CSE101 - Discussion Section 2/18/98
Dynamic Programming: Resource Allocation

1 Problem Description

n units of resource are to be allocated to to r Projects. If j units of the resource are allocated to project i, the resulting profit is $N(i,j)$. The problem is to allocate the resource to the r projects in such a way as to maximize total profit. We assume in this case that $N(i,j_1) \leq N(i,j_2)$ if $j_1 < j_2$, i.e. if we allocate more resources to a project, we do not get less profit (*). This seems to make sense. Anyway, it does not change the resulting algorithm very much.

2 Problem Solving

2.1 Characterization of the structure of an optimal solution

In order to allocate the resources, we have to make $r-1$ decisions: 1. "How many resources for project 1?", 2. "How many for project 2?" and so on. The rth project is assigned the remaining resources ($\rightarrow$(*)). Therefore the problem may be formulated as an $r+1$ stage graph problem as shown in figure 1 (for $r=3$ and $n=4$). Each path from $(1,0)$ to $(4,4)$ represents a distribution of resources among the projects. Following a path to a point (i,j) can be interpreted as: j resources are already allocated and I am allocating resources for project i next. Now we are able to reformulate the above problem as follows: Find a path from (0,1) to (4,4) such that the sum of the values $N(i,j)$ assigned to the edges that we take is maximal.

2.2 Recursive definition of the value of an optimal solution

Looking at the sample $r=3$ and $n=4$ in figure 1, we can see that an optimal solution consists of one step from $(1,0)$ to $(2,j)$, and the optimal path from $(2,j)$ to $(4,4)$. Taht means, once we have arrived at $(2,j)$, we have the problem to allocate the remaining $n-j$ resources to the remaining projects $2,3$.

This can be generalized: If we are at the point (i,j), we have to allocate the remaining $n-j$ resources to the remaining $r-i+1$ projects $i, i+1, ..., r$ in an optimal way. Now it is easy to see that the principle of optimality applies to this problem: every solution contains solutions to subinstances of the problem. Let's define $P[i,j]$ to be the value of an optimal solution of the "(i,j)th" subinstance mentioned above, i.e. the maximal profit we can get from project $i, i+1, ..., r$ if we have $n-j$ units of resource to allocate to them. Then it is

$$P[i,j] = \max_{l=0}^{n-j}\{N[i,l] + P[i+1, l+j]\}$$

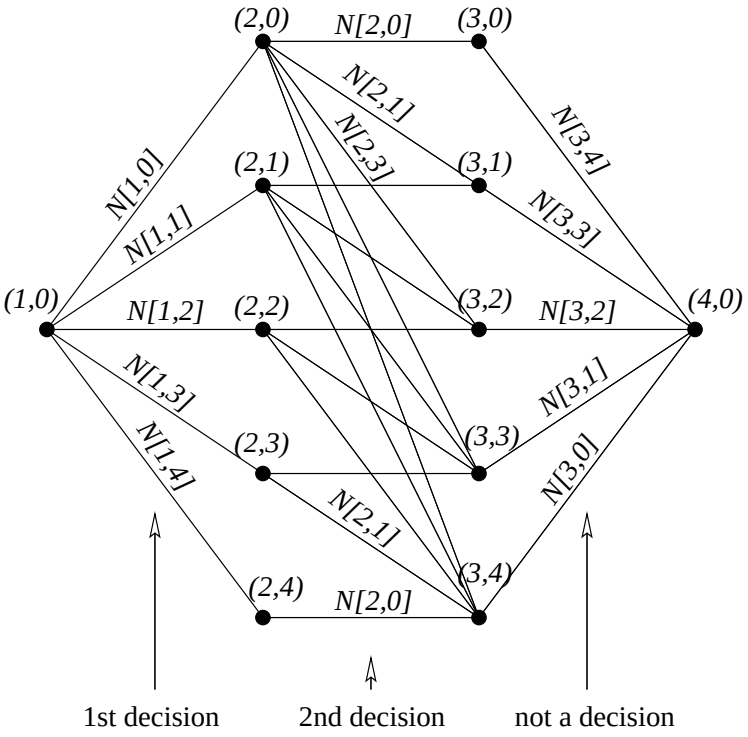

Figure 1: Multistage Graph

2.3 Computation of the value of the optimal solution "bottom-up"

The first question is: What is the base case? Looking at the recursion, one can see that we can compute the value of $P[i,j]$ from a known value $N[i,l]$ and $P[i+1,l+j]$. Therefore, we have to compute the $i+1$th column of the matrix P before we can compute the ith column. Thus, the base case is to compute $P[r,j]$, $j=0,...,n$. This is easy, because we can't make decisions anymore at that stage (see the example in figure 1). Here is the algorithm that fills in the array P:

1. for $j := n$ downto 0
2. { $P[r,j] = N[r,n-j]$ } //fill in the base case
3. for $i = r-1$ downto 2
4. { for $j = n$ downto 0 do
5. { $P[i,j] = \max_{l=0}^{n-j}\{N[i,l] + P[i+1,l+j]\}$ }}
6. $P[1,0] = \max_{l=0}^{n}\{N[1,l] + P[2,l]\}$