**ORIE 5530: Modeling Under Uncertainty**

# Lecture 13 (Infinite Horizon Discounted Dynamic Programmin)

*Professor Mark S. Squillante,*
*Adapted from Professor Itai Gurvich's original notes*

First, a few words about static vs. dynamic optimization, building on the pricing and the secretary problems we saw.

Let us start with the secretary problem and think of a linear optimization based heuristic way to obtain the optimal hiring rule. Recall: we are interviewing $n$ secretaries and have a hiring budget of $k$. There are secretaries of four possible qualities: the quality is $k$ with probability $p_k$.

Say we solve the dynamic programming problem and have a policy.

Let's call

$$x_i = \text{ the average number of quality-}i\text{ secretaries that the policy hires in expectation.}$$

Then, these four variables $x_1, x_2, x_4, x_4$ must satisfy

i. $x_1 + x_2 + x_3 + x_4 \leq k$

ii. $x_i \leq np_i =$ expected number of quality-$i$ candidates we will see in $n$ interviews.

iii. The total reward we get from hiring quality-1 candidates is $x_1$, quality-2 candidates $2x_2$, etc. Thus, the optimal value equals
$$x_1 + 2x_2 + 3x_3 + 4x_4.$$

Pretending that we can freely set these, we have an upper bound equal to the solution of the *linear-optimization problem*

$$\begin{aligned} \max\ & x_1 + 2x_2 + 3x_3 + 4x_4 \\ \text{s.t.}\ & x_1 + x_2 + x_3 + x_4 \leq k, \\ & x_i \leq np_i, \text{ for } i = 1, 2, 3, 4, \\ & x_i \geq 0, \text{ for } i = 1, 2, 3, 4. \end{aligned}$$

Four questions:

- Why is the solution of this an upper bound on the optimal value but not necessarily equal to the optimal value.
- How can you interpret this policy to come up with a dynamic heuristic for hiring candidates?
- Will the heuristic achieve the upper bound value, why no?
- Why would a dynamic policy do better that this heuristic?

A first static analysis gives a good sense, however, into structural properties of good policies as well as an implementable simple heuristic.

**Question:** What is the static problem for the inventory-pricing problem?

## 13.1 Infinite horizon discounted dynamic programming

In the last class we saw how finite horizon formulations are solved via backward recursion. In finite horizon dynamic programs the optimal action depends on the time period (on how many steps to go) so there is never rarely policy. Also, they may be difficult to compute if the horizon $n$ is large.

Infinite horizon discounted problems take care of some of these complications and they are also a proxy for finite horizon in that larger weights are put on what happens in the shorter term. By adjusting the discount constant $\beta$ you can adjust how much you care about the far future. The discount factor $\beta$ also has a natural monetary interpretation.

In infinite-horizon problems, we seek to find a dynamic and adaptive decision rule that maximizes the collected reward

$$\mathbb{E}_x[\sum_{k=0}^{\infty} \beta^k r(X_k, A_k)],$$

where $X_k$ is the state of the process we are tracking at time $k$ and $A_k$ is the action we take in this period. The evolution of $X_k$ will depend on the actions we take. As before the reward $r(X_k, A_k)$ is allowed to depend on the state and the action. In an inventory problem this might be the amount of inventory we have and how much we order.

Some of the what follows should be familiar. Suppose that the control is fixed and $X_k, k = 0, 1, 2, \ldots$ is a given (finite state space) Markov chain and you are studying

$$V(x) = \mathbb{E}_x[\sum_{k=0}^{\infty} \beta^k r(X_k)].$$

We saw in class that we can compute $V$ by solving the system of equation

$$V(x) = r(x) + \beta \sum_y P(x, y)V(y), \tag{13.1}$$

which has the solution (in vector form) $V = (I - \beta P)^{-1} r$.

Thus, it seems that if the chain is not given but rather it is optimized, a variant of the equation (13.1) should give us a way to find the optimal control. Meaning that if the rewards depend on the control $u$, and thus the probability transition matrix, we should have that the optimal value satisfies the equations

$$V(x) = \max_{\delta \in \Delta(x)} \left\{ r(x, \delta) + \beta \sum_y P_\delta(x, y)V(y) \right\}. \tag{13.2}$$

In principle this is a so-called **fixed-point** equation. We have $V$ on both sides. How do we solve this thing?

**Example 1** *You have $c$ units of inventory. A customer arrives every period. If you price the product at $p$, the customer buys an item with probability $\lambda(p)$. This the same example we used in the finite-horizon context. Here, the tradeoff is a bit different. If you wait, you are not necessarily going to be stuck with remaining inventory (because there is infinite horizon), BUT selling very far in the future brings a discounted value. You'd rather get the cash now! So you are trying to maximize*

$$V(x) = \mathbb{E}_x[\sum_{i=0}^{\infty} p_i R_i(p_i)].$$

*In contrast to the finite horizon setting, notice that here we do not have a dependence of $V$ on the period $i$. This means we also lose this terminal condition we had which said that $V_n(x) = 0$. The question is how does your price schedule look like if you wish to maximize the infinite horizon discounted reward?* ∎

To solve the Bellman equation (13.2), we cannot invert as we did with a fixed control but we will have a rather simple algorithm. Let's return to our pricing example:

$$V(c) = \max_{p\geq 0}\{p\lambda(p) + (1-\lambda(p))V(c) + \lambda(p)(V(c-1))\}.$$

Let us put a concrete structure on the price so that we can actually do something here. Also, to make things interesting let's use here $\lambda(p) = e^{-p}$ instead of $\lambda(p) = 1-p$ that we used in the finite horizon case. Then,

$$\begin{aligned} V(c) &= \max_{p\geq 0}\{pe^{-p} + \beta(1-e^{-p})V(c) + \beta e^{-p}(V(c-1))\} \\ &= \beta V(c) + \max_{p\geq 0}\{pe^{-p} + \beta e^{-p}(V(c-1) - V(c))\}. \end{aligned}$$

How do we maximize the inside. We take a derivative and compare to 0. We get (fill in the details) that

$$p^*(c) = 1 - \beta(V(c-1) - V(c)) = 1 + \beta(V(c) - V(c-1)).$$

This give us a way to get the optimal price **once** we know the value. This expression make intuitive sense – the optimal price is the average valuation (which is 1) plus $\beta(V(c) - V(c-1))$ which is how much we are losing from not having that extra item a period from now. Why does it make sense that we never price below average valuation? Think if you had only one item at what price would you offer it ?

How do we get the function $V(c)$? We know $V(0) = 0$ so we can even do so in excel. Indeed, plugging in the expression $p^*(c)$ for the optimal price we have that

$$\begin{aligned} V(c) &= \beta V(c) + \max_{p\geq 0}\{pe^{-p} + \beta e^{-p}(V(c-1) - V(c))\} \\ &= \beta V(c) + e^{-p^*(c)}(p^*(c) + \beta(V(c-1) - V(c))) \\ &\qquad \text{use } p^*(c) = 1 + \beta(V(c) - V(c-1)) \\ &= \beta V(c) + e^{-p^*(c)} \end{aligned}$$

Plugging back the definition of $p^*(c)$ we have **the point equation**

$$(1-\beta)V(c) = e^{-(1+\beta(V(c)-V(c-1)}.$$

Using $V(0) = 0$ (if there is no inventory, there is no revenue), we can now solve this using even an excel solver. We have variables $V(1), V(2), \ldots$ and equations as above. See the worksheet called "optimization" in the excel file *RM_DP_Example.xlsx*

This is not a useful generalizable procedure because it depends on our ability in this simple example to express the optimal action in a clean form as a function of the value function.

In practice, we will rarely enjoy such simplicity. Instead, we use a procedure called value iteration.

## 13.2 Value Iteration

We start with a first guess $V^0(x)$ for the value function for each $x$. The easiest guess is to set $V^0(x) = 0$ for all states $x$. Then, we improve upon it by solving **for each $x$ in the state space**

$$V^1(x) = \max_{\delta\in\Delta(x)}\{r(\delta, x) + \beta\sum_y P_\delta(x,y)V^0(y)\}.$$

We then continue iterating. For each $n$, we compute **for all state $x$**:

$$V^n(x) = \max_{\delta \in \Delta(x)} \{r(\delta, x) + \beta \sum_y P_\delta(x, y) V^{n-1}(y)\}.$$

The beautiful fact is that this converges to the true value; namely, as $n \to \infty$,

$$V^n(x) \to V(x),$$

where $V$ is the real optimal value. Once we have that, we know how to find the optimal policy.

**Remark 1** 1. You could have better candidates for the initial values $V^0(x)$ sometimes. The algorithm will converge regardless of your guess (that is the powerful result here) but the closer your guess to the true value, the faster the algorithm will converge.

2. *Only if you are curious: Why does it work?* This is in case you are interested in an informal proof of why this works. The formal proof follows something called "the contraction principle" but we can actually see this from a simpler argument.

   Take $V^0(x) = 0$ as we did and notice the following: Since $V^0(x) = 0$ and assuming the reward per $r(\delta, x)$ is always non-negative, we have

   $$V^1(x) = \max_{\delta \in \Delta(x)} \{r(\delta, x) + \beta \sum_y P_\delta(x, y) V^0(y)\} = \max_{\delta \in \Delta(x)} \{r(\delta, x)\} \geq 0 = V^0(x).$$

   So we conclude that

   $$V^1(x) \geq V^0(x) \text{ for all } x.$$

   We are now going to use an induction argument. Suppose that $V^k(x) \geq V^{k-1}(x)$ for all $k \leq n-1$ and consider now the equation for $k = n$:

   $$\begin{aligned} V^n(x) &= \max_{\delta \in \Delta(x)} \{r(\delta, x) + \beta \sum_y P_\delta(x, y) V^{n-1}(y)\} \\ &\geq \max_{\delta \in \Delta(x)} \{r(\delta, x) + \beta \sum_y P_\delta(x, y) V^{n-2}(y)\} \\ &= V^{n-1}(x), \end{aligned}$$

   where the first inequality follows from the induction assumption and the last inequality from the definition of value-iteration by which $V^{n-1}(x) = \max_{\delta \in \Delta(x)} \{r(\delta, x) + \beta \sum_y P_\delta(x, y) V^{n-2}(y)\}$.

   Thus, we conclude, for all $n$, that

   $$V^n(x) \geq V^{n-1}(x) \text{ for all } x.$$

   You now have to recall a basic fact from analysis that monotone increasing sequences converge to a number or diverge to infinity. If the per-period reward $r(\delta, x)$ is bounded, then the sequence is bounded and hence converges to some function

   $$f(x) = \lim_{n \to \infty} V^n(x).$$

   It is fair for you to ask now why is the limit the solution to the original Bellman (dynamic programming equation) but notice that since both $V^n(x) \to f(x)$ for all $x$ and $V^{n-1}(y) \to f(y)$ as $n \to \infty$ then

   $$\begin{array}{ll} V^n(x) & = \max_{\delta \in \Delta(x)} \{r(\delta, x) + \beta \sum_y P_\delta(x, y) V^{n-1}(y)\} \\ \downarrow & \qquad\qquad\qquad\qquad\qquad\qquad \downarrow \\ f(x) & = \max_{\delta \in \Delta(x)} \{r(\delta, x) + \beta \sum_y P_\delta(x, y) f(y)\} \end{array}.$$

   Hence, $f$ solves the dynamic programming equation and $f = V$.

■

Let's again return to the single-product pricing example. We have reached the dynamic-programming (Bellman) equation:

$$V(c) = \beta V(c) + \max_{p \geq 0}\{pe^{-p} + \beta e^{-p}(V(c-1) - V(c))\}.$$

The value iteration is:

$$V^n(c) = \beta V^{n-1}(c) + \max_{p \geq 0}\{pe^{-p} + \beta e^{-p}(V^{n-1}(c-1) - V^{n-1}(c))\}.$$

Following similar derivation to what we did above, the optimal $p$ **in the** $n^{th}$ **iteration** is

$$p^{*,n}(c) = 1 + \beta(V^{n-1}(c) - V^{n-1}(c-1)),$$

and we can plug this back in to get

$$\begin{aligned} V^n(c) &= \beta V^{n-1}(c) + e^{-p^{*,n}(c)} \\ &= \beta V^{n-1}(c) + e^{-(1+\beta(V^{n-1}(c) - V^{n-1}(c-1)))}. \end{aligned}$$

*Value iteration will look like this:*

0. Take $V^0(c) = 0$ for all $c$.
1. Take $p^{*,1}(c) = 1 + \beta(V^0(c) - V^0(c-1)) = 1$ and
$$V^1(c) = e^{-1}.$$
2. Repeat step 1 for all $n$. For example, given $V^1(c)$ we compute the price $p^{*,2}(c) = 1 + \beta(V^1(c) - V^1(c-1))$ and compute
$$V^2(c) = \beta V^1(c) + e^{-(1+\beta(V^1(c) - V^1(c-1)))}.$$

See the worksheet "value iteration" in the file *RM_DP_Example.xlsx*.

In general, you will not be able to solve for the optimal equation in a clean way from the equation. You might actually have to solve an optimization problem for each $n$ and $x$. Things get a bit simpler when you have a discrete set of possible actions – in that case you just have to compare values; see HW5 problem 2.

## 13.3 Synthesis of optimal policy

Value iteration gives you the optimal value $V(x)$ but **does not give you the optimal action**. The prices $p^{*,n}(c)$ that we computed while iterating are just instruments to find the optimal value. By themselves they do not mean anything. We want the optimal price that arises once we have the true value function $V(x)$. In the example above, we were able to identify in closed form

$$p^*(c) = 1 + \beta(V(c) - V(c-1)).$$

Once your algorithm converges and you have the value function $V$ you can compute the optimal prices.

More generally, we recall that optimal actions are the maximizers in the equation

$$V(x) = \max_{\delta \in \Delta(x)}\{r(\delta, x) + \beta \sum_y P_\delta(x, y)V(y)\}.$$

Since we already have $V$ from value iteration and do not need to solve for it, the optimal actions are the maximizers:

$$\delta^*(x) = \underset{\delta \in \Delta(x)}{\operatorname{argmax}} \{r(\delta, x) + \beta \sum_y P_\delta(x, y) V(y)\},$$

where $\delta^*(x)$ is a function from the state $x$ to the best action. In the case that the set of actions is discrete and finite, we can plug in different actions and see which one returns the maximal value.

If the state space is discrete (like the level of inventory in our example) we can finally **represent** $\delta^*(x)$ **by a table** that specifies for each $x$ what is the optimal action.

Here is a full matlab code for this problem where $c$ is allowed to be as high as 50.

```
beta=0.95;

v=zeros(50);
num˙iter = 100;
for k=1:num˙iter
v(1)=beta* v(1)+exp(-(1+beta*v(1)));
for c=2:50
v(c)=beta*v(c) +exp(-(1+(beta*v(c)-v(c-1))));
end
k=k+1;
end
v this output v
```

**policy synthesis**

```
price=zeros(50);
price(1)= 1+beta*v(1);
for c=2:50
price(c)=1+beta*(v(c)-v(c-1));
end
delta this outputs the optimal price
.....
```

Using $\beta = 0.95$ gives for $c = 1$, 1.6 which is consistent with the excel file. The optimal price it gives for $c = 1$ for example is 2.52.