

# IE170: Algorithms in Systems Engineering: Lecture 13

Jeff Linderoth

Department of Industrial and Systems Engineering  
Lehigh University

February 21, 2007



## Taking Stock

### Last Time

- Assembly Line Balancing
- (Intro to) Lot Sizing
- Knapsack Problem

### This Time: DP + Ilya's Favorite Algorithm

- Lot Sizing
- Greedy Algorithm



## Uncapacitated Lot Sizing

- Lot sizing is **the** canonical production planning problem
- Given a planning horizon  $\mathcal{T} = \{1, 2, \dots, T\}$
- You must meet given demands  $d_t$  for  $t \in \mathcal{T}$
- You can meet the demand from a combination of production ( $x_t$ ) and inventory ( $s_{t-1}$ )
- Production cost:

$$c(x_t) = \begin{cases} K + cx_t & \text{if } x_t > 0 \\ 0 & \text{if } x_t = 0 \end{cases}$$

- Inventory cost:  $I(s_t) = h_t s_t$



## Let's Solve it with DP

- What should our stages be?
- Hint: Typically stages have type "from beginning until now" (like  $S_{ij}$ ) or from "now until end" (like in capital budgeting)

### Stage

Let  $f_t(s)$ : be the minimum cost of meeting demands from  $t, t + 1, \dots, T$  if  $s$  units are in inventory at the beginning of period  $t$



## Let's Solve an Example

- $T = 3$
- $d = [2, 1, 2]$
- $h = [1, 1, 0]$
- $K = 2, c = 1$

### Busy Going Backwards

- $f_3(0) = 2 + 2(1) = 4$
- $f_3(1) = 2 + 1(1) = 3$
- $f_3(2) = 0$



## In General

### A General Recursive Relationship

$$f_t(s) = \min_{x \in \{0, 1, 2, \dots\}} \{c_t(x) + h_t(s + x - d_t) + f_{t+1}(s + x - d_t)\}.$$

- Let's do a couple by hand.
- This gets tedious – so let's code it up...



## Oh Dear!

- What if  $K = 250, d = [220, 280, 360, 140, 270], c_t = 2, h_t = 1$
- This might be a problem, as you need to consider producing **every** possible amount between 0 and 1270
- Instead, as is often the case in dynamic programming, we look for **structural properties** of an optimal solution that will make the algorithm more efficient.



## I Love Lemmas

### Lemma (Fact) 1

Let  $x^*$  be an optimal policy (production schedule). If  $x_t^* > 0$ , then  $x_t^* = \sum_{j=0}^{T-t} d_{t+j}$  for some  $j \in \{0, 1, \dots, T-t\}$

### Why? Oh Why?

If Lemma 1 was false, then there would be some period  $t$  and some subsequent period  $t + j$  such that production  $x_t^*$  only partially satisfied the demand in  $t + j$ . Say this is a quantity  $0 < p < d_{t+j}$ . If you produce  $p$  less at  $t$ , you still meet demands up to  $j - 1$ , save holding costs, and incur no additional setup cost (since production was going to have to happen in  $j$  anyway). Thus,  $x_t^*$  couldn't have been optimal



## Mmmmmmmmm. More Lemmas.

### Lemma (Factoid) 2

Let  $x^*$  be an optimal policy (production schedule). If  $x_t^* > 0$  then  $s_{t-1} < d_t$ .

### Why? Oh Why?

It's a similar argument. If Lemma 2 was false, then there is some  $t$  such that  $x_t^* > 0$  and  $s_{t-1} \geq d_t$ . If you defer production by one period, you will save holding costs, and incur no additional charges, so  $x_t^*$  couldn't be optimal.



## How Does This Help?

- For simplicity, assume that  $s_0 = 0$  (we can fix this up later...)
- These results *really* helps us cut down on the size of the state space. In fact, we need only (recursively) compute the minimum cost during periods  $t, t + 1, \dots, T$  as

$$f_t(0) = \min_{j \in \{0, 1, \dots, T-t\}} \{(c_{tj} + f_{t+k+1}(0))\}$$

- Where  $c_{tj}$  is the cost incurred for periods  $t, t + 1, \dots, t + j$  if production during  $t$  *exactly* meets demands for  $t, t + 1, \dots, t + j$ :

$$c_{tj} = K + c \left( \sum_{k=0}^j d_{t+k} \right) + h \left( \sum_{k=1}^j k d_{t+k} \right).$$



## Another OR Application

- We have a set  $\mathcal{A} = \{1, 2, \dots, n\}$  of activities that require exclusive use of a common resource.
  - Could be a machine or a classroom, for example
- Activity  $i \in \mathcal{A}$  has "start time"  $s_i$  and finish time  $f_i$

### Activity Selection Problem

Select the **largest set** of nonoverlapping (mutually compatible) activities



## More on Activity Selection

- Let  $S_{ij} \subseteq \mathcal{A}$  be the set of activities that start **after** activity  $i$  needs to finish and **before** activity  $j$  needs to start:

$$S_{ij} \stackrel{\text{def}}{=} \{k \in S \mid f_i \leq s_k, f_k \leq s_j\}$$

- Let's assume that we have sorted the activities such that

$$f_1 \leq f_2 \leq \dots \leq f_n$$

- Then:  $i \geq j \Rightarrow S_{ij} = \emptyset$ 
  - **Proof:**
- Our goal is to optimally schedule all jobs in  $S_{ij}$
- Then, if we add two "dummy activities" ( $s_0 = -\infty, f_0 = 0$ ), ( $s_{n+1} = \infty, f_{n+1} = \infty$ ), we need to optimally schedule jobs in  $S_{0,n+1}$



## Building up a Solution

- What does an optimal solution to problem on activities  $S_{ij}$  look like?
- Let  $A_{ij} \subseteq S_{ij}$  be an optimal set of activities for  $S_{ij}$
- We know that  $|A_{ij}| \geq 1$  as long as  $S_{ij} \neq \emptyset$
- Suppose  $k \in A_{ij}$ . That is, suppose job  $k$  is in an optimal solution to  $S_{ij}$ . This decomposes the problem into an optimal solution **before**  $k$  and an optimal solution **after**  $k$ .
- Specifically, we have

$$A_{ij} = A_{ik} \cup \{k\} \cup A_{kj}$$



## Building a Recursion

- From this, we can write a recursive solution. Let  $c_{ij}$  be the size of a maximum-sized subset of mutually compatible jobs in  $S_{ij}$ .
- If  $S_{ij} = \emptyset$ , then  $c_{ij} = 0$
- If  $S_{ij} \neq \emptyset$ , then  $c_{ij} = c_{ik} + 1 + c_{kj}$  for some  $k \in S_{ij}$ . We pick the  $k \in S_{ij}$  that maximizes the number of jobs:

$$c_{ij} = \begin{cases} 0 & \text{if } S_{ij} = \emptyset \\ \max_{k \in S_{ij}} c_{ik} + c_{kj} + 1 & \text{if } S_{ij} \neq \emptyset \end{cases}$$

- Note we need only check  $i < k < j$



## We Can Make It Easy

### Solution Theorem

Let  $S_{ij} \neq \emptyset$  and let  $m$  be the activity with the **earliest** finish time in  $S_{ij}$ :

$$m \in \arg \min_{k \in S_{ij}} \{f_k\},$$

then

- 1 Activity  $m$  is used in **some** optimal solution (maximum size compatible subset) of  $S_{ij}$
- 2  $S_{im} = \emptyset$

**Proof:**



## Theorems Are Great!

- Characterizing the optimal solution in this manner makes our algorithmic lives much, much easier.

	Before Theorem	After Theorem
# subproblems in recursion	2	1
# choices in recursion	$j - i - 1$	1

### To Solve $S_{ij}$

- 1 Choose  $m \in S_{ij}$  with the earliest finish time. **The Greedy Choice**
- 2 Then solve problem on jobs  $S_{mj}$



## When Greedy?

### How did we show that greedy works?

- 1 Determine optimal substructure of problem
- 2 Develop a recursive solution
- 3 **Prove** that at every stage of recursion, one of the optimal choices is a greedy choice.
- 4 Show that all but one of the subproblems induced by the greedy choice are empty



## Properties of Greedy

### Optimal Substructure

This is just the same as dynamic programming. An optimal solution contains within it optimal solutions to smaller problems.

### Greedy Choice Property

When we are considering which choice to make, we make the solution that looks best to us now—without considering the impact on subsequent problems



## Dynamic Versus Greedy

- DP and Greedy: Make a choice at each stage.
- DP: The choice **depends** on knowing the optimal solution to smaller problems. Thus, we have to solve from the “bottom up”. Get the solution to *all* smaller problems first in order to arrive at the solution to the bigger problem.
- Greedy: The choice can be made **before** solving the subproblems.



## Next Time

- Intro to Graphs
- More Homework Due Monday:
  - Problem 16.2-1 (Show that fractional knapsack has greedy choice property)
  - Problem 16-1 (a), (c), and (d) (Making change)

