Finite Markov Decision Processes

In this paper, I develop some of the theory behind Markov decision theory with finite state spaces, actions, and time. This is not entirely self-contained, particularly the diversion into utility functions. I use *Markov Decision Processes* by Martin Puterman (1994) as my primary reference, and *Microeconomic Theory* by Mas-Colell, Whinston, and Green (1995) Chapter 1 contains the relevant background on preference relations I use.

Background and Notation

Before covering the theory behind MDPs, we ought to define exactly what one is and the notation used. Informally, a Markov Decision Problem represents sequential decision making when facing some level of uncertainty. That is, we have some process which evolves over time and a goal to maximize some objective function over the entire time period *without* perfect knowledge of how the system changes. This captures the notion that decisions do not happen in isolation; any decision made today (or this minute, or this second, etc.) impacts all future decisions. Uncertainty enters primarily as a consequence of the decisions we make—an action may have more than one result which is determined probabilistically.

Markov Decision Problems

A MDP builds off the basic framework of a regular Markov chain. Like a Markov chain, it has some number of system states and transition probabilities. Extending this model, it also contains decision epochs, actions which we choose at each decision epoch, rewards and costs which depend on our action and state, and finally it allows the transition probabilities to depend on the action as well as the state.

The first few of these—decision epochs, system states, and actions—form the structure any particular MDP. The number of decision epochs determines how long the system runs, the system states contain all possible states for the Markov chain, and the actions are the potential decisions the problem solver makes at each decision epoch. Formally, we can write these as sets of numbers, states, and actions for each state:

$T = \{1, 2, \ldots, N\}$ with $N < \infty$ is the set of **decision epochs**.

$S = \{s_1, s_2, \ldots, s_n\}$ is the set of **system states**.

$A_s = \{a_{s,1}, a_{s,2}, \ldots, a_{s,k}\}$ is the set of possible **actions** at state $s \in S$, and $A = \bigcup_{s \in S} A_s$ is the union of all actions for all states $s \in S$.

The restriction to finite states, actions, and time periods avoids a number of technicalities in the analysis and solution of these problems.

We can generalize this formulation in two ways in order to describe a wider variety of problems without requiring any more subtle mathematics (though it is more arithmetically complicated). In the simplest case described above, we choose actions deterministically. That is, whenever a decision must be made we pick an action a with certainty. Instead, the actions $a \in A_s$ may be chosen randomly according to some probability distribution $q(s) \in \mathcal{P}(A_s)$. We assign each action $a \in A_s$ some probability $p_a \in [0, 1]$ of being chosen satisfying

$$\sum_{a \in A_s} p_a = 1.$$

Deterministic decisions now occur as a special case when $p_a = 1$ for some action a. This considerably complicates the computation of solutions to these models though it can lead to more optimal outcomes.

The second generalization allows states and actions to depend on time. By considering S_t as the set of states S at time $t \in T$, actions A_{s_t} can vary as time progresses. A particularly relevant application comes from game theory as a finite-stage Iterated Prisoners Dilemma. Under some circumstances, an optimal strategy requires cooperation at every stage until the last round in which defection is preferable. This happens when both players use trigger strategies, cooperating until the first defection and never cooperating again.

The final two parts of the MDP are the rewards and transition probabilities. The **reward**

$$r : S \times A \to \mathbb{R}$$

is a function $r_t(s, a)$ which depends on state s and action a. It assigns a real-valued reward (or cost, if negative) to every action at every state. Sometimes, we want to consider a reward in the final time period during which no decision is made, and we denote this $r_N(s)$. The solution to a MDP involves maximizing these rewards over every time period $t \leq N$. The time subscript signifies the reward earned at t, though in practice it usually does not matter whether the payoffs are rewarded as the MDP progresses or after it has finished. We leave it to keep track of rewards at each decision epoch. The **transition probability** $p(j \mid s, a)$ denotes the probability of moving to state j given state s and action a. We assume that

$$\sum_{j \in S} p(j \mid s, a) = 1.$$

These probabilities can vary with time, but adds little to the theory and does not have the same benefit as tracking rewards over time. With all parts in place, we can now define a Markov Decision Problem.

Definition. A **Markov Decision Problem** is the collection of objects

$$\{T, S, A, r_t(s, a), p(j \mid s, a)\}.$$

Markov refers to the nature of the transition probabilities which depend only on the current state of the system. *Decision Problem* signifies that we want to decide on actions

that maximize some function of the rewards. This process with decisions need not be Markov, as one could make decisions based on the entire history of the process.

Decision Rules and Policies

The concept which can (but need not) cause this non-Markovian behavior are decision rules. A **decision rule** describes which actions to use at each state of the MDP. They range from deterministic, Markovian decision rules which pick an action to use with certainty based only on the current state to randomized, history-dependent rules that allow for a probability distribution of actions based on the entire history of the MDP. They are functions

$$d_t : S \to A_s$$

which take a state as an argument and map to an action to use. Generalizing to a randomized history-dependent rule, we get

$$d_t : S_1 \times A_1 \times S_2 \times A_2 \times \ldots \times A_{t-1} \times S_t \to \mathcal{P}(A_s)$$

in which the decision rule requires all previous states and actions and returns a probability distribution of actions. The subscripts represent the decision epoch of each state and action. This violates the memoryless property of a regular Markov chain as decisions consider the history of states and govern future states as well. Table 1 lists four kinds of decisions rules.

Table 1: Four Kinds of Decisions Rules

	Deterministic	Randomized
Markov	d_t^{MD}	d_t^{MR}
History Dependent	d_t^{HD}	d_t^{HR}

Of these four kinds, MD decisions rules are a special case of MR which in turn are a special case of HR. Likewise, MD rules are a case of HD rules which are also a case of HR rules. The simplest to analyze rules are MD and so we restrict attention to these.

The final ingredient describes a collection of decision rules over time. These are called **policies**, which we define as an ordered tuple of decision rules

$$\pi = (d_1, d_2, \ldots, d_{n-1})$$

where each d_t represents the decision rule at time t. For the solution algorithm in the next section to make sense, we treat all rules as of the same kind (the most general type which all rules fit into) with simpler rules as degenerate cases.

An Example Problem

To illustrate these points, let us now consider and solve a simple, two-period MDP drawn from Problem 2.5 in *Markov Decision Processes*. Our objective function will be the sum of rewards and costs at each period,

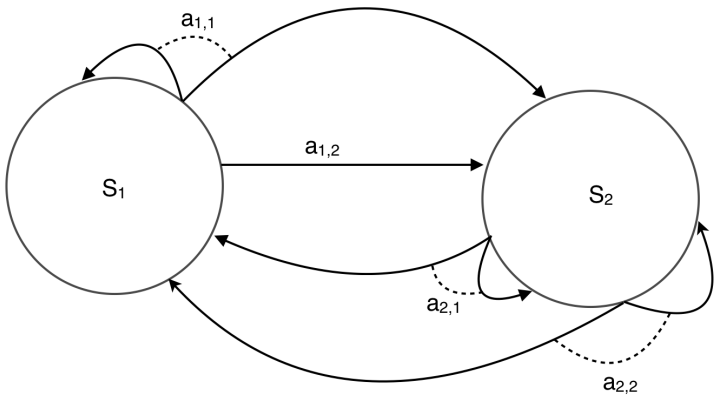
$$r_1(s,a) + \sum_{j \in S} p(j \mid s,a) \cdot r_2(j)$$

which considers the reward from the action at $t = 1$ and then the weighted average of payoffs at $t = 2$. We want to find the reward-maximizing action a_s^* such that

$$r_1(s,a_s^*) + \sum_{j \in S} p(j \mid s,a_s^*) \cdot r_2(j) = \max_{a \in A_s} \left\{ r_1(s,a) + \sum_{j \in S} p(j \mid s,a) \cdot r_2(j) \right\}. \tag{1}$$

Figure 1 diagrams the MDP we want to solve. The circles s_1 and s_2 are the two possible states of the MDP. Arrows (or arrows connected by dashed lines) represent actions with dashed lines connecting arrows indicating that two outcomes are possible, each with nonzero probability. Note that $a_{1,2} \in A_{s_1}$ has no dotted line because it moves the chain to s_2 with probability 1. The table below the diagram lists the rewards and transition probabilities for each action.

Figure 1: A two-state MDP



Action	Reward	Transition Probabilities	
$a_{1,1}$	$r_1(s_1, a_{1,1}) = 5$	$p(s_1 \mid s_1, a_{1,1}) = 0.5$	$p(s_2 \mid s_1, a_{1,1}) = 0.5$
$a_{1,2}$	$r_1(s_1, a_{1,2}) = 10$	$p(s_1 \mid s_1, a_{1,2}) = 0$	$p(s_2 \mid s_1, a_{1,2}) = 1$
$a_{2,1}$	$r_1(s_2, a_{2,1}) = -1$	$p(s_1 \mid s_1, a_{2,1}) = 0.8$	$p(s_2 \mid s_1, a_{2,1}) = 0.2$
$a_{2,2}$	$r_1(s_2, a_{2,2}) = 1$	$p(s_1 \mid s_1, a_{2,2}) = 0.1$	$p(s_2 \mid s_1, a_{2,2}) = 0.9$

First we can simplify to the scenario when $r_2(s_1) = r_2(s_2) = 0$ so that the final reward does not matter. It is easy to see that the reward-maximizing actions at each state are $a_{1,2}$ and $a_{2,2}$ since they have rewards 10 and 1. Thus, we can write our decision rule as

$$d_1(s) = \begin{cases} a_{1,2} & \text{if } s = s_1 \\ a_{2,2} & \text{if } s = s_2 \end{cases}$$

and the policy we wish to use as the 1-tuple $\pi = (d_1)$. This toy problem feels unsatisfyingly simple, however, so we can make it a little bit more interesting by letting the final rewards vary. If we let $r_2(s_1) = x$ and $r_2(s_2) = y$, we now need to consider the expected value of each action.

$$\mathbb{E}\left[m(s_1, a_{1,1})\right] = 5 + 0.5x + 0.5y \qquad \mathbb{E}\left[m(s_1, a_{1,2})\right] = 10 + y$$
$$\mathbb{E}\left[m(s_2, a_{2,1})\right] = -1 + 0.8x + 0.2y \quad \mathbb{E}\left[m(s_2, a_{2,2})\right] = 1 + 0.1x + 0.9y$$

To solve this, we can set the expected values equal to find values of x and y when both actions at each state are equally good. On either side of these curves, one solution will produce a larger expected value and thus maximize the reward. I use weak inequalities to demonstrate the solutions we get when solving for values when one action is at least as good as the other.

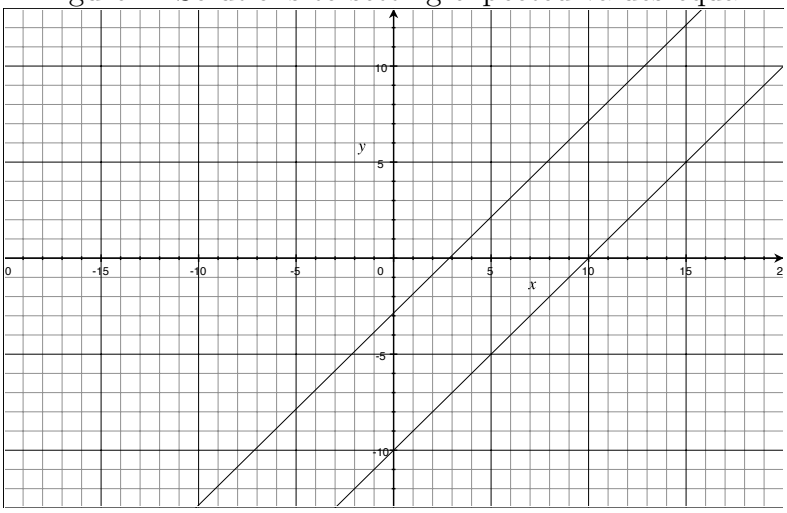
$$\mathbb{E}\left[m(s_1, a_{1,1})\right] \geq \mathbb{E}\left[m(s_1, a_{1,2})\right] \Rightarrow y \leq x - 10 \tag{2}$$
$$\mathbb{E}\left[m(s_2, a_{2,1})\right] \geq \mathbb{E}\left[m(s_2, a_{2,2})\right] \Rightarrow y \leq x - \frac{20}{7} \tag{3}$$

It is interesting to note that in this case, the lines are parallel so that only three solutions exist. That is, we cannot have equation (2) hold while equation (3) does not, so choosing both $a_{1,1}$ and $a_{2,2}$ never maximizes the objective function. When x and y cause either equation to have exact equality, both actions for that state maximize the reward. Otherwise, we can write the decision rules as:

$$d_1(s_1) = \begin{cases} a_{1,1} & \text{if } y < x - 10 \\ a_{1,2} & \text{if } y > x - 10 \end{cases} \qquad d_1(s_2) = \begin{cases} a_{2,1} & \text{if } y < x - \frac{20}{7} \\ a_{2,2} & \text{if } y > x - \frac{20}{7} \end{cases}$$

This has a nice graphical interpretation as well. Figure 2 plots equations (2) and (3) in $\mathbb{R}^2$ if we use equalities. Each (partially) bounded region represents a pair of actions, one for each state, that satisfies equation (1). Above both lines the range of $d_1(s)$ is $\{a_{1,1}, a_{2,1}\}$, between the two lines the range is $\{a_{1,2}, a_{2,1}\}$, and above both lines the range is $\{a_{1,2}, a_{2,2}\}$. When (x, y) lies exactly on one of the lines, the range increases to contain three elements as both actions for one of the states have the same expected reward. You can observe that no fourth region exists as would happen with non-parallel lines, as this would cause the range of the decision rule to be $\{a_{1,1}, a_{2,2}\}$ which cannot happen. Additionally, the range can never be all four actions as the lines to not intersect.

Figure 2: Solutions to setting expected values equal



Optimal Policies and Solution Algorithms

The previous example is easy to solve since there is only one decision epoch, two states, and two actions for each state. As the number of decision epochs, states, and actions grow finding policies which maximize rewards gets much more complicated and computationally infeasible. A naive approach which starts at the first decision and analyzes each possible outcome, then moves to the second epoch and does this again quickly becomes unreasonably long and complex even with a computer. In fact, this method may be impossible if these components are not discrete. A central question in regarding MDPs then is how to efficiently find an optimal policy.

Existence and Form of Objective Functions

Before starting our search for these solutions, we would like to know they exist. This requires an explicit objective function which we seek to maximize. To make things easier, the objective function should take real values and preserve the order of rewards we prefer. By this, I mean that if we prefer y to z both from some set X, written $y \succeq z$ where $\succeq$ is a preference relation on X, we would like the objective function $f : X \to \mathbb{R}$ to satisfy $f(y) \geq f(z)$. Fortunately, this holds under quite general conditions. An objective function with these qualities exists if $\succeq$ is a complete and transitive (or total) order on X.[1]

A sketch of the proof of this proposition for any countable X involves partitioning X into equivalence classes using $\succeq$ (which is an equivalence relation when complete and transitive). This partition has an obvious ordering $\succeq^*$ based on the original preference relation, where $E_y \succeq^* E_z$ if and only if $y \succeq z$ for $y \in E_y$ and $z \in E_z$. Since it is at most countably infinite, there is a bijection f^* from the set of equivalence classes to a subset the natural numbers. Of this bijection, we only require that $E_y \succeq^* E_z \Rightarrow f^*(E_y) \geq f^*(E_z)$. If we then define $f : X \to \mathbb{R}$ as $f(y) = f^*(E_y)$, we have constructed the objective function we want, since the

[1]This is actually an if and only if condition, but I only need one direction here.

following chain of implications holds

$$y \succeq z \Rightarrow E_y \succeq^* E_z \Rightarrow f^*(E_y) \geq f^*(E_z) \Rightarrow f(y) \geq f(z)$$

and demonstrates that f preserves the ordering of $\succeq$. This objective function also maintains order up to a strict monotonic transformation.

As the rewards over time is a vector—an ordered tuple $(r_1, r_2, \ldots, r_N) \in \mathbb{R}^N$—comparisons of reward vectors have no total order and thus do not have a real-valued objective functions with the properties we want (see footnote 1). In place of considering an entire vector, we use a utility function $u : \mathbb{R}^N \to \mathbb{R}$ which sums the reward at each time period and applies (if relevant) a discount factor λ in $[0, 1]$. When $\lambda = 1$ there is no discounting and when $\lambda = 0$ only the current periods reward matters. The function which begins summing rewards at time t takes the form

$$u(r_t, r_{t+1}, \ldots, r_N) = \sum_{k=t}^{N} r_k \lambda^{k-t}$$

which we call a linear additive utility function. This represents the preferences of someone risk-neutral. Now that we draw preferences of rewards over a totally ordered set, $\mathbb{R}$, an objective function exists. In this case, we simply define the utility function u as the objective function.

Optimal Policies

We introduced policies as a way to compactly describe decisions over time. This leads naturally to the question of an **optimal policy** π^*, a list of optimal actions at every epoch. Since actions usually do not move the chain to a state with certainty, we will use the expected value of rewards to compare policies.

To write this expected utility, we choose between introducting more notation, as Puterman does, or a longer form using only what we have developed so far. I opt for the second choice since it is sufficient and makes the underlying ideas clearer. Starting at some state s at epoch $t = 1$, the expected utility of a policy π is

$$u_t^{\pi}(s) = \sum_{k=t}^{N} \left(\sum_{j \in S} r_k(j, d_k(j)) \cdot p(j | j_{k-1}, a_{k-1}) \right) \tag{4}$$

where $d_k(s) = a_k$ is the action at time k dictated by the decision rules in π and s_k is the state at time k. We set $p(s|s_{t-1}, a_{t-1}) = 1$ and $r_N(s, a) = r_N(s)$ so that this equation is well-defined. The expected utility uses the transition probabilities to find the expected value of the reward at each epoch. The optimal policy π^* maximizes equation (4) and we denote this maximum $u_t^{\pi^*}(s)$. More precisely this means that for all initial states $s \in S$ and policies π, $u_t^{\pi^*}(s) \geq u_t^{\pi}(s)$.

By restricting ourselves to finite states, actions, and epochs, we assure the existence of this optimal policy. Combinatorial calculations for a specific MDP using the number of states one can reach from every action provide an exact count of the possible policies. Under less strict calculations, we achieve a larger but still finite upper bound on this number. With

N periods ($N-1$ decision epochs), n states, and k actions at each state which can move the chain to every state with nonzero probability, we have $n^N k^{N-1}$ total MR policies. The MR policy is important as it restricts us to considering only the current state rather than the history (though this would still be finite, just far larger). MD policies do not lower this bound since we assume that all actions can move to all states and so randomizing over actions does not increase the number of available states at the next epoch. As we have finitely many policies, the set

$$\left\{u_t^\pi(s) \in \mathbb{R} \mid \pi \text{ is a policy for the MDP}\right\}$$

is finite and thus has a maximum. Hence for some state s, we have the existence of an policy which maximizes that set.

Definition. The **optimal policy** π^* in a given Markov decision policy for initial state s is

$$\pi^* = \arg\max_\pi \left\{u_t^\pi(s) \in \mathbb{R} \mid \pi \text{ is a policy for the MDP}\right\}$$

or the policy which maximizes the expected utility of the rewards. For use in the algorithm below to find optimal policies we define $u_t^*(s) = u_t^{\pi^*}(s)$.

Algorithms to Find Optimal Policies

With the existence and definition of optimal policies finally settled, we can describe how to find such a policy. Given finiteness, we could iterate through every possible policy and pick the optimal policy. However, this grows exponentially with the number of decision epochs and polynomially (of deg $= N$ or $N-1$) with the state spaces and possible actions which quickly poses limitations on the problems we can solve. Furthermore, this does not generalize easily to MDP with infinite time horizons, actions, or states. Instead, we use a backwards induction algorithm to find an optimal policy.

A simple notion underlying backwards induction leads to efficient calculation. By knowing the final rewards at time N, we can calculate the action at time $N-1$ which maximizes this reward for each state at $t = N-1$ and call it d_{N-1}. Now to find the optimal action at time $N-2$, we already know the optimal action at each possible future state and again pick the action which maximizes the reward which is now d_{N-2}. Continuing recursively, at each decision epoch t we simply pick the action which leads to the optimal policy beginning at time $t+1$ and assign it to d_t. The algorithm stops after reaching $t = 1$ as it has determined the action at each epoch. Aligning these as a policy, we achieve the optimal policy $\pi^* = (d_1, d_2, \ldots, d_{N-1})$.

Contrasting this to the forward calculation approach, the number of calculations at each stage does not increase but stays the same, at most $k \cdot n$ or the number of states times the number of actions. Since we do this for each epoch, we have at most $(N-1) \cdot k \cdot n$ computations, which grows much more slowly than the total number of possible policies derived above does. Essentially, by collapsing future rewards into optimal policies from the end, each state has only one value associated with it in the next time period. This differs from a tree-type approach which branches off into numerous possibilities at every stage as forward calculations do.

Of course, we would like to describe this process using the language of MDPs we have developed. This involves three steps which occur recursively until the algorithm finishes. Let us call this the Optimal Policy Algorithm.

Definition. The **Optimal Policy Algorithm** finds an optimal policy for a given Markov Decision Problem. It follows these steps:

(1) Set $t = N$ and for all $s_N \in S$, compute

$$u_t^*(s_N) = r_N(s_N).$$

(2) Substitute $t-1$ for t and compute $u_t^*(s_t)$ for each $s_t \in S$ by

$$u_t^*(s_t) = \max_{a \in A_{s_t}} \left\{ r_t(s_t, a) + \sum_{j \in S} p(j \mid s_t, a) \cdot u_{t+1}^*(j) \right\},$$

and set

$$A_{s_t,t}^* = \arg\max_{a \in A_{s_t}} \left\{ r_t(s_t, a) + \sum_{j \in S} p(j \mid s_t, a) \cdot u_{t+1}^*(j) \right\}$$

(3) If $t = 1$, stop. Otherwise return to step (2).

This algorithm recursively defines the set of optimal actions for all times $t \leq N$ and all states $s_t \in S$. Then define the decision rule $d_t^*(s_t) = a_{s_t} \in A_{s_t,t}^*$. If $A_{s_t,t}^*$ has two or more elements, pick one to choose with certainty as the choice does not matter. Finally, the optimal policy is

$$\pi^*(s) = \left(d_1^*(s), d_2^*(s), \ldots, d_{N-1}^*(s)\right).$$

Concluding Remarks

This ends our development of Markov decision theory. By restricting ourselves to finite problems, we can succinctly prove the existence of and derive an algorithm for finding optimal policies. Many of the ideas here extend to more complicated problems, particularly in countably infinite time and state spaces (infinite actions are more subtle). Existence of optimal policies is no longer guaranteed but holds under more technical conditions of semi-continuity and the optimal policy algorithm finds ϵ-optimal policies, a weaker form of optimality than the one described above.