Markov Decision Processes Lecture Notes 04

Value Functions

Authors: Benjamin Van Roy *October 6, 2025*

1 Value Functions

A reward function expresses immediate reward earned from taking an action at a state. It is often unwise to take an action only because it maximizes immediate reward. This is because the action also influences future opportunities for earning reward. Value functions predict returns rather than just immediate reward. Because of this, they are better suited to guide strategic decisions that balance between immediate rewards and future opportunities.

In Lecture 03, we formulated several types of objectives, each taking the form of an expected return. There are similarly multiple types of value functions, which we will define in the next section. We will restrict attention in this lecture to MDPs with finite state and action spaces, though the concepts largely extend to infinite spaces.

1.1 Total Value

The *total value function* under policy π is defined by

$$V_{\pi,T}(s) = \mathbb{E}_{s,\pi}\left[\sum_{t=0}^{T-1} R_{t+1}\right]. \tag{1}$$

This value function $V_{\pi,T}$ provides the expected value over T time steps, starting at state s. We also define the *optimal total value function*:

$$V_{*,T}(s) = \max_{\pi} V_{\pi,T}(s). \tag{2}$$

Note that, for an optimal policy π_*, $V_* = V_{\pi_*}$. Further, such an optimal policy will typically be nonstationary because, as time progresses, the horizon approaches its end.

1.2 Discounted Value

The *discounted value function* under policy π is defined by

$$V_{\pi,\gamma}(s) = \mathbb{E}_{s,\pi}\left[\sum_{t=0}^{\infty} \gamma^t R_{t+1}\right]. \tag{3}$$

We also define the *optimal discounted value function*:

$$V_{*,\gamma}(s) = \max_{\pi} V_{\pi,\gamma}(s) \tag{4}$$

1.3 Relative Value

Denote the gain under stationary policy π, starting at state s, by

$$\lambda_\pi(s) = \lim_{T\to\infty} \mathbb{E}_{s,\pi}\left[\frac{1}{T}\sum_{t=0}^{T-1} R_{t+1}\right].$$

We restrict attention to stationary policies because for nonstationary policies this limit does not necessarily exist. We then define the bias by

$$V_\pi(s) = \lim_{\gamma \uparrow 1} \mathbb{E}_{s,\pi} \left[\sum_{t=0}^{\infty} \gamma^t (R_{t+1} - \lambda_\pi(S_t)) \right]. \tag{5}$$

The optimal gain is defined by

$$\lambda_*(s) = \max_\pi \lambda_\pi(s). \tag{6}$$

Let $\Pi_*(s)$ be the set of stationary policies that attain this maximum. We then define the optimal bias by

$$V_*(s) = \max_{\pi \in \Pi_*(s)} V_\pi(s). \tag{7}$$

Note that if we were not to restrict the domain of optimization to $\Pi_*(s)$, it can sometimes possible to identify a policy that increases the gain while reducing the bias. We view optimization of gain and bias as lexicographic, with gain prioritized.

1.4 Relation Between Relative and Discounted Value

Maximizing gain and bias is akin to maximizing expected discounted return with an asymptotically large discount factor. As such, it is possible to closely approximate the expected discounted return $V_{\pi,\gamma}(s)$ for γ close to one in terms of the gain $\lambda_\pi(s)$ and bias $V_\pi(s)$. In particular, the Laurent series expansion around $1 - \gamma \approx 0$ gives us

$$V_{\pi,\gamma}(s) = \frac{1}{1-\gamma} \lambda_\pi(s) + V_\pi(s) + O(1-\gamma). \tag{8}$$

The last term vanishes with $1 - \gamma$. Note that we use the Laurent rather than the Taylor series expansion to allow for the negative power $(1-\gamma)^{-1}$ that appears in the first term.

1.5 Examples

We offer a couple examples to illustrate value functions.

1.5.1 Queueing

Our first example is a variation of the queueing system from Lecture 01. A key difference is that we will limit the buffer size to keep the state space finite. In particular, we allow a maximum of twenty customers waiting in the queue. If a customer arrives when the queue is full, that customer abandons and this incurs a cost of 500. This is large relative to the cost of service, which is 30 for the fast mode and 0 for the slow mode. Each waiting customer incurs one unit of cost per time step.

Over each time step, a customer arrives with probability 0.5. If a customer is present then their service is completed over the next time step with probability 0.7 or 0.4, depending on whether the fast mode or slow mode of service is deployed.

Expected discounted return, with a discount factor of $\gamma = 0.99$, is maximized by a policy that uses the slow mode of service when there are no more than 12 waiting customers and, otherwise, the fast mode.

Figure 1 plots discounted value functions. There are plots for the optimal value function and the value function for the policy that always uses the fast mode of service. Clearly, the optimal value function dominates. The values become similar, though, as the queue length grows.

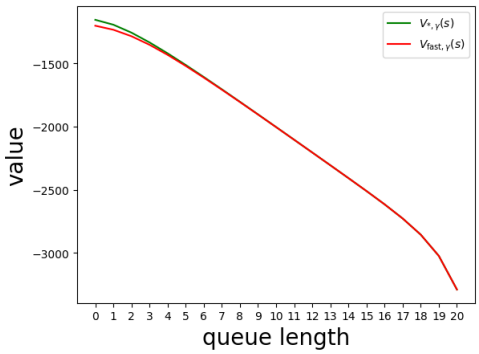

Figure 1: Value functions for the queueing system.

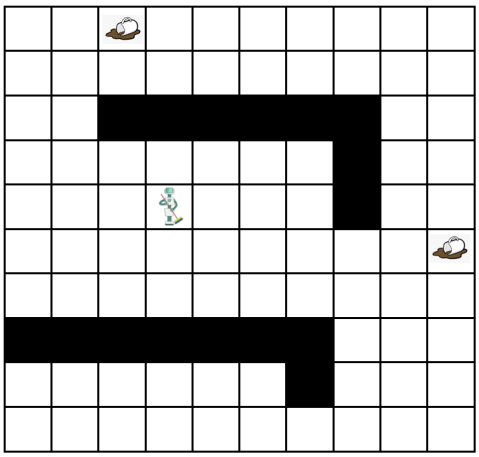

Figure 2: A MopBot environment.

1.5.2 MopBot

As a second example, we introduce the mopbot environment, as illustrated in Figure 2. In this environment, spills occur at random times and random locations, and the mopbot navigates to clean them.

Let us consider a very simple version of this environment in which the mobpot is restricted to a single hallway made up of five cells, as illustrated in Figure 3. The state of the environment encodes the mopbot's position and the locations of any spills that remain to be cleaned up. Over each time step and for each location, a spill occurs with probability $p_{\text{spill}} = 0.1$.

Over each time step, the mopbot can stay in its current cell or move to an adjacent cell. A spill is cleaned when the mopbot resides in the same cell for one timestep. The cost incurred over a timestep, equivalent to the negative reward, is the number of spills. Consider maximizing gain and then bias.

Figure 3: A very simple MobBot environment.

Figure 4 (left) indicates optimal actions across mopbot locations when there is a single spill in the leftmost cell. In this case, the mopbot makes a beeline for the spill. Figure 4 (right) indicates optimal actions across mopbot locations when there are no spills. In this case, the mopbot travels to the center cell.

Figure 4: Optimal actions when there is one spill or no spills.

Figure 5 plots the optimal relative value function across locations when there are no spills. The value at cells 0 and 4 are smaller than others because from there it could take the mopbot longer to reach new spills. The value across states 1, 2, and 3, are identical. This is because from each of these cells, the next will be the center cell (cell 2), which would optimally position the mopbot for future spills.

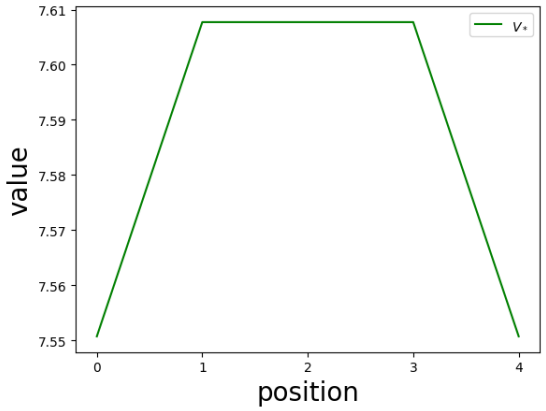

Figure 5: The optimal relative value function.

2 From Value to Policy

Value functions indicate desirability of states under particular policies. As such, they can guide decisions by prioritizing actions that lead to high-value states. In this section, we begin to explore this use case.

2.1 Action Values

In order to guide decisions, we need to attribute value to actions. This is done by *action value functions*. An action value function maps each state-action pair to a value. For the case of the total value function, action values are given by

$$Q_{\pi,T}(s,a) = r(s,a) + \sum_{s' \in \mathcal{S}} P_{ass'} V_{\pi,T-1}(s'). \tag{9}$$

Each action value is the sum of immediate reward and expected value started at the next time step.

We can similarly define discounted, and relative action value functions:

$$Q_{\pi,\gamma}(s,a) = r(s,a) + \gamma \sum_{s' \in \mathcal{S}} P_{ass'} V_{\pi,\gamma}(s'), \tag{10}$$

$$Q_\pi(s,a) = r(s,a) - \lambda_\pi(s) + \sum_{s' \in \mathcal{S}} P_{ass'} V_\pi(s'). \tag{11}$$

We can also define an average action value function:

$$\eta_\pi(s,a) = \sum_{s' \in \mathcal{S}} P_{ass'} \lambda_\pi(s'). \tag{12}$$

Note that the reward function does not appear in this definition because the average future reward does not depend on reward realized over one time step.

Each action value function we have defined via Equations (9, 10, 11) is for a given policy. If we take π to be an optimal policy π_*, we obtain definitions for optimal action value functions:

$$Q_{*,T}(s,a) = r(s,a) + \sum_{s' \in \mathcal{S}} P_{ass'} V_{*,T-1}(s'), \tag{13}$$

$$Q_{*,\gamma}(s,a) = r(s,a) + \gamma \sum_{s' \in \mathcal{S}} P_{ass'} V_{*,\gamma}(s'), \tag{14}$$

$$Q_*(s,a) = r(s,a) - \lambda_*(s) + \sum_{s' \in \mathcal{S}} P_{ass'} V_*(s'), \tag{15}$$

$$\eta_*(s,a) = \sum_{s' \in \mathcal{S}} P_{ass'} \lambda_*(s'). \tag{16}$$

2.2 Greedy Policies

For any function $Q : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, we say a policy π is greedy with respect to Q if, for all $s \in \mathcal{S}$, the support of $\pi(\cdot|s)$ is contained within $\arg\max_{a \in \mathcal{A}} Q(s,a)$. Hence, a greedy policy π with respect to Q would only ever select an action A_t that maximizes $Q(S_t, \cdot)$.

The following theorem speaks to how greedy policies can improve performance.

Theorem 1. (policy improvement: discounted value) *For any $\gamma \in [0,1)$, policy π, and policy π' that is greedy with respect to $Q_{\pi,\gamma}$:*

1. *$V_{\pi',\gamma} \geq V_{\pi,\gamma}$,*
2. *if $V_{\pi,\gamma} \neq V_{*,\gamma}$ then there exists $s \in \mathcal{S}$ such that $V_{\pi',\gamma}(s) > V_{\pi,\gamma}(s)$.*

Hence, the greedy policy π' does at least as well as π starting from any state and does better for at least one initial state if π' is not optimal. While this fact is intuitive, we will revisit it in Lecture 06 to better understand why it is true.

An implication of Theorem 1 is that a greedy policy with respect to an optimal discounted action value function is optimal. For example, given an optimal discounted value function $Q_{*,\gamma}$, if π_* is greedy with respect to $Q_{*,\gamma}$ then π_* is optimal. Hence, given $Q_{*,\gamma}$ an optimal action can be selected at time t simply by maximizing $Q_{*,\gamma}(S_t, \cdot)$. In this way, access to $Q_{*,\gamma}$ converts a multi-time-step decision problem into a single-time-step decision problem. While maximizing $r(S_t, \cdot)$ leads to poor decisions that do not account for delayed consequences, maximizing $Q_{*,\gamma}(S_t, \cdot)$ yields optimal actions.

An analog to Theorem 1 holds for total value:

Theorem 2. (policy improvement: total value) *For any policy π and any policy π' that is greedy with respect to $Q_{\pi,\tau}$ for $\tau = 1, 2, \ldots, T$:*

1. *$V_{\pi',\tau} \geq V_{\pi,\tau}$ for $\tau = 1, 2, \ldots, T$,*
2. *if, for all for $\tau = 1, 2, \ldots, T$, $V_{\pi,\tau} \neq V_{*,\tau}$ then there exists $s \in \mathcal{S}$ and $\tau = 1, 2, \ldots, T$ such that $V_{\pi',\gamma}(s) > V_{\pi,\gamma}(s)$.*

The situation is slightly more complicated with gain/bias. The policy of interest is greedy with respect to the gain, with ties broken by the bias. In particular, given functions $\eta : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ and $Q : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, we say that π is greedy with respect to η then Q if, for any $s \in \mathcal{S}$, the support of $\pi(\cdot|s)$ is a subset of $\arg\max_{a \in \mathcal{A}_*(s)} Q(s,a)$, where $\mathcal{A}_*(s) = \arg\max_{a \in \mathcal{A}} \eta(s,a)$. We have the following policy improvement result.

Theorem 3. (policy improvement: gain/bias) *For any stationary policy π and any policy π' that is greedy with respect to η_π then Q_π:*

1. $\lambda_{\pi'} \geq \lambda_\pi$
2. *if* $\lambda_{\pi'} = \lambda_\pi$ *then* $V_{\pi'} \geq V_\pi$
3. $\lambda_{\pi'} \neq \lambda_*$ *then either*
   (a) *there exists* $s \in \mathcal{S}$ *such that* $\lambda_{\pi'}(s) > \lambda_\pi(s)$ *or*
   (b) *there exists* $s \in \mathcal{S}$ *such that* $V_{\pi'}(s) > V_\pi(s)$.

2.3 Example: Queueing

Let us revisit the queueing system described in Section 1.5.1. Figure 6 plots discounted action value functions for the policy that only ever chooses the fast action (left) and differences between those plotted curves (right). When the queue length s is short, $Q_\pi(s, \text{slow}) > Q_\pi(s, \text{fast})$. When the queue length is long, $Q_\pi(s, \text{slow}) < Q_\pi(s, \text{fast})$. A greedy policy π' with respect to π, which always chooses the fast action, would instead choose the slow action when s is sufficiently small. This would improve performance relative to π.

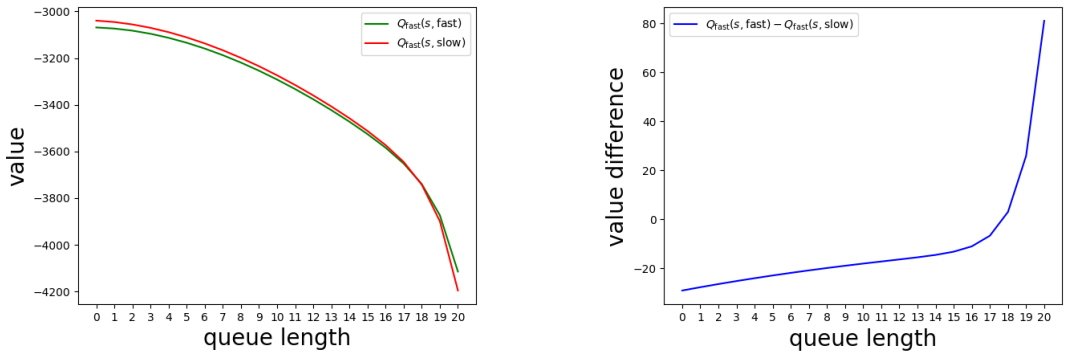

Figure 6: Discounted action value functions for the queueing system, and their difference.

Figure 6 plots optimal discounted action value functions (left) and differences between those plotted curves (right). Again, $Q_*(s, \text{slow}) > Q_*(s, \text{fast})$ when the queue length s is short and $Q_*(s, \text{slow}) < Q_*(s, \text{fast})$ when long. A greedy policy with respect to Q_* is optimal.

2.4 Rollouts

Suppose we wish to maximizes expected discounted return. Given a policy π, Theorem 1 suggests a means of improvement: instead of π, use a policy π' that is greedy with respect to $Q_{\pi,\gamma}$. In Lecture 05 we will study means for computing $Q_{\pi,\gamma}$. But those means require computation to scale with the size of the state space, which in many problems of practical interest can be onerous. Rollouts provide a tractable means for computing $Q_{\pi,\gamma}(S_t, \cdot)$ when at state S_t. The result can then be used to generate the next action $A_t \in \arg\max_{A \in \mathcal{A}} Q_{\pi,\gamma}(S_t, a)$ as though sampled from a greedy policy π'.

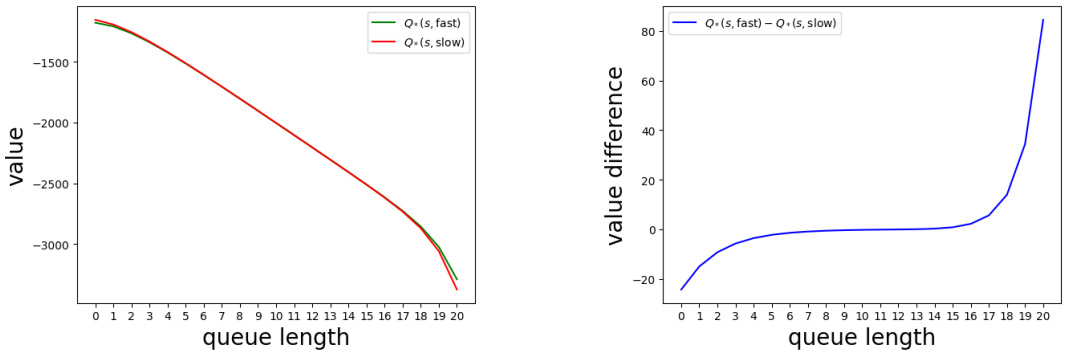

Figure 7: Discounted optimal action value functions for the queueing system.

A rollout is a simulated trajectory starting at a specified state-action pair. For example, given an MDP $(\mathcal{S}, \mathcal{A}, \mathcal{W}, f, \nu)$, to generate a rollout starting at state $S_0 = s$ and action $A_0 = a$, for $t = 0, 1, 2, \ldots$, we would generate a sample W_{t+1} from $\nu(\cdot|S_t, A_t)$ and then let $S_{t+1} = f(S_t, A_t, W_{t+1})$.

We can average over some number K of rollouts to arrive at an estimate of the action value:

$$Q_{\pi,\gamma}(s, a) \approx \frac{1}{K} \sum_{k=1}^{K} \sum_{t=0}^{\infty} \gamma^t r(S_t^k, A_t^k). \tag{17}$$

If we do this for each action $a \in \mathcal{A}$, with sufficiently large K, and select the one with the larger estimate, we are likely to arrive at an action that is greedy or nearly greedy with respect to $Q_{\pi,\gamma}$. Rollouts can be used similarly to improve total value or gain/bias.

Use of the rollout approach can be computational taxing as the number of simulations requires scales with the $K \times |\mathcal{A}|$. In general, K may need to be very large. But for cases when dynamics are deterministic, $K = 1$ suffices. By *deterministic dynamics* we mean that $\nu(\cdot|s, a)$ is always one-hot or, equivalently, there is a function f such that $S_{t+1} = f(S_t, A_t)$. In this case, all rollouts starting at a given state-action pair would be identical and, therefore, a single rollout suffices.

2.5 Example: Solitaire

We discuss application of the rollout approach to a version of solitaire with deterministic dynamics.

2.5.1 Background

Solitaire came into existence when fortune-telling with cards gained popularity in the eighteenth century. Many people play this game every day, yet simple questions such as *What is the chance of winning? How does this chance depend on the version I play? What is a good strategy?* remain beyond mathematical analysis.

Many variations of solitaire exist today, such as Klondike, Freecell, and Carpet. Popularized by Microsoft Windows, Klondike has probably become the most widely played. We will study a version of solitaire in which the identity of each card at each position is revealed to the player at the beginning of the game but the usual Klondike rules still apply. Figure 8 provides a rendering of the state of a game.

Our version of solitaire – which we will call *thoughtful solitaire* – is played by a number of serious solitaire players as a much more difficult version than standard Klondike. A prominent American mathematician spent a number of years studying the game and found a way to achieve a win rate of 36.6%. With this background, it is natural to wonder how well an optimal player can perform at thoughtful solitaire. We will

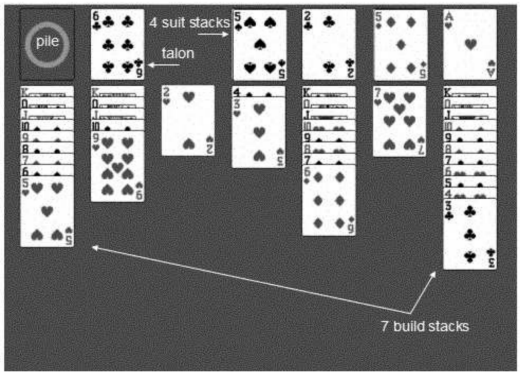

Figure 8: Thoughtful solitaire.

discuss an application of rollouts that leads to a win rate of about 70%. A more through presentation of this work is available in [Yan et al., 2004].

2.5.2 Base Policy

The Microsoft Windows Klondike app comes with a scoring system that assigns points to individual moves. This constitutes a function $Q : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$. We use a base policy π_0 that is greedy with respect to Q.

2.5.3 Rollout

Given π_0, we generate actions according to an improved policy π_1 based on rollouts. In particular, actions are selected as follows:

1. For each possible move a at S_t, generate a rollout, simulating the trajectory and selecting subsequent moves using π_0 until the game is won or lost.
2. If any of the rollouts resulted in victory, choose uniformly at random among them and let A_t be the corresponding move.
3. If none of the rollouts resulted in victory, sample A_t from $\pi_0(\cdot|S_t)$.

2.5.4 Iterated Rollouts

The procedure described above can be repeated to generate actions according to further improved policies π_2, π_3, and so on. In particular, to act according to $\pi_{\ell+1}$:

1. For each possible move a at S_t, generate a rollout, simulating the trajectory and selecting subsequent moves using π_ℓ until the game is won or lost.
2. If any of the rollouts resulted in victory, choose uniformly at random among them and let A_t be the corresponding move.
3. If none of the rollouts resulted in victory, sample A_t from $\pi_\ell(\cdot|S_t)$.

While the computational requirements grow exponentially with ℓ, we can apply several iterations within reasonable compute time. Simulation results are provided in the following Table 1 and Figure 9. We randomly generated a large number of games and played them with our algorithms in an effort to approximate the success probability with the percentage of games actually won. To determine a sufficient number of games

to simulate, we used the Central Limit Theorem to compute confidence bounds on the success probability for each algorithm with a confidence level of 99%. For the base policy and 1 through 3 rollout iterations, we attained confidence bounds of ±1.4%. For 4 and 5 rollout iterations, due to time constraints, we simulated fewer games and obtained weaker confidence bounds. Interestingly, however, after 5 rollout iterations, the resulting strategy wins almost twice as frequently as our esteemed mathematician.

Player	Success Rate	Games Played	Average Time Per Game	99% Confidence Bounds
Human expert	36.6%	2,000	20 minutes	±2.78%
heuristic	13.05%	10,000	.021 seconds	±.882%
1 rollout	31.20%	10,000	.67 seconds	±1.20%
2 rollouts	47.60%	10,000	7.13 seconds	±1.30%
3 rollouts	56.83%	10,000	1 minute 36 seconds	±1.30%
4 rollouts	60.51%	1,000	18 minutes 7 seconds	±4.00%
5 rollouts	70.20%	200	1 hour 45 minutes	±8.34%

Table 1: Simulation results from iterated rollouts.

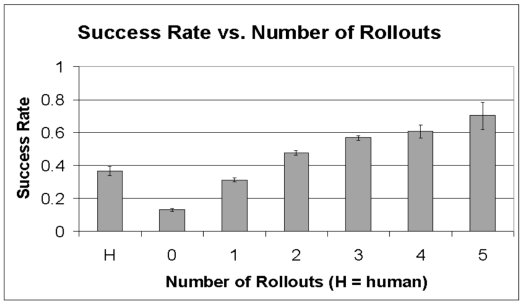

Figure 9: Simulation results from iterated rollouts.

3 Bellman Equations

Our policy improvement theorems indicate that:

1. Given a policy value function, we can compute action values and use them to generate an improved policy.
2. Given an optimal value function, we can compute optimal action values and use them to generate an optimal policy.

Policy value functions and optimal value functions can be computed by solving Bellman equations, which we introduce in this section.

3.1 Discounted Value

Theorem 4. *Let π be a stationary policy. Then, $V_{\pi,\gamma}$ is the unique solution to the system of equations*

$$V_{\pi,\gamma}(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(r(s,a) + \gamma \sum_{s' \in \mathcal{S}} P_{ass'} V_{\pi,\gamma}(s') \right) \qquad \forall s \in \mathcal{S}. \tag{18}$$

Theorem 5. *$V_{*,\gamma}$ is the unique solution to the system of equations*

$$V_{*,\gamma}(s) = \max_{a \in \mathcal{A}} \left(r(s,a) + \gamma \sum_{s' \in \mathcal{S}} P_{ass'} V_{*,\gamma}(s') \right) \qquad \forall s \in \mathcal{S}. \tag{19}$$

3.2 Total Value

Theorem 6. *Let π be a state-modulated policy. $(V_{\pi,0}, \ldots, V_{\pi,T})$ is the unique solution to*

$$V_{\pi,\tau+1}(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(r(s,a) + \sum_{s' \in \mathcal{S}} P_{ass'} V_{\pi,\tau}(s') \right) \qquad s \in \mathcal{S}, \tau = 1, 2, \ldots \tag{20}$$

$$V_{\pi,0}(s) = 0 \tag{21}$$

Theorem 7. *$(V_{*,0}, \ldots, V_{*,T})$ is the unique solution to*

$$V_{*,\tau+1}(s) = \max_{a \in \mathcal{A}} \left(r(s,a) + \sum_{s' \in \mathcal{S}} P_{ass'} V_{*,\tau}(s') \right) \qquad s \in \mathcal{S}, \tau = 1, 2, \ldots \tag{22}$$

$$V_{*,0}(s) = 0 \tag{23}$$

3.3 Relative Value

Theorem 8. *Let π be a stationary policy. The set of solutions to the system of equations*

$$\lambda(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \sum_{s' \in \mathcal{S}} P_{ass'} \lambda(s') \qquad \forall s \in \mathcal{S} \tag{24}$$

$$V(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(r(s,a) - \lambda(s) + \sum_{s' \in \mathcal{S}} P_{ass'} V(s') \right) \qquad \forall s \in \mathcal{S} \tag{25}$$

is the set of pairs of the form $(\lambda_\pi, V_\pi + g)$ such that

$$g(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \sum_{s' \in \mathcal{S}} P_{ass'} g(s') \qquad \forall s \in \mathcal{S}. \tag{26}$$

The analysis is a bit more complicated for optimal value functions. The following result asserts only that (λ_*, V_*) satisfy Bellman equations but not that they are the unique solutions.

Theorem 9. *Optimal value functions λ_* and V_* satisfy*

$$\lambda(s) = \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} P_{ass'} \lambda(s') \qquad \forall s \in \mathcal{S} \tag{27}$$

$$V(s) = \max_{a \in \mathcal{A}_*(s)} \left(r(s,a) - \lambda(s) + \sum_{s' \in \mathcal{S}} P_{ass'} V(s') \right) \qquad \forall s \in \mathcal{S}, \tag{28}$$

where $\mathcal{A}_*(s)$ is the set of actions that attains the maximum in (27). Further, for any pair (λ, V) that satisfy these equations, $\lambda = \lambda_*$ and any policy that attains that maximum in (28) is gain and bias optimal.

4 Geometric Interpretation

Recall that

$$\max_{\pi} \lambda_\pi = \max_{\mu \in \mathcal{P}} \sum_{s \in \mathcal{S}} \sum_{a \in \mathcal{A}} \mu(s,a) r(s,a), \tag{29}$$

where $\mathcal{P}$ is the polytope defined by linear constraints:

$$\sum_{a' \in \mathcal{A}} \mu(s',a') = \sum_{s \in \mathcal{S}} \sum_{a \in \mathcal{A}} \mu(s,a) P_{ass'} \qquad \forall s' \in \mathcal{S} \tag{30}$$

$$\sum_{s \in \mathcal{S}} \sum_{a \in \mathcal{A}} \mu(s,a) = 1 \tag{31}$$

$$\mu(s,a) \geq 0 \qquad \forall s \in \mathcal{S}, a \in \mathcal{A}. \tag{32}$$

Hence, (29) is a linear program. Shadow prices (aka othe optimal dual solution) of a linear program measure sensitivity to changes in constrains. For example, one could ask how the maximum objective value would change if we were to add a tiny amount ϵ to the right-hand-side of (30) for some $s' \in \mathcal{S}$. It turns out that the shadow prices of (30) provide optimal relative values V_*.

To see why, we formulate the dual linear program for the case where each policy has a single recurrent class thus the average value function is constant across states:

$$\begin{array}{ll} \min_{\lambda, V} & \lambda \\ \text{s.t.} & V(s) \geq r(s,a) - \lambda + \sum_{s' \in \mathcal{S}} P_{ass'} V(s') \qquad s \in \mathcal{S}. \end{array} \tag{33}$$

It can be shown that any optimal solution (λ, V) satisfies Equations (24) and (25). We will leave showing this for later, when that will be very simple to do using tools we will develop.

References

Xiang Yan, Persi Diaconis, Paat Rusmevichientong, and Benjamin Van Roy. Solitaire: Man versus machine. In L. Saul, Y. Weiss, and L. Bottou, editors, *Advances in Neural Information Processing Systems*, volume 17. MIT Press, 2004.