

Dynamic Programming 2

Chapter 7, Algorithm Design

Hantao Zhang

<http://www.cs.uiowa.edu/~hzhang/c31>

Properties of a typical problem that can be solved with dynamic programming

- ◆ **Simple Subproblems**
 - » We should be able to break the original problem to smaller subproblems that have the same structure
- ◆ **Optimal Substructure of the problems**
 - » The solution to the problem must be a composition of subproblem solutions
- ◆ **Subproblem Overlap**
 - » Optimal subproblems to unrelated problems can contain subproblems in common

dynprog - 2

Segmented Least Squares

- ◆ Given a list P of n points in the plane, $P = [p_1, p_2, \dots, p_n]$, where $p_i = (x_i, y_i)$ and $x_1 < x_2 < \dots < x_n$
- ◆ Given a line L defined by $y = ax + b$, the error of L with respect to P is
$$\text{Error}(L, P) = \sum_{k=1..n} (y_k - ax_k - b)^2$$
- ◆ How to find L to minimize $\text{Error}(L, P)$?

- ◆ There is a solution from Calculus:
 - » $a = (n\sum_i x_i y_i - (\sum_i x_i)(\sum_i y_i)) / (n\sum_i x_i^2 - (\sum_i x_i)^2)$
 - » $b = (\sum_i y_i - a\sum_i x_i) / n$

dynprog - 3

How Many Lines?

- ◆ One line may produce too much errors.
- ◆ $n - 1$ lines for n points have no errors, but too many lines.
- ◆ **Problem Formulation:** Given $P = [p_1, p_2, \dots, p_n]$, where $p_i = (x_i, y_i)$ and $x_1 < x_2 < \dots < x_n$, divide n points into m segments P_1, P_2, \dots, P_m , where $P = P_1 P_2 \dots P_m$, such that each segment P_i is approximated by a line L_i so that the value $Cm + \sum_i \text{Error}(L_i, P_i)$ is minimal, where C is a constant.

dynprog - 4

A Recursive Solution

- ◆ Let $\text{OPT}(j)$ be the optimal solution for the first j points in P . Then $\text{OPT}(n)$ is the solution.
- ◆ Let $P' = [p_i, p_{i+1}, \dots, p_j]$, and $e_{i,j} = \text{Error}(L', P')$.
- ◆ Then $\text{OPT}(j) = \min_{1 \leq i < j} (e_{i,j} + C + \text{OPT}(i - 1))$
- ◆ In particular,
$$\text{OPT}(n) = \min_{1 \leq i < n} (e_{i,n} + C + \text{OPT}(i - 1))$$

dynprog - 5

The Algorithm

- ◆ Let $P' = [p_i, p_{i+1}, \dots, p_j]$, and $e_{i,j} = \text{Error}(L', P')$.
- ◆ $\text{OPT}(j) = \min_{1 \leq i < j} (e_{i,j} + C + \text{OPT}(i - 1))$

Segmented-Least-Square(n)

	<u>Cost</u>
array $M[0..n]$	
$M[0] := 0$	$O(1)$
For all pairs $i < j$, compute $e_{i,j}$	$O(n^3)$
For $j = 1, 2, \dots, n$	
$M[j] = \min_{1 \leq i < j} (e_{i,j} + C + M[i - 1])$	$O(n^2)$
Return $M[n]$	$O(1)$

dynprog - 6

The Algorithm

- Let $P' = [p_i, p_{i+1}, \dots, p_j]$, and $e_{i,j} = \text{Error}(L', P')$.
- $\text{OPT}(j) = \min_{1 \leq i < j} (e_{i,j} + C + \text{OPT}(i - 1))$

Find-Segments(j)

If (j = 0) output nothing

Else

Find i that minimizes $(e_{i,j} + C + M[i - 1])$

Output $[p_i, p_{i+1}, \dots, p_j]$ and the result of

Find-Segments(i-1)

dynprog - 7

Subset Sum Problem

- Formulation 1:** Given n items $\{ 1, \dots, n \}$, and each has a given nonnegative weight w_i , and weight bound W. We want to select a subset S of the items so that $\sum_{i \in S} w_i \leq W$ and $\sum_{i \in S} w_i$ is maximal.

» A special case of Knapsack Problem, when $v_i = w_i$.

- Formulation 2:** Given n integers $\{ a_1, \dots, a_n \}$, and an integer t. We want to select a subset S of the integers so that $\sum_{i \in S} a_i = t$.

dynprog - 8

Subset Sum Problem

- Formulation 2:** Given n integers $\{ a_1, \dots, a_n \}$, and an integer t. We want to select a subset S of the integers so that $\sum_{i \in S} a_i = t$.
- Let $\text{OK}(j, t) = \text{true}$ iff the first j integers have a subset which sums up to t.
- $\text{OK}(j, 0) = \text{true}$; $\text{OK}(0, t) = \text{false}$ for $t > 0$
- $\text{OK}(j, t) = \text{OK}(j-1, t)$ if $a_j > t$
- $\text{OK}(j, t) = \text{true}$ if $a_j = t$
- $\text{OK}(j, t) = \text{OK}(j-1, t) \vee \text{OK}(j-1, t-a_j)$ if $a_j < t$

dynprog - 9

Subset Sum Problem

- Let $OK(j, t)$ = true iff the first j integers have a subset which sums up to t .
- $OK(j, 0) = \text{true}$; $OK(0, t) = \text{false}$ for $t > 0$
- $OK(j, t) = OK(j-1, t)$ if $a_j > t$
- $OK(j, t) = OK(j-1, t) \vee OK(j-1, t-a_j)$ if $a_j \leq t$

Subset-Sum(n, t)

Array $M[0..n, 0..t]$

Initialize $M[j, 0] = \text{true}$ for any j
and $M[0, i] = \text{false}$ for $i > 0$

For $j := 1, 2, \dots, n$

For $i := 1, 2, \dots, t$

if $a_j > t$ $M[j, i] := M[j-1, i]$

else $M[j, i] = M[j-1, i] \vee M[j-1, i-a_j]$

Return $M[n, t]$

dynprog - 10

Summary: Dynamic programming

- DP is a method for solving certain kind of problems
- DP can be applied when the solution of a problem includes solutions to subproblems
- We need to find a recursive formula for the solution
- We can recursively solve subproblems, starting from the trivial case, and save their solutions in memory
- In the end we'll get the solution of the whole problem

dynprog - 11
