Dynamic Programming

The general idea

Dynamic programming is essentially a recursive algorithm, which is often useful if the problem can be represented as a parametrized set of subproblems, such that each subproblem is reducible to others with smaller parameters.

The formulation can yield an efficient algorithm if the following two conditions are satisfied:

- The number of different subproblems is not too large (does not grow exponentially).
- The iterative reductions finally reduce the task to some initial cases for which the solution is already known or directly computable.

We show the principle through an example.

Example

A telecom company wants to create an extension plan for its network, year by year over a time horizon of T years, such that the network will have enough capacity to keep up with the growing demand in every year, and the total cost is minimum. The following information is available as input:

- $cost[t,k]$ is the known cost we have to pay if in year t we extend the capacity by k units (k is assumed to be an integer).
- $demand[t]$ is the known capacity demand in year t.

Goal: find an extension plan by specifying how much new capacity has to be bought in each year, such that in every year the accumulated capacity satisfies the demand, and the total cost is minimum.

Solution

For easy description, let us use the following notations:

- k will denote the amount of capacity.
- The time (year) will be denoted by t. This variable runs over a time horizon T, that is $t = 1, \ldots, T$.
- $cost[t, k]$ denotes the known cost we have to pay if in year t we buy k units of capacity.
- $demand[t]$ denotes the known demand for year t.
- Let $A[t, k]$ be the minimum achievable accumulated cost of a decision sequence up to the end of year t, if it achieves a total capacity of k.

Let us now formulate the recursive algorithm:

The optimal accumulated cost up to the end of year t with accumulated capacity k can be obtained recursively as follows.

If the last amount of capacity we have bought is r and now we have altogether k, then by the end of the previous year we must have had $k-r$. If we already know the optimum cost up to year $t-1$ for all possible values of the capacity, then we can express the updated total cost as

$$A[t, k] = A[t-1, k-r] + cost[t, r].$$

Since r is a free parameter, the updated optimum will be the minimum with respect to r:

$$A[t,k] = \min_r A[t-1, k-r] + cost[t,r]$$

where the minimum is taken over all possible values of $r \leq k$ that corresponds to the possible amounts of capacity we can buy.

Of course, k capacity in year t is acceptable only if $k \geq demand[t]$, that is, the accumulated capacity satisfies the demand. To exclude those values that violate this requirement, we can prevent using them by assigning infinite cost (or some very large number) to these cases. Thus, the final recursice formula will be

$$A[t,k] = \begin{cases} \infty & \text{if } k < demand[t] \\ \min_r A[t-1,k-r] + cost[t,r] & \text{if } k \geq demand[t] \end{cases}$$

We can apply the above recursive formula whenever $t > 1$. But how do we handle the first year, that is, how do we start the recursion? When $t = 1$, there is no previosly accumulated capacity yet, so then we simply take $A[1,k] = cost[1,k]$ for every k.

In this way the $A[t,k]$ values can be computed recursively for $t = 1, \ldots, T$.

Having done the iteration, the optimum cost will be

$$OPTCOST = \min_{k \geq demand[T]} A[T,k]$$

Comment:

This systematic solution solves the problem in time bounded by a polynomial of the time horizon length (T) and the maximum value of k (capacity) as opposed to the full decision tree where the running time would grow exponentially.