COMPLEXITY OF APPROXIMATELY SOLVED PROBLEMS

J.F. Traub

Computer Science Department

Columbia University

Presented at the Symposium on Complexity of Approximately Solved Problems, April 17, 1985.

As the name of this symposium indicates, its scope is delimited in two ways. We concern ourselves with problems which are *approximately solved* and we are interested in the *complexity* of such problems.

I'll begin by discussing complexity. For the purposes of this symposium, by complexity we restrict ourselves to computational complexity. That's a huge and important area but it's not the only notion of complexity. Since this Symposium is multidisciplinary, I'll introduce several fundamental notions of computational complexity at the risk of boring my computational complexity colleagues for the next couple of moments. By the computational complexity of a problem we mean its intrinsic difficulty as measured by the time, space, or other quantity *required* for its solution. For simplicity, I'll confine myself here to time complexity. Equivalently, the computational complexity of a problem is the cost of the optimal algorithm for its solution. Thus computational complexity defines optimal algorithm. For brevity I will usually refer to computational complexity simply as complexity.

Complexity is an invariant of a problem. It is independent of the algorithm, but may depend on the model of computation. Since I've explored the notion of complexity as an invariant at some length in a recent paper (Traub (1985)), I won't pursue it here.

In general, determining the complexity of a problem is difficult. One establishes a lower bound by proving that a faster way of performing a task cannot exist; and an upper bound, which is the cost of a particular algorithm. The actual complexity is caught from above and below by these bounds. The deeper question is the lower bound, but good upper bounds are also very important.

There is sometimes confusion because people use the word complexity when they refer to the cost of an algorithm. When I say complexity I'll always mean complexity of the problem.

The second scope delimiter of this Symposium is *approximately solved problems.* There are two reasons why we solve a problem approximately; we *can't* solve it exactly or we *choose* not to do so.

To illustrate why we can't solve problems exactly I'll use one of my favorite examples;

it's taken from the study of human vision.

How are humans able to see the world? The late David Marr of M.I.T. and his colleagues have developed a computational model of the human visual system. I'm going to give a simplified description of a small portion of this model. I want to point out, parenthetically, that similar issues arise when we design a vision system for a robot.

Imagine you're looking at an automobile. You can see what shape it has because, roughly speaking, your brain has performed a number of processes at various stages upon images it has received. For example, at one stage it has outlined the images of the various surfaces of the car by detecting the edges that separate them, such as the edge that separates the image of the windshield from the image of the hood. We can detect this edge because there's a sudden change in the slope of the surface; the window and hood do not join *smoothly.*

In the next stage, the human visual system identifies the three-dimensional shapes of the various surfaces. This stage will serve as our example.

How do we infer the shape of the hood? A depth value is the subjective distance to a point on the object as perceived by the viewer. The model assumes that by binocularity or other means we obtain a *finite* number of depth values. In general, between any pair of depth values the hood could have any shape. However, the visual system uses the assumption that the hood is smooth and therefore cannot change too much between depth values. (This notion of smoothness can be made mathematically precise.) Knowing the finite number of depth values and the smoothness of the surface, the visual system *approximately* determines the shape of the hood.

Now I'm going to use this example of determining the shape of the hood to introduce some fundamental concepts.

The first concept is *information.* I do not mean information in the sense of Claude Shannon and information theory. For present purposes, information is what we know about the problem to be solved. In determining the shape of the hood, the information is the finite number of depth values and the assumed smoothness of the surface. Because I may want to regard the surface smoothness as fixed, and study the effect of varying the

depth values, I'll often regard the set of depth values as the information.

The number of depth values is finite. Many different surfaces may have the same depth values; there are not enough depth values to uniquely determine the surface. We say the information is limited, or *partial.*

Furthermore, the subjective distance perceived by the viewer is only an estimate of the true distance. Thus the information is *contaminated* by error.

Because the information is partial and contaminated, we can solve the problem of determining the shape of the hood only approximately. Alternatively, I can say there must be *uncertainty* in the answer and this uncertainty is inherently caused by the available information. It should be clear that partial or contaminated information always leads to inherent uncertainty.

As a second example, I'll use a mathematical problem. It is a simple problem, the computation of a definite integral. For most integrands we cannot compute the integral utilizing the fundamental theorem of the calculus since the antiderivative is not a "simple" function. We have to approximate the integral numerically. Usually, the integrand is evaluated at a finite number of points. The information is the values of the integrand at these points. Since an infinite number of integrands have the same values at these points, the information is partial. The integral is estimated by combining the integrand values. In addition, there will be round-off error in evaluating the integrand, and therefore the information is contaminated. Since with the information we're using we don't know the integrand, there is intrinsic uncertainty in the answer.

This example differs from the previous one in that we started with complete and exact information. The integrand was specified exactly as a function. But we couldn't use that information to solve our problem. We had to throw away our complete and exact information and replace it by partial and contaminated information.

These are just examples, but problems with partial and contaminated information arise in many disciplines: in economics, psychology, computer science, physics, chemistry, control theory, information theory, signal processing, prediction and estimation, scientific and engineering computation, biology, medicine, geophysics, decision theory, and artificial

intelligence. I'm sure other disciplines occur to you.

There is a third reason why we cannot solve problems exactly. We restrict what we mean by algorithm; that is we restrict how the information can be used to provide an answer. I won't pursue this cause of uncertainty today. (See Traub and Woźniakowski (1984, pp. 50-51)).

We've seen that problems can't be solved exactly because the available or utilized information is partial and contaminated. Now I'm going to consider problems where the information is complete and exact. In principle we can always solve these problems exactly. I want to discuss why we *choose* not to do so.

We may choose not to solve a problem exactly because the complexity is too high. We are willing to live with uncertainty in order to reduce complexity. I'll illustrate this with four examples.

The first example is the approximate solution of NP-complete problems. The standard monograph in this area is Garey and Johnson (1979). An instance of an NP-complete problem is bin packing and we therefore believe that its complexity is exponential in the number of bins. Karmarkar and Karp (1982) have shown that if we're willing to settle for a packing that uses $(1 + \varepsilon)$ times as many bins as the optimal one, then the cost is much less, and this holds for arbitrarily small positive ε. Thus bin packing is an instance where we can choose to solve approximately in order to lower the complexity. There are NP-complete problems where you don't have that choice; the problem remains NP-complete even if we're willing to settle for an approximate solution.

A second example of choosing not to solve exactly is provided by the use of heuristics in artificial intelligence and elsewhere. An instance of the use of heuristics is provided by chess. In chess we don't really have a choice; solving approximately is forced on us by the complexity of the game.

The problem is to find a winning strategy for white (if it exists) against all possible strategies of black. The information is complete and exact and the problem can be solved exactly. Indeed, we have the following gedanken algorithm. Generate the tree of all possible moves. If there exist one or more winning strategies against all moves by black, choose

one of those strategies. This is an algorithm which guarantees a win. If no such strategy exists, no algorithm for winning exists.

Such a "brute-force" approach would be far too expensive and so we use heuristics.

A third example is provided by probabilistic algorithms. A well-known instance is primality testing. The use of probabilistic algorithms for primality testing was pioneered by Rabin (1976) and by Solovay and Strassen (1977). To decrease complexity we settle for an answer with uncertainty; that is, we sometimes get the wrong answer. However, the probability of a wrong answer is "small".

My final example of choosing not to solve exactly is provided by the use of iterative algorithms for problems which could be solved exactly by direct methods. An instance is provided by the solution of large linear systems of order n. Neglecting round-off error, direct methods solve the system exactly at cost proportional to n^3. (Direct methods based on fast matrix multiplication are not used in practice.) That's a nice polynomial cost except that the values of n arising in practice are so large that direct methods take too much time and space.

Thus iterative algorithms are often used, especially for large sparse systems. The linear systems problem has complete information since we're given the matrix and the righthand side. However, the information used by these algorithms are some vectors, consisting of the righthand side and the products of certain matrix-vector multiplication. Thus we turn a problem with complete information into one with partial information in order to reduce complexity.

To be specific, consider any linear system whose matrix is symmetric positive definite and has bounded condition number. Results of Nemirovsky and Yudin (1983) and Traub and Woźniakowski (1984) have shown that the complexity of this problem is *linear* in the order although the complexity also depends on the condition number and on the uncertainty in the solution.

These are four examples of choosing not to solve exactly. I'm sure others occur to you.

An area of great current and future interest in computer science, economics, and other

fields is the study of distributed systems. For simplicity, I will consider a parallel computer as an instance of a distributed system. I want to discuss how distributed systems fit into the theme of this conference.

There are two reasons for distributed systems. The first is that although a centralized system could be used, we select a distributed system for the sake of, say, efficiency. The second reason is that the problem is inherently distributed; examples include resource allocation in a decentralized economy, traffic networks, and reservation systems.

Consider now a large distributed system. One possibility is that the total system has complete information but the nodes have only local information, say, about themselves and their neighbors. Thus the information is distributed over the system. To give the nodes information about the total system would cost too much in time and/or space. Thus, decisions are made at nodes which have only partial information and that means a solution with uncertainty. In a dynamic system, even if complete information is initially available at the nodes, we cannot afford to update that information over time.

So far, I've assumed the total system has complete information. Of course, often even the total system has only partial or contaminated information and what I've said holds in spades.

An implication of this discussion is that even the problems that are now exactly solved on a uniprocessor will be only approximately solved in the distributed environments of the future. New models and new analyses will be required.

I've been discussing the available information. To discuss complexity we must have a model of computation, that is, we must decide what is permitted and how much it costs. I am now going to bring together information and complexity.

To include complexity, we must decide whether to charge for information. For the travelling salesman problem the information is the intercity distances. For linear programming the information is the matrix and the linear cost function. In the formulation of these and many other problems the information is assumed to be free. The complexity is then the minimum cost of operating on the information to obtain a solution.

For other problems, such as the vision and integration examples I discussed earlier, we

assume that information costs. There are many other examples. For instance, in mineral exploration, a seismologist might set off explosions whose effect is measured by sensors. That's an expensive process. If you assume that information costs, then the complexity is the minimum of the sum of two costs: the cost of the information plus the cost of combining the information to obtain a solution.

I will use the example of linear programming to illustrate an important dichotomy. In linear programming the information is assumed to be complete, exact, and free. These assumptions are also made for many other important problems, for example, for NP-complete problems. Those assumptions are typically not stated explicitly but they are important. I will use the phrase *combinatorial complexity* to denote the study of complexity when the information is complete, exact, and free. I will use the phrase *information-based complexity* when the information is partial, contaminated, and priced.

Information-based complexity has both an information level and a combinatorial level. At the information level we answer questions such as:

- What is the intrinsic uncertainty in the solution due to the available information?
- How much information is needed to limit the uncertainty to a specified level. For example, in geophysical exploration, how many measurements must the seismologist make?

At the combinatorial level we answer questions such as:

- What is the minimal cost of combining the information to obtain an answer?

The central question of information-based complexity is the following:

- What is the computational complexity of solving a problem for a given level of uncertainty?

Answering this question requires both the information and combinatorial levels.

At the information level the adversary arguments are based on the quantity and quality of the available information. The information level is essentially independent of the model of computation. Speaking technically for a moment, there *is* a model at the information

level. It is sometimes called an oracle model, but an oracle model is extremely weak and nonrestrictive.

At the combinatorial level the results of information-based complexity *are* dependent on the model of computation. Since for problems with complete and exact information there is only the combinatorial level, the model of computation plays a key role in combinatorial complexity.

Often, but not always, the complexity bounds are much tighter for information-based complexity than for combinatorial complexity. For example, for certain classes of problems we can prove that the combinatorial cost is small compared to the information cost and we can therefore use the great power of the information level arguments to obtain extremely tight bounds, sometimes good to within the cost of one arithmetic operation.

On the other hand, Papadimitriou and Tsitsiklis (1984) study a problem of decentralized control where the information is partial. They show that the amount of information required is polynomial in the reciprocal of the uncertainty. However, the complexity of combining the information is NP-complete. I think this is an interesting bridge paper between information-based complexity and combinatorial complexity, and I anticipate it will be the first of many papers which have the flavor of both these kinds of complexity.

In both combinatorial complexity and information-based complexity we judge algorithms by their performance in various settings including worst case, average case, and probabilistic. Since many of the speakers will refer to these, I want to remind you what the criteria are in the various settings.

In a *worse case* setting we want to minimize the cost for the most difficult problem instance. This is sometimes called a minimax criterion.

In an *average case* setting we want to minimize the expected cost.

In a *probabilistic* setting we want to minimize the cost while requiring that the probability of a large error is small. If a problem has a yes or no answer, then we require that the probability of a wrong answer is small.

In closing I'd like to summarize what I see as the essential differences and similarities

between combinatorial complexity and information-based complexity. I'll start with the differences.

In combinatorial complexity the information is complete, exact, and free. Only the combinatorial level is of interest. We can solve problems exactly unless the complexity makes that impossible; then the problem is said to be intractable.

In information-based complexity the information is partial, contaminated, and it costs. Some questions can be answered at the information level, others at the combinatorial level, and still others require both levels. Usually, problems cannot be solved exactly with finite complexity.

What are the similarities? As I indicated earlier, in both combinatorial complexity and information-based complexity we study the performance of algorithms in various settings. The most important commonality is that in both areas we study the intrinsic difficulty of solving problems and we seek optimal algorithms for their solution.

I've discussed the complexity of approximately solved problems in great generality. For the remainder of this Symposium you'll hear about results at the frontiers from the researchers who established those frontiers.

References

- Garey, M.R., and Johnson, D.S. (1979), "Computers and Intractability", Freeman, San Francisco, California.
- Karmarkar, N., and Karp, R.M. (1982), An efficient approximation scheme for the one-dimensional bin-packing problem, *23rd Annu. Symp. Found. Comput. Sci.*, 312-320.
- Nemirovsky, A.S. and Yudin, D.B. (1983), "Problem Complexity and Method Efficiency in Optimization", Wiley-Interscience, New York. Translated from "Slozhnost' zadach i effektivnost' metodov optimizatsii".
- Papadimitriou, H., and Tsitsiklis, J. (1984), Intractable Problems in Control Theory, Technical Report, Stanford University.
- Rabin, M.O. (1976), Probabilistic algorithms, *in* "Algorithms and Complexity: New Directions and Recent Results", (J.F. Traub ed.) Academic Press, New York, 21-39.
- Solovay, R., and Strassen, V. (1977), A fast Monte-Carlo test for primality, *SIAM J. Comput.* **6**, 84-85; erratum, **7**, 118(1978).
- Traub, J.F. (1985), Information, Complexity, and the Sciences, University Lecture, Columbia University.
- Traub, J.F., and Woźniakowski, H. (1984), Information and Computation, chapter in *Advances in Computers* **23**, 35-92.
- Traub, J.F., and Woźniakowski, H. (1984), On the Optimal Solution of Large Linear Systems, *JACM* **31**, 545-559.