IE170: Algorithms in Systems Engineering: Lecture 11

Jeff Linderoth

Department of Industrial and Systems Engineering
Lehigh University

February 9, 2007

Taking Stock

Last Time

- Easiest Quiz Ever

This Time

- Intro to Dynamic Programming

Dynamic Programming

- Not really an algorithm but a technique.
- Not really "programming" like Java programming

Dynamic Programming in a Nutshell

1. Characterize the structure of an optimal solution
2. Recursively define the value of an optimal solution
3. Compute the value of an optimal solution "from the bottom up"
4. Construct optimal solution (if required)

Capital Budgeting

- A company has $5 million to allocate to its three plants for possible expansion.
- Each plant has submitted different proposals on how it intends to spend the money.
- Each proposal gives the cost of the expansion c and the total revenue expected r.

Investment Possibilities

	Plant 1		Plant 2		Plant 3	
Proposal	c_1	r_1	c_2	r_2	c_3	r_3
1	0	0	0	0	0	0
2	1	5	2	8	1	4
3	2	6	3	9	-	-
4	-	-	4	12	-	-

More Setup

- Each plant will only be permitted to enact one of its proposals.
- The goal is to maximize the firm's revenues resulting from the allocation of the $5 million.
- Assume that any of the $5 million we don't spend is lost

Solve It!

How would you solve this problem?

Solution Methods

- One way—Enumeration: only $2 \times 3 \times 4 = 24$ possibilities, and many of these don't obey the budget constraint
- This doesn't scale well.
- Let's think of another way:

Building a Solution

- Let's break the problem into three stages: each stage represents the money allocated to a single plant.
- Each stage is divided into states. A state encompasses the information required to go from one stage to the next. In this case the states for stages 1, 2, and 3 are
    1. $\{0, 1, 2, 3, 4, 5\}$: the amount of money spent on plant 1, represented as x_1
    2. $\{0, 1, 2, 3, 4, 5\}$: the amount of money spent on plants 1 and 2, represented as x_2
    3. $\{5\}$, the amount of money spent on plants 1, 2, and 3 (x_3)

States

- Associated with each state is a revenue.
- Note that to make a decision at stage 3, it is only necessary to know how much was spent on plants 1 and 2, not how it was spent.
- Also notice that we will want $x_3 = 5$
- Let's calculate the revenues associated with each state.
- This is easy for stage 1:

Available capital x_1	Optimal Proposal	Revenue
0	1	0
1	2	5
2	3	6
3	3	6
4	3	6
5	3	6

Stage 2

- In this case, we want to find the best solution for both plants 1 and 2. Just try all proposals. For example: if $x_2 = 4$, we could do
  1. Proposal 1, revenue 0, leaves 4 for stage 1, revenue 6, total 6
  2. Proposal 2, revenue 8, leaves 2 for stage 1, revenue 6, total 14
  3. Proposal 3, revenue 9, leaves 1 for stage 1, revenue 5, total 14
  4. Proposal 4, revenue 12, leaves 0 for stage 1, revenue 0, total 12

Stage 2—All Optimal Policies

Available capital x_2	Optimal Proposal	Revenue for 1 and 2
0	1	0
1	1	5
2	2	8
3	2	13
4	2 or 3	14
5	4	17

Stage 3

- We only care about $x_3 = 5$.
  1. Proposal 1, revenue 0, leaves 5 for previous stage, revenue 17, total 17
  2. Proposal 2, revenue 4, leaves 4 for previous stage, revenue 14, total 18

Optimal Solution

Proposal 2 at plant 3, Proposal 2 or 3 at plant 2, and proposal 3 or 2 (resp.) at plant 1

Not Recursion Again!?!?!?!

- All calculations are done recursively:
- Stage 2 calculations are based on stage 1, stage 3 only on stage 2.
- If you are at a state, all future decisions are made independent of how you got to the state
- This is the principle of optimality, and all of dynamic programming rests on this assumption.

- $r(k_j), c(k_j)$: Revenue and cost for proposal k_j at stage j
- $f_j(x_j)$: Revenue of state j in stage j
- The following two equations hold:

$$\begin{aligned} f_1(x_1) &= \max_{\{k_1 \mid c(k_1) \le x_1\}} \{r(k_1)\} \\ f_j(x_j) &= \max_{\{k_j \mid c(k_j) \le x_j\}} \{r(k_j) + f_{j-1}(x_j - c(k_j))\} \quad j = 2, 3 \end{aligned}$$

Another Way

- y_1: amount allocated to stages 1, 2, and 3,
- y_2: amount allocated to stages 2 and 3,
- y_3: amount allocated to stage 3

$$f_3(y_3) = \max_{\{k_3 \mid c(k_3) \leq y_3\}} \{r(k_3)\}$$
$$f_j(y_j) = \max_{\{k_3 \mid c(k_j) \leq y_j\}} f_{j+1}(y_j - c(k_j))\}$$

- Sometimes backwards recursion is faster
- Sometimes forward recursion is faster
- Sometimes it doesn't matter

DP for Assembly Line Scheduling

- You have been hired to optimize the Yugo Factory in Prattville, AL
- There are two assembly lines. Each line has n different stations:

$$S_{11}, S_{12}, \ldots, S_{1n} \text{ and } S_{21}, S_{22}, \ldots, S_{2n}.$$

- Stations S_{1j} and S_{2j} perform the same function, put take a different amount of time: (a_{1j} and a_{2j})
- Once a Yugo is processed at station S_{ij}, it can either
  1. Stay on the same line (i) with no time penalty
  2. Transfer to the other line, but is then delayed by t_{ij}

Your Mission

Problem:
Given this setup, what stations should be chosen from each line in order to minimize the time that a car is in the factory?

- Note: We can't (efficiently) just check all possibilities?
- How many are there?

A Better Way

- A better way to find an optimal solution is to think about what properties an optimal solution must have.

Question

- What is the fasest way to get through station S_{1j}?
- If $j = 1 : a_{11}$
- If $j \geq 2$, then we have two choice for how to get through S_{1j}
  - Through $S_{1,j-1}$ then to S_{1j}
  - Through $S_{2,j-1}$ then to S_{1j}

Key Obervation

- Suppose fastest way through S_{1j} is through $S_{1,j-1}$
- We must have taken a fastest way to get through $S_{1,j-1}$ in this fastest solution through S_{1j}.
- If there was a faster way through $S_{1,j-1}$, we could have used it instead to get through S_{1j} faster.
- Likewise, suppose the fastest way through S_{1j} is from $S_{2,j-1}$. We must have used a fastest way through $S_{2,j-1}$

Optimal Substructure

An optimal solution to the problem (The fastest way through S_{1j}) contains within it an optimal solution to subproblems: either the fastest way through $S_{1,j-1}$ or $S_{2,j-1}$

Optimal Substructure

- Fastest way through S_{1j} is either (fastest of)
  - fastest way through $S_{1,j-1}$ then directly through S_{1j}
  - fastest way through $S_{2,j-1}$, transfer lines, then through S_{1j}
- Fastest way through S_{2j} is either (fastest of)
  - fastest way through $S_{2,j-1}$ then directly through S_{2j}
  - fastest way through $S_{1,j-1}$, transfer lines, then through S_{2j}

A Recursive Solution

- Suppose that we have entry times e_i and exit times x_i
- Let $f_i(j)$ be the fastest time to get through $S_{ij}\ \forall i = 1,2\ \forall j = 1,2,\ldots n$

A DP for the Optimal Solution Value

$$\begin{aligned}
f^* &= \min(f_1(n) + x_1, f_2(n) + x_2) \\
f_1(1) &= e_1 + a_{11} \\
f_2(1) &= e_2 + a_{21} \\
f_1(j) &= \min(f_1(j-1) + a_{1j}, f_2(j-1) + t_{2,j-1} + a_{1j}) \\
f_2(j) &= \min(f_2(j-1) + a_{2j}, f_1(j-1) + t_{1,j-1} + a_{2j})
\end{aligned}$$

Analyze the recursion

- Let's compute how many times we reference/compute $f_i(j) : r_i(j)$
- $r_1(n) = r_2(n) = 1$
- $r_1(j) = r_2(j) = r_1(j+1) + r_2(j_1)$ for $j = 1, \ldots n-1$
- Problem 15.1.2 – Show that $r_i(j) = 2^{n-j}$

Bottom's Up

- The number of references to $f_i(j)$ is so large only because we compute f^* in a top down fashion
- Really $f_i(j)$ only depends on times from its immediate predecessor stations $f_1(j-1)$ and $f_2(j-1)$
- In this case, we should compute $f_i(j)$ in increasing order of j
- It essentially amounts to "building a table" of the value functions $f_i(j)$ for each $i = 1, 2$ and $j = 1, 2, \ldots n$
- If you want to know the optimal solution, you need to "keep track" as you go.
- $\ell_i(j)$: Line number whose $j-1$ station was used to find the fastest way through i