AN INTRODUCTION TO SOME CURRENT RESEARCH IN NUMERICAL COMPUTATIONAL COMPLEXITY

J. F. Traub

Department of Computer Science
Carnegie-Mellon University
Pittsburgh, Pa.

October, 1973

This research was supported in part by the National Science Foundation under Grant GJ32111 and the Office of Naval Research under Contract N0014-67-A-0314-0010, NR 044-422.

ABSTRACT

This is an expository invited paper presented at a Conference on the Influence of Computing on Mathematical Research and Education, at the summer meeting of the American Mathematical Society at Missoula, Montana, August, 1973.

AN INTRODUCTION TO SOME CURRENT RESEARCH IN NUMERICAL COMPUTATIONAL COMPLEXITY

J. F. Traub

This is a Conference on the Influence of Computing on Mathematical Research and Education. I want to talk about a particular area of numerical mathematics, numerical computational complexity, which has a very large intersection with mathematics. On the one hand, it requires the development of new mathematical techniques while on the other hand, certain of the questions it tries to answer are primarily mathematical.

In this talk I will focus on research rather than educational issues. However, I will mention that some non-trivial results and questions can be formulated so as to be accessible to both undergraduate and graduate students. This is at least partially the case because the field is so new.

I'll not attempt a survey of current work in computational complexity, rather, I'll restrict myself to problems and algorithms from numerical mathematics. Later in this conference, Professor Karp will discuss non-numerical complexity. Within numerical computational complexity, I will try to give you the flavor of some of the recent work and state a few open problems. I will draw on work done at Carnegie-Mellon University for many of my examples.

A number of the papers I'll cite were presented at a Symposium on Complexity of Sequential and Parallel Numerical Algorithms at Carnegie-Mellon University in May, 1973. Incidentally, there have been at least four Symposia in the United States from April to July, 1973 devoted entirely or in major part to computational complexity.

I won't give a formal definition of complexity here. Instead I will begin by a particular problem, that of matrix multiplication, to illustrate some basic ideas.

<u>Matrix Multiplication</u>

Let A, B be (n,n) matrices and let C = AB. The definition of matrix product gives us the following algorithm for forming the n^2 elements of C - take the scalar product of rows of A with columns of B. This algorithm takes $O(n^3)$ mult and $O(n^3)$ additions. Until about five years ago no one asked if there might be better algorithms. Then Winograd [68] showed that matrix multiplication could be done in $\frac{1}{2}n^3 + O(n^2)$ multiplications and Strassen [69] gave an algorithm which required fewer than $O(n^3)$ arithmetic operations. Consider in particular the problem of multiplying two (2,2) matrices. Classically, this requires eight scalar multiplications. Strassen showed how to multiply the matrices in seven scalar multiplications. Furthermore, his algorithm does not depend on commutivity of multiplication of the elements and can therefore be applied to partitioned matrices. He showed that two (n,n) matrices could be multiplied in $O(n^{\log_2 7})$ <u>arithmetic operations</u>. Since $\log_2 7 \sim 2.81$, the multiplication of two matrices can be done in $O(n^{2.81})$ arithmetic operations.

Now for very large matrices the fact that matrix multiplication can be done in $O(n^{2.81})$ arithmetic operations might be of interest. It is, however, the research engendered by this result and its theoretical consequences which is of greater interest than the practical applications.

Can we do better than $O(n^{2.81})$? Not by using 2 by 2 matrices since it's been established that seven multiplications are optimal. What about using 3 by 3 matrices? The minimum number of multiplications for multiplying two 3 by 3 matrices is open.

Can we say anything in general about the minimum number of arithmetic operations needed to multiply any n by n matrices? Since the problem has $2n^2$ inputs and n^2 outputs, $O(n^2)$ must be a lower bound on the number of arithmetics. Now $O(n^2)$ is linear in the number of inputs and outputs. Is there a non-linear lower bound? We don't know. Our present state of knowledge is

$$O(n^2) \leq \# \text{ of arithmetic operations} \leq O(n^{\log_2 7}).$$

A survey of what is known is given by Borodin [73].

I will use the matrix multiplication problem to illustrate some general features of <u>algebraic complexity</u>. There is a parameter which is a measure of the problem size. In the matrix problem this is the order of the matrix. There are one or more fundamental operations. In the matrix multiplication case this is the number of scalar multiplications or the number of scalar arithmetic operations. There is an algorithm for solving the problem in a finite number of fundamental operations. This algorithm gives us an upper bound on the difficulty of the problem. We also want lower bounds and preferably tight lower bounds. For the matrix problem we saw $O(n^2)$ was a lower bound linear in the number of inputs/outputs. We regard such a lower bound as trivial; we want non-linear lower bounds. Few such lower bounds are known today.

The difficulty of a problem we call its computational complexity. We also refer to the complexity of a particular algorithm. How the difficulty is measured depends on what we designate as the fundamental operation or operations. For example, for a parallel computer it is usually the time rather than the number of operations that we try to optimize. I'll return to this later.

The area I'm discussing is often called concrete or specific complexity to distinguish it from the abstract complexity theory of Hartmanis, Blum and others. John Hopcroft wishes to reserve the name computational complexity to lower bound results. He feels that upper bounds represent computational simplicity. From a purely theoretical point of view this may be true. From a pragmatic point of view, good upper bounds provided by algorithms are very useful.

I find it convenient to divide computational complexity into two parts: algebraic computational complexity and analytic computational complexity.

Algebraic computational complexity deals with problems for which there are finite algorithms. Analytic computational complexity deals with problems for which there are no finite algorithms. I will discuss analytic complexity later. Here I want to add several more examples of algebraic complexity to the matrix problem discussed above.

The following problems deal with nth degree polynomials and they all take $O(n^2)$ operations if classical algorithms are used.

Multiplication of two polynomials

Division by a polynomial of degree $n/2$.

Evaluation of a polynomial at n points

Interpolation at $n+1$ points

Evaluation of a polynomial and all its derivatives at one point

Recently, algorithms have been developed for doing the first two problems in $O(n \log n)$ arithmetic operations and the last three in $O(n \log^2 n)$ arithmetic operations. Results here are due to Borodin, Horowitz, Kung, Strassen, and others. (See Borodin [73].) Using arguments from algebraic geometry, Strassen [73] has shown that the interpolation problem requires O(n log n) multiplications. This is a non-linear lower bound. Thus for the interpolation problem we have tight bounds from above and below

$$O(n \log n) \leq \# \text{ of arithmetic} \leq O(n \log^2 n).$$

The results above are asymptotic. But most problems arising in practice are for small n. For one of these problems, the evaluation of a polynomial and its derivatives at a point, we do have a result which is better than the classical algorithm for all n. The classical algorithm is the iterated Horner method which uses $\frac{n(n+1)}{2}$ multiplications and $\frac{n(n+1)}{2}$ additions. Shaw and Traub [73] have introduced an algorithm which requires 3n-2 multiplications and divisions and $\frac{n(n+1)}{2}$ additions. Thus the number of multiplications and divisions are linear rather than quadratic. There are a number of open questions including

1. Is there an algorithm using only a linear number of additions?
2. It is easy to show at least n+1 multiplications are required. An upper bound is 3n-2 multiplications and divisions. Can these bounds be tightened?
3. How many multiplications are required if divisions are not permitted?

An area which has seen a great deal of recent activity is the fast solution of linear systems, particularly of sparse systems arising, for example, from the discretization of partial differential equations. Recent surveys are given by Bunch [73] and Birkhoff and George [73].

One particular problem is the solution of a linear system whose matrix M is n^2 by n^2 and in block tridiagonal form, $M = [-I,T,-I]$, where T is an n by n tridiagonal matrix and I is the n by n identity. There are therefore $O(n^2)$ non-zero elements. Bank, Birkhoff, and Rose [73] give an $O(n^2)$ algorithm. This gives an upper bound linear in the number of non-zero elements and we can't hope for better than that. However, stability is crucial in this area and this algorithm is not stable. P. Swartztrauber (private communication) has developed a stable $O(n^2 \log \log n)$ algorithm.

So far, I've confined myself to problems for which there are finite algorithms. I used the adjective algebraic to characterize the complexity of problems in this area. I will now turn to an area where there are no finite algorithms. I'll refer to this area as analytic computational complexity.

Recent work in this area which I will not describe further today includes Eisenstat and Schultz [73] on the complexity of partial differential equations, Rice [73] on approximation, Brent [73b] on systems of non-linear equations, Brent, Winograd and Wolfe [73] on optimal iteration.

As a simple first illustration of a problem in analytic complexity, let's consider the calculation of α, $\alpha = \sqrt{A}$. A well-known method for calculating α is as the limit of the sequence $\{x_i\}$ defined by

$$x_{i+1} = \frac{1}{2}(x_i + \frac{A}{x_i}). \tag{1}$$

This is Newton iteration applied to $f = x^2 - A$. We can ask about the complexity of $\sqrt{A}$. An upper bound is obtained from the complexity of the particular algorithm specified by (1). We'll return to this problem after we've developed some tools.

Let x_0 be given and let

$$x_{i+1} = \varphi(x_i)$$

be a scalar iteration with $x_i \to \alpha$. Thus φ, x_0 defines an algorithm for computing α. What is the "efficiency" of this algorithm? I have to first introduce the basic concept of order. Let $p = p(\varphi)$ denote the order of φ. Roughly speaking

$$x_{i+1} - \alpha = 0[(x_i - \alpha)^p].$$

Let $C(\varphi)$ denote the "cost" of calculating x_{i+1} from x_i. We will give examples of various costs later. They have the critical property that $C(\varphi \circ \varphi) = 2C(\varphi)$. Define the efficiency of φ by

$$e(\varphi) = \frac{\log p(\varphi)}{C(\varphi)}.$$

All logarithms are to base two.

This efficiency has the following properties:

1. It is inversely proportional to the total cost of estimating α to within ϵ, for a small $\epsilon > 0$.
2. It is invariant under self composition. That is,

$$e(\varphi \cdot \varphi) = e(\varphi).$$

There are various quantities that can be used for C. For example, we can take

1. $C = M$ = Number of multiplications or divisions to compute x_{i+1} from x_i.
2. $C = \bar{M}$. This is the same as M except that multiplication by constants is not counted.

Let

$$e = \frac{\log p}{M}, \quad \bar{e} = \frac{\log p}{\bar{M}} .$$

Clearly

$$e \leq \bar{e}.$$

Kung [73a] proved that if φ is <u>any rational function</u>, then

$$\bar{e} \leq 1$$

and this bound is sharp. Therefore $e \leq 1$. Since a computer performs rational operations, this is not restrictive. Next, Kung [73b] asked for what iterations is $e = 1$ and for what iterations is $\bar{e} = 1$? He completely settled this question as follows:

If $e = 1$, then α is rational.

If $\bar{e} = 1$, then α is rational or quadratic irrational.

Thus only easy problems have optimal efficiency.

The above result concerns optimal φ over all problems α. Now let's consider a particular α. To fix ideas, let $\alpha = \sqrt{A}$. What's the best iteration for calculating $\sqrt{A}$? The most widely known algorithm is

$$x_{i+1} = \frac{1}{2}(x_i + \frac{A}{x_i}).$$

Then $p = 2$, $\bar{M} = 1$, $M = 2$, and therefore

$$\bar{e} = \frac{\log 2}{\bar{M}} = 1, \quad e = \frac{\log 2}{M} = \frac{1}{2} .$$

Thus in the $\bar{e}$ measure, Newton-Raphson iteration is optimal. This was first pointed out by Paterson [72]. However, in the e measure, the question is open.

What's the optimal iteration for computing a quadratic irrational in the multiplicative efficiency measure e? One can ask similar questions for classes of mathematical problems.

Observe that the cost in the above development is limited to multiplications. The reason for this is technical. Multiplications can affect the degree growth of a polynomial. Additions do not cause polynomial degree to increase and therefore analysis based on degree growth cannot be used for additions. Morganstern [73] has made recent progress on using other growth arguments for addition.

I'm going to change gears here and talk about approximating a zero α of a scalar function f by a sequence of iterations x_i. This problem is a prototype of many problems of applied mathematics where we seek a zero of an operator equation (Traub [72]). Examples are the numerical solution of integral equations and partial differential equations. With the exception of recent work by Brent [73b] and Wozniakowski [74] on finding a zero of a vector-valued function, that is solving a system of non-linear equations, almost all work has dealt with zeros of scalar functions.

Kung and Traub [73b] introduced a new measure for the efficiency of an algorithm φ with respect to a problem f (see also Traub [73b]). I don't want to get into technical details here other than to mention that they include the combinatorial cost of φ, which is the number of arithmetic operations used by an iteration even if the evaluation of f and its derivatives were free. It turns out to be crucial to include the combinatorial cost.

There is a major open conjecture in the area of iteration algorithms for calculating zeros of a function f. I remind you that, roughly speaking, the order of an iteration φ is a number p such that

$$x_{i+1} - \alpha = 0((x_i-\alpha)^p).$$

Let the total number of evaluations of f or its derivatives that φ uses in going from x_i to x_{i+1} be n.

Kung and Traub [73a] have constructed an algorithm using n evaluations which is of order 2^{n-1} and they conjecture this to be optimal. That is, the order of any iteration without memory (iteration is precisely defined by Kung and Traub [73a]) based on n evaluations is of order at most 2^{n-1} for every positive integer n. The conjecture has been proven for n = 1,2 (Kung and Traub [73c].

Up to now we have implicitly assumed we were dealing with a sequential computer. Thus the cost associated with an algorithm has been in terms of fundamental operations, such as the number of arithmetics. On a parallel machine, the cost associated with an algorithm is time and this is the quantity we optimize.

To a first approximation you can think of a parallel computer as consisting of a set of arithmetic units which you can use simultaneously. I emphasize this is a first approximation only. There are major differences in the parallel and vector computers now becoming available. Furthermore, when preparing programs for these machines, matters of data handling and data structure become paramount.

I think algorithms for parallel computers are one of the most interesting new research areas in numerical mathematics for a number of reasons:

1. From a practical point of view, because the parallel machines are coming. Machines which should be available within a year or so

include Carnegie-Mellon University's Multi-Mini Computer, CDC STAR, Burroughs' ILLIAC 4, Texas Instruments ASC. We need algorithms to put on them. In general, the best classical algorithms will not be very good on parallel machines.

2. From a theoretical point of view, because constructive mathematics has been implicitly sequential. The problems raised by parallelism raise a whole new world of interesting mathematical issues.

On a parallel machine we will be concerned with the <u>time</u> taken by an algorithm. One basic quantity to be analyzed is the parallel speed-up ratio defined as follows. For a problem of size m, the speed-up is defined as

$$S(m,k) = \frac{\text{optimal computation time on a one-processor computer}}{\text{optimal computation time on a k-processor computer}} .$$

In general we do not know the optimal times and therefore can only get estimates and bounds on $S(m,k)$.

Stone [73a] considers the maximal speed-up which can be achieved for various problems. Trivially, $S(m,m) \leq m$. A simple example of a problem with linear speed-up is the addition of two matrices of order m. On a sequential machine this takes m^2 addition times. On a parallel machine with m^2 processors this can be done in one addition time. Hence the speed-up is linear in the number of processors. Problems such as linear recurrence investigated by Stone [73b] and Heller [73], polynomial evaluation (Munro and Paterson [73]), rational evaluation (Brent [73a]), have speed-up at least $\frac{m}{\log m}$ which is close to linear. The numerical solution of partial differential equations also seems amenable to parallel computation. The solution of tridiagonal systems, which arise in the discretization of partial differential equations, is analyzed by Traub [73a].

The study of parallel algorithms is in an embryonic state. I believe it will be one of the major research areas in numerical mathematics in the next decade.

I've tried to give you a taste of some of the current research in numerical computational complexity. I'd like in conclusion to summarize some of my reasons for studying complexity:

1. To construct good new algorithms. There are, however, many components besides complexity to be considered in constructing a good algorithm.
2. To filter out bad algorithms.
3. To create a theory of algorithms which will establish theoretical limits on computation.
4. To investigate the intrinsic difficulty of mathematical problems.

ACKNOWLEDGMENT

I would like to thank H. T. Kung for his comments on this paper.

BIBLIOGRAPHY

Bank, Birkhoff, and Rose [73]	Bank, R., Birkhoff, G., and Rose, D. J., An $O(N^2)$ Method for Solving Constant Coefficient Boundary Value Problems in Two Dimensions. Report, Harvard University, 1973.
Birkhoff and George [73]	Birkhoff, G. and George, A., Elimination by Nested Dissection. Complexity of Sequential and Parallel Numerical Algorithms, Academic Press, 1973, pp. 221-270.
Borodin [73]	Borodin, A., On the Number of Arithmetics Required to Compute Certain Functions - Circa May 1973. Complexity of Sequential and Parallel Numerical Algorithms, Academic Press, 1973, pp. 149-180.
Brent [73a]	Brent, R., The Parallel Evaluation of Arithmetic Expressions in Logarithmic Time. Complexity of Sequential and Parallel Numerical Algorithms, Academic Press, 1973, pp. 83-102.
Brent [73b]	Brent, R., Some Efficient Algorithms for Solving Systems of Nonlinear Equations. SIAM J. Numer. Anal., Vol. 10, No. 2, April, 1973, pp. 327-344.
Brent, Winograd and Wolfe [73]	Brent, R., Winograd, S., and Wolfe, P., Optimal Iterative Processes for Root-Finding. Numerische Mathematik, 20, 1973, pp. 327-341.
Bunch [73]	Bunch, J. R., Complexity of Sparse Elimination. Complexity of Sequential and Parallel Numerical Algorithms, Academic Press, 1973, pp. 197-220.
Eisenstat and Schultz [73]	Eisenstat, S. C. and Schultz, M. H., The Complexity of Partial Differential Equations. Complexity of Sequential and Parallel Numerical Algorithms, Academic Press, 1973, pp. 271-282.
Heller [73]	Heller, D., A Determinant Theorem with Applications to Parallel Algorithms. To appear in SIAM Journal on Numerical Analysis, 1973. (Also available as a CMU Computer Science Department report.)

Kung [73a] Kung, H. T., A Bound on the Multiplication Efficiency of Iteration. JCSS, Vol.7, No.4, 1973, pp. 334-342. (Also available as a CMU Computer Science Department report.)

Kung [73b] Kung, H. T., The Computational Complexity of Algebraic Numbers. To appear in SIAM Journal on Numerical Analysis, 1973. (Also available as a CMU Computer Science Department report.)

Kung and Traub [73a] Kung, H. T. and Traub, J. F., Optimal Order of One-Point and Multipoint Iteration. To appear in the Journal of the ACM, 1973. (Also available as a CMU Computer Science Department report.)

Kung and Traub [73b] Kung, H. T., and Traub, J. F., Computational Complexity of One-Point and Multipoint Iteration. To appear in the Proceedings of the Symposium on the Complexity of Real Computation, edited by R. Karp, American Mathematical Society, 1973. (Also available as a CMU Computer Science Department report.)

Kung and Traub [73c] Kung, H. T., and Traub, J. F., Optimal Efficiency and Order for Iterations Using Two Evaluations. Report, Department of Computer Science, Carnegie-Mellon University, 1973.

Morgenstern [73] Morgenstern, J., Note on a Lower Bound of the Linear Complexity of the Fast Fourier Transform. JACM, April 1973.

Munro and Paterson [73] Munro, I. and Paterson, M., Optimal Algorithms for Parallel Polynomial Evaluation. Journal of Computer and System Sciences, Vol. 7, No. 2, April 1973, pp. 189-198.

Paterson [72] Paterson, M. S., Efficient Iterations for Algebraic Numbers. Complexity of Computer Computations (edited by R. E. Miller and J. W. Thatcher), Plenum Press, 1972, pp. 41-52.

Rice [73] Rice, J. R., On the Computational Complexity of Approximation Operators. Report, Purdue University, 1973.

Shaw and Traub [73] Shaw, M. and Traub, J. F., On the Number of Multiplications for the Evaluation of a Polynomial and Some of its Derivatives. To appear in the Journal of the ACM, 1973. (Also available as a CMU Computer Science Department report.)

Stone [73a]	Stone, H. S., Problems of Parallel Computation. Complexity of Sequential and Parallel Numerical Algorithms, Academic Press, 1973, pp. 1-16.
Stone [73b]	Stone, H. S., An Efficient Parallel Algorithm for the Solution of a Tridiagonal System of Equations. Journal of the ACM, Vol. 20, 1973, pp. 27-38.
Strassen [69]	Strassen, V., Gaussian Elimination is Not Optimal. Numerische Mathematik, 13, 1969.
Strassen [73]	Strassen, V., Die Berechnungskomplexitat von elementarsymmetrischen Funkteonen und von Interpolationskoeffizienten, Numer. Math. Vol. 20, 1973, pp. 238-251.
Traub [72]	Traub, J. F., Computational Complexity of Iterative Processes. SIAM J. Comput., Vol. 1, No. 2, June 1972, pp. 167-179. (Also available as CMU Computer Science report.)
Traub [73a]	Traub, J. F., Iterative Solution of Tridiagonal Systems on Parallel or Vector Computers. Complexity of Sequential and Parallel Numerical Algorithms, Academic Press, 1973, pp. 49-82. (Also available as CMU Computer Science report.)
Traub [73b]	Traub, J. F., Theory of Optimal Algorithms. To appear in the Proceedings of the Conference on Software for Numerical Mathematics, Loughborough, England, 1973. (Also available as CMU Computer Science report.)
Winograd [68]	Winograd, S., A New Algorithm for Inner-Product. IEEE Trans. C-17, 1968, pp. 693-694.
Wozniakowski [74]	Wozniakowski, H., Maximal Stationary Iterative Methods for the Solution of Nonlinear Equations. To appear, 1974.