# Dynamic Programming: The Matrix Chain Algorithm

Andreas Klappenecker

[partially based on slides by Prof. Welch]

## Matrix Chain Problem

Suppose that we want to multiply a sequence of rectangular matrices. In which order should we multiply?

$$A \times (B \times C) \quad \text{or} \quad (A \times B) \times C$$

# Matrices

An n x m matrix A over the real numbers is a rectangular array of nm real numbers that are arranged in n rows and m columns.

For example, a 3 x 2 matrix A has 6 entries

$$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \end{pmatrix}$$

where each of the entries $a_{ij}$ is e.g. a real number.

# Matrix Multiplication

Let A be an n x m matrix

B an m x p matrix

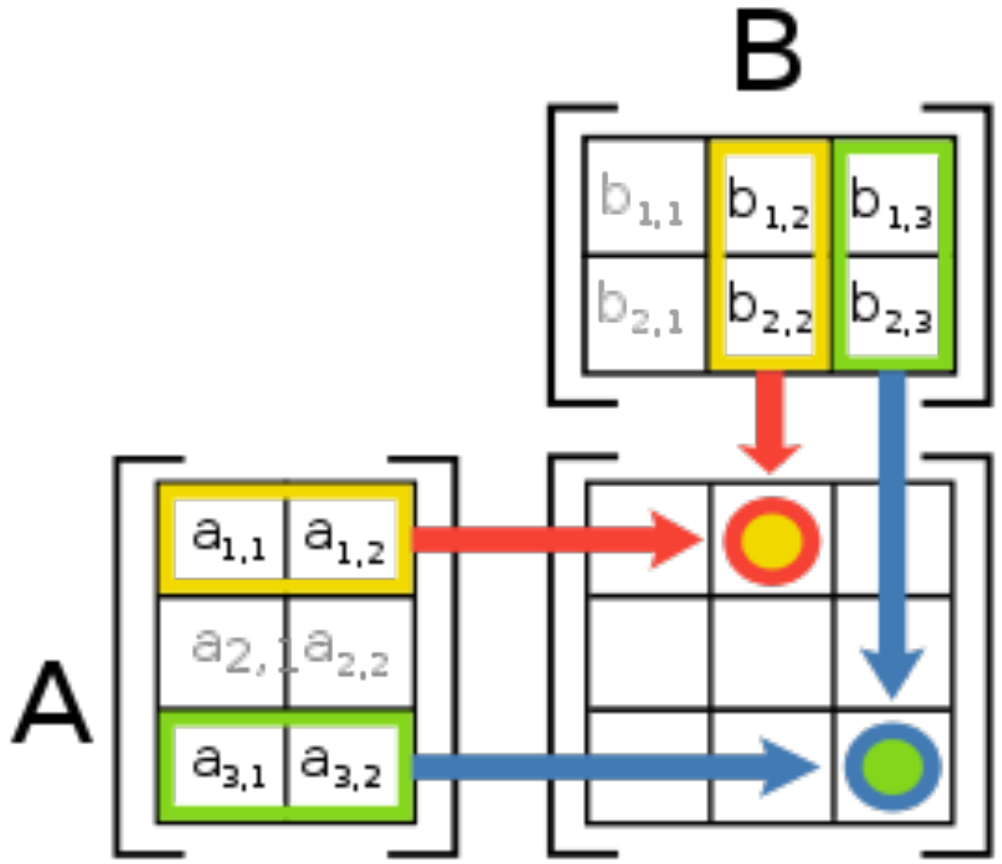
The product of A and B is n x p matrix AB whose (i,j)-th entry is

$$\Sigma_{k=1}^{m} a_{ik} b_{kj}$$

In other words, we multiply the entries of the i-th row of A with the entries of the j-th column of B and add them up.

## Matrix Multiplication

$$\begin{aligned} x_{1,2} &= (a_{1,1}, a_{1,2}) \cdot (b_{1,2}, b_{2,2}) \\ &= a_{1,1}b_{1,2} + a_{1,2}b_{2,2} \\ x_{3,3} &= (a_{3,1}, a_{3,2}) \cdot (b_{1,3}, b_{2,3}) \\ &= a_{3,1}b_{1,3} + a_{3,2}b_{2,3}. \end{aligned}$$



# Complexity of Matrix Multiplication

Let A be an n x m matrix, B an m x p matrix. Thus,

AB is an n x p matrix. Computing the product AB takes

nmp scalar multiplications

n(m-1)p scalar additions

for the standard matrix multiplication algorithm.

## Matrix Chain Order Problem

Matrix multiplication is associative, meaning that (AB)C = A(BC). Therefore, we have a choice in forming the product of several matrices.

What is the least expensive way to form the product of several matrices if the naïve matrix multiplication algorithm is used?

[We use the number of scalar multiplications as cost.]

## Why Order Matters

Suppose we have 4 matrices:

A: 30 x 1

B: 1 x 40

C: 40 x 10

D: 10 x 25

((AB)(CD)) : requires 41,200 scalar multiplications

(A((BC)D)) : requires 1400 scalar multiplications

## Matrix Chain Order Problem

Given matrices $A_1$, $A_2$, ..., $A_n$,

where $A_i$ is a $d_{i-1} \times d_i$ matrix.

[1] What is minimum number of scalar multiplications required to compute the product $A_1 \cdot A_2 \cdot \ldots \cdot A_n$?

[2] What order of matrix multiplications achieves this minimum?

We focus on question [1], and sketch an answer to [2].

## A Possible Solution

Try all possibilities and choose the best one.

Drawback: There are too many of them (exponential in the number of matrices to be multiplied)

We need to be smarter: Let's try dynamic programming!

## Step 1: Develop a Recursive Solution

- Define M(i,j) to be the minimum number of multiplications needed to compute $A_i \cdot A_{i+1} \cdot \ldots \cdot A_j$
- Goal: Find M(1,n).
- Basis: M(i,i) = 0.
- Recursion: How can one define M(i,j) recursively?

# Defining M(i,*j*) Recursively

- Consider all possible ways to split $A_i$ through $A_j$ into two pieces.
- Compare the costs of all these splits:
  - best case cost for computing the product of the two pieces
  - plus the cost of multiplying the two products
- Take the best one
- $M(i,j) = \min_k(M(i,k) + M(k+1,j) + d_{i-1}d_kd_j)$

## Defining M(i,j) Recursively

$$\underbrace{(A_i \cdot \ldots \cdot A_k)}_{P_1} \cdot \underbrace{(A_{k+1} \cdot \ldots \cdot A_j)}_{P_2}$$

- minimum cost to compute $P_1$ is $M(i,k)$
- minimum cost to compute $P_2$ is $M(k+1,j)$
- cost to compute $P_1 \cdot P_2$ is $d_{i-1}d_kd_j$

# Step 2: Find Dependencies Among Subproblems

M:

|   | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 0 |  |  |  |  |
| 2 | n/a | 0 |  |  |  |
| 3 | n/a | n/a | 0 |  |  |
| 4 | n/a | n/a | n/a | 0 |  |
| 5 | n/a | n/a | n/a | n/a | 0 |

# Step 2: Find Dependencies Among Subproblems

M:

|   | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 0 |   |   |   | ◯ |
| 2 | n/a | 0 |   |   |   |
| 3 | n/a | n/a | 0 |   |   |
| 4 | n/a | n/a | n/a | 0 |   |
| 5 | n/a | n/a | n/a | n/a | 0 |

⇐ GOAL!

# Step 2: Find Dependencies Among Subproblems

M:

| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 0 | | | | ○ |
| 2 | n/a | 0 | | 🟥 | |
| 3 | n/a | n/a | 0 | | |
| 4 | n/a | n/a | n/a | 0 | |
| 5 | n/a | n/a | n/a | n/a | 0 |

$\Leftarrow$ GOAL!

## Step 2: Find Dependencies Among Subproblems

M:

| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 0 | | | | ○ |
| 2 | n/a | 0 | | | |
| 3 | n/a | n/a | 0 | | |
| 4 | n/a | n/a | n/a | 0 | |
| 5 | n/a | n/a | n/a | n/a | 0 |

GOAL!

# Step 2: Find Dependencies Among Subproblems

M:

| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 0 | | | | ○ |
| 2 | n/a | 0 | | | |
| 3 | n/a | n/a | 0 | | |
| 4 | n/a | n/a | n/a | 0 | |
| 5 | n/a | n/a | n/a | n/a | 0 |

⇐ GOAL!

computing the pink square requires the purple ones: to the left and below.

## Defining the Dependencies

Computing $M(i,j)$ uses

everything in same row to the left:

$M(i,i), M(i,i+1), \dots, M(i,j-1)$

and everything in same column below:

$M(i,j), M(i+1,j), \dots, M(j,j)$

## Step 3: Identify Order for Solving Subproblems

Recall the dependencies between subproblems just found

Solve the subproblems (i.e., fill in the table entries) this way:

- go along the diagonal
- start just above the main diagonal
- end in the upper right corner (goal)

# Order for Solving Subproblems

M:

|   | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 0 |   |   |   | ○ |
| 2 | n/a | 0 |   |   |   |
| 3 | n/a | n/a | 0 |   |   |
| 4 | n/a | n/a | n/a | 0 |   |
| 5 | n/a | n/a | n/a | n/a | 0 |

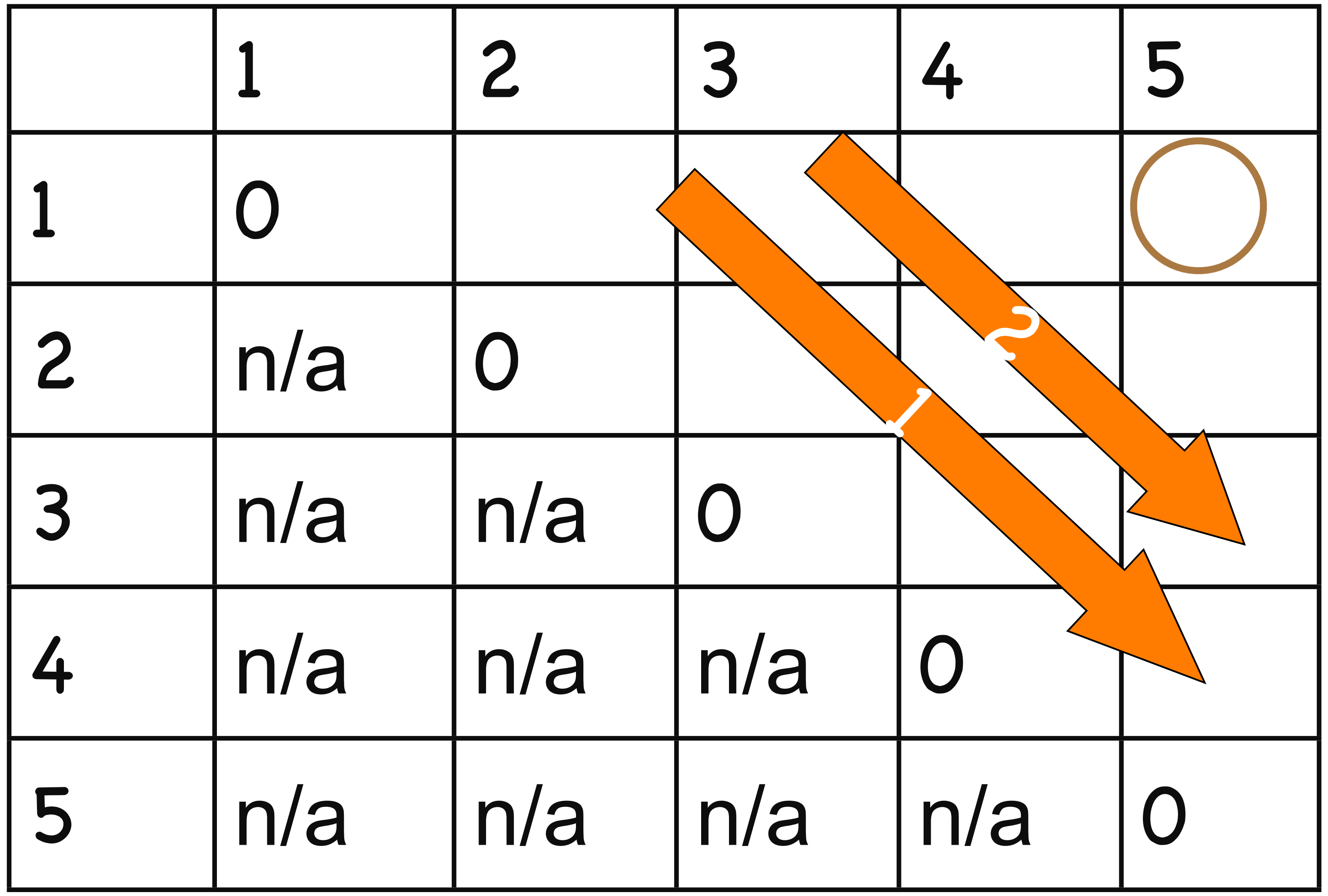
# Order for Solving Subproblems

M:

|  | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 0 |  |  |  |  |
| 2 | n/a | 0 |  |  |  |
| 3 | n/a | n/a | 0 |  |  |
| 4 | n/a | n/a | n/a | 0 |  |
| 5 | n/a | n/a | n/a | n/a | 0 |

# Order for Solving Subproblems

M:

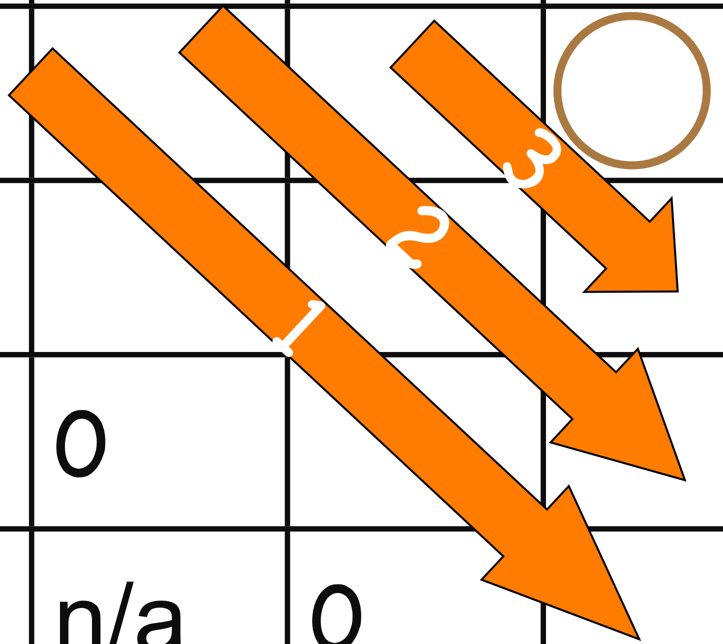


|   | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 0 |  |  |  |  |
| 2 | n/a | 0 |  |  |  |
| 3 | n/a | n/a | 0 |  |  |
| 4 | n/a | n/a | n/a | 0 |  |
| 5 | n/a | n/a | n/a | n/a | 0 |

# Order for Solving Subproblems

M:

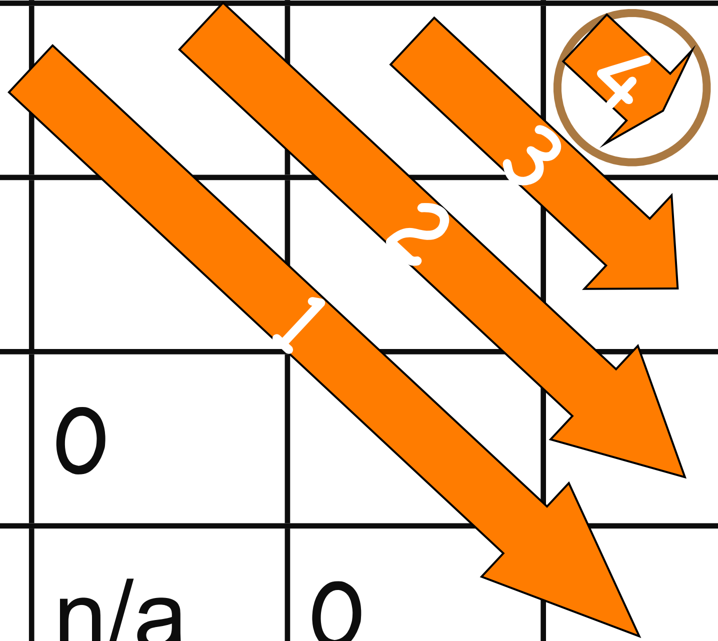
|   | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 0 |   |   |   |   |
| 2 | n/a | 0 |   |   |   |
| 3 | n/a | n/a | 0 |   |   |
| 4 | n/a | n/a | n/a | 0 |   |
| 5 | n/a | n/a | n/a | n/a | 0 |



# Order for Solving Subproblems

M:

| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 0 | | | | |
| 2 | n/a | 0 | | | |
| 3 | n/a | n/a | 0 | | |
| 4 | n/a | n/a | n/a | 0 | |
| 5 | n/a | n/a | n/a | n/a | 0 |



## Pseudocode

```
for i := 1 to n do M[i,i] := 0
for d := 1 to n-1 do  // diagonals
  for i := 1 to n-d to // rows w/ an entry on d-th diagonal
    j := i + d          // column corresp. to row i on d-th diagonal
    M[i,j] := infinity
    for k := i to j-1 to
      M[i,j] := min(M[i,j], M[i,k]+M[k+1,j]+$d_{i-1}d_kd_j$)
    endfor
  endfor
endfor
```

# Pseudocode

```
for i := 1 to n do M[i,i] := 0
for d := 1 to n-1 do  // diagonals
  for i := 1 to n-d to // rows w/ an entry on d-th diagonal
    j := i + d           // column corresp. to row i on d-th diagonal
    M[i,j] := infinity
    for k := i to j-1 to
      M[i,j] := min(M[i,j], M[i,k]+M[k+1,j]+$d_{i-1}d_kd_j$)
    endfor
  endfor
endfor
```

running time $O(n^3)$

# Pseudocode

```
for i := 1 to n do M[i,i] := 0
for d := 1 to n-1 do  // diagonals
  for i := 1 to n-d to // rows w/ an entry on d-th diagonal
    j := i + d          // column corresp. to row i on d-th diagonal
    M[i,j] := infinity
    for k := i to j-1 to
      M[i,j] := min(M[i,j], M[i,k]+M[k+1,j]+$d_{i-1}d_kd_j$)
    endfor
  endfor
endfor
```

pay attention here to remember actual sequence of mults.

running time $O(n^3)$

# Example

M:

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | 0 | 1200 | 700 | 1400 |
| 2 | n/a | 0 | 400 | 650 |
| 3 | n/a | n/a | 0 | 10,000 |
| 4 | n/a | n/a | n/a | 0 |

1: A is 30x1
2: B is 1x40
3: C is 40x10
4: D is 10x25

BxC: 1x40x10

(BxC)xD:
400 + 1x10x25

Bx(CxD):
... + 10,000

## Keeping Track of the Order

- It's fine to know the cost of the cheapest order, but what is that cheapest order?
- Keep another array S and update it when computing the minimum cost in the inner loop
- After M and S have been filled in, then call a recursive algorithm on S to print out the actual order

# Modified Pseudocode

```
for i := 1 to n do M[i,i] := 0
for d := 1 to n-1 do  // diagonals
  for i := 1 to n-d to // rows w/ an entry on d-th diagonal
    j := i + d          // column corresponding to row i on d-th diagonal
    M[i,j] := infinity
    for k := i to j-1 to
      M[i,j] := min(M[i,j], M[i,k]+M[k+1,j]+$d_{i-1}d_kd_j$)
      if previous line changed value of M[i,j] then S[i,j] := k
    endfor
  endfor
endfor
```

keep track of cheapest split point found so far: between $A_k$ and $A_{k+1}$

# Example

M:

S:

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | 0 | $1200_{1}$ | $700_{1}$ | $1400_{1}$ |
| 2 | n/a | 0 | $400_{2}$ | $650_{3}$ |
| 3 | n/a | n/a | 0 | $10{,}000_{3}$ |
| 4 | n/a | n/a | n/a | 0 |

1: A is 30x1
2: B is 1x40
3: C is 40x10
4: D is 10x25

A x (BCD)
A x ((BC) x D)
A x ((BxC) x D)

# Using S to Print Best Ordering

Call Print(S,1,n) to get the entire ordering.

```
Print(S,i,j):
    if i = j then output "A" + i    //+ is string concat
    else
        k := S[i,j]
        output "(" + Print(S,i,k) + Print(S,k+1,j) + ")"
```