Lecture 10: Feb 21, 2013

In this notes, we formalize the process of using dynamic programming to solve various optimization problems.

The most important thing to keep in mind when setting up dynamic programming formulation is: that the point of using dynamic programming is that it allows us to *break down* large and complex problems into (possibly a lot of) small, do-able *subproblems*.

By "breaking down" a large problem, we mean that we tackle the problem in *stages*. In each stage, we consider a number of cases, or *states*. At each state in each stage, we deal with "easy problems"; this means that the set of allowable decisions is normally simple, small, and easy to describe.

1 Steps for setting up DP formulation

The following is the summary of the steps that we have to do to solve problems using dynamic programming.

1. Specify the stages
2. Specify the states
3. Specify the sets of allowable decisions at each state in each stage
4. Describe in words the optimization function to be solved at each state in each stage
5. Specify boundary conditions
6. Write the recurrence relation for the optimization function described in step 4
7. Compute the value of the optimization function for all states in each stage
8. Trace backwards (relative to the order of computation in step 7) to find the optimal objective function and the optimal decision.

Note that steps 1 through 6 describe the setup of the formulation while steps 7 and 8 are the computation that follows. The difficult part of dynamic programming is normally the setup stage, because it often takes experience to identify the appropriate "stages" and "states". Once you do the setup correctly, computation is not difficult although there might be a large number of computation that you have to do.

Also note that although we have outlined the steps above "linearly" (i.e., first think of the stages, then think of the states, etc.), the process in steps 1 through 6 do not come sequentially. That is, we cannot come up with the stages, states, allowable decisions, and optimality functions completely separately, because they together form one formulation. However, we hope that formalizing them in the above fashion provides a systematic guidance into the tricky process of learning how to use dynamic programming.

In what follows, we will go through each of the steps in detail.

1.1 Specify the stages (k)

Think of a way to break down the problem into "several easier/smaller problems of the same type". For example, in computing the shortest-path from s to t in a layered graph, we break down the problem into shortest-path problems from s to node i in a layer that is closer to s. So, the stages corresponds to the layers in the graph.

When breaking down the problem, think about how an optimal solution of the smaller problem can help you find the solution to a slightly larger problem easily. That is, think about how the optimal solution to a subproblem in stage k can help you find the solution to a problem in stage $k-1$ or stage $k+1$.

Once you identify what "stage" should correspond to, then identify how many stages you will have to do in your computation.

1.2 Specify the states (i)

Once you break down the problem into smaller stages, think of the various cases that you have to consider within each stage. These different cases are/correspond to your states.

For example, in the shortest-path problem in a layered graph, within each layer, we need to consider each node in that layer. So, in stage k, the states are the nodes in layer k.

1.3 Specify the sets of allowable decisions at each state in each stage ($Q_{k,i}$)

Suppose that you are currently at state i in the kth stage. Here, you are faced with a smaller version of problem as your original problem.

Provided that you set up the states and stages wisely, your optimization problem in state i at stage k can be broken down into smaller subproblems, each of which corresponds to a problem in stage $k-1$ (or $k+1$, depending on how you chose your stages).

This means that the set of decisions that you need to consider are just decisions that will allow you to refer to another problem in stage $k-1$ (or $k+1$).

1.4 Describe in words the optimization function to be solved at each state in each stage ($f_k^*(i)$

We often denote the optimization function as $f_k^*(i)$, denoting the optimal value of the subproblem that we're dealing with at state i, in stage k.

For example, in our shortest-path in a layered graph problem, $f_k^*(i)$ is the length of the shortest path from s to node i, where node i is a node in the kth layer.

Note that there is some stage k and some state i such that $f_k^*(i)$ corresponds to the optimal value of the original problem.

In our shortest-path in a layered graph example (where there are 4 layers), $f_4^*(t)$ is the length of the shortest path from s to t.

1.5 Specify the boundary conditions

This step involves identifying which stage corresponds to the "smallest" subproblem.

In our shortest-path in a layered graph problem, note that the easiest case will be the shortest path from s to s, whose optimization function is $f_0^*(s) = 0$. We can also consider a slightly less-trivial subproblem as our boundary condition: $f_1^*(i) = c_{si}$, the problem in stage 1, which involves nodes i that is only one edge away from s, hence the shortest path from s to i is just the unique edge (s, i) that connects s to i.

1.6 Write the recurrence relation for $f_k^*(i)$

Here, we would like to express $f_k^*(i)$, the optimal value in stage k, in terms of optimal values in stage $k-1$ (or $k+1$): we consider each of the allowable decisions that we can take. Depending on this decision, we look at a particular state in the stage $k-1$.

For example, in the shortest path in a layered graph problem, suppose that we would like to know the length of the shortest path from s to a node i in the kth layer. We assume that we know the shortest path from s to any node in the $k-1$th layer (because it is an easier/smaller problem). Then, the only decisions/possibilities that we need to consider is what edge to take from some node j in the $k-1$th layer to go to our current node i which is in the kth layer. The number of allowable decisions is relatively small because we only consider an edge between layer $k-1$ and layer k. So, the shortest path from s to i is just the minimum among various possible nodes j in the $k-1$th layer of (the length of the shortest path from s to j + the length of the edge from j to i).

2 Solving inventory planning problems using DP

2.1 An inventory planning problem

The following is an example of a "basic" inventory planning problem. There are other inventory planning models that are more complicated and have more components, but this example should illustrate the basic ideas of how dynamic programming is used to solve inventory planning problems in general.

A company is to plan its production for the next T periods: periods $1, 2, \ldots, T$ (T could be in order of weeks, month, etc.). In each period, the company needs to meet a certain amount of demand. We denote the demand in period k as d_k. The costs that are involved are per-unit production cost, fixed production cost, and per-unit holding cost. We denote the per-unit production cost in period k as c_k, the fixed production cost in period k as F_k (this is a "set-up cost" which is charged if the company is to produce any nonzero quantity in period k), and the holding cost in period k as h_k (this is the cost for holding one unit of item that is left at the end of period k).

We assume that we have zero units in the inventory at the beginning of period 1. How many units should the company produce in each period so that its total cost is minimized?

To summarize the problem:

- Input:
  - T = number of periods in the "planning horizon"
  - For each period $k \in \{1, 2, \ldots, T\}$, there is a demand d_k, a per-unit production cost c_k, a fixed cost of production F_k, and a per-unit holding cost h_k.

- Decision to make: to determine the quantity to be produced in each period
- Constraint: demand in each period must be satisfied by the end of the period
- Objective: to minimize total cost

Example 2.1. The following is an example involving four periods.

Period (k)	d_k	c_k	F_k	h_k
1	10	3	5	0.2
2	40	2	20	0.3
3	20	4	10	0.5
4	50	3	10	0.8

Also assume that the production quantities can only be in multiples of 10.

2.2 Dynamic programming formulation

2.2.1 Specify the stages

Stage k corresponds to period k. So, there will be five stages: $1, 2, 3, 4, 5$, where the fifth stage is a dummy stage, indicating the end of the fourth period.

2.2.2 Specify the states

The state I corresponds to inventory level I at the beginning of the period.

So, at each stage k, we will consider various cases: what is our optimal cost if we begin the period with an inventory level of I? The set of all possible states is called the state space, which might be the same or different for each of the stages.

There are several ways to do this, but in this lecture note, we will choose the state space to be the same for all the stages for simplicity purposes:

$$S_k = \{0, 10, 20, \ldots, 120\}.$$

The reason we choose at most 120 is because it is the sum of all demands. It is quite obvious that we won't ever need to store more than this many units in the inventory.

During lecture, we had a different setup: $S_k = \{0, 10, \ldots, \sum_{i=k}^{n} d_i\}$. That is, we recognize that at the beginning of each stage, we will not want to have more than the sum of future demands (because if we have extras, we have to pay holding costs for them). The advantage of that formulation is that there are fewer states to consider, which ease computation.

(Either setup is equally correct!)

2.2.3 Specify the sets of allowable decisions at each state in each stage

Let x_k denote the quantity to produce at stage k. If we start with an inventory of I units at the beginning of stage k, then we want to make sure that:

- We have enough items to satisfy demand: $I + x_k \geq d_k$

- And since we assume that our inventory level is at most 120, we need only consider x_k such that

$$I + x_k - d_k \leq 120.$$

This second constraint is less crucial than the first one. The first constraint is intrinsic to the inventory problem ("we must satisfy demand!") but the second constraint is due to our choice of state spaces in step 2.

Note that if, as in what we did in lecture, we chose $S_k = \{0, 10, \ldots, \sum_{i=k}^{n} d_i\}$, then the second constraint above will be replaced by:

$$I + x_k - d_k \leq \sum_{i=k}^{n} d_i.$$

So, in summary, we formulate the set of allowable decisions at state I in stage k as:

$$Q_{k,I} = \{x \in \{0, 10, \ldots\} \mid I + x_k \geq d_k \text{ and } I + x_k - d_k \leq 120\},$$

which can be simplified as:

$$Q_{k,I} = \{x \in \{0, 10, \ldots\} \mid d_k - I \leq x_k \leq 120 - (I - d_k)\}.$$

For example:

$$\begin{aligned}
Q_{1,0} &= \{10, 20, \ldots, 130\} \\
Q_{1,10} &= \{0, 10, \ldots, 120\} \\
Q_{1,20} &= \{0, 10, \ldots, 110\} \\
&\vdots \quad \vdots \\
Q_{2,0} &= \{40, 50, \ldots, 160\} \\
Q_{2,10} &= \{30, 40, \ldots, 150\} \\
Q_{2,20} &= \{20, 30, \ldots, 140\} \\
Q_{2,30} &= \{10, 20, \ldots, 130\} \\
Q_{2,40} &= \{0, 10, \ldots, 120\} \\
Q_{2,50} &= \{0, 10, \ldots, 110\} \\
&\vdots \quad \vdots \\
Q_{3,0} &= \{20, 30, \ldots, 140\} \\
Q_{3,10} &= \{10, 20, \ldots, 130\} \\
&\vdots \quad \vdots
\end{aligned}$$

2.2.4 Describe in words the optimization function to be solved at each state in each stage

Consider a state I in stage k. That is, the inventory level at the beginning of period k is I. Then,

$$f_k^*(I) = \text{the minimum total cost to meet demands in period } k \text{ until period } T+1$$

2.2.5 Specify boundary conditions

Note that the above word-description of $f_k^*(I)$ suggests that the "easiest" subproblem is when $k = T + 1$. Indeed, this provides the boundary conditions. In our case, since $T = 4$, then stage $T + 1 = 5$ provide the boundary conditions:

$$f_5^*(I) = 0, \ \forall I \in \{0, 10, \ldots, 120\}.$$

Why is $f_5^*(I) = 0$? Suppose that we have an inventory of I at the beginning of stage 5, what is the minimum cost to meet all demands in stage 5? The answer is the cost is zero, because there is no more cost to incur: we don't need to produce more to satisfy more demand, and there is no cost incurred for "throwing away" the leftovers in the inventory. (Note that in some other variant of the inventory problem, you may have to pay a disposal fee to get rid of leftovers at the end of the planning horizon. In yet some other variant, you may be able to "sell" the remaining inventory for a small revenue, etc. In these cases, then $f_{T+1}^*(I)$ might not be zero, but is still easy to compute.)

2.2.6 Write the recurrence relation for the optimization function described in step 4

We call our equation a "recurrence relation" because we are about to express f_k^* in terms of f_{k+1}^* (there is a a recurrence of f^*)

The following is the process that we might go through while thinking about how we should come up with the recurrence relation.

- Our boundary condition indicates that the larger values of k corresponds to "easier" problems
- So, think of expressing f_k^* in terms of f_{k+1}^*
- Suppose that we are currently at state I in stage k. How can we express

  "the minimum cost to satisfy demands in periods k to $T + 1$ given that our starting inventory is I"

  in terms of

  "the minimum cost to satisfy demands in periods $k + 1$ to $T + 1$ given that our starting inventory is J"

  for some state J?
- Suppose we decide to produce x_k units in period k, then the inventory level at the beginning of period $k + 1$ is
  $$I + x_k - d_k.$$
- So, if we decide to produce x_k units in this period, then our total cost is

  cost due to producing x_k in period k + total cost in periods $k + 1$ to $T + 1$ given that the starting inventory is $I + x_k - d_k$.
- Then, the best we can do if we produce x_k units this period, is to have a total cost of

cost due to producing x_k in period k + the minimum total cost in periods $k+1$ to $T+1$ given that the starting inventory is $I + x_k - d_k$.

- So, the minimum total cost in periods k to $T+1$ is achieved when we consider all allowable values of x_k and taking the minimum:

$$\min_{x_k \in Q_{k,I}} \left\{ \text{cost due to producing } x_k \text{ in period } k + f^*_{k+1}(I + x_k - d_k) \right\},$$

Hence,

$$f^*_k(I) = \min_{x_k \in Q_{k,I}} \left\{ \underbrace{(x_k * c_k + \mathbb{1}_{x_k>0} F_k + (I + x_k - d_k) * h_k)}_{\text{cost due to producing } x_k \text{ in period } k} + f^*_{k+1}(I + x_k - d_k) \right\}.$$

2.2.7 Compute the value of $f^*_k(I)$ for each state $I \in S_k$, for each stage k

To compute $f^*_k(I)$, we only need to use the recurrence relation we described in step 6. In order to do the computation, since each f^*_k refer to a value f^*_{k+1}, we do the computation starting from $k = T+1$ (the boundary conditions), then $k = T, T-1, \ldots, 1$.

Within each stage k, we solve for $f^*_k(I)$ for all states $I \in S_k$, before proceeding to stage $k-1$. That is, we complete the following table starting from each row in the leftmost column, proceeding to the next columns.

The completed table is as follows: