ON REALITY AND MODELS

Joseph F. Traub

I. Introduction

Recently, I heard a researcher present a colloquium on
computational aspects of protein-folding. Although this man was
obviously an expert on the topic, he casually mentioned in passing
that, of course, ``protein-folding is NP-complete''.

Protein-folding is a biological process that nature performs swiftly.
One question that scientists would like to answer is: Given a linear
sequence of amino acids, into what three-dimensional configuration will
the sequence fold? Experience to date is that this process is very
difficult to simulate on the most powerful supercomputers. Fraenkel
(1993) proved that a particular mathematical model (minimal energy) of
protein-folding is NP-complete in the Turing machine model of
computation.

Note that four worlds come into play here; see Figure 1. Above the
horizontal line are two real worlds; the world of biological phenomena
and the computer world, where simulations are performed. These are
worlds of atoms and electrons. Below the horizontal line are two
formal models: a mathematical model of the biological phenomenon and a
model of computation. In the formal models, representations are in
bits.

Figure 1. Four worlds.

The mathematical model is an abstraction of the natural world while the
model of computation is an abstraction of the computer world. We get
to select both of these abstractions and the next section will be
devoted to a discussion of these choices.

Discussions of multiple worlds may also be found in Traub and
Wo\'zniakowski (1991), Traub (1992), Jackson (1994) and Casti (1996).

The statement ``protein-folding is NP-complete'' co-mingles a real-world
phenomenon with formal models. This is a not uncommon shortcut but if
we are to make progress on a theory of scientific limits, it will be
important to keep the distinction between reality and models clear.

What is the current situation with respect to protein-folding in these
four worlds?

•Protein-folding: Nature does it fast.

•Computer simulation: Protein folding cannot be done on the
fastest computers.

•Formal models: A particular model (minimal energy) has
been proven NP-complete in the Turing machine model of computation.
Most experts believe that NP-complete problems are computationally
intractable.

After we have built our understanding of some of the issues regarding
mathematical and computer models we will explore, in the concluding
section, the dissonance among nature, simulation, and models.

I'll summarize the remainder of the chapter. In the next section I will
discuss formal models focusing on models of computation. In Section III
the intrinsic difficulty of solving a mathematical model, as

measured by its computational complexity, will be discussed. In the concluding section I will return to protein-folding and apply what we've learned to present some reasons for the dissonances in our current state of knowledge.

II. Formal Models

Computational complexity results and the limits they imply for the computer solution of mathematical models depend on the model of computation; that is, on the abstract model of the computer. The model of computation should be appropriate to the mathematical model, which in turn depends on our idea of reality.

Here, physical phenomena will be used as my real world illustration. Although some physicists believe that space and/or time is ultimately discrete, most physicists seem to believe that they are continuous. Furthermore, if space and/or time are discrete, it is at scales many orders of magnitude smaller than the Planck length.

What about the mathematical models built by physicists or applied mathematicians? There is, of course, considerable interest in discrete models such as cellular automata. However, most mathematical models are continuous. These include the dynamical systems of classical physics and the operator equations and path integrals of quantum mechanics. That is, in their mathematical models, physicists use number fields such as the real and complex numbers. For simplicity I will refer only to the reals in what follows.

It is well understood that the real numbers are an abstraction. That is, it would take an infinite number of bits to represent a single real number; an infinite number of bits are not available in the universe. Real numbers are utilized because they are a powerful and useful construct.

Let us accept that today continuous models are central to mathematical physics and that they will continue to occupy that role for at least the foreseeable future. But the computer is a finite state machine. What should we do when the continuous mathematical model meets the finite-state machine?

I will compare and contrast two models of computation: the Turing machine and the real-number model. In the interest of full disclosure I want to tell you that I've always used the real-number model in my work but will do my best to present balanced arguments. I will assume the reader is familiar with the Turing machine as the abstraction of a digital computer. In the real-number model we assume that we can store and perform arithmetic operations and comparisons on real numbers exactly and at unit cost. Of course, this is an abstraction and the test is how useful and close the abstraction is to reality.

The real-number model has a long history. It was used for polynomial evaluation (Ostrowski (1954)), in optimal iteration theory (Traub (1964)), algebraic complexity (Borodin and Munro (1975)), information-based complexity (Traub, Wasilkowski and Wo\'zniakowski (1988)), and in continuous combinatorial complexity (Blum, Shub, and Smale (1989)). See also Moore (1995), for recursion theory on the reals and a chaotic dynamical systems approach surveyed by Siegelmann (1995).

What are the pros and cons of these two models? I'll begin with the pros of the Turing machine model.

The attraction of the Turing machine is its simplicity and economy of

description. Turing's definition of computability is equivalent to
other definitions and according to the Church-Turing thesis it may be
considered a universal definition of computability. See, however,
Moore (1995) and Siegelmann (1995); Siegelmann claims her model is a
``super-Turing'' machine.

I'll turn to the cons of the Turing machine model. I believe it is not
natural to use the discrete model in conjunction with continuous
mathematical models. Furthermore, estimated running times are not
predictive of scientific computation on digital computers.

I'll move now to the pros of the real-number models. Many mathematical
models in physics, and generally in science and engineering, are
continuous and use the real number system. For such formulations it
seems natural to also use the real numbers in the model of computation.

For example, investigation of the computational complexity of path
integrals has recently been initiated; see Wasilkowski and Wo\'zniakowski
(1995). The real-number model is used; I believe a Turing machine
model would not be natural.

Most scientific computation use finite-precision, floating point
arithmetic. Modulo stability, computational complexity in the real
number model is the same as for finite precision floating point.
Therefore, the real-number model is predictive of running times for
scientific computation.

The final pro that I'll mention here is that by using the real-number
model one has at hand the full power of continuous mathematics. Here's
just one example of the significance of that. There has been
considerable interest in the physics community in the result that there
exist differential equations with computable initial conditions and
non-computable solutions. (Whether physicists should be concerned
about non-computability is an issue that I will take up in another
paper.) This follows from a theorem on ill-posed problems established
by Pour-El and Richards. They use computability theory to establish
their result and devote a large portion of a monograph, Pour-El and
Richards (1988), to develop the mathematical underpinnings and to prove
the theorem.

An analogous result on ill-posed problems has been established using
information-based complexity, which relies on the real-number model.
(Information-based complexity will be discussed in Section IV.) The
proof takes about one page; see Werschulz (1987). More importantly, in
information-based complexity it is natural to consider the average
case. It was recently shown that every ill-posed problem is well-posed
on the average for every Gaussian measure; see Traub and Werschulz
(1994) for a survey. There is no corresponding result using
computability theory. The theme of average behavior will play a
prominent role in the final two sections.

An eloquent argument for the real-number model is given in the
``Manifesto'' by Blum, Cucker, Shub, and Smale (1995).  They write ``Our
point of view is that the Turing model\dots is fundamentally inadequate
for giving a foundation to the theory of modern scientific
computation.''

The con of using the real-number model is that it would be attractive
to use a finite-state model for a finite state machine.

The pros and cons of the Turing machine and real-number models are
summarized in Table 1.

Table 1. Pros and Cons of Two Models.
Turing Machine Model

Pro:
• Simple, robust

Con:
• Not predictive for scientific computation

Real-Number Model

Pro:
• ``Natural'' for continuous mathematical models

Pro:
• Predictive for scientific computation

Pro:
• Utilizes the power of continuous
mathematics

Con:
• Attractive to use finite-state model for finite-state
machine

Some of my colleagues are uncomfortable with the use of the real-number
model because they believe that both the mathematical models and the
model of computation should be finite. See, for example, the brief
notes by Casti, Jackson, and Landauer in Casti and Traub (1994).

Note that the Turing machine model is not finite since it uses an
unbounded tape. I would characterize the Turing machine as discrete
but unbounded. Then, why not use a finite model of computation? There
are such models (for example, circuit models and linear bounded
automata), but they are special purpose. See Table 2 for the
distinctions.

Table 2. Finite and Unbounded Models.

Finite Models:

• Circuits

• Linear bounded automata

Unbounded Models:

Discrete

• Turing Machine

Continuous

• Real Number Model

The idea of using only finite mathematical and computational models is
certainly attractive. We'll have to wait and see if scientists succeed
in building them.

III. Computational Complexity

Computational complexity measures the minimal computational

resources required to solve a mathematically-posed problem. For brevity, I'll often use ``complexity'' for the remainder of this paper. I'll comment on this informal definition:

• Consider all possible algorithms for solving a problem; those known and those existing only in principle. The complexity is the minimal cost over all possible algorithms.

• In Traub (1991), I suggest how ideas analogous to those used in complexity might be used to prove limits to scientific knowledge. I will not pursue that theme here.

• The complexity may be regarded as measuring the intrinsic difficulty of a mathematically posed problem.

• The ``computational resources'' may be time, memory, area on a chip, etc. In this paper the resource will always be time. Then the complexity is the minimal time required to solve a problem exactly or to prescribed accuracy.

• The complexity depends on the problem, not on the algorithm for solving it. It also depends on the model of computation and on the guarantee we offer regarding the solution (the ``setting''). I'll return to this below.

• Computational complexity is both the difficulty of a problem and the name of a field of study. The meaning is usually clear from context.

• Complexity may be thought of as the ``thermodynamics of computation'' with intrinsic limits on what any heat engine can do replaced by limits on what any algorithm can do. See Packel and Traub (1987).

Computational complexity comes in various flavors. The structure is shown schematically in Figure 2. The top node in this tree is all of complexity. This may be divided into discrete and continuous complexity.

The node labelled discrete represents discrete combinatorial problems. Typical here is the well-known Traveling Salesman Problem (TSP). The input is the location of n cities; these locations are usually represented with a finite number of bits. The input specifies a single TSP; the information is complete.

Figure 2. Schema of computational complexity.

Continuous complexity may be divided into two parts; information-based complexity (IBC), and continuous combinatorial complexity. Typical problems of IBC are multivariate and path integration. Most integration problems that occur in practice have to be solved numerically. The mathematical input is the integrand but the information available for solving the problem consists of a finite number of integrand evaluations. This information usually does not specify an integrand uniquely; the information is partial.

Finally, a typical problem of continuous combinatorial complexity is 4-satisfiability; does a system of quartic polynomial equations have a

real zero? The input to this problem consists of the coefficients, taken as real numbers.

Table 3 distinguishes among these three areas of computational complexity with respect to the model of computation and available information. Note that combinatorial complexity, whether discrete or continuous, makes the same assumption about information. The difference is that discrete combinatorial complexity uses the Turing machine or an equivalent model, whereas continuous combinatorial complexity uses the real-number model. Note that IBC and continuous combinatorial complexity use the real-number model but make opposite assumptions about information.

I will briefly indicate results, starting with combinatorial complexity. How does the complexity grows with the size of the input? For example, in TSP, the size of the input is the number of cities, n. Typically, we do not know the complexity of combinatorial problems. We don't even know if the complexity grows polynomially or exponentially with the size of the input.

If the complexity grows polynomially we say the problem is tractable; if the growth is superpolynomial, e.g., exponential, we say it is intractable.

Since we don't know the complexity of a combinatorial problems we have to settle for a complexity hierarchy. Perhaps the hierarchy collapses, at least partially. The famous conjecture $P\ne NP$ states that at least a portion of the hierarchy does not collapse; see, for example, Papadimitriou (1994).

Table 3. Flavors of complexity.

Today we do not know if TSP is tractable or intractable. Most experts believe $P \ne NP$ and that TSP is therefore intractable. But that remains only a conjecture.

What we do know is that many combinatorial problems are equivalent from the complexity viewpoint. They are all tractable or all intractable. The "hardest" problem in the class of NP problems, in the sense of reduction, is said to be NP-complete. Blum, Shub, and Smale (1989) gave a certain formalization of the real-number model, often called a BSS machine. They established that 4-satisfiability is NP-complete over the reals.

We conclude this section with results from information-based complexity. Here we do often have tight bounds on complexity and do not have to content ourselves with a complexity hierarchy. I'll use the diagram of Figure 3 to explain why we can obtain complexity bounds in IBC.

Figure 3. Schema for information-based complexity.

The mathematical problem to be solved is specified by the operator S that maps the mathematical input, I_m, into the mathematical output O_m. This is very general since one can think of all computation as taking inputs into outputs.

Suppose now that the mathematical input is a real multivariate function. Such a function cannot be input to a digital computer. Thus the function has to be replaced by a finite set of numbers, say, evaluating the function at a finite number of points. The operator N maps the mathematical input, I_m, into the computer input

I_c. It's crucial that N is a many-to-one operator, i.e.,
knowing I_c does not give us I_m. Indeed, in IBC there
are typically an infinite number of indistinguishable mathematical
inputs corresponding to a computer input.

A computer algorithm maps the computer input, I_c, into the computer
output O_c. Note that $O_c\ne O_m$. Since N is
many-to-one, we can't know which mathematical problem we're solving and
therefore can, at best, solve the problem only approximately.
Mathematically stated, N composed with ϕ does not
commute with S.

Now I can explain why we can often get tight lower and upper bounds on
the computational complexity of IBC problems. We can use arguments
based on how powerful the information operator has to be. (Indeed,
this is why the field is called information-based complexity). See the
monograph by Traub, Wasilkowski, and Wo\'zniakowski (1988) for rigorous
mathematical formulation and analysis, and Traub and Wo\'zniakowski
(1994) for a more informal treatment.

For combinatorial problems the computer input is usually the same as
the mathematical input and there are no information-based arguments.

Although IBC is an abstract theory developed over abstract spaces, the
typical applications are to multivariate functions. Here is a typical
result from IBC. The problem is integration of a function defined on
the unit cube in d dimensions. Assume we are in the worst case
deterministic setting. That is, we guarantee an error at most
ϵ for every integrand in some class of integrands and
randomization is not permitted.

Let the class of integrands be continuous; smoothness is not assumed.
Then it is easy to see that the complexity is infinite for all ϵ;
that is, the problem is unsolvable.

Assume next that the class of integrands is once continuously
differentiable with uniformly bounded derivatives. Then the complexity
is proportional to $(1/\epsilon)^d$. That is, the
complexity increases exponentially with dimension and the problem is
intractable.

For many classes of functions that have fixed smoothness (in the sense
of Sobolev), integration is intractable. For the precise result see,
for example, Traub and Wo\'zniakowski (1991, 1994).

Thus the integration problem is unsolvable or intractable in the worst
case deterministic setting. But this is not an anomaly; typically
multivariate continuous problems are intractable.

Since this is a complexity result we can't beat the intractability
result by inventing a clever new algorithm. The only way to possibly
break intractability is to weaken the assurance.

I'll mention two settings with weaker guarantees. One is the Monte
Carlo (randomized) setting. The guarantee here is that the expected
error, with respect to the distribution on the sample points, is less
than ϵ. Then the complexity of integration is proportional to
$1/\epsilon^2$, independent of d, even when the class of integrands
is only continuous. (Recall that in the worst case deterministic
setting this problem was unsolvable).

The second setting is the average case deterministic setting. Assume a
Wiener measure on the continuous functions. The guarantee is now that

the expected error with respect to the Wiener measure is less
than ϵ.

Since this is a deterministic setting, the evaluation points must be
given. This was a long-open problem of optimal design solved by
Wo\'zniakowski (1991). He established a connection to low discrepancy
sequences in number theory and showed that the complexity is
proportional to $1/\epsilon$, modulo a polylog factor in $1/\epsilon$.
Numerical tests on a problem of mathematical finance involving
integration in 360 dimensions (Paskov and Traub (1995)) suggest that
evaluation at low discrepancy points may be superior to Monte Carlo for
certain problems in mathematical finance.

That completes our brief tour of concepts and results from
computational complexity. In the concluding section I'll return to the
protein-folding problem, applying what we've learned.

IV. Application to Protein Folding

Now that we are equipped with an arsenal of ideas from computational
complexity, I'll return to the issue raised at the beginning of this
paper regarding protein-folding. Here's what is known about the
current status of this biological problem:

• Nature does it quickly

• We cannot simulate the process on even the most powerful
supercomputers

• A particular mathematical formulation is believed to be
computationally intractable in a particular model of computation

It seems to me that a natural question is how does the time that nature
uses to do protein-folding depend on the length of the sequence of
amino acids. Since nature folds proteins very fast and since the
length of N, the amino acid string is large, the time is not
exponential. Is the time superlinear, sublinear, or even constant,
independent of n? It is my understanding from a conversation with
Jonathan King that this is a question that experimentalists have not
asked.

Note that there are two separate issues here:

• to explain how protein-folding occurs in nature, and

• to ask if we can perform a computer simulation of this process?

A similar dichotomy occurs in vision research. We want to

• understand the human visual system, and

• give machines similar abilities.

Both issues are of interest. Humans have excellent visual and pattern
recognition skills due to millions of years of evolution. It has
proven very difficult to give computers such abilities. Progress on
one issue might help with the other but not necessarily.

Should we be concerned that nature does protein-folding easily while,
with our current knowledge, simulation seems hard in practice and
theory? Not necessarily, but in the list presented below I will
suggest some possible reasons for the difference. First, I'll remind
the reader of the current theoretical status. Fraenkel (1993) proved

that a minimal energy model based on a discrete graph representation is
NP-complete in the Turing machine model. That is, if the sequence of
amino acids is of length n and if the conjecture $P \ne NP$
is true, then the problem cannot be solved in running time that is a
polynomial in n.

a) Nature has selected for proteins that fold easily.

b) NP-completeness is a worst case theory. Perhaps the solution of
the mathematical model is easy on the average but we don't know the
prior. Note that nature using selection is one example of an unknown
prior.

The average behavior can be totally different than the worst case
behavior; in Section III, I used the example of high dimensional
integration to show that a problem that is unsolvable or intractable
in the worst case can be tractable in the average case. Here is a
second example. Before the Karmarkar algorithm, the simplex algorithm
was the algorithm of choice for solving linear programming. Although,
due to a result of Klee and Minty, the cost of the algorithm was known
to be exponential in the worst case, practitioners reported that the
cost was a low degree polynomial in the size of the problem. Then
Borgwardt (1982) and Smale (1983) independently proved that the average
cost of the simplex algorithm is a low degree polynomial; this gave a
theoretical explanation of what practitioners had experienced. Thus
there is an exponential difference between the worst and average case
running time. (Note that I'm careful to talk about cost and not
complexity, since these are only properties of a particular
algorithm.)

c) A minimal energy model is commonly used. Perhaps there are other
mathematical models that are not computationally intractable.

d) Fraenkel assumes that the mathematical model is exactly solved but
perhaps it's enough to solve it approximately. IBC problems can
only be approximately solved because the information is partial.
Combinatorial problems can often be exactly solved. It is possible for a
combinatorial problem that is intractable if an exact answer is demanded
to become tractable if an approximate answer suffices. Thus we may
choose to solve a combinatorial problem approximately. See, for
example, Garey and Johnson (1979).

e) The set of inputs must be specified. For some problems, such as
TSP, any set of n points in the Euclidean plane can be an input.
For other problems, the choice of input set can totally change the
problem complexity. I'll illustrate the point with the simple example
of univariate integration in the worst case setting. If the class of
inputs consists of continuous functions then the problem is unsolvable.
If the class of inputs consists of continuous differential functions
with uniformly bounded derivative, then the complexity of computing an
approximation with error at most ϵ is proportional to $1/\epsilon$.

f) Nature may be using massive parallelism, say, of order
10^{23}. Such parallelism might eventually be provided by quantum computation
(see Di Vincenzo (1995) for a recent survey and the references given
there), or biological computation (see Adelman 1995), but this is
currently highly speculative.

g) The NP-completeness result uses the Turing machine model of
computation. Perhaps a different model of computation might be more
appropriate. Possibilities are the real-number model or the
"super-Turing" model mentioned in Section II.

h) We should not forget that intractability of NP-complete problems is only conjectured.

i) Nature may have ways of cutting the complexity of protein folding. See the last section of Fraenkel (1993) for some examples. Fraenkel gives an excellent general discussion of the ramifications of his NP-completeness result.

How might this affect the dissonance between reality and simulation? I'll consider three possibilities from the above list. Algorithms that guarantee good average behavior can be very different than those that guarantee good sworst-case behavior (Point b). Algorithms that are guaranteed to solve a problem approximately can be very different than those that solve a problem exactly (Point d). Thus weakening the guarantee regarding the solution might lead to algorithms that are much cheaper than those in current use.

Finally, the complexity depends on the set of inputs (Point e). For the protein-folding problem the inputs are linear sequences of amino acids of length n. Any prior knowledge restricting the class of inputs might reduce the complexity.

How hard will simulation of protein-folding be in one to two decades? Opinions among biologists vary. When I was asking scientists in the early nineties for their candidates for very hard problems a number of them mentioned simulation of protein-folding as a candidate. On the other hand, Leroy Hood thought it would be routinely solved in one to two decades.

We will see.

Acknowledgements

References

Adleman, L. M. ``Molecular Computation of Solutions to Combinatorial Problems,'' Science, 266 (1994), 1021--1024.

Blum, L., Cucker, F., Shub, M. and Smale, S. ``Complexity and Real Computation: A Manifesto.'' Technical Report TR--95--042, International Computer Science Institute, Berkeley, CA, 1995.

Blum, L., Shub, M., and Smale, S., ``On a Theory of Computation and Complexity over the Real Numbers: NP-Completeness, Recursive Functions and Universal Machines'', BAMS 21 (1989), 1--46.

Borgwardt, K. H. ``The Average Number of Steps Required by the Simplex Method is Polynomial'', Zeitschrift fur Operations Research, 26 (1982), 157--177.

Borodin, A. and Munro, I. The Computational

Complexity of Algebraic and Numeric Problems. Elsevier, New York, 1975.

Casti, J. ``On the Limits to Scientific Knowledge,''
Scientific American, to appear July 1996.

Casti, J. and Traub, J.F. ``On Limits'',
Santa Fe Institute Working Paper, WP--94--10--056, 1994.

DiVincenzo, D. P. ``Quantum Computation'',
Science, 270 (1995)", 255--261.

Fraenkel, A. S. ``Complexity of Protein-Folding'', BMB
55 (1993), 1199--1210.

Garey, M. R. and Johnson, D. S. Computers and
Intractability. W. H. Freeman, San Francisco, 1979.

Jackson, E. A. ``The Second Metamorphosis of Science: A
Working Paper'', Center for Complex Systems Research, Beckman
Institute, U. of Illinois, UIUC--BI--CCSR--94--1, 1994.

Moore, C., ``Recursion Theory on the Reals and
Continuous-time Computation'', Santa Fe Institute Working Paper,
WP--95--09--079, 1995.

Ostrowski, A. M. ``On Two Problems in Abstract Algebra
Connected with Horner's Rule'', in Studies Presented to R. von
Mises, Academic Press, New York, 1954, pp. 40--48.

Packel E., and Traub, J. F. ``Information-based
Complexity'', Nature, 328 (1987), 29--33.

Papadimitriou, C. H., Computational
Complexity. Addison-Wesley, Reading, MA, 1994.

Paskov, S. and Traub, J. F., ``Faster Valuation of
Financial Derivatives'', Journal of Portfolio Management, 22
(1995), 113--120.

Pour-El, M. B. and Richards, J. I.,
Computability in Analysis and Physics. Springer-Verlag, Berlin, 1988.

Siegelmann, H. T. ``Computation
Beyond the Turing Limit'', Science, 268 (1995), 545--548.

Smale, S., ``The problem of the average speed of the simplex
method'', in Proceedings of the 11th International Symposium on
Mathematics, Springer-Verlag, 1983.

Traub, J. F., Iterative Methods for the Solution of
Equations. Prentice-Hall, Englewood Cliffs, N. J., 1964 (reissued by
Chelsea Press, New York, 1982).

Traub, J. F., ``What is Scientifically Knowable?'', in
Twenty-Fifth Anniversary Symposium, School of Computer Science,
Carnegie-Mellon University, Addison-Wesley, Reading, MA, 1991,
pp. 489--503.

Traub, J. F., ``Can We Prove There are Limits to What is
Knowable About the Universe?'', Computer Science Department, Columbia
University, 1992 (Presented at The Mainichi Shimbun Symposium,
December, 1992, Tokyo and Osaka, Japan).

Traub, J. F. and Werschulz, A. G., ``Linear
Ill-Problems are Solvable on the Average for All Gaussian Measures'',
The Mathematical Intelligencer, 16, No. 2 (1994), 42--48.

Traub, J. F. and Wo\'zniakowski, H., ``Breaking
Intractability'', Scientific American, January 1994, 102--107.

Traub, J. F. and Wo\'zniakowski, H., ``Theory and
Applications of Information-Based Complexity''. Presented in the 1990
Lectures in Complex Systems at the Santa Fe Institute, Addison-Wesley,
Reading, MA, 1991, pp. 163--193.

Traub, J. F. and Wo\'zniakowski, H.,
``Information-based Complexity: New Questions for Mathematicians'',
The Mathematical Intelligencer, 13 (1991), 34--43.

Traub, J. F., Wasilkowski, G.
W., and Wo\'zniakowski, H., Information-Based Complexity.
Academic Press, San Diego, CA, 1988.

Wasilkowski, G. and
Wo\'zniakowski, H., ``On Tractability of Path Integration'', J.
Math. Physics, to appear 1996.

Werschulz, A. G., ``What is the Complexity of Ill-Posed
Problems?'', Num. Func. Anal. Optim., 9 (1987), 945--967.

Wo\'zniakowski, H., ``Average Case Complexity of
Multivariate Integration'', BAMS 24 (1991), 185--194.