IE170: Algorithms in Systems Engineering: Lecture 12

Jeff Linderoth

Department of Industrial and Systems Engineering
Lehigh University

February 12, 2007

Taking Stock

Last Time

- Intro to Dynamic Programming: Capital Budgeting

This Time: More DP

- Assembly Line Balancing
- Lot Sizing

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)

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

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

Capital Budgeting
Assembly Line Balancing

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 $r_i(j) = 2^{n-j}$

Jeff Linderoth IE170:Lecture 12

Capital Budgeting
Assembly Line Balancing

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 $j = 1, 2, \ldots n$
- This "keep track instead of recomputing" is sometimes called memoization

Jeff Linderoth IE170:Lecture 12

Capital Budgeting
Assembly Line Balancing

Knowing the Solution

- If you want to know the optimal solution, you also 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
- Let's do our example...

Jeff Linderoth IE170:Lecture 12

What Makes a Dynamic Program?

1. The problem can be divided into stages with a decision required at each stage.
    - In the capital budgeting problem the stages were the allocations to a single plant. The decision was how much to spend.
    - In the assembly-line balance problem, the stages were the stations, and the decision was which line to go to next
2. Each stage has a number of states associated with it.
    - The states for the capital budgeting problem corresponded to the amount spent at that point in time. (Or equivelently, how much money was remaining)
    - The state in the assembly-line balance problem was the line the car currently was on.

What Makes a Dynamic Program? (cont.)

1. The decision at one stage transforms one state into a state in the next stage.
    - In capital budgeting: the decision of how much to spend gave a total amount spent for the next stage.
    - In Assembly line balance:The decision of where to go next defined where you arrived in the next stage.
2. Given the current state, the optimal decision for each of the remaining states does not depend on the previous states or decisions.
    - In the budgeting problem, it is not necessary to know how the money was spent in previous stages, only how much was spent.
    - In the assembly line problem, it was not necessary to know how you got to a node, only that you did.

What Makes a Dynamic Program? (cont.)

1. There exists a recursive relationship that identifies the optimal decision for stage j, given that stage j+1 has already been solved.
    - These were the recursions we wrote for each problem

What's the Hard Part!?

The big skill in dynamic programming, and the art involved, is to take a problem and determine stages and states so that all of the above hold. (You will be asked to think about this a bit in lab today). If you can, then the recursive relationship makes finding the values relatively easy.

Uncapacitated Lot Sizing

- Lot sizing is the canonical production planning problem
- Given a planning horizon $\mathcal{T} = \{1, 2, \ldots, T\}$
- 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 $t, t+1, \ldots T$ 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

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

- 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 $j \in \{0, 1, \ldots T - t\}$

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

Mmmmmmmmmm. 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 $t, t + 1, \ldots T$ as

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

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

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

Happy Days!

- No Class on Friday 2/16 or Monday 2/19
- Today's lab and homework due on 2/26