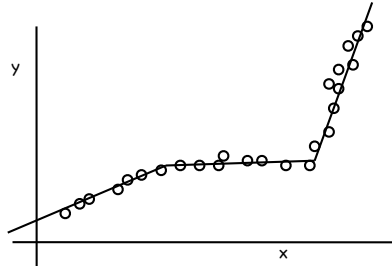


Segmented Least Squares

- Points lie roughly on a sequence of several line segments.
- Given n points in the plane $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$ with $x_1 < x_2 < \dots < x_n$, find a sequence of lines that fits well.



1

Multiway Choice

- $OPT(j)$ = minimum cost for points p_1, p_2, \dots, p_j .
- $e(i, j)$ = minimum error segment for points p_i, p_{i+1}, \dots, p_j .
- To compute $OPT(j)$:
 - Last segment uses points p_i, p_{i+1}, \dots, p_j for some i .
 - Cost = $e(i, j) + c + OPT(i-1)$.

$$OPT(j) = \begin{cases} 0 & \text{if } j=0 \\ \min_{1 \leq i \leq j} \{ e(i, j) + c + OPT(i-1) \} & \text{otherwise} \end{cases}$$

2

Segmented Least Squares: Algorithm

```
Segmented-Least-Squares() { Cost
  for j = 1 to n
    for i = 1 to j
      compute the least square error  $e_{ij}$  for
      the segment  $p_i, \dots, p_j$   $O(n^3)$ 

  M[0] = 0
  for j = 1 to n
    M[j] =  $\min_{1 \leq i \leq j} (e_{ij} + c + M[i-1])$   $O(n^2)$ 

  return M[n]
}  $\text{Total} = O(n^3)$ 
```

3

The Price is Right!

(or shopping with somebody else's money)

- ⦿ Spend as much money as possible without going over \$100.

CD	\$18	Jeans	\$40
DVD	\$35	Dinner	\$15
Book	\$8	Ice cream	\$5
Shoes	\$61	Pizza	\$7

4

Greedy Solution?

- ◉ Sort by decreasing cost:
Shoes \$61, DVD \$35
Total: \$96
- ◉ Does it always work well?
Counterexample: \$51, \$50, \$50

5

Greedy Solution 2?

- ◉ Sort by increasing cost:
Ice cream \$5, Pizza \$7, Book \$8, Dinner \$15,
CD \$18, DVD \$35
Total \$88
- ◉ Less good than first attempt

6

Dynamic Programming Solution

- ◉ Define $OPT(j)$ to be the most we can spend considering just the first j items.
- ◉ Case analysis:
 - ◉ $OPT(j)$ does not include a purchase of item j
 - ◉ $OPT(j)$ does include a purchase of item j

7

Defining OPT

- ◉ If $OPT(j)$ does not include item j , how do we calculate $OPT(j)$?
 $OPT(j) = OPT(j-1)$
- ◉ If $OPT(j)$ includes item j , how do we calculate $OPT(j)$?
Harder! It's not obvious which of the previous items "conflict".

8

Dynamic Programming: False Start

- $OPT(i)$ = max value subset of items 1, ..., i.
 - Case 1: OPT does not select item i.
 - OPT selects best of { 1, 2, ..., i-1 }
 - Case 2: OPT selects item i.
 - accepting item i does not immediately imply that we will have to reject other items
 - without knowing what other items were selected before i, we don't even know if we have enough room for i
- Conclusion. Need more sub-problems!

9

Defining OPT 2: Adding a Variable

- Define $OPT(j, W)$ to be the maximum amount we can spend considering the first j items and not exceeding amount W.
- If $OPT(j, W)$ does not include item j, how do we calculate $OPT(j, W)$?
$$OPT(j, W) = OPT(j-1, W)$$
- If $OPT(j)$ includes item j, how do we calculate $OPT(j)$?
$$OPT(j, W) = w_j + OPT(j-1, W - w_j)$$

10

OPT Recurrence

- $OPT(j, W) = OPT(j-1, W)$ if $w_j > W$ (it doesn't fit)
- $OPT(j, W) = \max(OPT(j-1, W), w_j + OPT(j-1, W-w_j))$ otherwise

11

Algorithm

```
Subset-Sum (n, w) {
  initialize  $M[0, w] = 0$  for  $w = 0, 1, \dots, W$ 
  for  $i = 1..n$  {
    for  $w = 0..W$  {
      if  $w_i > w$  {
         $M[i, w] = M[i-1, w]$ 
      }
      else {
         $M[i, w] = \max(M[i-1, w], w_i + M[i-1, w-w_i])$ 
      }
    }
  }
}
```

Running time. $O(n W)$.
Not polynomial in input size!
"Pseudo-polynomial."

12

Using the Algorithm

Candy \$2
Burger \$4
Soda \$1
Pizza \$6

\$8 to
spend

	0	1	2	3	4	5	6	7	8
Candy	0	0	2	2	2	2	2	2	2
Burger	0	0	2	2	4	4	6	6	6
Soda	0	1	2	3	4	5	6	7	7
Pizza	0	1	2	3	4	5	6	7	8

13