Dynamic Programming

Dynamic programming (DP) is the most important technique to solve many optimozation problems. In most applications, dynamic programming obtains solutions by working *backwards* from the end of a problem to the beginning, in such a way that a large, complicated problem is broken up into a series of smaller, more tractable problems.

1 Simple examples of dynamic programming

1.1 Which is the heaviest coin

We have 21 coins and are told that all the coins are of the same weight except one is heavier than any of the other coin. How many weighings on a balance will it take to find the heaviest coin?

Analysis: The answer of the question is not so clear at first glance. However, let us think backwards. What is the maximum number of coin we should have left in order for the last weighing to successfully tell the heaviest coin. This is an easy question to answer: 3 coins. We only need to pick two of three coins and weigh them on the balance. If they are of the same weight, then the one coin left is the heaviest coin. If they are not, the heavier one is the heaviest coin in the bunch. However, if we have 4 or more coins left, we cannot always tell which one is the heaviest if only one weighing is left.

Now let us go back one step and ask what is the maximum number of coin we should have left in order for the last two weighing to successfully tell the heaviest coin. Suppose we have $m \leq 9$ coins left. What we can do is to divide the coin into three groups (a, a, b) such that $a + a + b = m$, and $a \leq 3$, $b \leq 3$. Then we weight the first two groups on the balance. If they have the same weight, then the heaviest coin is in the group of b coins. If they are not, then the heaviest coin is in the heavier group of a coins. Either way, since $a \leq 3, b \leq 3$, we can always tell the heaviest coin in the last weighing. Now assume that we have $m \geq 10$ coins, and we divide into three groups (a, a, b). A weighing on balance will tell which group the heaviest coin belongs to. No matter how we do it, there is at least a group with at least 4 coins. If it turns out that the heavier coin is in that group, we are in trouble, because the last weighing cannot tell the heaviest coin. Therefore, we can tell the heaviest coin from at most 9 coins, and cannot not tell if there are at least 10 coins.

Go on step futher, it is easy to see that if we have at most $3 \cdot 9 = 27$ coins, we can tell the heaviest one in three weighings. Therefore, for 21 coins, we need at most 3 weighings. Not only does the above procedure gives the number of weighing need, it also yields how should the weighings go: We will divide the coins into three groups (a, a, b) such that $a \leq 9, b \leq 9$. Say we take $a = b = 7$. The first weighing will tell us the group the heaviest coin belongs to. Then we divide that group (of 7 coins) into three groups $(2, 2, 3)$ or $(3, 3, 1)$. The second weighing will tell which group the heaviest coin belong to, and the last weighing will only need to tell the heaviest coin from at most 3 coins.

Remark: The above procedure indeed solves a more general problem: suppose we have $n \geq 2$ coins to start with, then the number of weighings needed to tell the heaviest coin is k with $3^{k-1} < n \leq 3^k$. In other words, k weighings will tell the heaviest coin from at most 3^k coins. A mathematical description for the above process is as follows. Let

$$v_k \doteq \max\{n : \text{ the heaviest coin can be distinguished from at most } k \text{ weighings}\}.$$

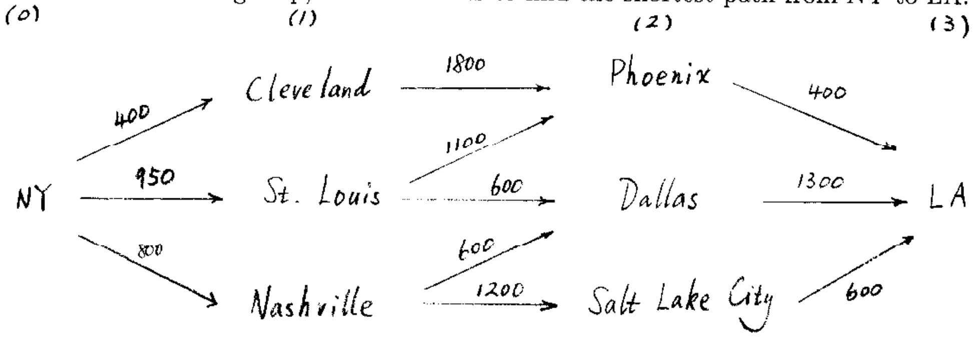
We start with $v_1 = 3$ and we have $v_{k+1} = 3v_k$. Therefore, $v_k = 3^k$.

The analysis also tells how to weigh these n coins in k weighting: each time divide the coins into three groups (a, a, b) such that $a \leq 3^{l-1}$, $b \leq 3^{l-1}$ if there are l weighings left.

Remark: The coin problem can be viewed as a optimization problem: minimize the number of weighings to tell the heaviest coin. The above analysis, though simple, give some basic characteristics of the dynamic programming: not only does DP gives a systematic way to solve an optimization problem, it also tells you *more* information on the *optimal policy.* In the coin problem, the policy corresponds to the process of weighing.

1.2 Shortest path problem

The following example is a simple ***shortest path*** problem, which we will discuss later in great generality. Consider the following map, and one wants to find the shortest path from NY to LA.



This shortest path problem has a very distinctive feature in that the problem can be divided into several stages.

$$\begin{aligned}
\text{Stage 0} &= \{\text{NY}\} \\
\text{Stage 1} &= \{\text{Cleveland, St. Louis, Nashville}\} \\
\text{Stage 2} &= \{\text{Phoenix, Dallas, Salt Lake City}\} \\
\text{Stage 3} &= \{\text{LA}\}
\end{aligned}$$

Each stage consists of one or several cities, and one always moves from a city in one stage to a city in the next stage. A direct approach to the problem is to enumerate all the possible paths and choose the one with the minimal distance. Such an approach will work in such a small-scale example. But Imagine doing this for a complicated map!

The idea here is again working backwards. Let

$$V_k(x) = \text{shortest path to LA when you are at city } x \text{ of stage } k.$$

Clearly, $V_3(\text{LA}) = 0$

$$V_2(\text{Phoenix}) = 400, \quad V_2(\text{Dallas}) = 1300, \quad V_2(\text{Salt Lake City}) = 600.$$

And it is easy to see that

$$\begin{aligned}
V_1(\text{Cleveland}) &= \text{dist(Cleveland, Phoenix)} + V_2(\text{Phoenix}) = 1800 + 400 = 2200, \\
V_1(\text{St. Louis}) &= \min\{\text{dist(St.Louis, Phoneix)} + V_2(\text{Phoenix}), \text{dist(St.Louis, Dallas)} + V_2(\text{Dallas})\} \\
&= \min\{1100 + 400, 600 + 1300\} = 1500. \\
V_1(\text{Nashville}) &= \min\{\text{dist(Nashville, Dallas)} + V_2(\text{Dallas}), \\
&\qquad\qquad \text{dist(Nashville, Salt Lake City)} + V_2(\text{Salt Lake City})\} \\
&= \min\{600 + 1300, 1200 + 600\} = 1800.
\end{aligned}$$

It is easy now to have

$$\begin{aligned}
V_0(\text{NY}) &= \min\{\text{dist(NY, Cleveland)} + V_1(\text{Cleveland}), \text{dist(NY, St. Louis)} + V_1(\text{St. Louis}), \\
&\qquad \text{dist(NY, Nashville)} + V_1(\text{Nashville})\} \\
&= \min\{400 + 2200, 950 + 1500, 800 + 1800\} = 2450.
\end{aligned}$$

As before, the DP also tells us the shorest path is "NY→St. Louis→Salt Lake City→LA".

Remark: The above iterations can be written in a more compact form.

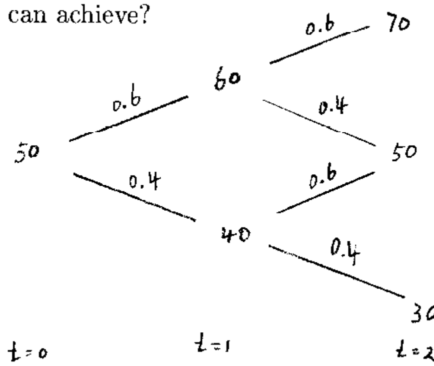
$$V_k(x) = \min\{\text{dist}(x, y) + V_{k+1}(y) : \quad y \in \text{Stage}(k+1)\},$$

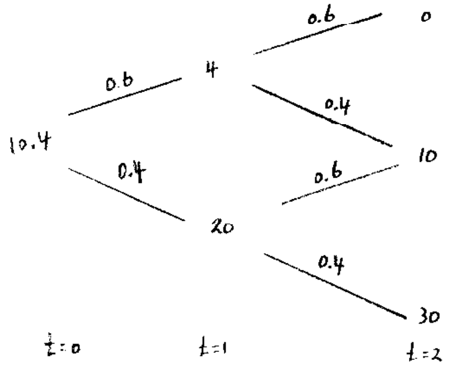
with the convention that $d(x, y) = \infty$ if there is no path from x to y. This is called the *dynamic programming equation* (DPE).

1.3 American option pricing

The next example is an American option pricing problem and can be viewed as a simple example of *Decision Analysis*. A bit of intuitive probabilities is involved. Even though the notation becomes more messy, the basic idea of dynamic programming remains same and simple.

Suppose you own a share of American put option on IBM with strike price 60 dollars. In other words, if you liquidate the option when the IBM stock price is S, you get a payoff $(60 - S)^+ = \max\{0,\ 60 - S\}$. Everyday the stock price has probability 0.6 to go up 10 dollars and probability 0.4 to go south 10 dollars. Figure 1 is a tree for the stock price. The option can be liquidated at any time $t = 0$, $t = 1$, or $t = 2$. You want to maximize your average payoff from the liquidation. Question: should you liquidate the option now, or should you wait? What is the maximal average payoff you can achieve?





Solution: The solution is not at all clear. If you liquidate now, you will get $(60-50)^+ = 10$ dollars. If you choosc to wait, it is hard to tell that whether you could get more or less.

Now let us work backwards. Suppose we are at $t = 2$, what should we do? This is obvious since we will liquidate if the price is below the strike to get a payoff $(60 - S)^+$, and the option is toilet paper is the stock price is above the strike. The payoff is calculated in Figure 2.

Go back one step, say we are at $t = 1$, and the stock price become 60. The liquidation of the option will yield a payoff of 0. If we choose to wait, we will get an average payoff $0.6 \cdot 0 + 0.4 \cdot 10 = 4$. Therefore, if we are at $t = 1$ and the stock price is 60, we should wait and expect an average payoff

$$\max\{(60-60)^+, 0.6 \cdot 0 + 0.4 \cdot 10\} = \max\{0, 4\} = 4.$$

What if at $t = 1$ the stock price is 40 dollars. If you liquidate, you will get a payoff 20 dollars. If not, you will on average get a payoff $0.6 \cdot 10 + 0.4 \cdot 30 = 18$ dollars. Therefore, if we are at $t = 1$ and the stock price is 40, we should immediately liquidate and get a payoff

$$\max\{(60-40)^+, 0.6 \cdot 10 + 0.4 \cdot 30\} = \max\{20, 18\} = 20.$$

Go back one more step, and we are $t = 0$. If we liquidate, we get a payoff 10 dollars. But if we wait, we will expect to get a payoff $0.6 \cdot 4 + 0.4 \cdot 20 = 10.4$. Therefore at $t = 0$, we should wait and expect an average payoff

$$\max\{(60-50)^+, 0.6 \cdot 4 + 0.4 \cdot 20\} = \max\{10, 10.4\} = 10.4$$

At $t = 1$, if the stock price is 60 dollars, keep waiting; if the stock is 40 dollars, liquidate.

Remark: Again, the DP works backwards, and tell us the information on the optimal strategy.

Remark: The above DP procedure can be written in a more compact form. Let

$$V_k(x) = \text{The maximal average payoff if you are at } t = k \text{ and the stock price is } x.$$

We are interested in $V_0(x)$ at $x = 50$. However, we have $V_2(x) = (60-x)^+$, and

$$V_k(x) = \max \ \left\{(60-x)^+, p_u V_{k+1}(x_u) + p_d V_{k+1}(x_d)\right\};$$

here $x_u = x + 10$, $x_d = x - 10$, $p_u = 0.6$, $p_d = 0.4$. This equation is again said to be the *dynamic programming equation* (DPE). The first term in the maximization corresponds to the payoff if you liquidate at $t = k$, while the second term corresponds to the average payoff if you choose to wait. Note that at any time, we have two strategies, either liquidate or wait.

2 A general framework of deterministic dynamic programming

Consider a system whose evolution is determined by

$$x_{n+1} = f_n(x_n, u_n), \qquad n = 0, 1, \cdots, N.$$

The parameters in the above equation has interpretation as

$$\begin{aligned} n &= \text{the label for the stage} \qquad \text{(usually it is naturally defined)},\\ x_n &= \text{the state at stage } n\\ u_n &= \text{the control (or, decision) variable at stage } n. \end{aligned}$$

In other words, suppose at stage n, the system is in state x_n. If a decision u_n is applied, the system will move to state s_{n+1} at stage $n+1$.

In the previous example of shortest path, the stage is artifically defined, and the state is the city, and the decision variable is which city to travel to. In the example of American option pricing, the stage is naturally defined as the time, and the state is the price of the IBM stock, and the decision variable is binary "liquidate" or "wait".

Let us consider the following *cost criteria.* The goal is to select a *decision (control) sequence* $U = \{u_0, u_1, \cdots, u_N\}$ so as to

$$g(x_{N+1}) + \sum_{n=0}^{N} c_n(x_n, u_n)$$

We can view

$$\begin{aligned} g(x) &= \textit{terminal cost} \text{ if the terminal state } x_{N+1} \text{ is } x \\ c_n(x,u) &= \textit{running cost} \text{ if at stage } n \text{, a decision } u_n = u \text{ is made at state } x_n = x. \end{aligned}$$

In the example of shortest path, the running cost $c_n(x,u)$ is nothing but the distance from x to destination determined by the control u. There is no terminal cost involved.

Remark: For American option pricing, the situation is a bit subtle since if a decision of "liquidation" is made at any stage, the stock price (the state) after that stage does not matter a bit. Therefore the problem cannot directly fit into the cost structure we mentioned above. However, the idea to solve the problem using DPE (which we will describe below) remains the same. We will come back to this later in the section of Probabilistic Dynamic Programming.

Remark: Sometimes the above cost criteria is called *additive.* Of course one can consider a *multiplicative* (or even mixed) cost criteria. A general multiplicative cost reads

$$g(x_{N+1}) \cdot \prod_{n=0}^{N} c_n(x_n, u_n).$$

The analysis of this cost is exactly the same as that of the additive cost criteria.

The **value function** is defined as

$$V_0(x) \doteq \min_U \left[g(x_{N+1}) + \sum_{n=0}^{N} c_n(x_n, u_n) \right] \qquad \text{given } x_0 = x.$$

To solve for V_0, we will *expand* the problem and work backwards. For notation convenience, we will denote by

$$V_j(x) \doteq \min_U \left[g(x_{N+1}) + \sum_{n=j}^{N} c_n(x_n, u_n) \right] \qquad \text{given } x_j = x,$$

for $j = 1, 2, \cdots, N+1$, with convention that $\sum_{N+1}^{N} \doteq 0$.

Dynamic Programming Equation (DPE)

It is not difficult to see that $V_{N+1}(x) \equiv g(x)$ for any x by definition. Furthermore, it is intuitive that the functions V_j should satisfy the following equation

$$\text{(DPE)} \qquad V_j(x) = \min_u \left[c_j(x,u) + V_{j+1}(f_j(x,u)) \right], \quad \text{for every } x.$$

This equation is called the *dynamic programming equation* (DPE). Knowing V_{N+1}, one can recursively solve all V_j. The interpretation of DPE is also clear. Suppose at stage j, the system is at state $x_j = x$. Choosing a control $u_j = u$, at stage $j+1$, the state will become

$$x_{j+1} = f_j(x_j, u_j) = f_j(x,u).$$

Therefore the DPE says "*The minimum of the cost from stage j to the end of the problem must be attained by choosing at stage j a decision that minimize the sum of the costs incurred during the current stage plus the minimum cost that can be incurred from stage $j+1$ to the end of the problem.*" We give a short (not so rigorous) proof below.

[redacted]

These two inequalities yield the (DPE). Furthermore, from the proof we verify that "*the optimal control at stage j is the control u^* that achieving the minimum in the RHS of the DPE.*"

Remark: Note that the DPE actually solve $V_0(x)$ for *any* x. In practice $x_0 = x$ is a specific value, but which is not important since one only need to plug this specific initial state into function V_0 to obtain the value of optimization problem associated with this specific x_0.

Remark: It is not hard to believe that for the multiplicative cost criteria,one can similarly define V_j (with convention $\prod_N^{N+1} = 1$), and the DPE will become

$$\text{(DPE)} \qquad V_j(x) = \min_u \left[c_j(x,u) \cdot V_{j+1}(f_j(x,u)) \right], \quad \text{for every } x,$$

with $V_{N+1}(x) \equiv g(x)$. Again, the optimal control at stage j is the minimizing u^* in the RHS of the DPE.

Remark: The above procedure remains true in a maximization problem – just replace all the "min" above by "max".

2.1 Examples of DP

Example: The owner of a lake must decide how many bass to catch and sell at the beginning of each year. Assume that the market demand for bass is unlimited. If x is the bass sold in year n, a revenue of $r(x)$ is earned. The cost to catch x bass is $c(x,b)$, where b is the number of bass in the lake at the beginning of the year. The number of bass in the lake at the beginning of a year is 20% greater than the number in the lake at the end of the previous year. The owner is interested in maximizing the overall profit over the next N years.

DP Formulation: In this case, the stage is naturally defined as the time n, which varies from 1 to N. The state variable is the number of bass at the beginning of year n, denoted by s_n. The decision variable at year n (control) is the number of bass caught, denoted by x_n.

The dynamic for the system is

$$s_{n+1} = 1.2(s_n - x_n); \qquad n = 1, 2, \cdots, N-1.$$

The objective is to maximize the overall profit

$$v(s) \doteq \max_{\{x_n\}} \sum_{n=1}^{N} \left[r(x_n) - c(x_n, s_n) \right], \qquad \text{given } s_1 = s.$$

The dynamic programming will work backwards: define

$$v_j(s) \doteq \max_{\{x_n\}} \sum_{n=j}^{N} \left[r(x_n) - c(x_n, s_n) \right], \qquad \text{given } s_j = s,$$

for $j = 1, \cdots, N$, and define $v_{N+1}(s) = 0$ for any s (note the terminal cost in the optimization problem is 0).

Clearly the quantity of interest is $v \equiv v_1$. The dynamic programming equation (DPE) can be written as

$$v_j(s) = \max_{0 \le x \le s} \left[r(x) - c(x,s) + v_{j+1}(1.2(s-x)) \right], \quad j = 1, 2, \cdots, N.$$

Since $v_{N+1} \equiv 0$, one can work backwards to solve for all v_j. At year j, the optimal amount of bass to sell x_j^* is the maximizing x in the above DPE, given that $s_j = s$ (i.e., at the beginning of year j the number of bass in the lake is s). □

Remark: A weakness of the previous formulation is that the profits received during later years are weighted the samc as profits received during earlier years. Now we consider the following *discounting factor* $0 < \beta < 1$: \$1 received in year $j+1$ is equivalent to β dollar received in year j. Then the optimization becomes (abusing the notation a bit)

$$v(s) \doteq \max_{\{x_n\}} \sum_{n=1}^{N} \beta^{n-1} \left[r(x_n) \quad c(x_n, s_n) \right], \quad \text{given } s_1 = s.$$

Similarly we define

$$v_j(s) \doteq \max_{\{x_n\}} \sum_{n=j}^{N} \beta^{n-j} \left[r(x_n) - c(x_n, s_n) \right], \quad \text{given } s_j = s.$$

v_j can be interpreted as the optimal profit from year j to year N (valued by the dollar in year j). Clearly $v_1 \equiv v$. The DPE becomes

$$v_j(s) = \max_{0 \le x \le s} \left[r(x) - c(x,s) + \beta v_{j+1}(1.2(s-x)) \right].$$

and $v_{N+1}(s) \equiv 0$. □

[redacted]

Exercise: Can you formulate an LP to solve this maximization problem?

Example: It's the last weekend of the 1996 campaign, and candidate Blaa Blaa is in NY. Before election day, he must visit Miami, Dallas, and Chicago and then return to his NY headquarters. Blaa Blaa wants to minimize the total distance he must travel. In what order he should visit the cities?

	NY	Miami	Dallas	Chicago
NY	–	1334	1559	809
Miami	1334	–	1343	1397
Dallas	1559	1343	–	921
Chicago	809	1397	921	–

Solution: It is not obvious how to work this problem into the structure of dynamic programming. The stage j is easily defined as the j-th stop of the trip, with $j = 0$ as the starting point and $j = 4$ as the ending point (both are NY in this case). The definition of the state, however, is a bit more subtle: The state is (I, S) where $I = \{\text{last city visited}\}$ and $S = \{\text{cities visited}\}$.

Let us define

$$f_j(I,S) \;=\; \text{the minimal distance that must be travelled to complete the tour if Blaa is at the } j\text{-th stop with } I \text{ being the last city visited and } S \text{ being all the } j \text{ cities visited.}$$

To ease notation, we will denote by {N, M, D, C} the four cities, and the distance between city I and city J is denoted by d_{IJ}. The DPE for this problem can be written as

$$f_j(I,S) = \min_{J \notin S} \left[d_{IJ} + f_{j+1}(J, S \cup \{J\})\right], \quad \forall\, 0 \le j \le 2.$$

with

$$f_3(I, \{\text{M,D,C}\}) = d_{IN} = \text{distance from city } I \text{ to NY}.$$

Recursively, we have

$$f_2(I,S) = \min_{J \notin S} \left[d_{IJ} + f_3(J, S \cup \{J\})\right] = \min_{J \notin S} \left[d_{IJ} + d_{JN}\right];$$

or

$$\begin{aligned}
f_2(M,\{M,D\}) &= d_{MC} + d_{CN} = 1397 + 809 = 2206;\\
f_2(D,\{M,D\}) &= d_{DC} + d_{CN} = 921 + 809 = 1730;\\
f_2(M,\{M,C\}) &= d_{MD} + d_{DN} = 1343 + 1559 = 2902;\\
f_2(C,\{M,C\}) &= d_{CD} + d_{DN} = 921 + 1559 = 2480;\\
f_2(D,\{D,C\}) &= d_{DM} + d_{MN} = 1343 + 1334 = 2677;\\
f_2(C,\{D,C\}) &= d_{CM} + d_{MN} = 1397 + 1334 = 2731;
\end{aligned}$$

and

$$f_1(I,S) = \min_{J \notin S} [d_{IJ} + f_2(J, S \cup \{J\})]$$

or

$$\begin{aligned}
f_1(M,\{M\}) &= \min_{J \in \{D,C\}} [d_{MJ} + f_2(J,\{M,J\})] \\
&= \min\{d_{MD} + f_2(D,\{M,D\}),\ d_{MC} + f_2(C,\{M,C\})\} \\
&= \min\{1343 + 1730,\ 1397 + 2480\} \\
&= 3073 \qquad \text{(with the minimum achieved at } J^* = D), \\
f_1(D,\{D\}) &= \min_{J \in \{M,C\}} [d_{DJ} + f_2(J,\{D,J\})] \\
&= \min\{d_{DM} + f_2(M,\{D,M\}),\ d_{DC} + f_2(C,\{D,C\})\} \\
&= \min\{1343 + 2206,\ 921 + 2731\} \\
&= 3549 \qquad \text{(with the minimum achieved at } J^* = M) \\
f_1(C,\{C\}) &= \min_{J \in \{M,D\}} [d_{CJ} + f_2(J,\{C,J\})] \\
&= \min\{d_{CM} + f_2(M,\{C,M\}),\ d_{CD} + f_2(D,\{C,D\})\} \\
&= \min\{1397 + 2902,\ 921 + 2677\} \\
&= 3598 \qquad \text{(with the minimum achieved at } J^* = D);
\end{aligned}$$

and finally

$$\begin{aligned}
f_0(N,\{N\}) &= \min_{J \in \{M,D,C\}} [d_{NJ} + f_1(J,\{J\})] \\
&= \min\{d_{NM} + f_1(M,\{M\}),\ d_{ND} + f_1(D,\{D\}),\ d_{NC} + f_1(C,\{C\})\} \\
&= \min\{1334 + 3073,\ 1559 + 3549,\ 809 + 3598\} \\
&= 4407 \qquad \text{(with the minimum achieved at } J^* = M \text{ or } J^* = C).
\end{aligned}$$

Therefore, the shortest tour will be either

$$N \to M \to D \to C \to N \quad \text{or} \quad N \to C \to D \to M \to N.$$

Both has total distance 4407 miles. Note these two tours are reverse to each other. □