Algorithm Theory

Chapter 3
Dynamic Programming

Part II:
Matrix Chain Multiplication

Fabian Kuhn

Dynamic Programming

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

- Only the *first time* a sub-problem 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!).

Dynamic programming / memoization can be applied if

- Optimal solution contains optimal solutions to sub-problems (recursive structure)
- Number of sub-problems that need to be considered is small

Time is at least linear in the number of subproblems.

Computing the solution:

- For each sub-problem, store how the value is obtained (according to which recursive rule).

Matrix-chain multiplication

Given: sequence (chain) $\langle A_1, A_2, \dots, A_n \rangle$ of matrices

Goal: compute the product $A_1 \cdot A_2 \cdot \dots \cdot A_n$

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

- a single matrix
- or the product of two fully parenthesized matrix products, surrounded by parentheses.

Example

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

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

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

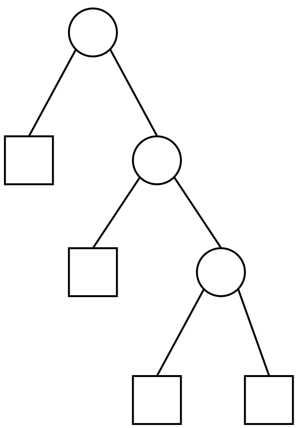
$$((A_1A_2)(A_3A_4))$$

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

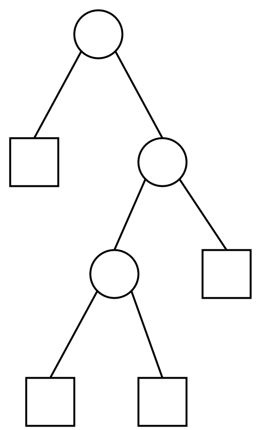
$$(((A_1A_2)A_3)A_4)$$

Different parenthesizations

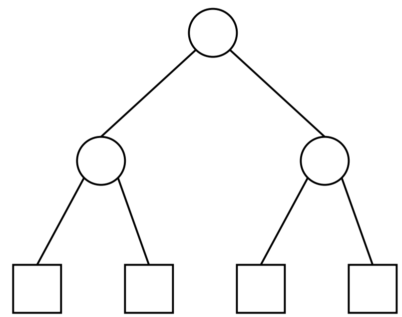
Different parenthesizations correspond to different trees:



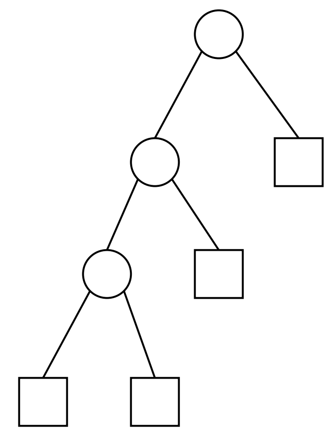
$$\left(A_1\left(A_2(A_3A_4)\right)\right)$$



$$\left(A_1\left((A_2A_3)A_4\right)\right)$$



$$\left((A_1A_2)(A_3A_4)\right)$$



$$\left(\left((A_1A_2)A_3\right)A_4\right)$$

Number of different parenthesizations

- Let $P(n)$ be the number of alternative parenthesizations of the product $A_1 \cdot \ldots \cdot A_n$:

$$P(1) = 1$$

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

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

exponential in n

$$P(n+1) = C_n \qquad (n^{th} \text{ Catalan number})$$

- Thus: Exhaustive search needs exponential time!

Multiplying Two Matrices

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

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

Algorithm *Matrix-Mult*

Input: $(p \times q)$ matrix A, $(q \times r)$ matrix B

Output: $(p \times r)$ matrix $C = A \cdot B$

1 **for** $i := 1$ **to** p **do**
2   **for** $j := 1$ **to** r **do**
3       $C[i,j] := 0;$
4   **for** $k := 1$ **to** q **do**
5          $C[i,j] := C[i,j] + A[i,k] \cdot B[k,j]$

Remark:

Using this algorithm, multiplying two $(n \times n)$ matrices requires n^3 multiplications. This can also be done faster, using only $O(n^{2.373})$ multiplications.

using divide-and-conquer

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

Matrix-chain multiplication: Example

Computation of the product $A_1 A_2 A_3$, where

A_1 : (50 × 5) matrix

A_2 : (5 × 100) matrix

A_3 : (100 × 10) matrix

a) Parenthesization $((A_1 A_2)A_3)$ and $\big(A_1(A_2A_3)\big)$ require:

$A' = (A_1 A_2)$: $50 \cdot 5 \cdot 100 = 25'000$ $\quad A'' = (A_2A_3)$: $5 \cdot 100 \cdot 10 = 5'000$

50×100 $\quad\quad 5 \times 10$

$A'A_3$: $50 \cdot 100 \cdot 10 = 50'000$ $\quad A_1A''$: $50 \cdot 5 \cdot 10 = 2'500$

Sum: $75'000$ $\quad 7'500$

Structure of an Optimal Parenthesization

- $(A_{\ell\ldots r})$: optimal parenthesization of $A_\ell \cdot \ldots \cdot A_r$

  For some $1 \leq k < n$: $(\boldsymbol{A_{1\ldots n}}) = \big((\boldsymbol{A_{1\ldots k}}) \cdot (\boldsymbol{A_{k+1\ldots n}})\big)$

- Any optimal solution contains optimal solutions for sub-problems

- Assume matrix A_i is a $(d_{i-1} \times d_i)$-matrix

- Cost to solve sub-problem $A_\ell \cdot \ldots \cdot A_r, \ell \leq r$ optimally: $C(\ell, r)$

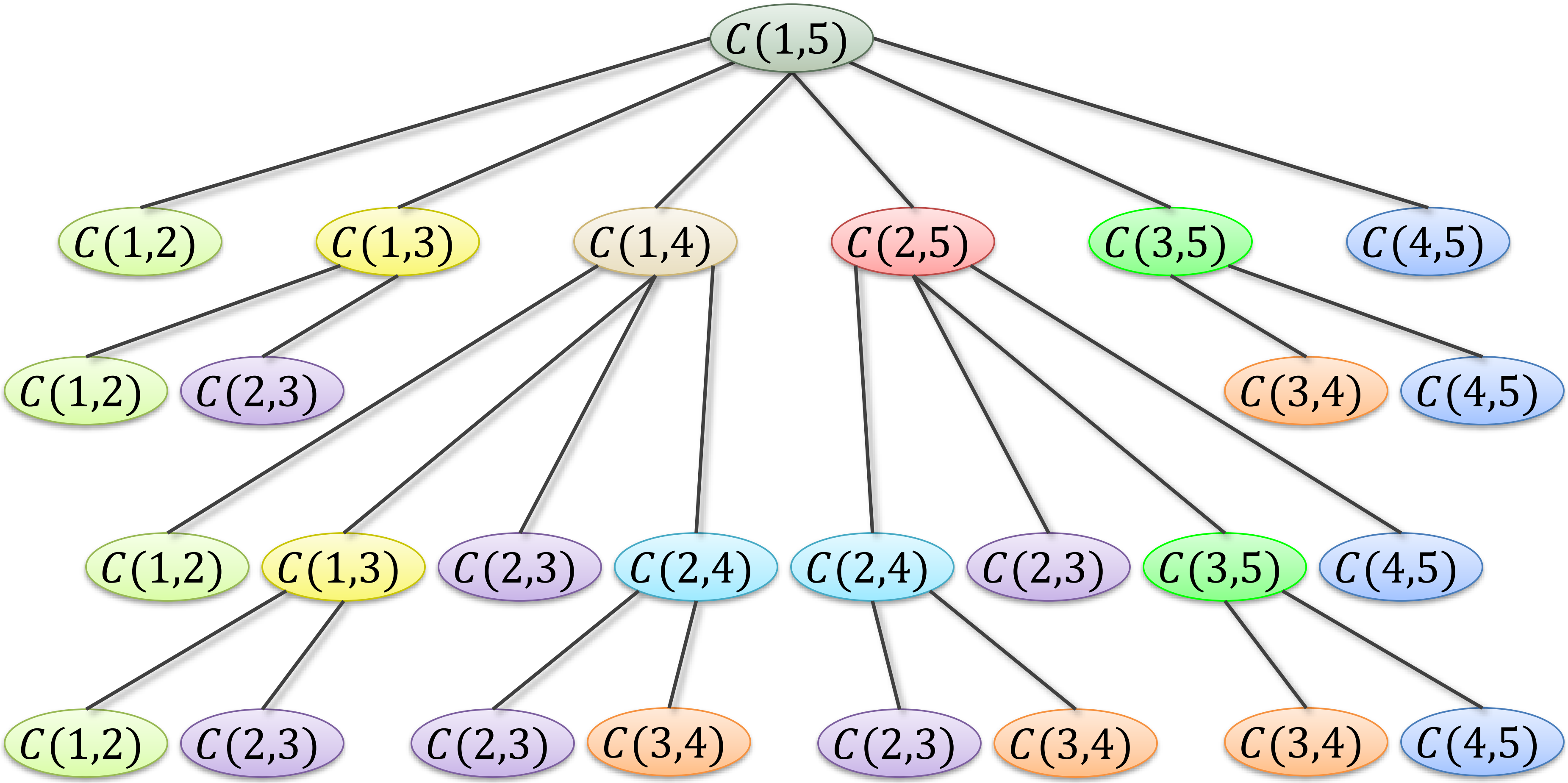
- Then:

$$C(\ell, r) = \min_{\ell \leq k < r} \{C(\ell, k) + C(k+1, r) + d_{\ell-1} d_k d_r\}$$

$$C(\ell, \ell) = 0$$

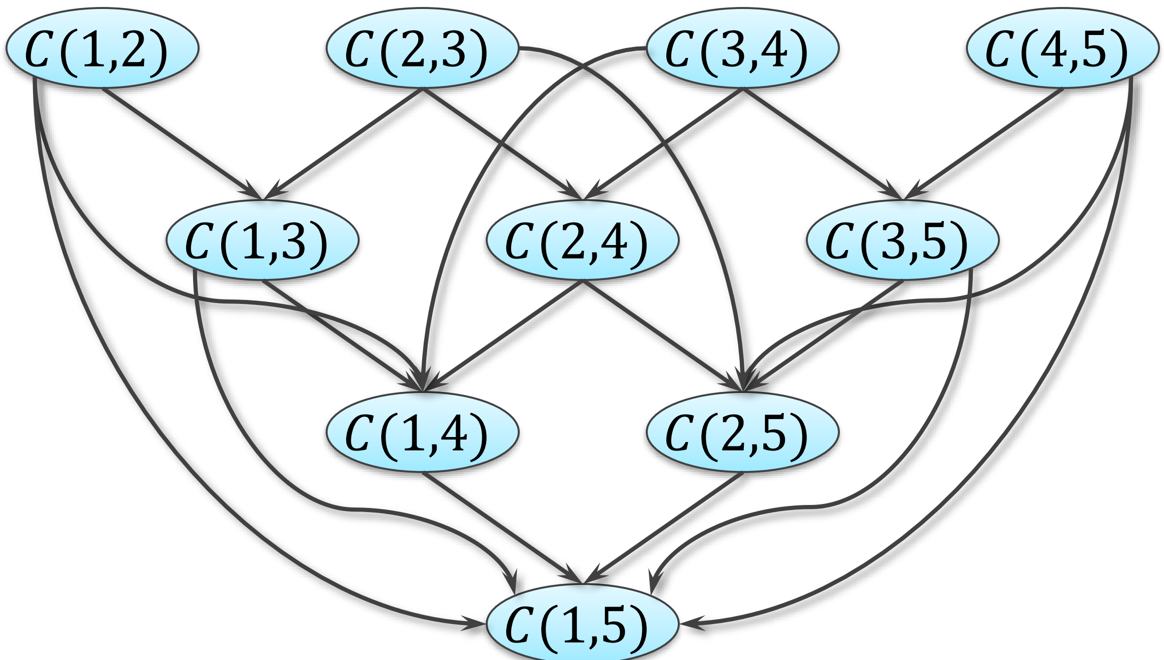
Recursive Computation of Opt. Solution

Compute $A_1 \cdot A_2 \cdot A_3 \cdot A_4 \cdot A_5$:



Using Meomization

Compute $A_1 \cdot A_2 \cdot A_3 \cdot A_4 \cdot A_5$:



Compute $A_1 \cdot \ldots \cdot A_n$:

- Each $C(i,j), i < j$ is computed exactly once → $O(n^2)$ values
- Each $C(i,j)$ dir. depends on $C(i,k), C(k,j)$ for $i < k < j$

Cost for each $C(i,j)$: $O(n)$ → overall time: $\boldsymbol{O(n^3)}$

Remarks about matrix-chain multiplication

1. There is an algorithm that determines an optimal parenthesization in time
$$O(n \cdot \log n).$$
[Hu, Shing; 1980]

2. There is a linear time algorithm that determines a parenthesization using at most
$$1.155 \cdot C(1, n)$$
multiplications.
[Hu, Shing; 1981]