_{k+1},\dots,s_T}\left(\prod_{t=k}^{T-1} p_\pi(s_{t+1}|s_t)\right)\left(\sum_{t=k}^{T-1}\gamma^{t-k} r(s_t,\pi(s_t))\right)\\
&= r(s_k,\pi(s_k))\underbrace{\sum_{s_{k+1},\dots,s_T}\prod_{t=k}^{T-1} p_\pi(s_{t+1}|s_t)}_{=1} + \sum_{s_{k+1},\dots,s_T}\left(\prod_{t=k}^{T-1} p_\pi(s_{t+1}|s_t)\right)\left(\sum_{t=k+1}^{T-1}\gamma^{t-k} r(s_t,\pi(s_t))\right)\\
&= r(s_k,\pi(s_k)) + \gamma\sum_{s_{k+1}} p_\pi(s_{k+1}|s_k)\sum_{s_{k+2},\dots,s_T}\left(\prod_{t=k+1}^{T-1} p_\pi(s_{t+1}|s_t)\right)\left(\sum_{t=k+1}^{T-1}\gamma^{t-(k+1)} r(s_t,\pi(s_t))\right)\\
&= r(s_k,\pi(s_k)) + \gamma\sum_{s_{k+1}} p_\pi(s_{k+1}|s_k)B_{k+1}(s_{k+1}),
\end{aligned}$$

with the base case $B_T(s_T) = 0$. Plugging this recursive expression into Equation 26 gives the expansion in Equation 1.

References

Abbeel, P. and Ng, A. Y. Apprenticeship learning via inverse reinforcement learning. In *In ICML 04: Proceedings of the twenty-first international conference on Machine learning*, 2004.

Boyd, Stephen and Vandenberghe, Lieven. *Convex Optimization.* Cambridge University Press, 2004.

LaValle, S. M. *Planning Algorithms.* Cambridge University Press, Cambridge, U.K., 2006. Available at http://planning.cs.uiuc.edu/.

Littman, Michael L., Dean, Thomas L., and Kaelbling, Leslie Pack. On the complexity of solving markov decision problems. In *In proc. of the 11th Int. Conf. on Uncertainty in Artificial Intelligence (UAI)*, pp. 394–402, 1995.

Puterman, Martin L. *Markov Decision Processes: Discrete Stochastic Dynamic Programming.* John Wiley and Sons, 1994.

Ratliff, Nathan, Bagnell, J. Andrew (Drew), and Zinkevich, Martin. Maximum margin planning. In *International Conference on Machine Learning*, July 2006.

Stengel, R. *Optimal Control and Estimation.* Dover, New York, 1994.