Objectives

- Dynamic Programming
  - Segmented Least Squares
  - Subset Sums problem

Review: Weighted Interval Scheduling

- Jobs have start time, end time, value/weight
  - Goal: schedule compatible jobs with maximum weight
- What was the key insight to solving the weighted interval scheduling problem?

Binary decision:
- Optimal solution for jobs i through j includes j or doesn't

- How do we pick the solution?

Choose the larger value of
- [choose j and the best solution of compatible jobs] OR [best solution if don't pick j]

Then what did we do?

Review

- What is the process for applying dynamic programming to a problem?

Dynamic Programming Process

- Determine optimal substructure of problem
  - Define the recurrence relation
- Define algorithm to find the ***value*** of optimal solution
- Optionally, change algorithm to an ***iterative*** rather than recursive solution
- Define algorithm to find ***optimal solution***
- Analyze running time of algorithms

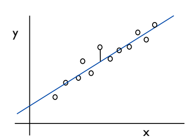
SEGMENTED LEAST SQUARES

Least Squares

- Foundational problem in statistic and numerical analysis
- Given n points in the plane: (x_1, y_1), (x_2, y_2), . . . , (x_n, y_n)
- Find a line $y = ax + b$ that minimizes the sum of the squared error
  - "line of best fit"

Sum of squared error

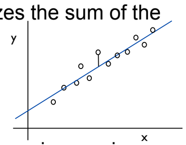
$$SSE = \sum_{i=1}^{n} (y_i - ax_i - b)^2$$



Least Squares

- Foundational problem in statistic and numerical analysis
- Given n points in the plane: $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$
- Find a line $y = ax + b$ that minimizes the sum of the squared error
  - "line of best fit"

Sum of squared error $SSE = \sum_{i=1}^{n} (y_i - ax_i - b)^2$

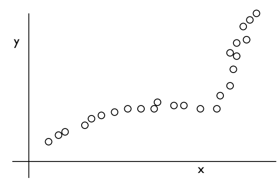


- Closed form solution. Calculus $\Rightarrow$ min error is achieved when

$$a = \frac{n \sum_i x_i y_i - (\sum_i x_i)(\sum_i y_i)}{n \sum_i x_i^2 - (\sum_i x_i)^2}, \quad b = \frac{\sum_i y_i - a \sum_i x_i}{n}$$

Least Squares

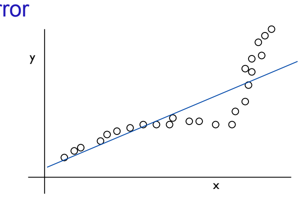
- What happens to the error if we try to fit one line to these points?



- What pattern does it seem like these points have?

Least Squares

- What happens to the error if we try to fit one line to these points?
  - Large error

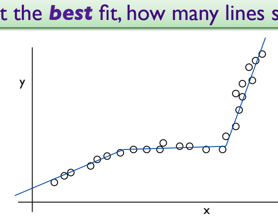


- Pattern: More like 3 lines

Segmented Least Squares

- Points lie roughly on a **sequence** of line segments
- Given n points in the plane $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$ with $x_1 < x_2 < \ldots < x_n$, find a **sequence** of **line segments** that **minimizes f(x)**

If I want the ***best*** fit, how many lines should I use?



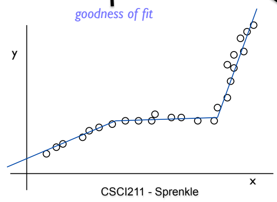
Segmented Least Squares

- Points lie roughly on a **sequence** of line segments
- Given n points in the plane $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$ with $x_1 < x_2 < \ldots < x_n$, find a sequence of line segments that **minimizes f(x)**

What's a reasonable choice for *f(x)* to balance *accuracy* and *parsimony*?

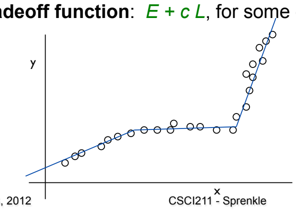
goodness of fit

number of lines



Segmented Least Squares

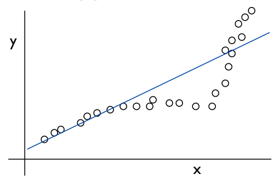
- Points lie roughly on a **sequence** of several line segments.
- Given n points in the plane $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$ with $x_1 < x_2 < \ldots < x_n$, find a sequence of line segments that minimizes:
  - *E*: sum of the sums of the squared errors in each segment
  - *L*: the number of lines
- **Tradeoff function**: $E + c\,L$, for some constant $c > 0$.



How should we define an optimal solution?

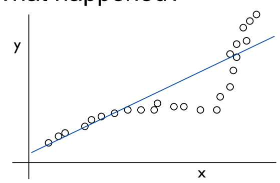
Segmented Least Squares

- What made it seem like the points were in 3 lines? What happened?



Segmented Least Squares

- What made it seem like the points were in 3 lines? What happened?



- Error increased
- Looking for *change* in linear approximation
  - Where to partition points into line segments

Recall: Properties of Problems for DP

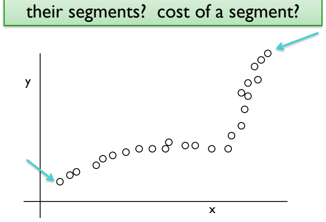
- Polynomial number of subproblems
- Solution to original problem can be easily computed from solutions to subproblems
- Natural ordering of subproblems, easy to compute recurrence

We need to:
- Figure out how to break the problem into subproblems
- Figure out how to compute solution from subproblems
- Define the recurrence relation between the problems

Toward a Solution

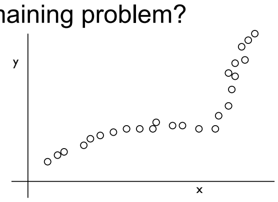
- Consider just the first or last point

What do we know about those points? their segments? cost of a segment?



Toward a Solution

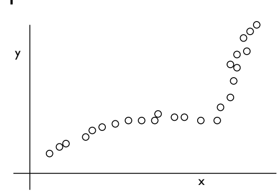
- p_n can only belong to one segment
  - Segment: p_i, …, p_n
  - Cost: c (cost for segment) + error of segment
- What is the remaining problem?



Toward a Solution

- p_n can only belong to one segment
  - Segment: p_i, …, p_n
  - Cost: c (cost for segment) + error of segment
- What is the remaining problem?
  - Solve for p_1, …, p_{i-1}

Next: Formulate as a recurrence



Dynamic Programming: Multiway Choice

- Notation.
  - **OPT(j)** = minimum cost for points $p_1, p_{i+1}, \ldots, p_j$.
  - **e(i, j)** = minimum sum of squares for points $p_i, p_{i+1}, \ldots, p_j$.
- How do we compute OPT(j)?
  - Last problem: binary decision (include job or not)
  - This time: **multiway** decision
    - Which option do we choose?

Dynamic Programming: Multiway Choice

- Notation.
  - **OPT(j)** = minimum cost for points $p_1, p_{i+1}, \ldots, p_j$.
  - **e(i, j)** = minimum sum of squares for points $p_i, p_{i+1}, \ldots, p_j$.
- To compute OPT(j):
  - Last segment contains points $p_i, p_{i+1}, \ldots, p_j$ for some i
  - Cost = e(i, j) + c + OPT(i-1).

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

Segmented Least Squares: Algorithm

```
INPUT: n, p1,…,pN , c

Segmented-Least-Squares()
   M[0] = 0
   e[0][0] = 0  # needed?
   for j = 1 to n
      for i = 1 to j
         e[i][j] = least square error for the
                   segment pi, …, pj

   for j = 1 to n
      M[j] = min 1 ≤ i ≤ j (e[i][j] + c + M[i-1])

   return M[n]
```

Costs?

Segmented Least Squares: Algorithm Analysis

How do we find the *solution*?

```
INPUT: n, p1,…,pN , c

Segmented-Least-Squares()
   M[0] = 0
   e[0][0] = 0
   for j = 1 to n
      for i = 1 to j
         e[i][j] = least square error for the
                   segment pi,…, pj

   for j = 1 to n
      M[j] = min 1 ≤ i ≤ j (e[i][j] + c + M[i-1])

   return M[n]
```

can be improved to $O(n^2)$ by pre-computing various statistics

$O(n^3)$

$O(n^2)$

- Bottleneck: computing e(i, j) for $O(n^2)$ pairs, O(n) per pair using previous formula

Post-Processing: Finding the Solution

```
FindSegments(j):
   if j = 0:
       output nothing
   else:
       Find an i that minimizes e_i,j + c + M[i-1]
       Output the segment {p_i, …, p_j}
       FindSegments(i-1)
```

Cost? $O(n^2)$

Dynamic Programming Process

- Determine optimal substructure of problem
  - Define the recurrence relation
- Define algorithm to find the ***value*** of optimal solution
- Optionally, change algorithm to an ***iterative*** rather than recursive solution
- Define algorithm to find ***optimal solution***
- Analyze running time of algorithms

SUBSET SUMS and KNAPSACKS

The Price is Right

Or, shopping with someone else's money

- **Goal**: Spend as much money as possible without going over \$100
  - CD \$18
  - Jeans \$40
  - DVD \$35
  - Dinner \$15
  - Book \$8
  - Ice cream \$5
  - Shoes \$62
  - Pizza \$7

Possible solutions?

Knapsack Problem

- Given n objects and a "knapsack"
- Item i weighs $w_i > 0$ kilograms and has value $v_i > 0$
  - Alternative: jobs require w_i time
- Knapsack has capacity of W kilograms
  - Alternative: W is time interval that resource is available

Goal: fill knapsack so as to maximize total **value**

W = 11

Item	Value	Weight
1	1	1
2	6	2
3	18	5
4	22	6
5	28	7

Towards a Recurrence…

- What do we know about the knapsack with respect to item i?

Towards a Recurrence…

- What do we know about the knapsack with respect to item i?
  - Either select item i or not
  - If don't select
    - Pick optimum solution of remaining items
  - Otherwise
    - What happens?
    - How does problem change?

Dynamic Programming: False Start

- Def. OPT(i) = max profit 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
      - No known conflicts
    - Without knowing what other items were selected before i, we don't even know if we have enough room for i

➡Need more sub-problems!

Looking Ahead

- Exam 2 due next Friday
  - Wednesday work period
- No wiki for next week

Mar 16, 2012 CSCI211 - Sprenkle 31