12 – Dynamic Programming (2)

Matrix-chain Multiplication

Segmented Least Squares

Optimal substructure

Dynamic programming is typically applied to
optimization problems.

An optimal solution to the original problem contains
optimal solutions to smaller subproblems.

Matrix-chain multiplication

Given: Sequence (chain) $\langle A_1, A_2, \ldots, A_n \rangle$ of matrices

Goal: Compute the product $A_1 \cdot A_2 \cdot \ldots . \cdot A_n$.

Problem: Parenthesize the product in a way that minimizes the number of scalar multiplications.

Definition: A product of matrices is *fully parenthesized* if it is either a single matrix or the product of two fully parenthesized matrix products, surrounded by parentheses.

Multiplying two matrices

$$A = (a_{ij})_{p \times q},\ B = (b_{ij})_{q \times r},\ A \cdot B = C = (c_{ij})_{p \times r},$$

$$c_{ij} = \sum_{k=1}^{q} a_{ik} b_{kj}.$$

Algorithm *Matrix-Mult*
Input: $(p \times q)$ matrix A, $(q \times r)$ matrix B
Output: $(p \times \mathrm{r})$ matrix $C = A \cdot B$

```
for i := 1 to p do
    for j :=1 to r do
        C[i, j] := 0
        for k := 1 to q do
            C[i, j] := C[i, j] + A[i, k] ·B[k,j]
```

Number of multiplications and additions: $p \cdot q \cdot r$

Remark: Using this algorithm, multiplying two $(n \times n)$ matrices requires n^3 multiplications. This can also be done using $O(n^{2.376})$ multiplications.

Matrix-chain multiplication: Example

Computation of the product $A_1 A_2 A_3$, where

A_1 : (10×100) matrix
A_2 : (100×5) matrix
A_3 : (5×50) matrix

Parenthesization $((A_1 A_2) A_3)$ requires

$A' = (A_1 A_2)$: $10 \cdot 100 \cdot 5 = 5000$

$A' A_3$: $10 \cdot 5 \cdot 50 = 2500$

Sum: 7500

Matrix-chain multiplication: Example

A_1 : (10 $\times$ 100) matrix
A_2 : (100 $\times$ 5) matrix
A_3 : (5 $\times$ 50) matrix

Parenthesization $(A_1 (A_2 A_3))$ requires

$A'' = (A_2 A_3)$: $100 \cdot 5 \cdot 50 = 25000$

$A_1 A''$: $10 \cdot 100 \cdot 50 = 50000$

Sum: 75000

Fully parenthesized matrix products

All possible fully parenthesized matrix products of the chain $\langle A_1, A_2, A_3, A_4 \rangle$ are:

$$(A_1(A_2(A_3A_4)))$$

$$(A_1((A_2A_3)A_4))$$

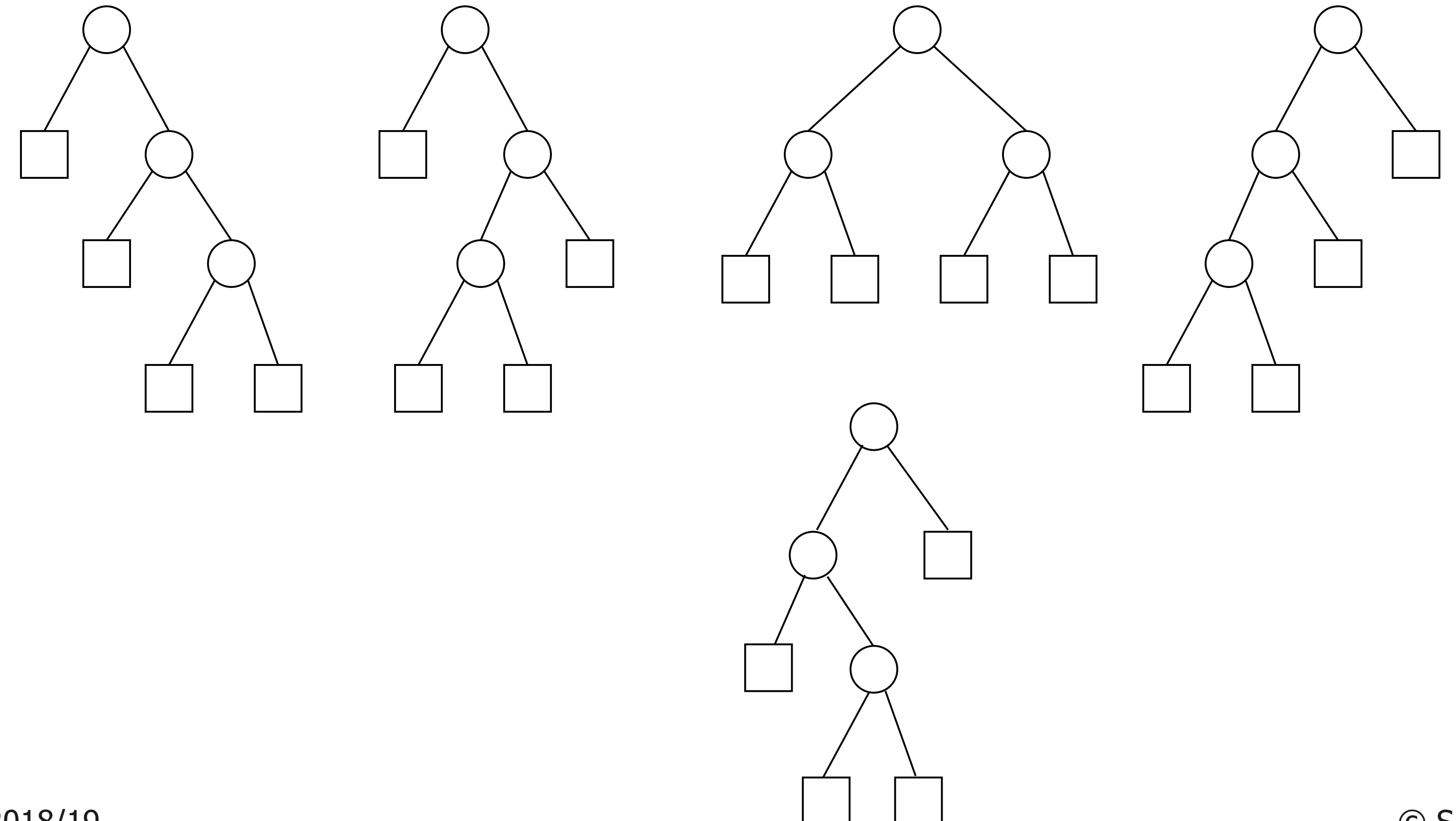
$$((A_1A_2)(A_3A_4))$$

$$((A_1(A_2A_3))A_4)$$

$$(((A_1A_2)A_3)A_4)$$

Number of different parenthesizations

Different parenthesizations correspond to different trees (selection).



Number of different parenthesizations

Let $P(n)$ be the number of alternative parenthesizations of the product $A_1 \ldots A_k A_{k+1} \ldots A_n$.

$$P(1) = 1$$

$$P(n) = \sum_{k=1}^{n-1} P(k) P(n-k) \quad \text{for } n \geq 2$$

$$P(n) = C_{n-1} \quad (n-1)-\text{st Catalan number}$$

$$C(n) = \frac{1}{n+1}\binom{2n}{n} \approx \frac{4^n}{n\sqrt{\pi n}} + O\left(\frac{4^n}{\sqrt{n^5}}\right)$$

Remark: Determining the optimal parenthesization by exhaustive search is not reasonable.

Matrix-chain multiplication

Given: Sequence $\langle A_1, A_2, \ldots, A_n \rangle$ of matrices
Matrix A_i has dimension $p_{i-1} \times p_i$, for $i = 1, \ldots, n$.

Goal: Parenthesize the product in a way that minimizes the number of scalar multiplications.

Structure of an optimal parenthesization

Subproblems $A_{i...j}$ $\quad 1 \le i \le j \le n$

$A_{i...i} = A_i$ $\quad A_{i...j} = (A_{i...k})(A_{k+1...j}) \quad i \le k < j$

Any optimal solution to the matrix-chain multiplication problem contains optimal solutions to subproblems. Determine an optimal solution recursively.

Let $m[i,j]$ be the minimum number of operations needed to compute the product $A_{i...j}$.

$m[i,j] = 0 \quad$ if $i = j$

$$m[i,j] = \min_{i \le k < j} \{ m[i,k] + m[k+1, j] + p_{i-1} p_k p_j \} \quad \text{otherwise}$$

$s[i,j]$ = optimal split value k, i.e. the optimal parenthesization of $A_{i...j}$ splits the product between A_k and A_{k+1}

Recursive matrix-chain multiplication

Algorithm *RecMatChain*(p, i, j)

Input: Sequence $p = \langle p_0, p_1, ..., p_n \rangle$,

where $(p_{i-1} \times p_i)$ is the dimensionen of matrix A_i

Invariant: RecMatChain(p, i, j) returns $m[i, j]$

```
if i = j then return 0;
m[i, j] := ∞;
for k := i to j – 1 do
      m[i, j] := min( m[i,j],  p_{i-1} p_k p_j +
                               RecMatChain(p, i, k) +
                               RecMatChain(p, k+1, j) );
return m[i, j];
```

Initial call: RecMatChain(p,1, n)

Recursive matrix-chain multiplication

Let $T(n)$ be the time taken by rec-mat-chain(p,1,n).

$$T(1) \geq 1$$

$$T(n) \geq 1 + \sum_{k=1}^{n-1} \left(T(k) + T(n-k) + 1 \right)$$

$$\geq n + 2\sum_{i=1}^{n-1} T(i)$$

$$\Rightarrow T(n) \geq 3^{n-1} \quad (\text{induction})$$

Exponential running time!

Solution using dynamic programming

Algorithm *DynMatChain*

Input: Sequence $p = \langle p_0, p_1, \ldots, p_n \rangle$ $(p_{i-1} \times p_i)$ the dimension of matrix A_i

Output: $m[1,n]$

```
n := length(p) – 1;
for i := 1 to n do m[i, i] := 0;
for l := 2 to n do                          /* l = length of the subproblem */
  for i := 1 to n – l + 1 do                /* i is the left index */
      j := i + l – 1;                       /* j is the right index */
      m[i, j] := ∞;
      for k := i to j - 1 do
          m[i, j] := min( m[i, j], p_{i-1} p_k p_j + m[i, k] + m[k + 1, j] );
return m[1, n];
```

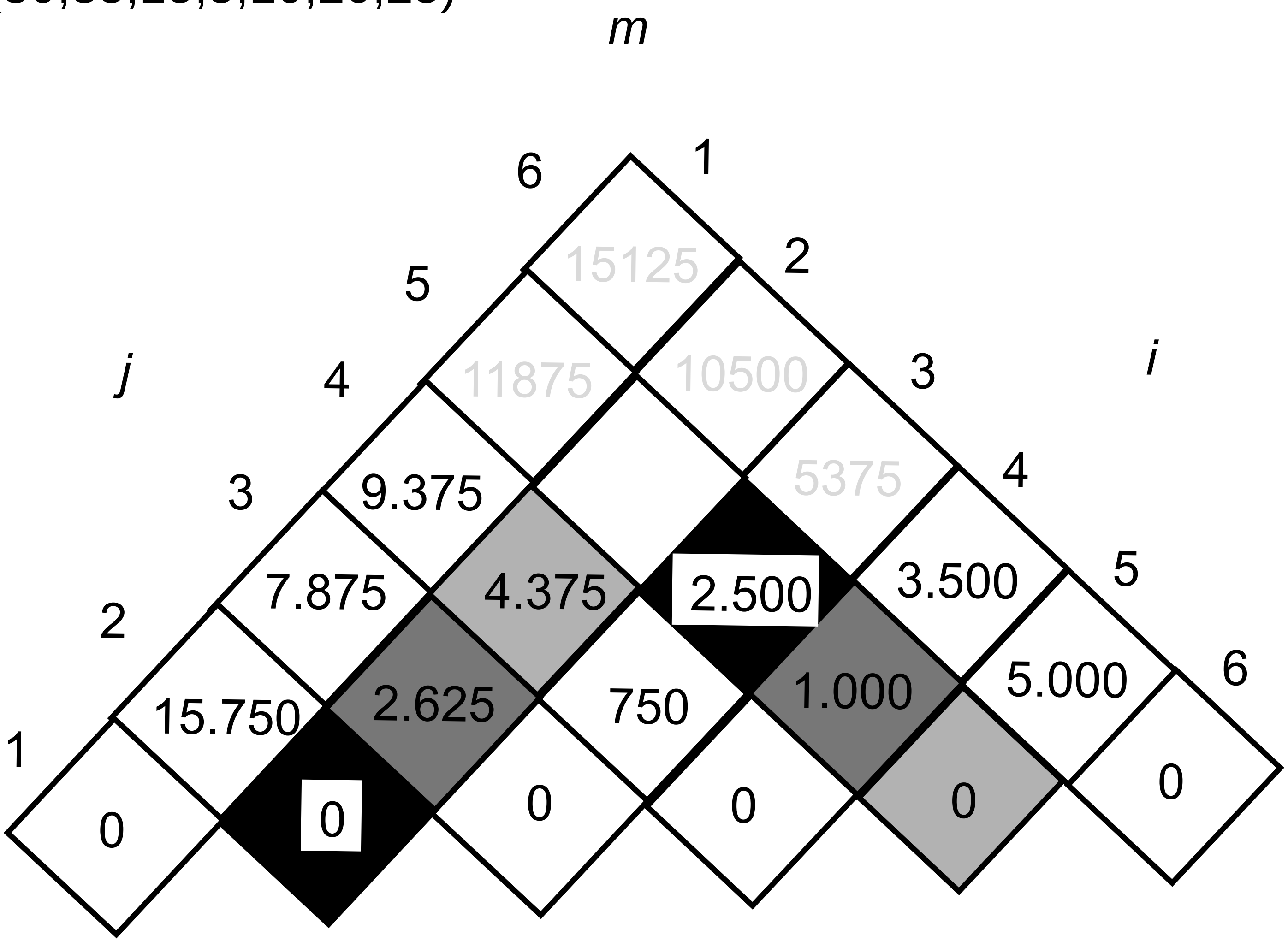
Example

A_1 (30×35)	A_4 (5×10)
A_2 (35×15)	A_5 (10×20)
A_3 (15×5)	A_6 (20×25)

$p = (30,35,15,5,10,20,25)$

Example

$p = (30,35,15,5,10,20,25)$



Example

$$m[2,5] = \min_{2 \le k < 5} \left(m[2,k] + m[k+1,5] + p_1 p_k p_5 \right)$$

$$= \min \begin{cases} m[2,2] + m[3,5] + p_1 p_2 p_5 \\ m[2,3] + m[4,5] + p_1 p_3 p_5 \\ m[2,4] + m[5,5] + p_1 p_4 p_5 \end{cases}$$

$$= \min \begin{cases} 0 + 2500 + 35 \cdot 15 \cdot 20 = 13000 \\ 2625 + 1000 + 35 \cdot 5 \cdot 20 = 7125 \\ 4375 + 0 + 35 \cdot 10 \cdot 20 = 11375 \end{cases}$$

$$= 7125$$

Including optimal split values

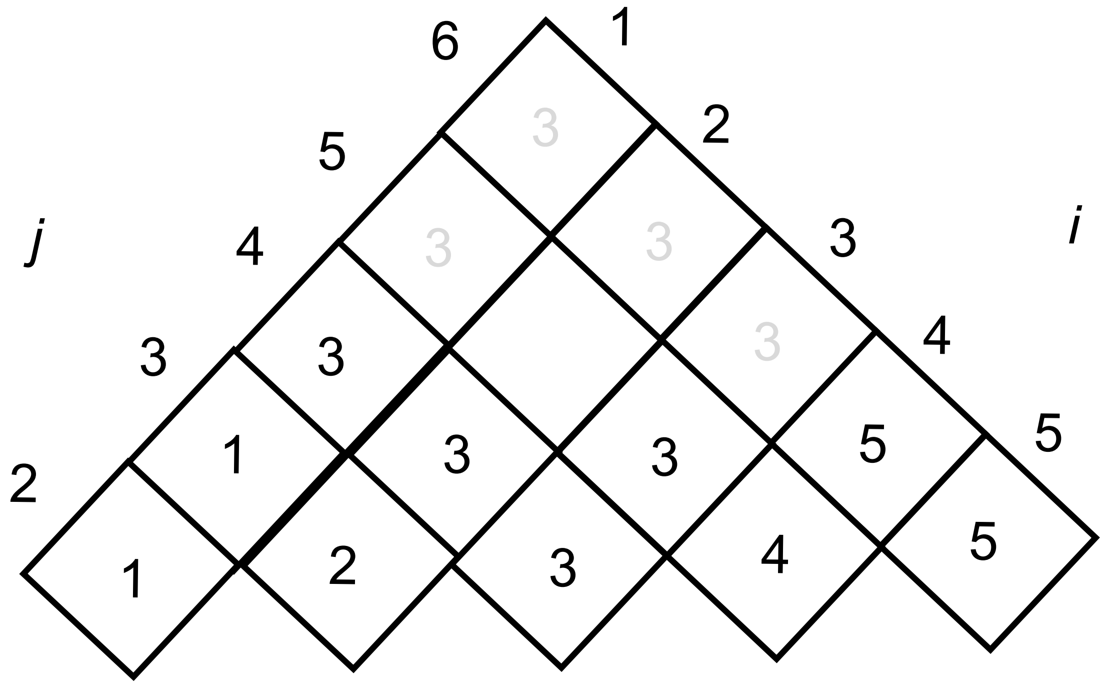
Algorithm *DynMatChain(p)*

Input: Sequence $p = \langle p_0, p_1, \ldots, p_n \rangle$, $(p_{i-1} \times p_i)$ the dimension of matrix A_i

Output: $m[1,n]$ and a matrix $s[i,j]$ containing the optimal split values

```
n := length(p) – 1;
for i := 1 to n do m[i, i] := 0
for l := 2 to n do
    for i := 1 to n – l +1 do
        j := i + l – 1;
        m[i, j] := ∞;
        for k := i to j – 1 do
            q := m[i, j];
            m[i, j] := min( m[i, j],  p_{i-1} p_k p_j + m[i, k] + m[k + 1, j] );
            if m[i, j] < q then s[i, j] := k;
 return (m[1, n], s);
```

Example of splitting values



Computation of an optimal parenthesization

Algorithm *OptParenthesization*

Input: Chain A of matrices, matrix s containing the optimal split values, two indices i and j

Output: An optimal parenthesization of $A_{i...j}$

if $i < j$
  then X := *OptParenthesization*(A, s, i, $s[i, j]$);
    Y := *OptParenthesization*(A, s, $s[i, j] + 1$, j);
    return $(X \cdot Y)$;
  else return A_i;

Initial call: *OptParenthesization*(A, s, 1, n)

Dynamic programming; top-down approach

„Memoization“ for increasing the efficiency of a recursive solution:

Only the *first time* a subproblem is encountered, its solution is computed and then stored in a table. Each subsequent time that the subproblem is encountered, the value stored in the table is simply looked up and returned (without repeated computation!).

Memoized matrix-chain multiplication

Algorithm *MemMatChain*(p, i, j)

Invariant: MemMatChain(p, i, j) returns $m[i, j]$;
the value is correct if $m[i, j] < \infty$

if $i = j$ **then return** 0;
if $m[i, j] < \infty$ **then return** $m[i, j]$;
for $k := i$ **to** $j - 1$ **do**
    $m[i, j] := \min(m[i, j],\ p_{i-1}\, p_k\, p_j +$ MemMatChain(p, i, k) $+$ MemMatChain($p, k + 1, j$));
return $m[i, j]$;

Memoized matrix-chain multiplication

Call:

```
n:= length(p) – 1;
for i := 1 to n do
    for j := 1 to n do
        m[i, j] := ∞;
MemMatChain(p,1,n);
```

The computation of all entries $m[i, j]$ using *MemMatChain* takes $O(n^3)$ time.

$O(n^2)$ entries.

Each entry $m[i, j]$ is computed once.

Each entry $m[i, j]$ is looked up during the computation of $m[i',j']$ if $i' = i$ and $j' > j$ or $j' = j$ and $i' < i$.

Thus $m[i, j]$ is looked up during the computation of at most $2n$ entries.

Remarks about matrix-chain multiplication

1. There is an algorithm that determines an optimal parenthesization in time $O(n \log n)$.

2. There is a linear time algorithm that determines a parenthesization using at most $1.155 \cdot M_{opt}$ multiplications.

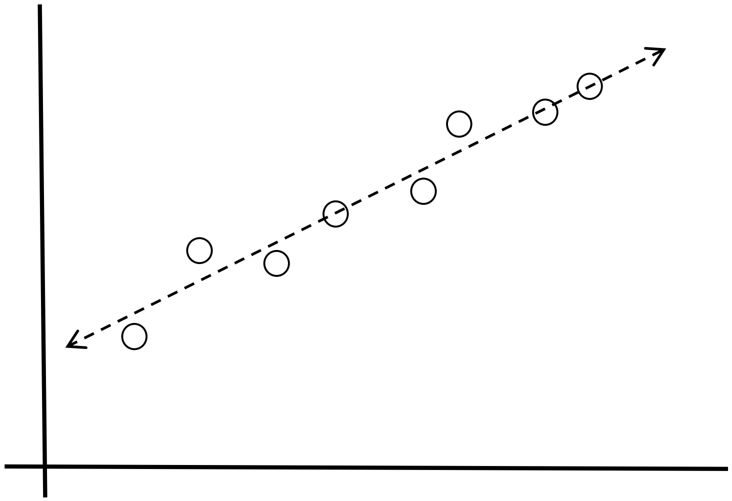
Segmented Least Squares

Problem in statistics and numerical analysis

$P = \{(x_1,y_1), (x_2,y_2),\dots,(x_n,y_n)\}$ $\quad x_1 < x_2 < \dots < x_n$

Find $L = ax+b$ that minimizes the error of L w.r.t. P.

$$Error(L,P) = \Sigma_{1\le i\le n} (y_i - ax_i - b)^2$$



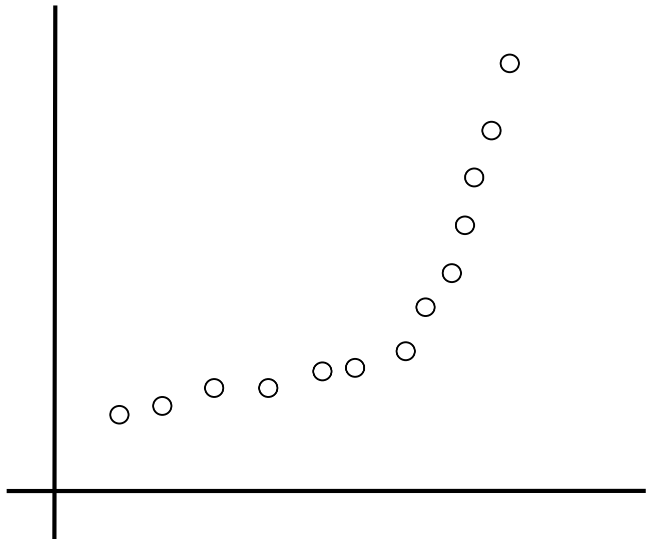
Solution

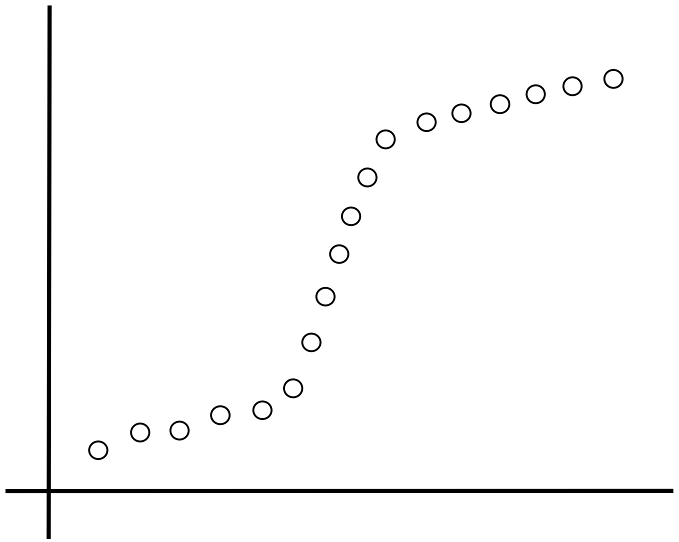
Line $L = ax+b$ where

$$a = \frac{n\sum_i x_i y_i - \left(\sum_i x_i\right)\left(\sum_i y_i\right)}{n\sum_i x_i^2 - \left(\sum_i x_i\right)^2}$$

$$b = \frac{\sum_i y_i - a\sum_i x_i}{n}$$

One line might not suffice





Segmented Least Squares

Problem: $P = \{p_1, p_2, \ldots, p_n\}$ where

$p_i = (x_i, y_i)$ for $i = 1,\ldots,n$ $x_1 < x_2 < \ldots < x_n$

A segment is a subset $\{p_i,\ldots, p_j\}$ where $i \le j$

Partition P into segments. Penalty is the sum of

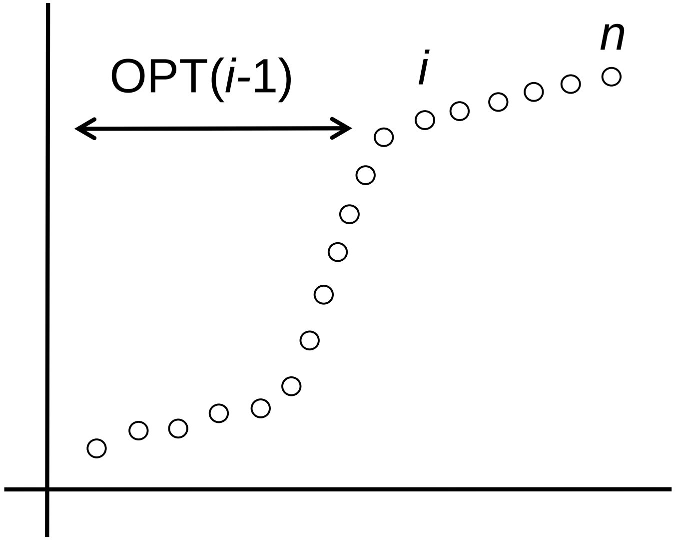
- $C \cdot$ # segments where $C > 0$
- For each segment, the error of the optimum line through it.

Goal: Find partition of minimum penalty.

Dynamic programming approach

Last segment is $\{p_i, \ldots, p_n\}$ for some $1 \leq i \leq n$.

The remaining segments are an optimal solution for $\{p_1, \ldots, p_{i-1}\}$.



Dynamic programming approach

OPT(j) = value of an optimal solution for $\{p_1, \ldots, p_j\}$ $\quad 1 \le j \le n$

OPT(0) := 0

$e_{i,j}$ = minimum error of any line through $\{p_i, \ldots, p_j\}$ $\quad 1 \le i \le j \le n$

$$\text{OPT}(n) = \min_{1 \le i \le n} (e_{i,n} + C + \text{OPT}(i-1))$$

$$\text{OPT}(j) = \min_{1 \le i \le j} (e_{i,j} + C + \text{OPT}(i-1))$$

Dynamic programming algorithm

Array m[0..n] contains the values of the optimal solutions.

Algorithm SegmentedLeastSquares(n)

m[0] := 0;
for all pairs $i \leq j$ **do**
    Compute $e_{i,j}$ for segment $\{p_i,\ldots,p_j\}$;
for j := 1 **to** n **do**
    m[j] := $\min_{1 \leq i \leq j} (e_{i,j} + C + \mathrm{m}[i-1])$;

Running time: $O(n^2)$

Computing a solution

Algorithm FindSegments(j)

if j = 0 **then**
    Output nothing;
else
    Find an i that minimizes $e_{i,j} + C + m[i-1]$;
    Output segment $\{p_i,\ldots,p_j\}$ and the result of FindSegments(i-1);
endif;