Objectives

Dynamic Programming

- Finish weighted scheduling
- Segmented least squares

Quote of the NCAA Tourney

This is the guy who has to get it done for Binghamton. He's their CPU if this is a computer…. He's the operating system…. He's the processing unit, the one that makes everything happen.

-- Clark Kellogg on Emanuel Mayben

Dynamic Programming: Key Idea?

Dynamic Programming: Key Idea

Memoization. Keep the previous results to reduce running time

- Tradeoff of space for time

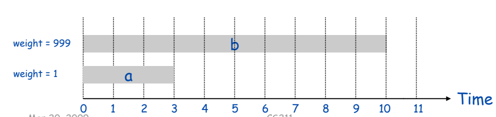
WEIGHTED INTERVAL SCHEDULING

Limitation of Greedy Algorithm

Recall. Greedy algorithm works if all weights are 1.

- Consider jobs in ascending order of finish time
- Add job to subset if it is compatible with previously chosen jobs

Observation. Greedy algorithm can fail spectacularly if arbitrary weights are allowed



Dynamic Programming: Binary Choice

Notation. OPT = value of optimal solution to the problem consisting of job requests 1, 2, ..., *j*

- Case 1: OPT selects job *j*
  - can't use incompatible jobs { p(j) + 1, p(j) + 2, ..., j - 1 }
  - must include optimal solution to problem consisting of remaining compatible jobs 1, 2, ..., p(*j*)
- Case 2: OPT does not select job *j*
  - must include optimal solution to problem consisting of remaining compatible jobs 1, 2, ..., *j*-1

$$OPT(j) = \begin{cases} 0 & \text{if } j = 0 \\ \max\{ v_j + OPT(p(j)),\ OPT(j-1)\} & \text{otherwise} \end{cases}$$

Choose the better of the two solutions

Weighted Interval Scheduling: Memoization

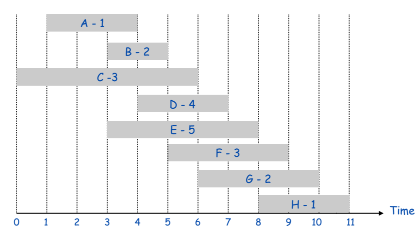
Memoization. Store results of each sub-problem in a cache; lookup as needed.

```
Input: n jobs (associated start time sj, finish time fj, and
value vj)

Sort jobs by finish times so that f1 ≤ f2 ≤ ... ≤ fn
Compute p(1), p(2), …, p(n)

for j = 1 to n
   M[j] = empty      <-- global array
M[0] = 0  <-- Because we have jobs whose p(j) = 0

M-Compute-Opt(j):
   if M[j] is empty:
      M[j] = max(vj + M-Compute-Opt(p(j)), M-Compute-Opt(j-1))
   return M[j]
```

Need to analyze runtime…

Example

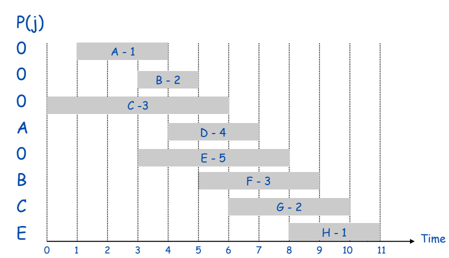
Jobs labeled with name - weight/value



M

0	A	B	C	D	E	F	G	H

Example



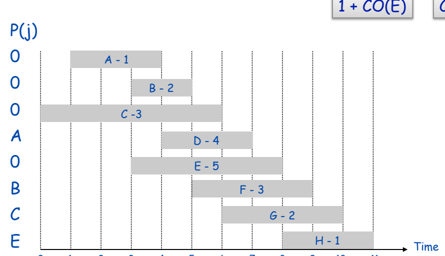
M

0	A	B	C	D	E	F	G	H
0								

Example

CO(H)

1 + CO(E) CO(G)



M

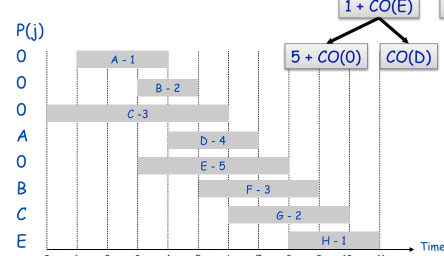
0	A	B	C	D	E	F	G	H
0								

Example

CO(H)

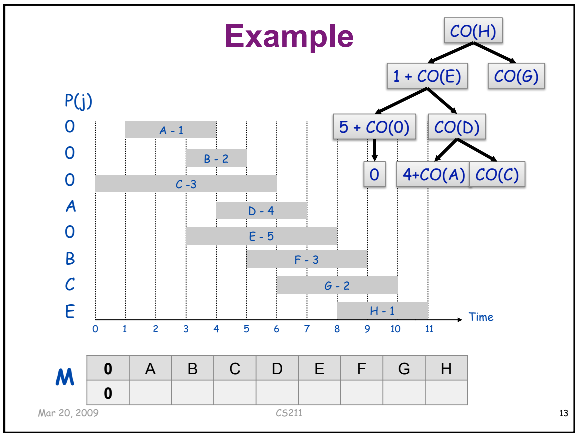
1 + CO(E) CO(G)

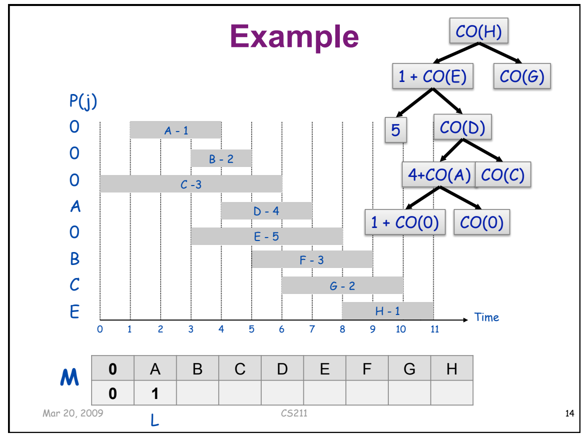
5 + CO(0) CO(D)

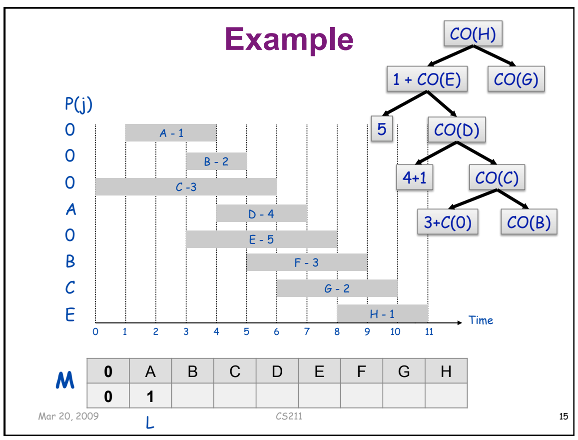


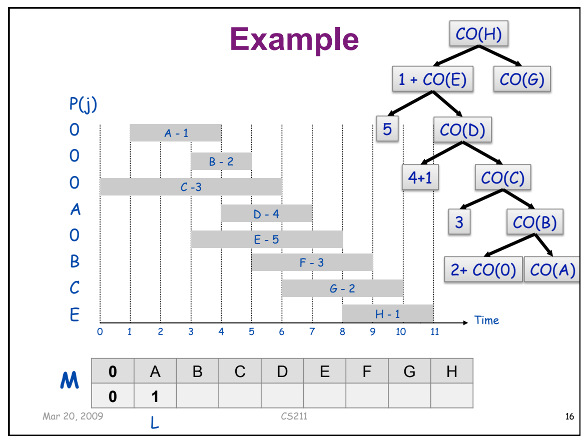
M

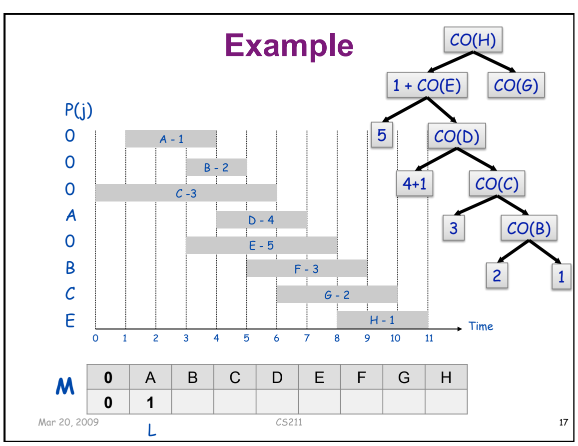
0	A	B	C	D	E	F	G	H
0								

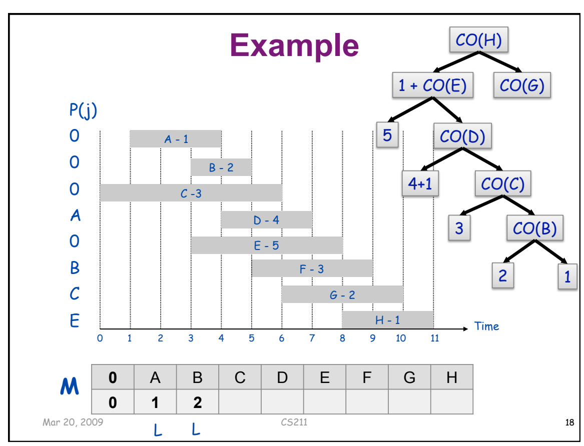

Example
CO(H)
1 + CO(E)
CO(G)
5 + CO(0)
CO(D)
0
4+CO(A)
CO(C)
P(j)
0
0
0
A
0
B
C
E
A - 1
B - 2
C - 3
D - 4
E - 5
F - 3
G - 2
H - 1
Time
0 1 2 3 4 5 6 7 8 9 10 11
M
0 A B C D E F G H
0
Mar 20, 2009
CS211
13


Example
CO(H)
1 + CO(E)
CO(G)
5
CO(D)
4+CO(A)
CO(C)
1 + CO(0)
CO(0)
P(j)
0
0
0
A
0
B
C
E
A - 1
B - 2
C - 3
D - 4
E - 5
F - 3
G - 2
H - 1
Time
0 1 2 3 4 5 6 7 8 9 10 11
M
0 A B C D E F G H
0 1
L
Mar 20, 2009
CS211
14

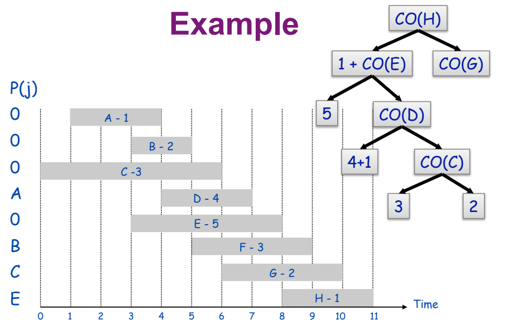

Example
CO(H)
1 + CO(E)
CO(G)
5
CO(D)
4+1
CO(C)
3+C(0)
CO(B)
P(j)
0
0
0
A
0
B
C
E
A - 1
B - 2
C - 3
D - 4
E - 5
F - 3
G - 2
H - 1
Time
0 1 2 3 4 5 6 7 8 9 10 11
M
0 A B C D E F G H
0 1
L
Mar 20, 2009
CS211
15


Example
CO(H)
1 + CO(E)
CO(G)
5
CO(D)
4+1
CO(C)
3
CO(B)
2+ CO(0)
CO(A)
P(j)
0
0
0
A
0
B
C
E
A - 1
B - 2
C - 3
D - 4
E - 5
F - 3
G - 2
H - 1
Time
0 1 2 3 4 5 6 7 8 9 10 11
M
0 A B C D E F G H
0 1
L
Mar 20, 2009
CS211
16


Example
CO(H)
1 + CO(E)
CO(G)
5
CO(D)
4+1
CO(C)
3
CO(B)
2
1
P(j)
0
0
0
A
0
B
C
E
A - 1
B - 2
C - 3
D - 4
E - 5
F - 3
G - 2
H - 1
Time
0 1 2 3 4 5 6 7 8 9 10 11
M
0 A B C D E F G H
0 1
L
Mar 20, 2009
CS211
17


Example
CO(H)
1 + CO(E)
CO(G)
5
CO(D)
4+1
CO(C)
3
CO(B)
2
1
P(j)
0
0
0
A
0
B
C
E
A - 1
B - 2
C - 3
D - 4
E - 5
F - 3
G - 2
H - 1
Time
0 1 2 3 4 5 6 7 8 9 10 11
M
0 A B C D E F G H
0 1 2
L L
Mar 20, 2009
CS211
18

Example

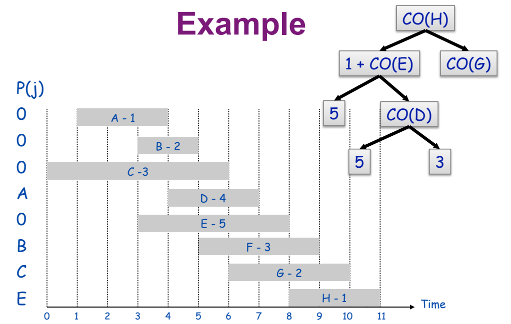


M

0	A	B	C	D	E	F	G	H
0	1	2	3					

L L L

Example

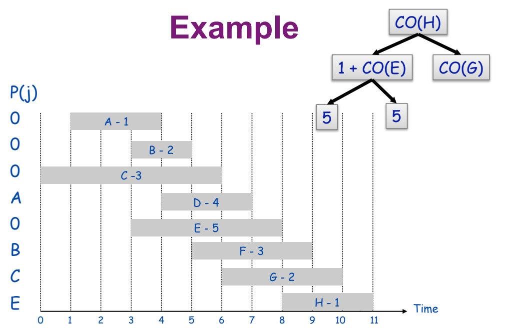


M

0	A	B	C	D	E	F	G	H
0	1	2	3	5				

L L L L

Example

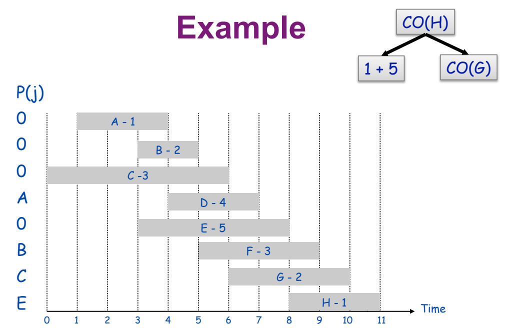


M

0	A	B	C	D	E	F	G	H
0	1	2	3	5	5			

L L L L L/R

Example

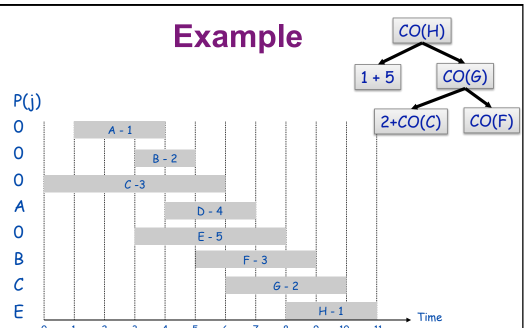


M

0	A	B	C	D	E	F	G	H
0	1	2	3	5	5			

L L L L L/R

Example

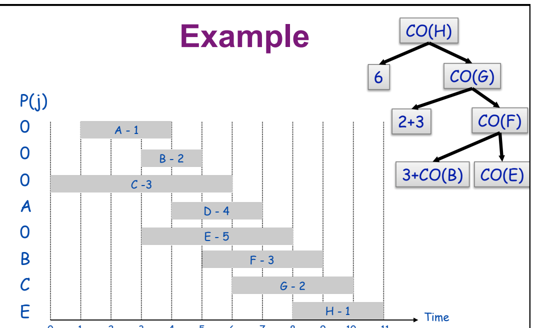


M

0	A	B	C	D	E	F	G	H
0	1	2	3	5	5			

L L L L L/R

Example

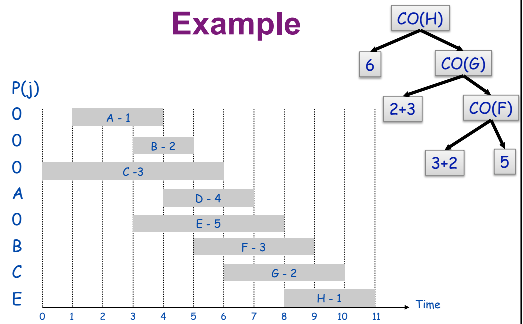


M

0	A	B	C	D	E	F	G	H
0	1	2	3	5	5			

L L L L L/R

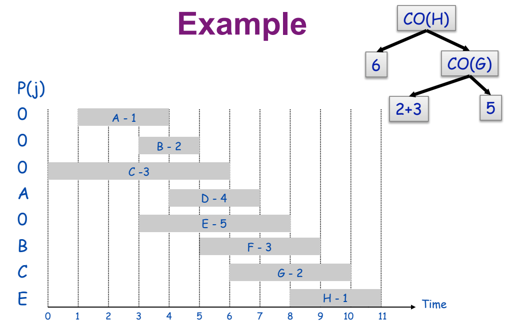
Example



M	0	A	B	C	D	E	F	G	H
	0	1	2	3	5	5	5		

L L L L L/R L/R

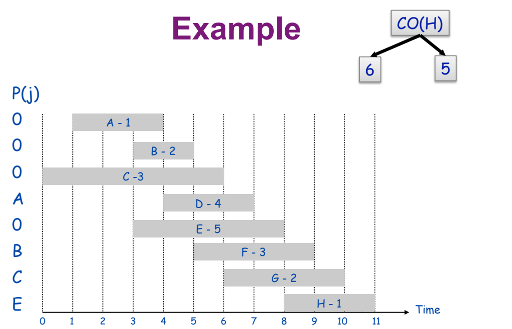
Example



M	0	A	B	C	D	E	F	G	H
	0	1	2	3	5	5	5	5	

L L L L L/R L/R L/R

Example



M	0	A	B	C	D	E	F	G	H
	0	1	2	3	5	5	5	5	6

L L L L L/R L/R L/R L

Weighted Interval Scheduling: Memoization Analysis

Costs?

```
Input: n jobs (associated start time s_j, finish time f_j, and
value v_j)

Sort jobs by finish times so that f_1 ≤ f_2 ≤ ... ≤ f_n
Compute p(1), p(2), …, p(n)

for j = 1 to n
   M[j] = empty
M[0] = 0

M-Compute-Opt(j):
   if M[j] is empty:
      M[j] = max(v_j + M-Compute-Opt(p(j)), M-Compute-Opt(j-1))
   return M[j]
```


Weighted Interval Scheduling: Memoization Analysis

Costs?

```
Input: n jobs (associated start time s_j, finish time f_j, and
value v_j)

Sort jobs by finish times so that f_1 ≤ f_2 ≤ ... ≤ f_n    O(n log n)
Compute p(1), p(2), …, p(n)  O(n)

for j = 1 to n
   M[j] = empty          O(n)
M[0] = 0

M-Compute-Opt(j):               O(n)
   if M[j] is empty:
      M[j] = max(v_j + M-Compute-Opt(p(j)), M-Compute-Opt(j-1))
   return M[j]
```


Weighted Interval Scheduling: Running Time

Claim. Memoized version of algorithm takes O(n log n) time

- Sort by finish time: O(n log n)
- Computing p(·): O(n) after sorting by start time
- `M-Compute-Opt(j)`: each invocation takes O(1) time and either
  - (i) returns an existing value `M[j]`
  - (ii) fills in one new entry `M[j]` and makes two recursive calls
- Progress measure Φ = # nonempty entries of `M[]`
  - (i) initially $\Phi = 0$, throughout $\Phi \le n$
  - (ii) increases Φ by 1 $\Rightarrow$ at most 2n recursive calls
- Overall running time of `M-Compute-Opt(n)` is O(n). ▪

Remark. O(n) if jobs are pre-sorted by start and finish times

Weighted Interval Scheduling: Finding a Solution

Q. Dynamic programming algorithms compute optimal value. What if we want the solution itself?

A. Do some post-processing

- Looking at M, how do we know which set of intervals were chosen?

M	0	A	B	C	D	E	F	G	H
	0	1	2	3	5	5	5	5	6
		L	L	L	L	L/R	L/R	L/R	L

Weighted Interval Scheduling: Finding a Solution

Q. Dynamic programming algorithms compute optimal value. What if we want the solution itself?

A. Do some post-processing

```
Run M-Compute-Opt(n)
Run Find-Solution(n)

Find-Solution(j):
   if j = 0:
      output nothing
   elif v_j + M[p(j)] > M[j-1]:
      print j
      Find-Solution(p(j))
   else:
      Find-Solution(j-1)
```


Turning it Around…

We solved the Fibonacci problem as both recursive/ memoized and an iterative algorithm

Can we write this algorithm as an iterative solution?

```
Input: n jobs (associated start time s_j, finish time f_j, and
value v_j)

Sort jobs by finish times so that f_1 ≤ f_2 ≤ ... ≤ f_n
Compute p(1), p(2), …, p(n)

for j = 1 to n
   M[j] = empty
M[0] = 0

M-Compute-Opt(j):
   if M[j] is empty:
      M[j] = max(v_j + M-Compute-Opt(p(j)), M-Compute-Opt(j-1))
   return M[j]
```


Iterative Solution

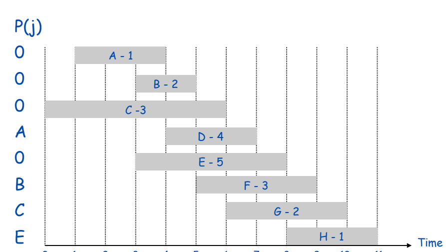
Build up solution from subproblems instead of breaking down

```
Input: n, s_1,…,s_n , f_1,…,f_n , v_1,…,v_n

Sort jobs by finish times so that f_1 ≤ f_2
≤ ... ≤ f_n.

Compute p(1), p(2), …, p(n)

Iterative-Compute-Opt
   M[0] = 0
   for j = 1 to n
      M[j] = max(v_j + M[p(j)], M[j-1])
```

Typically, approach we'll take

Runtime?

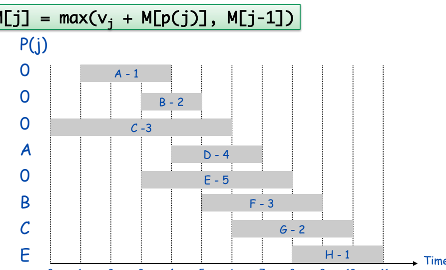
Example: Iteratively



M	0	A	B	C	D	E	F	G	H
	0								

Example: Iteratively

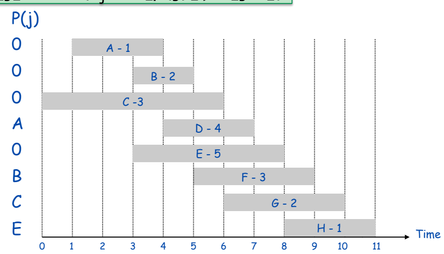
M[j] = max(v_j + M[p(j)], M[j-1])



M	0	A	B	C	D	E	F	G	H
	0								

Example: Iteratively

$M[j] = max(v_j + M[p(j)], M[j-1])$

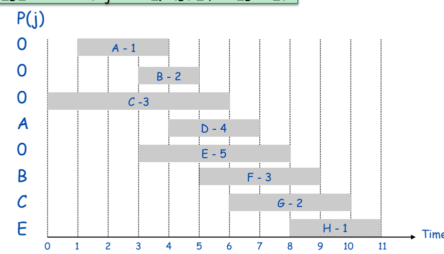


M

0	A	B	C	D	E	F	G	H
0	1							

Example: Iteratively

$M[j] = max(v_j + M[p(j)], M[j-1])$

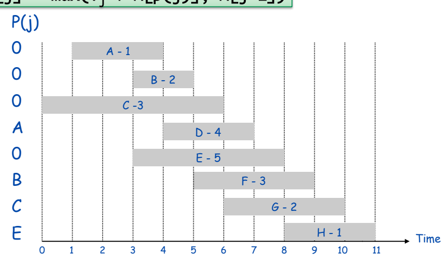


M

0	A	B	C	D	E	F	G	H
0	1							

Example: Iteratively

$M[j] = max(v_j + M[p(j)], M[j-1])$

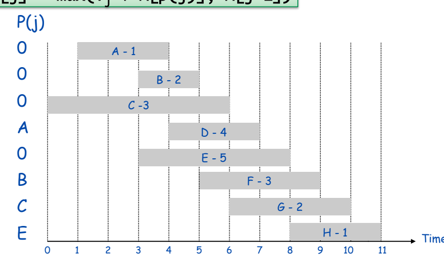


M

0	A	B	C	D	E	F	G	H
0	1	2						

Example: Iteratively

$M[j] = max(v_j + M[p(j)], M[j-1])$

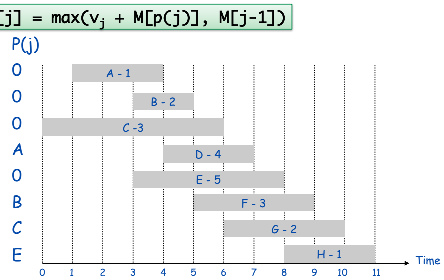


M

0	A	B	C	D	E	F	G	H
0	1	2	3					

Example: Iteratively

$M[j] = max(v_j + M[p(j)], M[j-1])$

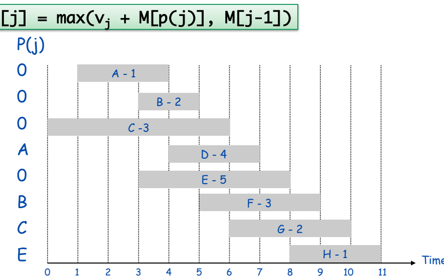


M

0	A	B	C	D	E	F	G	H
0	1	2	3	5				

Example: Iteratively

$M[j] = max(v_j + M[p(j)], M[j-1])$

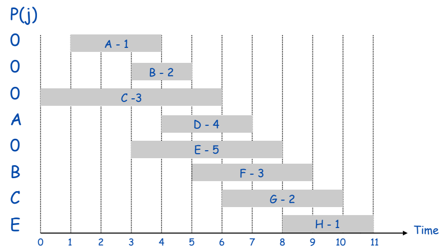


M

0	A	B	C	D	E	F	G	H
0	1	2	3	5				

And so on....

Example: Iteratively



M

0	A	B	C	D	E	F	G	H
0	1	2	3	5	5	5	5	6

Summary: 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

SEGMENTED LEAST SQUARES

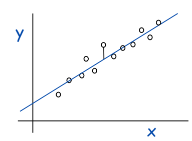
Least Squares

Foundational problem in statistic and numerical analysis

Given *n* points in the plane: $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$

Find a line $y = ax + b$ that minimizes the sum of the squared error

- "line of best fit"



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

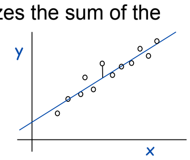
Least Squares

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"



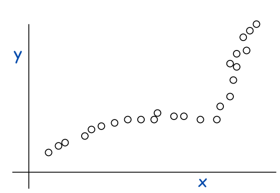
Sum of squared error $SSE = \sum_{i=1}^{n} (y_i - ax_i - b)^2$

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?

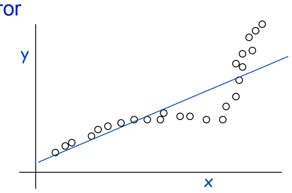


What pattern does it seem like these points have?

Least Squares

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

- Large error



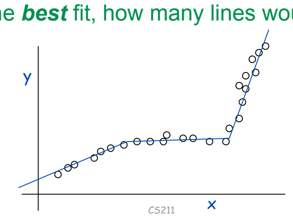
Pattern: More like 3 lines

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 lines that minimizes $f(x)$

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



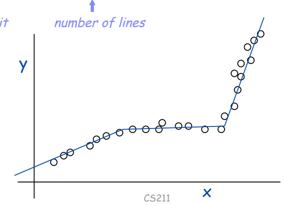
Segmented Least Squares

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 lines that minimizes $f(x)$

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

goodness of fit *number of lines*



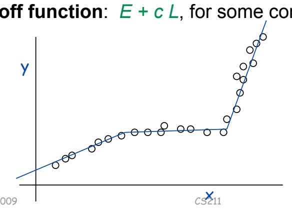
Segmented Least Squares

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 lines that minimizes:

- the sum of the sums of the squared errors *E* in each segment
- the number of lines *L*

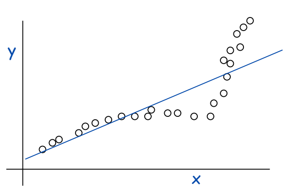
Tradeoff function: $E + c\,L$, for some constant $c > 0$.



How should we define an optimal solution?

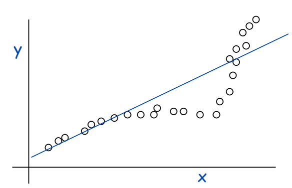
Segmented Least Squares

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



Segmented Least Squares

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



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 segments?