Multiplying a sequence of matrices

Bangyao Zhao, Zilin Wang, Zizheng Zhang

University of Michigan

4/20/2022

Problem Setup

- Let $M_1, M_2, ..., M_n$ be n matrices of dimensions $d_1 \times d_2, d_1 \times d_2, ..., d_n \times d_{n+1}$
- Assume multiplying a $p \times q$ matrix with a $q \times r$ matrix is pqr.
- **Goal:** Find the minimum cost for evaluate the product.

An Illustrative Example

- Suppose we have four matrices M_1, M_2, M_3, M_4 of dimensions $100 \times 1, 1 \times 50, 50 \times 20,$ and $20 \times 10.$
- Evaluating their product in the left-to right order, i.e. $((M_1 \times M_2) \times M_3) \times M_4$, costs 125,000.
- Evaluating in the optimal order, i.e. $M_1 \times ((M_2 \times M_3) \times M_4)$, is only 2,200.

Dynamic Programming: First Attempt

Defining $c(i,j)$ to be the minimum cost for evaluating $M_i, M_2, ..., M_j$, then the recurrence relationship is,

$$c(i,j) = \min_{i<k\leq j}[c(i,k-1) + c(k,j) + d_i d_k d_{j+1}]$$

This gives an $O(n^3)$ algorithm for computing $c(1,n)$.

Geometric Representation for the Problem

Yao, 1982 proposed an equivalent geometric representation for the problem using an $(n+1)$-sided convex polygon[1].

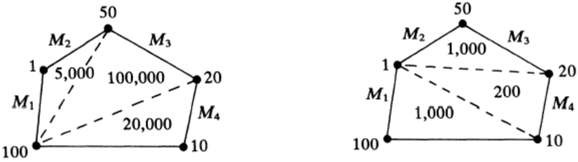


FIG. 2. *Geometric representation for the evaluation of a matrix chain.*

Some Properties of the Optimal Partition

Let $w_1 \leq w_2 \leq \ldots \leq w_{n+1}$ be the sorted d_i's in the non-decreasing order. Yao, 1982 discovered the lemma about the optimal partition, which significantly reduced the range of possible partitions to search.

LEMMA 1. *Let P be an n-gon with weights $w_1 \leqq w_2 \leqq \cdots \leqq w_n$. Then there exists an optimal partition π for which the following is true.*

(a) *w_1 and w_2 are adjacent (either by a side edge or by a chord); similarly for w_1 and w_3.*

(b) *if both w_1w_2 and w_1w_3 are side edges, then either w_1w_4 or w_2w_3 exists as a chord.*

Yao's Algorithm: Worst Case $O(c^n)$ Version

PROCEDURE Partition $[P]$
begin
if $|P| = 1$ or 2 **then return** $\varnothing$
else
if P is a triangle **then return** P
else
if w_1 and w_2 are not adjacent **then return** Partition $[P_{1,2}] \cup$ Partition $[P_{2,1}]$
else
if w_1 and w_3 are not adjacent **then return** Partition $[P_{1,3}] \cup$ Partition $[P_{3,1}]$
else
return better of {Partition $[P_{2,3}] \cup$ Partition $[P_{3,2}]$,
Partition $[P_{1,4}] \cup$ Partition $[P_{4,1}]$};
end.

Worst case exponentially complicated, but in practice it is faster than the $O(n^3)$ complicated algorithm.

Yao's Algorithm: Guaranteed $O(n^2)$ Version

DEFINITION. A (directed) chord $w_i w_j$ of P is called a *bridge*, if all weights w_k in P_{ij} satisfy $k \geqq \max\{i, j\}$.

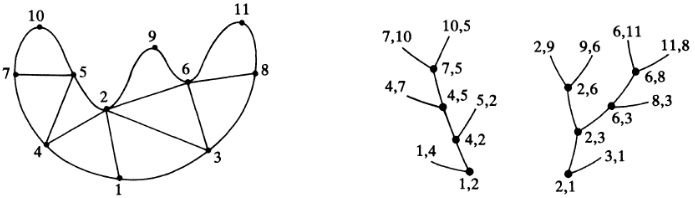


FIG. 3. *A polygon P and the corresponding forest T[<]. (The weights are represented by their indices only.)*

Yao's Algorithm: Guaranteed $O(n^2)$ Version

We can easily verify that the bridges must satisfy the following properties:

1. Two bridges never intersect (except possibly at the end points) $\implies \#$ of bridges $\leq O(n)$
2. Any nonleaf node $w_i w_j$ has exactly two sons, namely $w_i w_j$ and $w_k w_j$ where k is the smallest index in P_{ij} (aside from i, j). Assuming $i < j$, we refer to them as the *minson* and *maxson* of $w_i w_j$, respectively.

Yao's Algorithm: Guaranteed $O(n^2)$ Version

```
PROCEDURE MarkBridges [P];
begin
    find the minimum weight w₁;
    w ← w₁;
    repeat
        begin
            S ⇐ w;
            w ← nextweight;                       —Going clockwise from w₁.
            while top(S) > w do
                begin t ⇐ S;
                    output (top(S), t) and (t, w) as bridges;
                end;
        end
    until w = w₁;                                 —Halt after returning to w₁.
end.
```

Yao's Algorithm: Guaranteed $O(n^2)$ Version

DEFINITION. A subpolygon Q of P is called a *cone*, if $Q = P_{ij} \cup w_i w_j w_k$ where $b = w_i w_j$ is a bridge of P, and $k \leqq \min\{i, j\}$. We also denote a cone Q by (b, w_k)

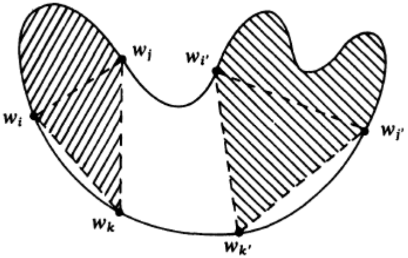


FIG. 4. *Example of two cones (the shaded regions).*

Yao's Algorithm: Guaranteed $O(n^2)$ Version

```
PROCEDURE DP-Partition [P]
begin
    for b = w_i w_j ∈ B do                 —B is the output of Markbridges [P].
    begin                                  —Assume that i < j.
        if b is a leaf then
            for all cones Q = (b, w_k) with k ≦ i do
                if w_k = w_i then Partition [Q] ← ∅                 —Q = w_i w_j
                          else Partition [Q] ← Q;                   —Q = w_i w_j w_k
        if b is not a leaf then
            for all cones Q = (b, w_k) with k ≦ i do
                if w_k = w_i then Partition [Q] ← Partition[(minson (b), w_i)] ∪
                                                  Partition[(maxson (b), w_i)]
                          else Partition [Q]
                          ← better of {Partition [(b, w_i)] ∪ w_i w_j w_k,
                                       Partition[(minson (b), w_k)] ∪
                                       Partition [(maxson (b), w_k)]};
    end;
    Partition [P] ← Partition [P_{1,2}] ∪ Partition [P_{2,1}];
end.
```

The algorithm runs in $O(n^2)$ time because there are at most $2n$ bridges, and at most n cones for a given bridge.

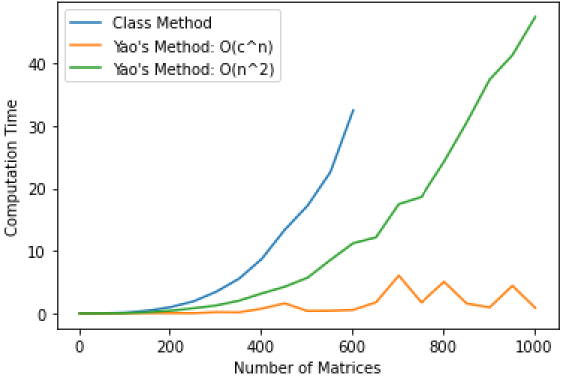
1. Introduction
2. Method
3. Implementation
4. **Simulation**

Correctness of Implementation

- Generated $d_1, d_2, ..., d_{n+1}$ using independent $Poi(20)$ distributions with $n = 20$.
- Verified the three algorithms had the same output
- Repeated this procedure for 1000 times

Simulation Study

- Increased n up to 1000 and measured the computation time of the three algorithms.
- Set time limit to be 30 s for all algorithms



[1] F. F. Yao. Speed-up in dynamic programming[J]. SIAM Journal on Algebraic Discrete Methods, 1982, 3(4): 532-540

Thanks!