CSE 421
Algorithms

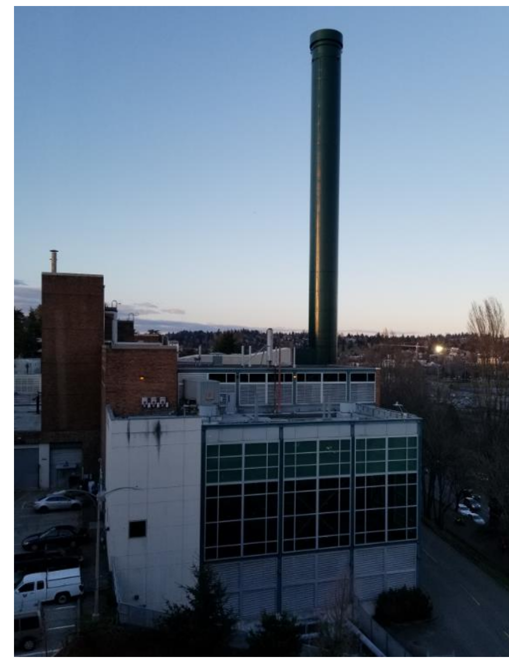
Richard Anderson

Lecture 17, Winter 2019

Dynamic Programming

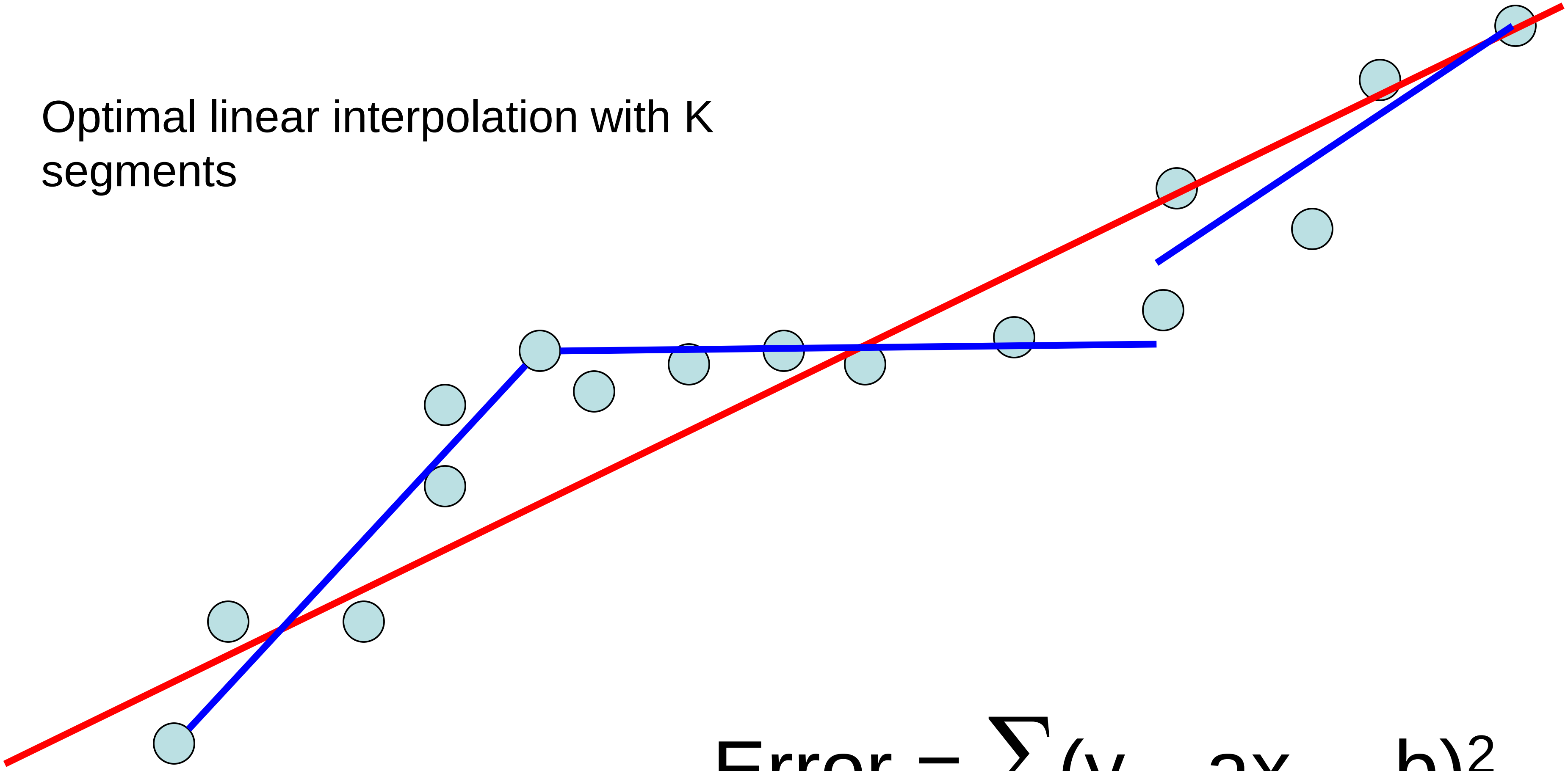
Announcements

- HW 7. Partially Posted
- I've moved
  - My new office: CSE2 344



Optimal linear interpolation

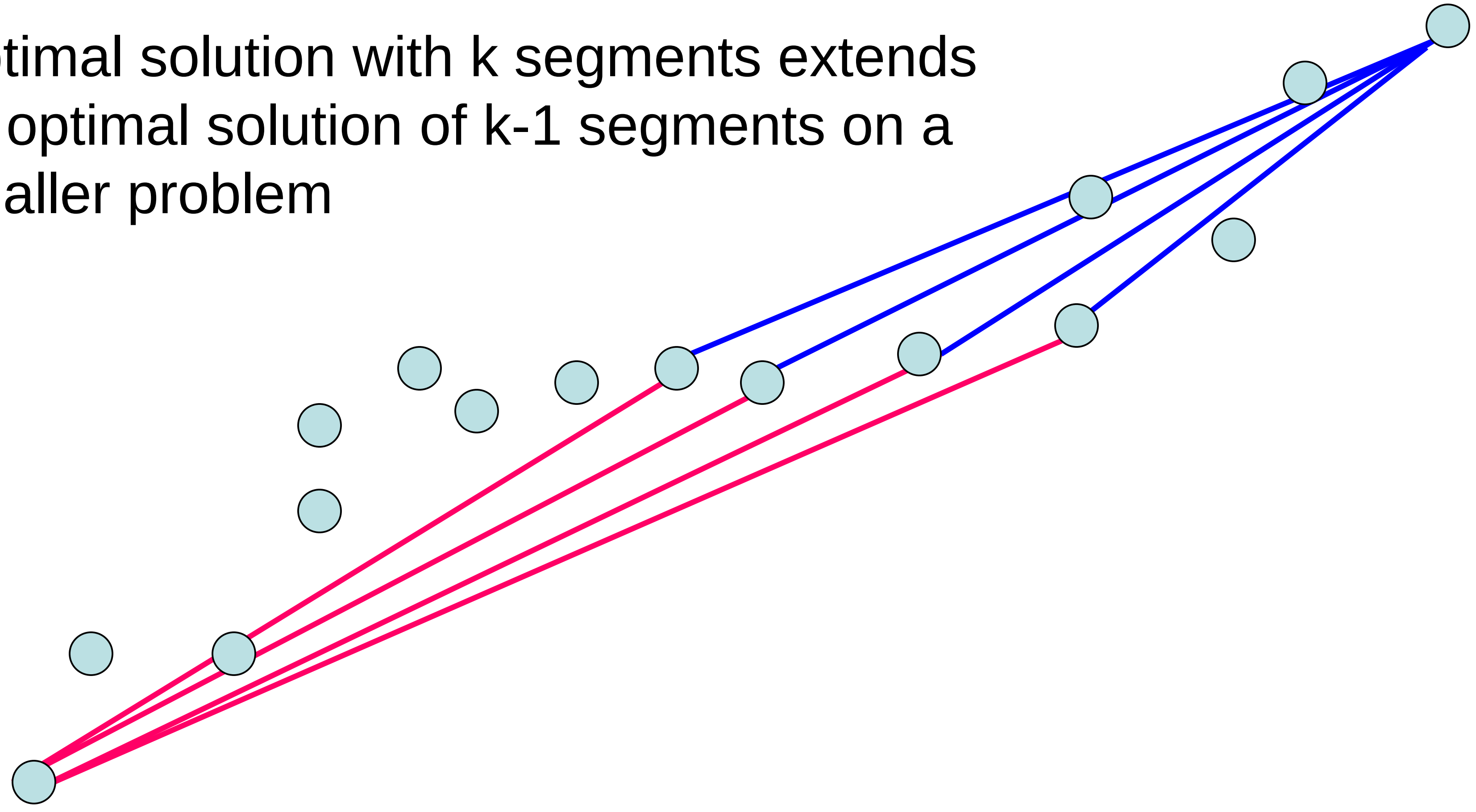
Optimal linear interpolation with K segments



$$\text{Error} = \sum (y_i - a x_i - b)^2$$

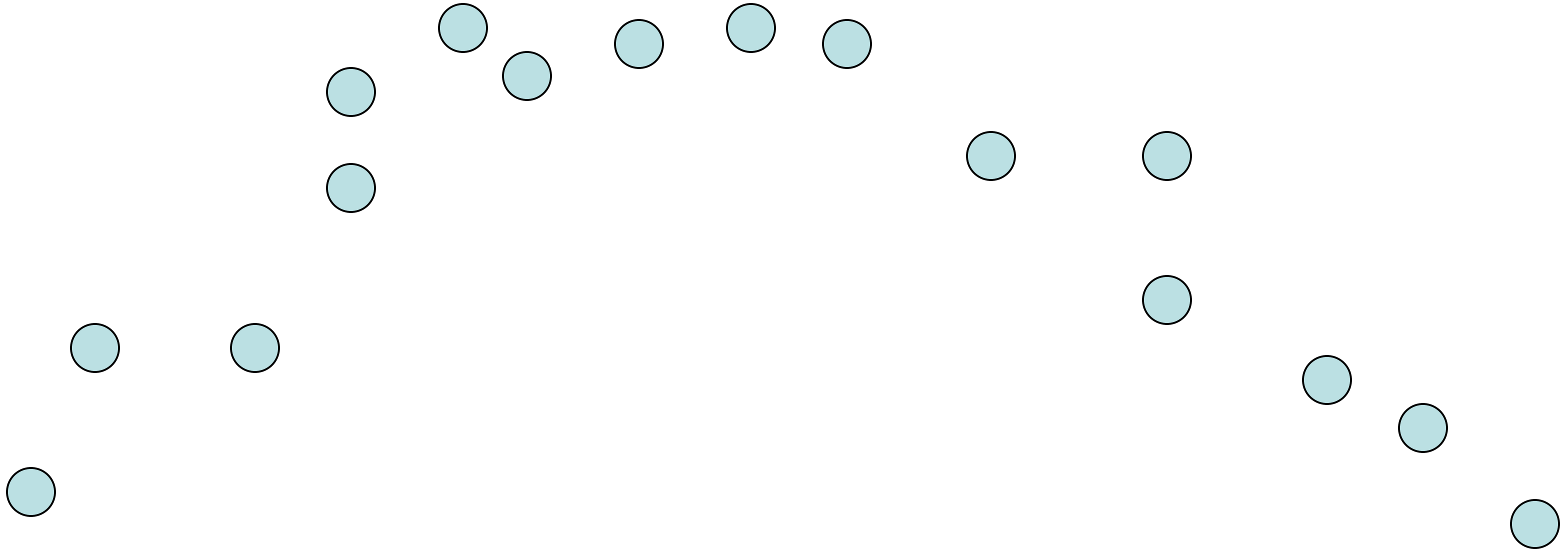
$$\text{Opt}_k[\ j\ ] = \min_i \{\ \text{Opt}_{k-1}[\ i\ ] + E_{i,j}\ \} \text{ for } 0 < i < j$$

Optimal solution with k segments extends an optimal solution of k-1 segments on a smaller problem



Variable number of segments

- Segments not specified in advance
- Penalty function associated with segments
- Cost = Interpolation error + C x #Segments



Penalty cost measure

- $Opt[\ j\ ] = \min(E_{1,j}, \min_i(Opt[\ i\ ] + E_{i,j} + P))$

Subset Sum Problem

- Let $w_1,\ldots,w_n = \{6, 8, 9, 11, 13, 16, 18, 24\}$
- Find a subset that has as large a sum as possible, without exceeding 50

Adding a variable for Weight

- Opt[j, K] the largest subset of $\{w_1, \ldots, w_j\}$ that sums to at most K
- {2, 4, 7, 10}
  - Opt[2, 7] =
  - Opt[3, 7] =
  - Opt[3,12] =
  - Opt[4,12] =

Subset Sum Recurrence

- Opt[j, K] the largest subset of $\{w_1, \ldots, w_j\}$ that sums to at most K

Subset Sum Grid

Opt[j, K] = max(Opt[j – 1, K], Opt[j – 1, K – w_j] + w_j)

4																	
3																	
2																	
1																	
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

{2, 4, 7, 10}

Subset Sum Code

```
for j = 1 to n
    for k = 1 to W
        Opt[j, k] = max(Opt[j-1, k], Opt[j-1, k-w_j] + w_j)
```

Knapsack Problem

- Items have weights and values
- The problem is to maximize total value subject to a bound on weght
- Items $\{I_1, I_2, \ldots I_n\}$
  - Weights $\{w_1, w_2, \ldots, w_n\}$
  - Values $\{v_1, v_2, \ldots, v_n\}$
  - Bound K
- Find set S of indices to:
  - Maximize $\sum_{i \varepsilon S} v_i$ such that $\sum_{i \varepsilon S} w_i <= K$

Knapsack Recurrence

Subset Sum Recurrence:

$$\text{Opt}[\, j, K] = \max(\text{Opt}[\, j - 1, K], \text{Opt}[\, j - 1, K - w_j] + w_j)$$

Knapsack Recurrence:

Knapsack Grid

Opt[j, K] = max(Opt[j – 1, K], Opt[j – 1, K – w_j] + v_j)

4																	
3																	
2																	
1																	
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Weights {2, 4, 7, 10} Values: {3, 5, 9, 16}

Dynamic Programming Examples

- Examples
  - Optimal Billboard Placement
    - Text, Solved Exercise, Pg 307
  - Linebreaking with hyphenation
    - Compare with HW problem 6, Pg 317
  - String approximation
    - Text, Solved Exercise, Page 309

Billboard Placement

- Maximize income in placing billboards
  - $b_i = (p_i, v_i)$, v_i: value of placing billboard at position p_i
- Constraint:
  - At most one billboard every five miles
- Example
  - {(6,5), (8,6), (12, 5), (14, 1)}

Design a Dynamic Programming Algorithm for Billboard Placement

- Compute Opt[1], Opt[2], . . ., Opt[n]
- What is Opt[k]?

Input $b_1, \ldots, b_n$, where $b_i = (p_i, v_i)$, position and value of billboard i

$Opt[k] = fun(Opt[0],\ldots,Opt[k-1])$

- How is the solution determined from sub problems?

Input $b_1, \ldots, b_n$, where $b_i = (p_i, v_i)$, position and value of billboard i

Solution

```
j = 0;              // j is five miles behind the current position
                     // the last valid location for a billboard, if one placed at P[k]
for k := 1 to n
        while (P[ j ] < P[ k ] – 5)
                j := j + 1;
        j := j – 1;
        Opt[ k]  = Max(Opt[ k-1] , V[ k ] + Opt[ j ]);
```