[Sparse] Matrix Chain Products

Uwe Naumann

Informatik 12:
Software and Tools for Computational Engineering (STCE)

RWTH Aachen University

Contents

Outline

Objective and Learning Outcomes

Motivation

- Matrix Chain Product (MCP)
- Proof of NP-Completeness of MCP

[Sparse] Matrix Chain Product Bracketing

- Dynamic Programming
- Implementation

Summary and Next Steps

Objective

- Introduction to dynamic programming motivated by the desirable evaluation of dense and sparse matrix chain products at optimal computational cost.

Learning Outcomes

- You will understand
  - NP-completeness of the MATRIX CHAIN PRODUCT problem
  - dense and sparse MATRIX CHAIN PRODUCT BRACKETING problems
  - dynamic programming.
- You will be able to
  - apply the dynamic programming algorithm with pen and paper
  - implement and use the dynamic programming algorithm.

Outline

We consider [sparse] matrix chain products

$$\prod_{\nu=p-1}^{0} A_\nu = A_{p-1} \cdot \ldots \cdot A_0 \quad \text{for } A_\nu = (a^\nu_{j,i})^{j=0,\ldots,m_\nu-1}_{i=0,\ldots,n_\nu-1} \in \boldsymbol{R}^{m_\nu \times n_\nu} \quad . \tag{1}$$

A matrix product $B = A_{\nu+1} \cdot A_\nu$ is evaluated as a sequence of fused multiply-add (`fma`) operations

$$b_{k,i} = b_{k,i} + a^{\nu+1}_{k,j} \cdot a^\nu_{j,i} \ ,$$

where initially $b_{k,i} = 0$.

The MATRIX CHAIN PRODUCT (MCP) problem asks for an `fma`-optimal evaluation of a given [sparse] matrix chain product.

Example: The matrix product

$$\begin{pmatrix} 6 & 0 \\ 0 & 7 \end{pmatrix} \begin{pmatrix} 7 & 0 \\ 0 & 6 \end{pmatrix} = \begin{pmatrix} 42 & 0 \\ 0 & 42 \end{pmatrix}$$

can be evaluated at the expense of a single `fma` by exploiting commutativity of scalar multiplication.

MCP is NP-complete.

The proof is based on reduction from ENSEMBLE COMPUTATION (EC).

Given a collection

$$C = \{C_\nu \subseteq A : \nu = 1, \ldots, |C|\}$$

of subsets $C_\nu = \{c_i^\nu : i = 1, \ldots, |C_\nu|\}$ of a finite set A and a positive integer K is there a sequence $u_i = s_i \cup t_i$ for $i = 1, \ldots, k$ of $k \leq K$ union operations, where each s_i and t_i is either $\{a\}$ for some $a \in A$ or u_j for some $j < i$, such that s_i and t_i are disjoint for $i = 1, \ldots, k$ and such that for every subset $C_\nu \in C$, $\nu = 1, \ldots, |C|$, there is some u_i, $1 \leq i \leq k$, that is identical to C_ν.

Example: $A = \{a_1, a_2, a_3, a_4\}$, $C = \{\{a_1, a_2\}, \{a_2, a_3, a_4\}, \{a_1, a_3, a_4\}\}$ and $K = 4$ yield "yes" as an answer with a corresponding solution to the optimization problem given by $C_1 = u_1 = \{a_1\} \cup \{a_2\}$; $u_2 = \{a_3\} \cup \{a_4\}$; $C_2 = u_3 = \{a_2\} \cup u_2$; $C_3 = u_4 = \{a_1\} \cup u_2$.

EC is NP-complete. (Garey/Johnson, 1979.)

Consider an arbitrary instance (A, C, K) of EC and a bijection $A \leftrightarrow \tilde{A}$, where $\tilde{A}$ consists of $|A|$ mutually distinct primes. A corresponding bijection $C \leftrightarrow \tilde{C}$ is implied.

Create an extension $(\tilde{A} \cup \tilde{B}, \tilde{C}, K + |\tilde{B}|)$ by adding unique entries from a sufficiently large set $\tilde{B}$ of primes not in $\tilde{A}$ to the $\tilde{C}_j$ such that they all have the same cardinality p. Note that a solution for this extended instance of EC implies a solution of the original instance of EC as each entry of $\tilde{B}$ appears exactly once.

Fix the order of the elements of the $\tilde{C}_j$ arbitrarily yielding $\tilde{C}_j = (\tilde{c}_i^j)_{i=1}^p$ for $j = 1, \ldots, |\tilde{C}|$. Let the factors in Equation (1) be diagonal matrices in

$$A_\nu = (a^\nu_{j,j})_{j=0}^{|\tilde{C}|-1} \in \boldsymbol{R}^{|\tilde{C}| \times |\tilde{C}|} \quad \text{such that} \quad a^\nu_{j,j} = \tilde{c}^j_{\nu+1} \; .$$

Union in EC becomes multiplication in MCP.

According to the fundamental theorem of arithmetic (Gauss, 1801) the elements of $\tilde{C}$ correspond to unique (up to commutativity of scalar multiplication) factorizations of the $|\tilde{C}|$ nonzero diagonal entries of $\prod_{\nu=p-1}^{0} A_\nu$. This uniqueness property extends to arbitrary subsets of the $\tilde{C}_j$ considered during the exploration of the search space of MCP.

A solution to the constructed (with effort polynomial in the size of the given arbitrary instance of EC) instance of MCP implies a solution of the associated extended instance of EC and, hence, of the original instance of EC.

A proposed solution for MCP is easily validated by counting the at most $|\tilde{C}| \cdot p$ scalar multiplications performed.

For the previously discussed sample instance of EC we get

$$A_2 \cdot A_1 \cdot A_0 = \begin{pmatrix} 11 & & \\ & 7 & \\ & & 7 \end{pmatrix} \cdot \begin{pmatrix} 3 & & \\ & 5 & \\ & & 5 \end{pmatrix} \cdot \begin{pmatrix} 2 & & \\ & 3 & \\ & & 2 \end{pmatrix}$$

as

$$A = \{a_1, a_2, a_3, a_4\} \Rightarrow \tilde{A} = \{2, 3, 5, 7\}$$
$$\tilde{B} = \{11\}$$
$$C = \{\{a_1, a_2\}, \{a_2, a_3, a_4\}, \{a_1, a_3, a_4\}\} \Rightarrow \tilde{C} = \{\{2, 3, 11\}, \{3, 5, 7\}, \{2, 5, 7\}\}$$
$$K + |\tilde{B}| = K + 1 = 5 \; .$$

Outline

Heuristically, we approach MCP by restriction of the search space to bracketing of the matrix chain product, e.g, let $B = A_3 \cdot A_2 \cdot A_1 \cdot A_0$ with dense

$$A_3 \in \boldsymbol{R}^{4\times 4}, A_2 \in \boldsymbol{R}^{4\times 2}, A_1 \in \boldsymbol{R}^{2\times 3}, \text{and } A_0 \in \boldsymbol{R}^{3\times 1} .$$

Associativity of matrix multiplication yields the following five bracketings and corresponding total number of `fma` operations:

- $A_3 \cdot (A_2 \cdot (A_1 \cdot A_0)) \Rightarrow 30$ `fma`
- $A_3 \cdot ((A_2 \cdot A_1) \cdot A_0)) \Rightarrow 52$ `fma`
- $(A_3 \cdot A_2) \cdot (A_1 \cdot A_0) \Rightarrow 46$ `fma`
- $(A_3 \cdot (A_2 \cdot A_1)) \cdot A_0 \Rightarrow 84$ `fma`
- $((A_3 \cdot A_2) \cdot A_1) \cdot A_0 \Rightarrow 68$ `fma`

There is a discrepancy by almost a factor of three between the best and the worst choices.

The [Sparse] Matrix Chain Product Bracketing ([S]MCPB) problem asks for a bracketing that optimizes the computational cost by minimizing the total number of `fma` operations.

The number of different bracketings of a matrix chain product of length p is defined by the recurrence

$$\gamma(p) = \begin{cases} 1 & \text{if } p = 1 \\ \sum_{i=1}^{p-1} \gamma(i)\gamma(p-i) & \text{if } p \geq 2 \end{cases} .$$

The induced sequence of Catalan numbers grows exponentially with p since $\gamma(p) = \mathcal{C}(p-1)$ and

$$\mathcal{C}(p) = \frac{1}{p+1}\binom{2p}{p} \approx \frac{4^p}{(p+1)\sqrt{\pi p}}$$

For example, $\mathcal{C}(p = 2, \ldots, 5) = (1, 1, 2, 5)$.

Any given instance of [S]MCPB can be regarded as a product of the results of two mutually independent subproblems as

$$(A_{p-1} \cdot \ldots \cdot A_{i+1}) \cdot (A_i \cdot \ldots \cdot A_0)$$

Recursively, this procedure yields a total of $\sum_{i=1}^{p-1} i = \binom{p}{2} = O(p^2)$ distinct subproblems which overlap such that the search space of the subproblem $A_{k,j} \equiv A_k \cdot \ldots \cdot A_j$ is a subset of the search space of any subproblem $A_{l,i} \equiv A_l \cdot \ldots \cdot A_i$ with $i \leq j \leq k \leq l$.

Moreover, every single one of this polynomial $(O(p^2))$ number of distinct subproblems needs to be solved exactly once due to the optimal substructure property. If $A_{k,j}$ is evaluated as part of an optimal bracketing of $A_{l,i}$, $i \leq j \leq k \leq l$, then $A_{k,j}$ itself must be bracketed optimally. Otherwise, the total cost could be decreased by an optimal bracketing of $A_{k,j}$ yielding a contradiction to the assumed optimality of the bracketing of $A_{l,i}$.

[S]MCPB can be solved at computational cost of $O(p^3)$ by the dynamic programming recurrence

$$\texttt{fma}_{k,i} = \begin{cases} 0 & k = i \\ \min_{i \le j < k} \left(\texttt{fma}_{k,j+1} + \texttt{fma}_{j,i} + \texttt{fma}_{k,j,i} \right) & k > i \end{cases}$$

through tabulating $\texttt{fma}_{k,i}$ for $k - i = 0, \ldots, p$ and where $\texttt{fma}_{k,j,i}$ is the cost of evaluating $A_{k,j} \cdot A_{j,i}$.

Exploitation of sparsity increases the computational cost due to the need for explicit evaluation of the sparsity patterns for all subproblems. This increase in computational cost is at most $O(n^3)$ for $n = \max\left(n_0, \max_{\nu=0,\ldots,p-1} m_\nu\right)$.

The algorithm proceeds as follows:

1. $\texttt{fma}_{i,i} = 0$ for $i = 3, \ldots, 0$
2. $\texttt{fma}_{3,2} = 4 \cdot 4 \cdot 2 = 32$ and $A_{3,2} \in \boldsymbol{R}^{4\times 2}$
3. $\texttt{fma}_{2,1} = 4 \cdot 2 \cdot 3 = 24$ and $A_{2,1} \in \boldsymbol{R}^{4\times 3}$
4. $\texttt{fma}_{1,0} = 2 \cdot 3 \cdot 1 = 6$ and $A_{1,0} \in \boldsymbol{R}^{2\times 1}$
5. $\texttt{fma}_{3,1} = \min\{\texttt{fma}_{2,1} + 4 \cdot 4 \cdot 3, \texttt{fma}_{3,2} + 4 \cdot 2 \cdot 3\} = 56$ and $A_{(3,2),2} \in \boldsymbol{R}^{4\times 3}$
6. $\texttt{fma}_{2,0} = \min\{\texttt{fma}_{1,0} + 4 \cdot 2 \cdot 1, \texttt{fma}_{2,1} + 4 \cdot 3 \cdot 1\} = 14$ and $A_{2,(1,0)} \in \boldsymbol{R}^{4\times 1}$
7. $\texttt{fma}_{3,0} = \min\{\texttt{fma}_{2,0} + 4 \cdot 4 \cdot 1, \texttt{fma}_{3,2} + \texttt{fma}_{1,0} + 4 \cdot 2 \cdot 1, \texttt{fma}_{3,1} + 4 \cdot 3 \cdot 1\} = 30$ and $A_{3,(2,(1,0))} \in \boldsymbol{R}^{4\times 1}$

The bracketing scheme $A_3 \cdot (A_2 \cdot (A_1 \cdot A_0))$ is optimal with a total of 30 fma performed.

Dynamic Programming

Implementation

```
#include <vector>

template<typename T>
T dp(const std::vector<std::pair<T,T>> &A,
              std::vector<std::vector<std::pair<T,T>>> &C) {
  int p=A.size();
  for (int j=0;j<p;j++)
    for (int i=j;i>=0;i--)
      if (i==j)
        C[j][i]=std::make_pair(0,0);
      else
        for (int k=i+1;k<=j;k++) {
          T cost=C[j][k].first+C[k-1][i].first+A[j].first*A[k].second*A[i].second;
          if (k==i+1||cost<C[j][i].first) C[j][i]=std::make_pair(cost,k);;
        }
  return C[p-1][0].first;
}
```

Dynamic Programming

Implementation

```
int main(int argc, char* argv[]) {
  assert(argc==2); std::ifstream in(argv[1]);
  using T=unsigned long;
  // Matrix Chain Product as sequence of m x n factors
  int p; in >> p; assert(p>0);
  std::vector<std::pair<T,T>> A(p,std::make_pair(0,0));
  // Dynamic Programming Table as // p x p lower triangular matrix
  // storing optimal cost and split position per subchain
  std::vector<std::vector<std::pair<T,T>>>
    C(p,std::vector<std::pair<T,T>>(p,std::make_pair(0,0)));
  dp(A,C);
  // Result ...
  return 0;
}
```

See live demo.

Dynamic Programming

Random Generation of MCP Instances

```
#include<iostream>
#include<cassert>
#include<random>

int main(int argc, char* argv[]) {
  assert(argc==3); int l=std::stoi(argv[1]), max_nm=std::stoi(argv[2]);
  std::random_device r;
  std::default_random_engine g(r());
  std::uniform_int_distribution<int> dnm(1,max_nm);
  std::cout << l << std::endl;
  int m=dnm(g), n=dnm(g);
  std::cout << m << " " << n << std::endl;
  for (int i=1;i<l;i++) {
    n=dnm(g);
    std::cout << n << " " << m << std::endl;
    m=n;
  }
  return 0;
}
```

See live demo.

Let $A_3 \cdot A_2 \cdot A_1 \cdot A_0$ be such that

$$A_3 \in \boldsymbol{R}^{4\times 4}, A_2 \in \boldsymbol{R}^{4\times 2}, A_1 \in \boldsymbol{R}^{2\times 3}, A_0 \in \boldsymbol{R}^{3\times 1},$$

and

$$A_3 = \begin{pmatrix} * & & & \\ & * & & \\ & & * & \\ & & & * \end{pmatrix}, \quad A_2 = \begin{pmatrix} * & \\ & * \\ * & \\ & * \end{pmatrix},$$

$$A_1 = \begin{pmatrix} * & & * \\ * & * & * \end{pmatrix}, \quad A_0 = \begin{pmatrix} * \\ \\ * \end{pmatrix}.$$

An optimal bracketing, e.g, $(A_3 \cdot A_2) \cdot (A_1 \cdot A_0)$, requires 12 `fma`.

Outline

Objective and Learning Outcomes

Motivation

- Matrix Chain Product (MCP)
- Proof of NP-Completeness of MCP

[Sparse] Matrix Chain Product Bracketing

- Dynamic Programming
- Implementation

Summary and Next Steps

Summary

- Introduction to dynamic programming for [SPARSE] MATRIX CHAIN PRODUCT BRACKETING problem motivated by NP-completeness of MATRIX CHAIN PRODUCT problem.

Next Steps

- Download, inspect and play with the code.
- Extend the sample implementation of [dense] MATRIX CHAIN PRODUCT BRACKETING to the sparse case (tutorial).
- Continue the course to find out more ...