Complexity Theory of Real Functions

Ker-I Ko

State University of New York at Stony Brook

Contents

Introduction

Starting with Cook's pioneering work on *NP*-completeness in 1970, polynomial complexity theory, the study of polynomial-time computability, has quickly emerged as the new foundation of algorithms. On the one hand, it bridges the gap between the abstract approach of recursive function theory and the concrete approach of analysis of algorithms. It extends the notions and tools of the theory of computability to provide a solid theoretical foundation for the study of computational complexity of practical problems. In addition, the theoretical studies of the notion of polynomial-time tractability sometimes also yield interesting new practical algorithms. A typical example is the application of the ellipsoid algorithm to combinatorial optimization problems (see, for example, Lovász [1986]). On the other hand, it has a strong influence on many different branches of mathematics, including combinatorial optimization, graph theory, number theory and cryptography. As a consequence, many researchers have begun to re-examine various branches of classical mathematics from the complexity point of view. For a given nonconstructive existence theorem in classical mathematics, one would like to find a constructive proof which admits a polynomial-time algorithm for the solution. One of the examples is the recent work on algorithmic theory of permutation groups.

In the area of numerical computation, there are also two traditionally independent approaches: *recursive analysis* and *numerical analysis*. Recursive analysis studies effective computability in classical analysis; that is, it studies which mathematical notions and proofs are computable and which are not computable. The notion of computational complexity is, however, rarely involved. Numerical analysis studies the computational complexity of numerical algorithms in a problem-oriented approach. The complexity analysis of a problem is based on the finite-precision floating-point model and the complexity measure is often a local measure tailored toward the specific problem under investigation. The relationship between these two approaches to numerical computation is more or less like the relationship between recursive function theory and analysis of algorithms for discrete computation. It is apparent that the two approaches have a big gap and a polynomial complexity theory of real functions is needed to serve both as the theoretical foundation for numerical analysis and as the framework for a complexity theory of algorithmic real analysis. In this book we develop such a polynomial complexity theory of real functions and demonstrate how this theory may be applied to classify the computational complexity of elementary numerical operations. In this theory, the basic notion of polynomial-time computability of real functions is established through a marriage of recursive analysis and discrete polynomial complexity theory. A number of fundamental numerical operations, such as integration, differentiation and root-finding, are examined in this framework. The main results are a hierarchical classification of the computational complexity of these operations in terms of the relationship among the complexity classes of discrete polynomial complexity theory such as *P*, *NP* and *PSPACE*.

We begin with an informal description of the computational model used in the theory. A real number x in this theory is represented by a sequence $\{r_n\}$ of rational numbers satisfying the property that $|r_n - x| \leq 2^{-n}$. Since each rational number r_n has a finite representation (in terms of symbols from a fixed finite alphabet), a real number may be considered as a type-1 function that maps a type-0 object (an integer n) to a type-0 object (a rational number r_n). The notions of computability and polynomial-time computability can then be introduced to real numbers accordingly. Namely, a real number x is *computable* if there is a machine M that computes, for any given integer n, a rational number r_n that approximates x within error 2^{-n}. The real number x is said to be *polynomial-time computable* if this machine M operates in time polynomial in n.

Since real numbers are represented by type-1 functions, a real function $f : \mathsf{R} \to \mathsf{R}$, where R is the set of real numbers, is regarded as a type-2 function that maps a type-1 function to a type-1 function. We follow the approach of recursive function theory to define the notion of computability of type-2 functions. Informally, we say a function $f : \mathsf{R} \to \mathsf{R}$ is computable if there is a machine M that computes, for any given real number x and any output precision n, a rational number s_n such that $|s_n - f(x)| \leq 2^{-n}$. Since x is a type-1 function that does not have a finite representation, machine M cannot directly "read" its input x. Instead, we must provide a more complicated mechanism to allow machine M to access the information about the real number x. In our computational model, we use the oracle machine to formalize the communication between the machine M and the input real number x. We may imagine that x is presented to oracle machine M by a black box that outputs, when given input m, a rational number r_m such that $|r_m - x| \leq 2^{-m}$. During the computation, the oracle machine M for f is allowed to ask the black box for values r_m. Thus, more formally, we say a function $f : \mathsf{R} \to \mathsf{R}$ is *computable* if there is an oracle machine M that, for any black box for x and any input n, outputs a rational number s_n such that $|s_n - f(x)| \leq 2^{-n}$. When we consider the time complexity of an oracle machine M, we charge only one unit of time for each query to the black box made by M. We say a function $f : \mathsf{R} \to \mathsf{R}$ is *polynomial-time computable* if it is computed by an oracle machine M that operates in time polynomial in n, independent of real number x.

The above definition of polynomial-time computable real functions is simple to use and is, as we will prove in Chapter 2, essentially independent of the underlying machine models. Note that the class of polynomial-time computable real functions is defined based on the notion of worst-case complexity. Although we will also consider the notion of average-case complexity in Chapter 5, most of the book will deal with the worst-case complexity. It is our opinion that the worst-case complexity is the simplest and most important complexity measure. It provides a solid foundation for a complexity theory. A thorough understanding of the worst-case complexity theory is important for studies of other types of complexity measures.

A comparison of our definition with the definitions in recursive analysis and numerical analysis is in order. For the definition of computable real functions, there are two different approaches in recursive analysis. The first approach studies computability of real functions in the context of classical real analysis. Nonrecursive objects, as well as nonconstructive arguments, are allowed. Therefore, a computable real function is defined on all real numbers, though most of them are noncomputable. Some earlier researchers following this approach are Grzegorczyk [1955, 1957, 1959], Lacombe [1955, 1957, 1959a] and Mostowski [1957, 1959]. More recently, Pour-El and Richards [1983a, 1983b, 1989] have developed a much deeper theory along this approach. The second approach is more constructive. It only studies recursive objects and only uses constructive logic. A computable real function is, thus, defined only on computable real numbers. Since this approach only deals with constructive objects, many results in this theory are contradictory to the results in classical analysis. For example, in this theory, there exists a function which is continuous on a compact domain but is not uniformly continuous on this domain. The main contributors to this approach include Moschovakis [1964a, 1964b], Goodstein [1961], Šanin [1968], Ceitin [1967] and Aberth [1980]. (There is a third and most constructive approach which uses intuitionistic logic and does not restrict itself to the notion of recursiveness. This approach has been taken by Bishop [1967] and Bishop and Bridges [1985]. This theory is usually called *constructive analysis* to distinguish it from the above two groups which work strictly within recursive function theory.) The general approach of our theory is the same as the first approach of recursive analysis, and our definition of computable real functions agrees exactly with the definition of the first approach. As both numerical analysis and discrete polynomial complexity theory are developed in the logic of classical mathematics, it is appropriate for our theory to be established on the same ground.

The floating-point model is the usual model of computation in numerical analysis [Knuth, 1973]. This is a finite-precision model which does not reflect the structure of the real number system. A well known example shows that the numbers represented in this model do not satisfy even the law of associativity. It is clear that it is not appropriate for a general complexity theory of numerical algorithms. Another model often used in the area of *arithmetic complexity theory* is the real number RAM (random access machine) [Borodin and Munro, 1975]. It assumes that each register of the real number RAM can store a real number with infinite precision. It also assumes that each arithmetical operation can be performed in a unit of time. Although the arithmetics in this model satisfies all mathematical laws, it is apparent that no physical implementation for this model is possible. Our model of computation tries to combine these two models of numerical computation. On the one hand, we use infinite precision representations for real numbers so that mathematical laws are satisfied by these representations. (There are certain inherent limitations to these representations. We will discuss them in Chapter 2.) On the other hand, we accept the limit of discrete processing of digital computers and will measure the cost of numerical algorithms by bit operations. We notice that in numerical analysis as well as in arithmetic complexity theory, bit operations are often used as the measure of cost in analysis of algorithms—despite the fact that this measure is not necessarily the most natural measure of the basic computational models in these theories. Some examples are integer multiplication, fast Fourier transforms, root approximation and Diophantine approximation [Aho, Hopcroft and Ullman, 1974; Henrici, 1974; Lovász, 1986].

Now we turn to the main results of the book. The typical question asked in the book is the following: for an elementary numerical operator F (for example, maximization), what is the computational complexity of F? The answer to this question depends on the precise formulation of this question. The natural formulation is to treat F as a type-3 function that maps a type-2 function to a type-2 function, and use the type-3 oracle machine as the computational model for F. A type-3 oracle machine may be informally considered as a machine with the ability to query the oracle for approximate values of the input function f at any rational point. We say such a machine computes a numerical operator F if for any real function f in the domain of F, any real number x and any integer n, it outputs an approximation s_n to $F(f)(x)$ within error 2^{-n} (where the real function f as well as the real number x are given to the machine in the form of oracles). The worst-case computational complexity of the type-3 machine is measured as a function of the output precision n only, and is independent of the oracle used. This complexity measure seems too strict: for almost all nontrivial numerical operators F we can easily find simple information-theoretic proofs for the nonpolynomial-time computability of the operator F. (The formal model and detailed discussions are included in Section 2.7.) Thus it is difficult to compare the complexity of different operators. One of the reasons for such simple lower bound proofs seems to be that the domain of the operator F is too big and covers many ill-behaved functions.

In this book, we take the approach of discrete polynomial complexity theory. We use the concepts of reducibility and completeness to give a finer classification of the inherent computational complexity of numerical operators. More precisely, for each numerical operator F, we ask whether the operator F maps all polynomial-time computable real functions to polynomial-time computable real functions. Therefore, the complexity problem about a type-3 function F is reduced to a complexity problem about a type-2 function $F(f)$, where f is a polynomial-time computable real function. It is interesting to see that although most basic numerical operators are not polynomial-time computable under the type-3 function model, they are *not known* to be nonpolynomial-time computable under this weaker approach. For instance, consider the maximization operator MAX that maps a function $f : [0,1]^2 \to \mathsf{R}$ to the function $g : [0,1] \to \mathsf{R}$, defined by $g(x) = \max\{f(x,y) \mid 0 \le y \le 1\}$. Then, the operator MAX is provably nonpolynomial-time computable under the type-3 function model, but the question of whether it maps all polynomial-time computable functions to polynomial-time computable functions is equivalent to the major open question "$P = ?NP$" in discrete complexity theory. In the terms of complexity theory, we may say that there is a polynomial-time computable real function f such that MAX(f) is *complete* for the class NP in the informal sense that if MAX(f) is polynomial-time computable then $P = NP$. This type of results thus reveals interesting connections between discrete complexity theory and numerical algorithms and allows us to classify the complexity of numerical problems by the techniques of discrete polynomial complexity theory.

The numerical problems considered in this book include maximization, integration, differentiation, roots, and ordinary differential equations. The computational complexity of these problems forms a hierarchy parallel to the complexity hierarchy in discrete polynomial complexity theory. Let us recall the following important complexity classes (see Chapter 1 for the formal definitions and a more complete treatment):

P: the class of sets accepted by deterministic Turing machines in polynomial time,

- FP: the class of functions computed by deterministic Turing machines in polynomial time,
- NP: the class of sets accepted by nondeterministic Turing machines in polynomial time,
- $LOGSPACE$: the class of sets accepted by deterministic Turing machines in logarithmic space,
- $PSPACE$: the class of sets accepted by deterministic Turing machines in polynomial space,
- $FPSPACE$: the class of functions computed by deterministic Turing machines in polynomial space,
- $\#P$: the class of functions that enumerate the number of accepting computations of polynomial-time nondeterministic Turing machines,
- UP: the class of sets accepted by unambiguous nondeterministic Turing machines (that have at most one accepting computation on any input) in polynomial time.

It is known that

$$LOGSPACE \subseteq P \subseteq UP \subseteq NP \subseteq PSPACE,$$

$$FP \subseteq \#P \subseteq FPSPACE,$$

but none of these inclusions is known to be a proper inclusion. Now the main results in this book may be summarized as follows:

(1) *Maximization.* Let MAX be the maximization operator defined above. Then, $P = NP$ iff for all polynomial-time computable real functions f, $\text{MAX}(f)$ is polynomial-time computable.

(2) *Integration.* Let INT be the operator that maps a function $f : [0,1] \to \mathsf{R}$ to the function $g : [0,1] \to \mathsf{R}$, defined by $g(x) = \int_0^x f(t)dt$. Then, $FP = \#P$ iff for all polynomial-time computable real functions f, $\text{INT}(f)$ is polynomial-time computable.

(3) *Inverse Functions: one-dimensional case.* Let INV_1 be the operator that maps a one-to-one function $f : [0,1] \to [0,1]$ to its inverse function f^{-1}. Then, $\text{INV}_1(f)$ is polynomial-time computable for all polynomial-time computable, one-to-one real functions f on $[0,1]$. On the other hand, $LOGSPACE = P$ iff for all log-space computable, one-to-one real functions f, $\text{INV}_1(f)$ is log-space computable.

(4) *Inverse Functions: two-dimensional case.* Let INV_2 be the operator that maps a one-to-one function $f : [0,1]^2 \to [0,1]^2$ to its

inverse function f^{-1}. Then, $P = NP$ implies that for all polynomial-time computable, one-to-one real functions f on $[0,1]^2$, $\textsc{Inv}_2(f)$ is polynomial-time computable, and this in turn implies $P = UP$.

(5) *Differentiation.* There exists a polynomial-time computable function f on $[0,1]$ such that the derivative f' exists but is not *computable*. On the other hand, if the second derivative f'' exists and is continuous on $[0,1]$, then f' must be polynomial-time computable. Furthermore, if f is analytic on $[0,1]$ then the sequence $\{f^{(k)}\}$ is a polynomial-time computable sequence of real functions.

(6) *Ordinary Differential Equations.* There is a natural weak Lipschitz condition on functions $f : [0,1]^2 \to [-1,1]^2$ such that $P = PSPACE$ iff for all first-order ordinary differential equations $y' = f(x,y)$ defined by polynomial-time computable functions f satisfying this weak Lipschitz condition, the solutions y are polynomial-time computable.

(7) Other results include the applications of discrete complexity theory to the study of average-case polynomial-time computability of real functions, based on the polynomial version of the Lebesgue measure theory, to the study of the polynomial-time version of the Weierstrass approximation theorem, and to the study of a specific optimization problem in control theory.[1]

In addition to the above exact characterizations for the operators such as $\textsc{Max}$ and $\textsc{Int}$, some weaker characterizations for weaker operators are also found. For instance, let $\textsc{Max}_1$ be the operator that maps each function $f : [0,1] \to \mathsf{R}$ to its maximum value $\max\{f(x) \mid 0 \le x \le 1\}$. Then, what is the complexity of $\textsc{Max}_1(f)$, if f is known to be polynomial-time computable? Note that the immediate analogous question in discrete complexity theory is meaningless, as the maximum value of a function $\phi : \{0,1\}^* \to \{0,1\}^*$ may not exist, and, even if it exists, it must be a finite string and has a constant time complexity. The more appropriate formulation is probably to ask what the complexity is of the function $\psi(n) = \max\{\phi(s) \mid \text{length of } s \text{ is } n\}$. Formulating this question into a language recognition problem, we found that the complexity of $\textsc{Max}_1(f)$ depends on the relation between polynomial-time computable *sparse* sets and

[1] A more recent work on the complexity of integral equations appears in Ko [1991].

NP-time computable *sparse* sets.[2] More precisely, we prove that if all sparse sets in Δ_2^P are actually in P then for all polynomial-time computable real functions f, $\textsc{Max}_1(f)$ is polynomial-time computable. On the other hand, if $P \neq NP$ on sparse sets, then $\textsc{Max}_1(f)$ is not polynomial-time computable for some polynomial-time computable real function f. Similar results are proved for the operators $\textsc{Int}_1$ ($\textsc{Int}_1(f) = \int_0^1 f(t)dt$) and $\textsc{Root}$ ($\textsc{Root}(f)$ on a one-to-one function f is the point $\langle x, y\rangle \in [0,1]^2$ such that $f(\langle x, y\rangle) = \langle 0, 1\rangle$).

Interestingly, the notion of sparse sets has itself played an important role in discrete polynomial complexity theory. In particular, it has a close connection with the notion of non-uniform complexity: all sparse sets are computable by circuits of a polynomial number of gates. Our results about the relation between the complexity of real numbers and the complexity of sparse sets allow us to obtain immediately many interesting properties of maximum values of polynomial-time computable functions. For instance, we know that they are not necessarily polynomial-time computable if $P \neq NP$ on sparse sets, but they cannot be NP-complete unless $P = NP$. Thus they form an intermediate class in between P and NP. (The complexity of the sets in this intermediate class is however, at the present time, not well understood yet.)

It is apparent that our research has just begun to show interesting, nontrivial relations between discrete complexity theory and numerical computation. The main results are about the complexity of fundamental numerical operations. The questions about the complexity of numerical problems involving infinite-dimensional space or abstract spaces other than the Euclidean space have remained untouched. As discrete complexity theory itself is still evolving, we expect deeper results to come in the coming years, in both discrete complexity theory and in our theory of numerical computation. Recently a number of other new theories have been developed for the study of computational complexity of real functions. These theories often use different models of computation and aim at different domains of numerical problems. Among these works, Schönhage [1982], Papadimitriou and Tsitsiklis [1986], Hoover [1987], Kreitz and Weihrauch [1982] and Müller [1987] use essentially the same compu-

[2] A sparse set is a set that has at most polynomially many strings of any give length; see Section 1.8.

tational model as ours, and most of them emphasize the importance of polynomial-time computability. Traub, Wasilkowski and Wozniakowski's [1983] information-based complexity theory uses a more general model and emphasizes the notion of information complexity rather than the notion of combinatorial complexity which is fundamental in discrete complexity theory. Blum, Shub and Smale [1989] proposed a new model which is close to Borodin and Munro's [1975] model of real number random access machines. This theory also emphasizes the importance of polynomial-time computability and they showed that some basic numerical problems are *NP-complete* under this model. (The relation between their new notion of *NP*-completeness and the notion of *NP*-completeness in discrete complexity theory is not clear.) We hope that this book will motivate more interesting studies on computational complexity of real functions.