Dynamic Programming Solution to the Matrix-Chain Multiplication Problem

Javed Aslam, Cheng Li, Virgil Pavlu

[this solution follows "Introduction to Algorithms" book by Cormen et al]

Matrix-Chain Multiplication Problem

Given a chain $< A_1, A_2, \ldots, A_n >$ of n matrices, where for $i = 1, 2, \ldots, n$, matrix A_i has dimension $p_{i-1} \times p_i$, fully parenthesize the product $A_1 A_2 \ldots A_n$ in a way that minimizes the number of scalar multiplications.

First we show that exhaustively checking all possible parenthesizations leads to exponential growth of computation. Denote the number of alternative parenthesizations of a sequence of n matrices by $P(n)$. When $n = 1$, we have just one way to fully parenthesize one matrix. When $n \geq 2$, a fully parenthesized matrix product may be splited into two fully parenthesized matrix subproducts, between the kth and $(k+1)$st matrices for some $k = 1, 2, \ldots, n-1$. Thus, we obtain the recurrence

$$P(n) = \begin{cases} 1 & \text{if } n = 1 \\ \sum_{k=1}^{n-1} P(k)P(n-k) & \text{if } n \geq 2 \end{cases}$$

We show that $P(n) \geq 2^{n-2} = \Omega(2^n)$. Clearly, the claim is true for $n = 1, 2$. Assume that the claim is true for each $k < n$.

$$\begin{aligned} P(n) &= \sum_{k=1}^{n-1} P(k)P(n-k) \\ &\geq P(1)P(n-1) + P(n-1)P(1) \\ &= 2P(1)P(n-1) \\ &= 2P(n-1) \\ &\geq 2 \cdot 2^{n-1-2} \\ &= 2^{n-2} \end{aligned}$$

In fact, $P(n) = \Omega(\frac{4^n}{n^{3/2}})$.

Methodology

(1) Characterize the Structure of an Optimal Solution.

Claim 1 *Suppose that in and optimal way to parenthesize $A_i A_{i+1} \ldots A_j$, we split the product between A_k and A_{k+1}. Then the way we parenthesize the "prefix" subchain $A_i A_{i+1} \ldots A_k$ within this optimal parenthesization of $A_i A_{i+1} \ldots A_j$ must be an optimal parenthesization of $A_i A_{i+1} \ldots A_k$. And the way we parenthesize the "suffix" subchain $A_{k+1} A_{k+2} \ldots A_j$ within this optimal parenthesization of $A_i A_{i+1} \ldots A_j$ must be an optimal parenthesization of $A_{k+1} A_{k+2} \ldots A_j$.*

$$(\overbrace{(A_i A_{i+1} \ldots A_k)}^{\text{prefix subchain}}\overbrace{(A_{k+1} A_{k+2} \ldots A_j)}^{\text{suffix subchain}})$$

Proof: By contradiction, suppose there were a less costly way to parenthesize $A_i A_{i+1} \ldots A_k$, then the optimal parenthesization of $A_i A_{i+1} \ldots A_j$ could be replaced with this parenthesization, yielding another way to parenthesize $A_i A_{i+1} \ldots A_j$ whose cost was lower than the optimum: a contradiction. An identical argument applies to the subchain $A_{k+1} A_{k+2} \ldots A_j$ in the optimal parenthesization of $A_i A_{i+1} \ldots A_j$. □

(2) Recursively Define the Value of the Optimal Solution. First, we define in English the quantity we shall later define recursively. Let $A_{i..j}$ be the matrix that results from evaluating the product $A_i A_{i+1} \ldots A_j$, where $i \leq j$. Let $C[i,j]$ be the minimum number of scalar multiplications needed to compute the matrix $A_{i...j}$. If $i = j$, no scalar multiplications are needed. Thus, $C[i,i] = 0$ for $i = 1, 2, \ldots, n$. When $i < j$, we assume that in an optimal parenthesization, the product $A_i A_{i+1} \ldots A_j$ is split between A_k and A_{k+1}, where $i \leq k < j$. Then $C[i,j] = C[i,k] + C[k+1,j] + p_{i-1}p_k p_j$. We don't know the value of k in the optimal parenthesization, but there are only $j - i$ possible values for k, namely $k = i, i+1, \ldots, j+1$. So we should check all these values to find the best. We thus have the following recurrence.

Claim 2

$$C[i,j] = \begin{cases} 0 & \text{if } i = j, \\ \min_{i \leq k < j}\{C[i,k] + C[k+1,j] + p_{i-1}p_k p_j\} & \text{if } i < j. \end{cases}$$

Proof: The correctness of this recursive definition is embodied in the paragraph which precedes it. □

(3) Compute the Value of the Optimal Solution Bottom-up. Consider the following piece os pseudocode, where $p =< p_0, p_1, \ldots, p_n >$, with $p.length = n + 1$, table $C[1..n, 1..n]$ stores the costs and table $S[1..n-1, 2..n]$ stores which index of k achieved the optimal cost in computing $C[i,j]$.

The way we fill in the tables C and S here is more tricky than that in the knapsack problem and checkboard problem. In knapsack problem and checkboard problem, we fill in the tables by rows. Here, the recurrence formula shows that the cost $C[i,j]$ of computing a matrix-chain product of $j - i + 1$ matrices depends only on the costs of computing matrix-chain products of fewer than $j - i + 1$ matrices. Thus, the tables C and S should be filled in by increasing lengths of the matrix chains. This corresponds to filling in the tables diagonally.

```
MATRIX-CHAIN-ORDER(p)
1  n = p.length − 1
2  let C[1..n, 1..n] and S[1..n − 1, 2..n] be new tables
3  for i = 1 to n
4      C[i, i] = 0
5  for l = 2 to n //l is the chain length
6      for i = 1 to n − l + 1
7          j = i + l − 1
8          C[i, j] = ∞
9          for k = i to j − 1
10             q = C[i, k] + C[k + 1, j] + p_{i−1} p_k p_j
11             if q < C[i, j]
12                 C[i, j] = q
13                 S[i, j] = k
14 return C and S
```

Claim 3 *When the above procedure terminates, $C[i,j]$ will contain the minimum number of scalar multiplications needed to compute the matrix $A_{i...j}$, and $S[i,j]$ will contain the index of k achieved the optimal cost in computing $C[i,j]$.*

Proof: The correctness of the above procedure is based on the fact that it correctly implements the recursive definition given above. The base case is properly handled in Lines 3-4, and the recursive case is properly handled in Lines 5-13. At each step, the $C[i,j]$ cost computed in lines 10-13 depends only on table entries $C[i,k]$ and $C[k+1,j]$ already computed. Lines 8-12 correctly compute $\min_{i \leq k < j}\{C[i,k] + C[k+1,j] + p_{i-1}p_k p_j\}$, and $C[i,j]$ is set to this value in Line 12. Lines 8-13 correctly compute $\arg\min_{i \leq k < j}\{C[i,k] +$

$C[k+1, j] + p_{i-1}p_kp_j\}$, and $S[i, j]$ is set to this value in Line 13. □

Figure 1 illustrates this procedure on a chain of $n = 6$ matrices.

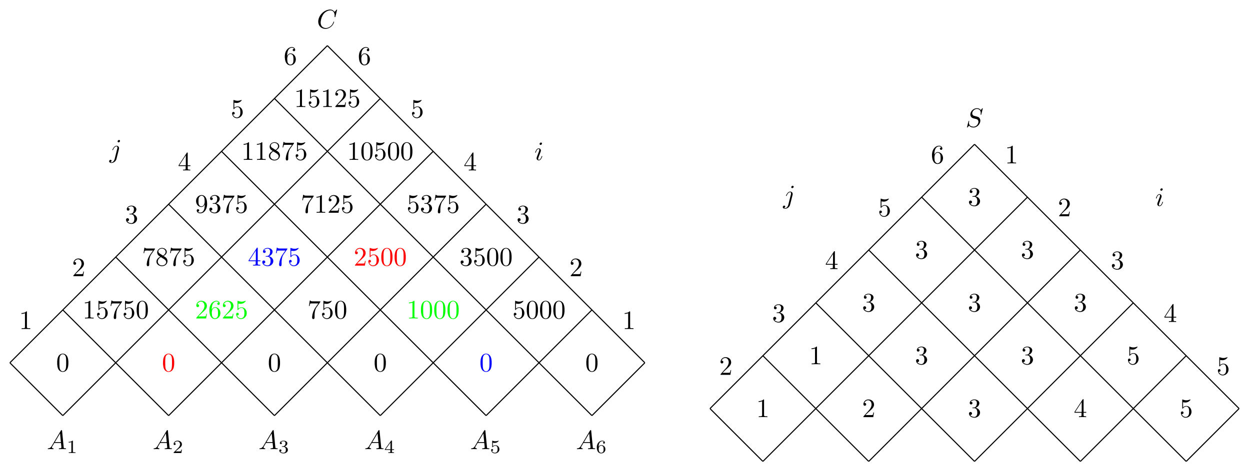
(4) Construct the Optimal Solution from the Computed Information. Consider the following piece of pseudocode, where S is the table computed by MATRIX-CHAIN-ORDER.

```
PRINT-OPTIMAL-PARENS(S, i, j)
if i == j
    print "A"_i
else print "("
    PRINT-OPTIMAL-PARENS(S, i, S[i, j])
    PRINT-OPTIMAL-PARENS(S, S[i, j] + 1, j)
print ")"
```

Claim 4 *The above procedure correctly prints an optimal parenthesization of* $< A_i, A_{i+1}, \ldots, A_j >$.

Proof: $S[i, j]$ indicates the value of k such that an optimal parenthesization of $A_iA_{i+1} \ldots A_j$ splits the product between A_k and A_{k+1}. The above procedure just recursively splits the parenthesization of a chain into the parenthesization of its prefix chain and the parenthesization of its suffix chain. □

(5) Running Time and Space Requirements. The procedure MATRIX-CHAIN-ORDER runs in $O(n^3)$ due to the nested loop defined in Lines 5, 6 and 9. We can also show that the running time of this algorithm is in fact also $\Omega(n^3)$ (exercise). The algorithm requires $\Theta(n^2)$ space to store the C and S tables. The procedure PRINT-OPTIMAL-PARENS runs in $\Theta(n)$ time and uses no additional space. The overall running time is $\Theta(n^3)$ and the space requirement is $\Theta(n^2)$.



C

j \ i	1	2	3	4	5	6
6	15125	10500	5375	3500	5000	0
5	11875	7125	2500	1000	0	
4	9375	4375	750	0		
3	7875	2625	0			
2	15750	0				
1	0					

S

j \ i	1	2	3	4	5	6
6	3	3	3	5	5	
5	3	3	3	4		
4	3	3	3			
3	1	2				
2	1					
1						

Figure 1: The C and S table computed by MATRIX-CHAIN-ORDER for $n = 6$ and the following matrix dimensions:

matrix	A_1	A_2	A_3	A_4	A_5	A_6
dimension	30×35	35×15	15×5	5×10	10×20	20×25

The tables at the top are rotated to make the main diagonal runs horizontally. The C table uses only the main diagonal and upper triangle, since $C[i, j]$ is only defined for $i \leq j$. The S table uses only the upper triangle. Of the colored entries, the pairs that have the same color are taken together in line 10 when computing

$$C[2,5] = \min \begin{cases} C[2,2] + C[3,5] + p_1 p_2 p_5 = 0 + 2500 + 35 \cdot 15 \cdot 20 = 13,000 \\ C[2,3] + C[4,5] + p_1 p_3 p_5 = 2625 + 1000 + 35 \cdot 5 \cdot 20 = 7125 \\ C[2,4] + C[5,5] + p_1 p_4 p_5 = 4375 + 0 + 35 \cdot 10 \cdot 20 = 11,375 \end{cases}$$
$$= 7125.$$

The tables at the bottom are normally orientated. The tables are filled in by diagonals.