AN INTRODUCTION TO THE COMPUTATIONAL COMPLEXITY OF MATRIX OPERATIONS

by Derek O'Connor, U.C.D.

1. INTRODUCTION

Computational complexity is the study of algorithms to determine the amount of 'effort' or 'work' they require in solving problems. We associate with every problem an integer (or set of parameters) called the size of the problem. The time needed by an algorithm for solving a problem, expressed as a function of the size of the problem, is called the time complexity of the algorithm. The limiting behavior of this complexity, as the size increases, is called the asymptotic time complexity of the algorithm. There are similar definition for space complexity, the storage space needed by the algorithm. In this paper we will be concerned only with time complexity and for the remainder of the paper the word 'time' is omitted.

We need to distinguish between the inherent complexity of a problem and the complexity of an algorithm used to solve the problem. The inherent complexity of a problem of size n is the amount of time that is both necessary and sufficient to solve the problem. This time, $T^*(n)$, is usually difficult to determine and we have to be content with $T'(n)$, a lower bound on $T^*(n)$. The complexity of an algorithm, $T''(n)$, is sufficient to solve the problem and is an upper bound on $T^*(n)$. The algorithm is optimal if $T''(n) = T^*(n)$.

In this paper we concentrate on the asymptotic complexity of algorithms for matrix operations. If an algorithm solves a problem of sized n in a time $T(n) = cn^2+n$ for some constant c, we say the complexity is $O(n^2)$. hence we are interested only in the functional form of $T(n)$ for large values of n. In general a function $g(n)$ is said to be $O(f(n))$ if there exists a constant c such that $g(n) < cf(n)$ for all but some finite set of non-negative values for n.

We consider only square matrices, but many of the results hold for non-square matrices. The size of an $n \times n$ matrix is n and we wish to determine the amounts of time needed to perform the familiar operations of addition, multiplication, inversion, equation solving and determinant evaluation, with time for each operation expressed as a function of n. We assume that our computer has the usual repertoire of operators (+, -, *, /) for reals and integers and that each operator is performed in a fixed amount of time. Hence we can view the time complexity of an algorithm as the number of basic arithmetic operations it performs.

2. LOWER BOUNDS ON MATRIX OPERATIONS

A general $n \times n$ matrix has n^2 elements. Hence, any operation that involves all elements requires at least $O(n^2)$ time. The usual algorithm for matrix addition/subtraction requires n^2 additions/subtractions and is therefore optimal. The ordinary algorithm for matrix multiplication requires n^3 multiplications and $n^2(n-1)$ additions. Surprisingly, the best-known lower bound is $O(n^2)$.

3. ORDINARY MATRIX MULTIPLICATION, INVERSION AND EQUATION SOLVING ARE $O(N^3)$ OPERATIONS

Multiplication: $C = AB$ where $c_{ij} = \sum_k a_{ik} * a_{kj}$

Each c_{ij} requires n multiplications and $n-1$ additions. The total number of operations is n^3 multiplications n^3-n^2 additions. Hence the complexity of multiplication is $O(n^3)$.

Equation solving: $Ax = b$

Triangularize A using Gaussian elimination. This gives $Ux = \hat{b}$ which can be solved by Back-Substitution because U is upper-triangular. Gaussian Elimination is $O(n^3)$ and Back-Substitution is $O(n^2)$. Hence the complexity of Equation Solving is $O(n^3)$.

Inversion: $AA^{-1} = I$

This is equivalent to solving the n sets of equations $Ax_j = e_j$, $j=1,2,\ldots n$ where x_j and e_j are the j^{th} columns of A^{-1} and I respectively. This, in turn, is equivalent to solving the n sets of equations $Ux_j = \hat{e}_j$. U is computed once using Gaussian Elimination in $O(n^3)$ time, each $\hat{e}_j$ can be computed in $O(n^2)$ time, and each set of equations is solved by backsubstitution in $O(n^2)$ time. The total time is $O(n^3) + 2nO(n^2)$. Hence, Inversion is $O(n^3)$.

4. IMPROVED UPPER BOUNDS FOR MATRIX MULTIPLICATION

Lemma 1: $(M_n, +_n, \cdot_n, 0_n, I_n)$ is a ring, where M_n is the set of all $n \times n$ matrices whose elements are chosen from arbitrary ring R.

Lemma 2: Let f be a partition of an $n \times n$ matrix into four $n/2 \times n/2$ matrices, assuming n is even,

$$\text{i.e.} \quad f(a) = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$$

Then, for any matrices A and B in M_n,

$$f(A+B) = f(A) + f(B) \quad \text{and} \quad f(AB) = f(A)f(B)$$

The above lemma allows us to transform an $n \times n$ matrix with elements from source ring R into a 2×2 matrix with elements from the ring of $n/2 \times n/2$ matrices.

Strassens Algorithm for Matrix Multiplication

The product $C = AB$ of two $n \times n$ matrices can be transformed into the product of two 2×2 matrices whose elements are $n/2 \times n/2$ matrices. Thus we have

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

where elements of C are defined as follows

$$\begin{aligned} C_{11} &= A_{11}B_{11} + A_{12}B_{21} \\ C_{12} &= A_{11}B_{12} + A_{12}B_{22} \\ C_{21} &= A_{21}B_{11} + A_{22}B_{21} \\ C_{22} &= A_{21}B_{12} + A_{22}B_{22} \end{aligned}$$

If we have an algorithm that computes the elements of C using m multiplications and a additions then we apply the algorithm recursively to smaller and smaller partitions until we are reduced to multiplying the elements of the underlying ring R. If the algorithm multiplies two n x n matrices in time $T_m(n)$ and n is a power of 2 ($=2^p$) then we have

$$T_m(n) \leq mT_m(n/2) + an^2/4, \quad n > 2.$$

This inequality can be used to show that $T_m(n) < kn^q$, $q = \log_2 m$, i.e., the complexity of the algorithm is $O(n^{\log m})$ and is independent of a. If $m = 8$ then the algorithm is $O(n^3)$, which is the same as ordinary matrix multiplication.

Strassen's Lemma: The product of two 2x2 matrices with elements from an arbitrary ring can be computed with 7 multiplications and 18 additions/subtractions.

Proof: Compute the elements of C = AB as follows:

Compute the products

$$\begin{aligned}
m_1 &= (a_{12} - a_{22})(b_{21} + b_{22}) \\
m_2 &= (a_{11} + a_{22})(b_{11} + b_{22}) \\
m_3 &= (a_{11} + a_{22})(b_{11} + b_{12}) \\
m_4 &= (a_{11} + a_{12} - b_{11}) \\
m_5 &= a_{11}(b_{12} - b_{22}) \\
m_6 &= a_{22}(b_{21} - b_{11}) \\
m_7 &= (a_{21} + a_{22})b_{11}
\end{aligned}$$

With these seven products compute

$$\begin{aligned}
c_{11} &= m_1 + m_2 - m_4 + m_6 \\
c_{12} &= m_4 + m_5 \\
c_{21} &= m_6 + m_7 \\
c_{22} &= m_2 - m_3 + m_5 - m_7
\end{aligned}$$

There are obviously 7 multiplications and 18 additions/subtractions. Simple algebraic manipulation, using only the ring axioms, shows that the c_{ij}'s are those required by the definition of multiplication.

This lemma leads to the following theorem.

Theorem1: Two $n \times n$ matrices with elements from an arbitrary ring can be multiplied in $O(n^{\log 7})$ arithmetic operations.

Proof: Assume $n = 2^p$, for some integer p. If T(n) is the number of arithmetic operations needed to multiply two $n \times n$ matrices, then using Strassen's Lemma on the matrices partitioned into four $n/2 \times n/2$ blocks we get the recurrence

$$T(n) = 7T(n/2) + 18(n/2)^2 \quad \text{for } n \geq 2 .$$

The first term on the right is the time required to multiply a 2×2 matrix whose elements are $n/2 \times n/2$ matrices, while the second term is the time required for the addition/subtraction of these matrices. By induction we get

$$T(n) = O(7^{\log n}) = O(n^{\log 7}) .$$

If n is not a power of 2, i.e., $2^p < n < 2^{p+1}$, then augment the matrices with enough rows and columns of zeroes to make $n = 2^{p+1}$. This will increase the size of the matrix by a factor of 2, at most. This in turn will increase T(n) by a factor of 7, and so T(n) is still $O(n^{\log 7})$.

It can be shown (see [AHU74]) that equation - solving, inversion, determinant evaluation, etc., are computationally equivalent. This implies that each of these operations can be performed in $O(n^{\log 7})$ time.

5. PRACTICAL CONSIDERATIONS

Although Strassen's improvement is theoretically significant it is not better than classical $O(n^3)$ multiplication, unless n is large (> 150). This is because of the many hidden, non-arithmetic operations that must be performed during the multiplication operation. Additional storage, for intermediate results, is also needed because the algorithm is recursive.

Nonetheless, Strassen's Algorithm has opened up a large new area of research (see below) whose results will have great practical significance as computers increase in size and speed.

6. RECENT IMPROVEMENTS

Strassen's work [Str69] stimulated a great deal of research into matrix multiplication and related operations. We give here a chronology of the improvements since 1969, taken from a recent paper [Pan81].

Exponent	Author	Date of Publication
2.8074	Strassen	1969
2.7950	Pan	1978
2.7804	Pan	1979
2.7801	Pan	1979
2.7799	Bini, et al.	1979
2.6088	Schonhage	1979
2.6054	Pan	1979
2.5480	Schonhage	To appear
2.5220	Pan & Winograd	To appear

7. A SHORT REFERENCE LIST

1. [AHU74] Aho, A., Hopcroft, J., & Ullman, J. : The Design and Analysis of Computer Algorithms, Addison Wesley, 1974.

2. [Pan81] Pan, V. : New Combinations of Methods for the Acceleration of Matrix Multiplication, Computers and Mathematics with Applications, Vol. 7, No.1, 1981.

3. [Str69] Strassen,V. : Gaussian Elimination is not Optimal, Numer. Math., Vol. 13, 354 - 356, 1969.