Matrix Chain Multiplication using Dynamic Programming

Đỗ Thành Đạt

Ngày 20 tháng 1 năm 2025

Tóm tắt nội dung

Matrix Chain Multiplication (MCM) là một bài toán tối ưu hóa nổi bật trong lĩnh vực lập trình động. Nhiệm vụ là tìm cách nhóm các ma trận trong một chuỗi sao cho tổng số phép toán nhân cần thiết là tối thiểu. Bài viết này trình bày tổng quan về bài toán, động lực nghiên cứu, các khái niệm liên quan, ví dụ minh họa từng bước và tập trung vào các giải pháp Dynamic Programming (DP). Ngoài ra, thuật toán Hu và Shing, với độ phức tạp $O(n \log n)$, cũng được giới thiệu và phân tích như một giải pháp tiên tiến.

1 Giới thiệu

Matrix Chain Multiplication (MCM) là bài toán tìm thứ tự nhân tối ưu cho một chuỗi các ma trận, nhằm giảm số phép toán nhân ma trận. Để nhân hai ma trận $A(p \times q)$ và $B(q \times r)$, cần thực hiện $p \times q \times r$ phép toán. Khi nhân nhiều ma trận, thứ tự nhóm chúng ảnh hưởng lớn đến tổng số phép toán. Do đó, mục tiêu là tìm thứ tự nhân để **tối thiểu hóa** chi phí tính toán.

1.1 Ý nghĩa

Bài toán MCM là một ví dụ điển hình về ứng dụng của Dynamic Programming trong tối ưu hóa. Nó có liên hệ chặt chẽ với các vấn đề về tối ưu hóa chuỗi, như nhân đa thức, chia đoạn, và các thuật toán sắp xếp thứ tự ưu tiên trong máy tính. Việc tối ưu hóa nhân ma trận đặc biệt quan trọng trong các lĩnh vực như xử lý tín hiệu, đồ họa máy tính và học sâu (deep learning).

1.2 Đặc điểm chính

- **Không thay đổi thứ tự ma trận**: Thứ tự nhân phải giữ nguyên thứ tự xuất hiện của ma trận trong chuỗi.
- **Tối ưu hóa phép toán nhân**: Chỉ cần thay đổi cách nhóm (đặt dấu ngoặc) giữa các ma trận để tối thiểu hóa số phép toán.
- **Ứng dụng Dynamic Programming**: Giải pháp đệ quy cơ bản có độ phức tạp hàm mũ, nhưng có thể giảm xuống $O(n^3)$ với DP.

2 Lý do nghiên cứu bài toán

2.1 Phát biểu Bài toán

Cho một dãy n ma trận $A_1, A_2, \ldots, A_n$ với kích thước $A_i = p_{i-1} \times p_i$. Nhiệm vụ là xác định thứ tự nhân (đặt dấu ngoặc) để tối thiểu hóa tổng số phép toán cần thiết để nhân tất cả các ma trận.

2.2 Ứng dụng Thực tế

- **Học sâu (Deep Learning)**: Trong các mạng nơ-ron, việc tối ưu thứ tự nhân các ma trận trọng số giúp tăng tốc huấn luyện.
- **Đồ họa máy tính**: Các phép biến đổi hình học sử dụng ma trận có thể được tối ưu để giảm thời gian kết xuất.
- **Tính toán phân tán**: Trong các hệ thống phân tán, tối ưu hóa nhân ma trận giúp giảm chi phí giao tiếp giữa các nút.

3 Giải pháp Dynamic Programming

3.1 Công thức truy hồi

Hàm $M[i][j]$ biểu diễn chi phí tối thiểu để nhân các ma trận từ A_i đến A_j. Công thức truy hồi là:

$$M[i][j] = \begin{cases} 0, & \text{nếu } i = j; \\ \min\limits_{i \le k < j} \big(M[i][k] + M[k+1][j] + p_{i-1} \cdot p_k \cdot p_j\big), & \text{nếu } i < j. \end{cases}$$

3.2 Thuật toán Dynamic Programming sử dụng mảng tra cứu

Algorithm 1 Dynamic Programming cho Matrix Chain Multiplication

Require: $p[0..n]$: Kích thước các ma trận
Ensure: Chi phí tối thiểu để nhân tất cả các ma trận

```
function MatrixChainOrder(p)
    n ← length(p) − 1
    M ← 2D array of size n × n, initialized to 0
    for l = 2 → n do                                  ▷ Chiều dài đoạn từ 2 đến n
        for i = 1 → n − l + 1 do
            j = i + l − 1
            M[i][j] ← ∞
            for k = i → j − 1 do
                q ← M[i][k] + M[k + 1][j] + p[i − 1] · p[k] · p[j]
                M[i][j] ← min(M[i][j], q)
            end for
        end for
    end for
    return M[1][n]
end function
```

3.3 Độ phức tạp của thuật toán

Thuật toán Dynamic Programming cho bài toán *Matrix Chain Multiplication* có độ phức tạp như sau:

1. **Vòng lặp chiều dài đoạn (l):** Chạy từ 2 đến n, nên có $n - 1$ vòng lặp.

2. **Vòng lặp bắt đầu (i):** Với mỗi l, i chạy từ 1 đến $n - l + 1$, tức tối đa n vòng lặp.
3. **Vòng lặp điểm tách (k):** Với mỗi đoạn, k chạy từ i đến $j - 1$, tức tối đa $l - 1$ vòng lặp.

Như vậy, tổng số lần lặp của thuật toán là:

$$\sum_{l=2}^{n} \sum_{i=1}^{n-l+1} \sum_{k=i}^{j-1} 1$$

Số lần tính toán q: Xấp xỉ $O(n^3)$, vì l, i, và k đều phụ thuộc tuyến tính vào n.

Độ phức tạp thời gian:

$$O(n^3)$$

Độ phức tạp không gian: - Thuật toán sử dụng một bảng $M[i][j]$ kích thước $n \times n$, do đó độ phức tạp không gian là:

$$O(n^2)$$

4 Mở rộng: Thuật toán của Hu và Shing – Giải pháp $O(n \log n)$

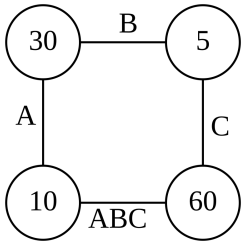
Thuật toán do **T. C. Hu** và **M.-T. Shing** phát triển là một bước đột phá trong việc giải quyết bài toán Matrix Chain Multiplication. Ý tưởng chính của thuật toán là chuyển bài toán từ không gian ma trận sang không gian hình học, cụ thể là bài toán **chia đa giác đều thành các tam giác (triangulation)**, từ đó giảm độ phức tạp xuống $O(n \log n)$.

4.1 Ý tưởng chính

- **Biểu diễn bài toán bằng đa giác đều**: Với n ma trận cần nhân, ta biểu diễn bài toán dưới dạng một đa giác đều có $n + 1$ đỉnh:
  - Các **cạnh** của đa giác biểu diễn các ma trận trong chuỗi.
  - Các **đỉnh** chứa thông tin về kích thước ma trận.
  - **Cạnh đáy** (base) biểu diễn kết quả cuối cùng của phép nhân.
- **Tương quan với tam giác hóa (triangulation)**:
  - Một cách đặt dấu ngoặc cho chuỗi nhân ma trận tương ứng với một cách chia đa giác thành các tam giác.
  - Mỗi tam giác đại diện cho một phép nhân ba ma trận với chi phí là:

    $$\text{Cost}(i, j, k) = p_i \cdot p_k \cdot p_j,$$

    với p_i, p_k, p_j là kích thước tại ba đỉnh của tam giác.
  - Chi phí tổng của một cách tam giác hóa là tổng chi phí của tất cả các tam giác.
- **Tối ưu hóa bằng chia để trị**: Thuật toán sử dụng phương pháp chia để trị để chia đa giác thành các phần nhỏ hơn, sau đó tìm cách tam giác hóa tối ưu.



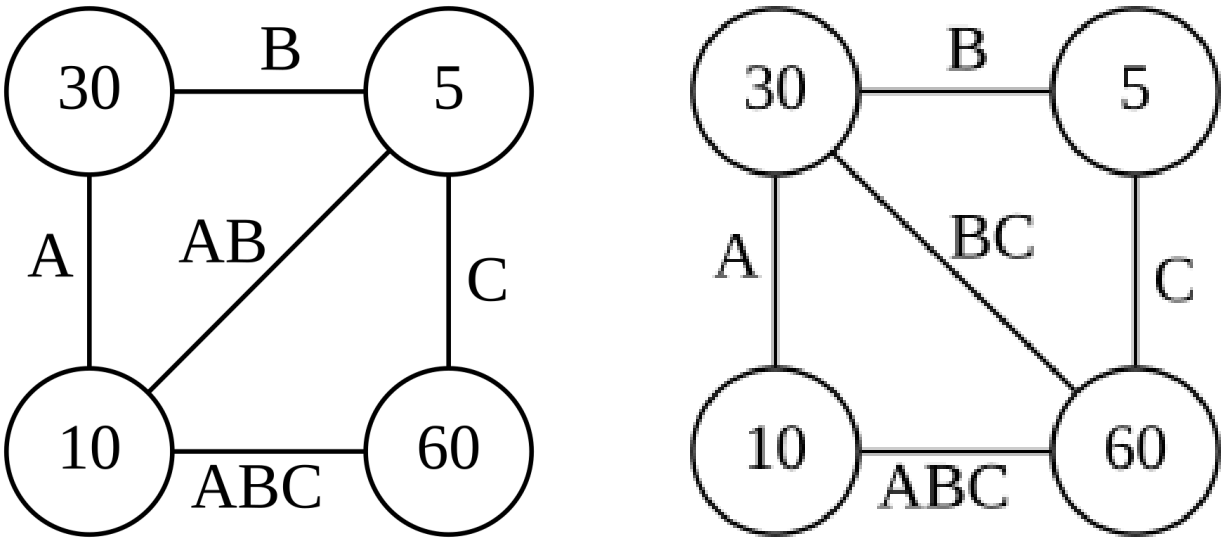
Hình 1: Ảnh minh họa biểu diễn bằng đa giác - Nguồn: Wikipedia

4.2 Ví dụ Minh họa

Xét ví dụ với $A(10 \times 30)$, $B(30 \times 5)$, $C(5 \times 60)$:

1. **Biểu diễn bằng đa giác**: Đa giác là một hình vuông (4-gon) với các đỉnh: $10, 30, 5, 60$. Cạnh đáy là kết quả cuối cùng (ABC).
2. **Cách tam giác hóa**: Có hai cách chia đa giác thành tam giác:
   - $(AB)C$:
   $$\text{Chi phí} = (10 \times 30 \times 5) + (10 \times 5 \times 60) = 1500 + 3000 = 4500.$$
   - $A(BC)$:
   $$\text{Chi phí} = (30 \times 5 \times 60) + (10 \times 30 \times 60) = 9000 + 18000 = 27000.$$

   Rõ ràng, cách tam giác hóa $(AB)C$ hiệu quả hơn.



(a) Cách (AB)C (b) Cách A(BC)

Hình 2: So sánh 2 cách tam giác hóa - Nguồn: Wikipedia

4.3 Thuật toán Hu và Shing

Thuật toán được thực hiện theo các bước:

Algorithm 2 Thuật toán Hu và Shing cho Matrix Chain Multiplication (Chia để trị)

Require: $p[0..n]$: Kích thước các ma trận, với $p[i]$ là số hàng của ma trận i và $p[i+1]$ là số cột của ma trận i

Ensure: Chi phí tối thiểu để nhân tất cả các ma trận

function HuShingMatrixChain(p, i, j)
  if $j - i \leq 1$ **then** ▷ Chỉ có một ma trận hoặc không có ma trận
    return 0 ▷ Không cần phép nhân nào
  end if
  $minCost \leftarrow \infty$ ▷ Khởi tạo chi phí lớn nhất
  for $k = i + 1 \rightarrow j - 1$ **do** ▷ Duyệt qua các điểm chia đoạn i, j
    Tính chi phí tam giác $cost \leftarrow p[i] \cdot p[k] \cdot p[j]$
    Gọi đệ quy để tính chi phí các phần con:
    $leftCost \leftarrow$ HuShingMatrixChain(p, i, k)
    $rightCost \leftarrow$ HuShingMatrixChain(p, k, j)
    $totalCost \leftarrow cost + leftCost + rightCost$
    $minCost \leftarrow \min(minCost, totalCost)$
  end for
  return $minCost$ ▷ Chi phí tối thiểu cho đoạn $[i, j]$
end function
HuShingMatrixChain($p, 0, n-1$) ▷ Gọi hàm với toàn bộ đoạn ma trận

4.4 Độ phức tạp

- **Thời gian:** Thuật toán **Hu Shing** đạt được độ phức tạp thời gian $O(n \log n)$ nhờ:
  - **Tam giác hóa (Triangulation):** Các ma trận được tổ chức để chia nhỏ bài toán, chỉ xử lý những phần cần thiết.
  - **Chia để trị (Divide and Conquer):** Chia bài toán lớn thành các đoạn con không chồng chéo, tính toán độc lập, và hợp nhất kết quả. Mỗi bước chia nhỏ có chi phí $O(n)$, tổng thời gian chạy là $O(n \log n)$.
- **Không gian:** Độ phức tạp không gian của thuật toán là $O(n)$, do:
  - Chỉ lưu các trạng thái cần thiết của tam giác hoặc đoạn con thay vì toàn bộ bảng $M[i][j]$ như trong Dynamic Programming.
- **Ưu điểm và nhược điểm:**
  - **Ưu điểm:** Thuật toán hiệu quả hơn Dynamic Programming ($O(n^3)$) trong các bài toán với cấu trúc đặc biệt, ví dụ: ma trận hiếm (sparse) hoặc đối xứng.
  - **Nhược điểm:** Cần hiểu rõ cấu trúc bài toán và thiết kế thuật toán một cách cẩn thận để đạt hiệu quả tốt nhất.

—

5 Kết luận

Thuật toán Hu và Shing mở ra một hướng tiếp cận hiệu quả cho bài toán MCM với độ phức tạp $O(n \log n)$. Tuy nhiên, phương pháp Dynamic Programming truyền thống với độ phức tạp $O(n^3)$ vẫn là lựa chọn phổ biến nhờ tính đơn giản và dễ triển khai.

Tài liệu tham khảo

- T. C. Hu and M.-T. Shing, 1984. *Combinatorial Algorithms for Matrix Multiplication*
- Mark A. Weiss, 3rded., 2005. *Data Structures and Algorithm Analysis in C++*
- Reema Thareja, Oxford University Express, 2nd edition. *Data Structures Using C*
- *https://www.techiedelight.com/matrix-chain-multiplication/*