Polynomial-Time Computability in Analysis

Ker-I Ko*
Department of Computer Science
State University of New York at Stony Brook
Stony Brook, NY 11794
keriko@cs.sunysb.edu

1 Introduction

Research in computational complexity of the past thirty years has clearly established polynomial-time complexity theory (or, the theory of NP-completeness) as the new foundation of algorithms. Recursion theory identifies the intuitive notion of effective computability with the formal notion of recursiveness, and hence provides a formal setting for us to study what problems are computable and what are not computable. Polynomial-time complexity theory identifies the intuitive notion of feasible computability with the formal notion of polynomial-time computability, and allows us to formally study what problems are feasible and what are infeasible. This new theory ties closely the tools in abstract mathematics with practical computational problems. It has made strong impact on almost every area of discrete computation, including graph theory, combinatorial optimization, computational geometry, computational number theory and modern cryptography (cf. Garey and Johnson [1979]). The influence on numerical computation came later but is definitely not weaker. Due to the continuous nature of numerical computation, the discrete polynomial-time complexity theory cannot be applied directly to numerical problems. Extra effort has to be made to formulate formal computational models of continuous functions to allow us to apply discrete complexity theory to numerical problems. Different approaches aiming at different areas of numerical problems may adopt different computational models, but it is universal to identify the notion of polynomial-time computability with the notion of feasible computability.

In this paper, we present a short survey on one of the polynomial-time complexity theories of numerical computation based on the model of recursive analysis. We will call this the *theory of polynomial-time analysis*. In this theory, a real number x is represented by a sequence of rational numbers that converges to x in a predefined rate of convergence; that is, it is represented as a *type-1 function* satisfying the convergence condition. A real function f is then represented by a *type-2 function* satisfying some continuity condition.

*Research supported in part by NSF grant CCR 9121472.

The advantage of using this model of computation is that it is compatible with the model of discrete complexity theory, and so the complexity of numerical problems may be studied and compared with discrete problems. Using this model, we are able to classify the computational complexity of numerical problems in terms of the complexity classes in the discrete polynomial-time complexity theory such as *P, NP* and *PSPACE*. This approach to polynomial-time complexity theory of numerical computation has been first introduced by Ko and Friedman [1982], and further established since then in Ko [1991, 1992, 1995a, 1995b, 1997], Friedman [1984], Hoover [1987, 1990], Papadimitriou and Tsitsiklis [1986], Chou and Ko [1995] and Ko and Weihrauch [1996].

This paper is organized as follows. First we introduce the computational model and some computability results in recursive analysis, including some recent results by Pour-El and Richards [1983a, 1983b, 1988]. Next we add the complexity measures to the computational model of recursive analysis to form the model for polynomial-time analysis. Then, a hierarchical classification of the computational complexity of numerical problems is presented in terms of the relations among discrete complexity classes. The problems to be considered including maximization, root-finding, integration, ordinary differential equations, integral equations and measures of two-dimensional regions. We will not include any formal proofs but will give precise definitions and statements of theorems. Finally we discuss different approaches to the polynomial-time complexity theory of real functions, including the works of Blum et al. [1989] and Traub et al. [1988].

2 Recursive Analysis

In this section, we give a short summary of some recent results in recursive analysis. We do not intend to present a complete survey on recursive analysis. Instead, we only include some results to motivate the study of these problems in the context of polynomial-time complexity theory of analysis. The reader is referred to Pour-El and Richards [1988] and Weihrauch [1995, 1997a] for more complete treatments. Other interesting results include Metakids and Nerode [1982] and Metakids, Nerode and Shore [1985].

2.1 Computable Real Functions

As it is well known, 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. A computable real function is defined on all real numbers, though most of them are noncomputable [Pour-El and Richards, 1988]. 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 [Šanin, 1968; Aberth 1980].[1] We will only discuss recursive

[1]A third and most constructive approach is the constructive analysis of Bishop [1967] and Bishop and Bridges [1985] which uses intuitionistic logic and does not restrict itself to the notion of recursiveness, and so is quite different from the above two approaches of recursive analysis. Nevertheless, their work is closely

analysis in the first approach.

The model of computation in recursive analysis is based on the Turing machine (TM) model.[2] Turing machines were designed to deal with discrete objects, more specifically, finite strings of symbols. Therefore, when we deal with real numbers and real functions, we must first establish a representation system in which real numbers and real functions are represented as functions or functionals on finite strings. We first fix the set of rational numbers $\mathbf{Q}$ as the base set of numbers which are representable by finite strings; namely, a rational number p/q, in the reduced form, is represented as s_p/s_q, where s_p and s_q are two strings over $\{+,-,0,1\}$ representing integers p and q. Based on the notion of Cauchy sequence representation of real numbers, a real number x is represented by a function $\phi : \mathbf{N} \to \mathbf{Q}$ having the property that $\{\phi(n)\}$ converges to x.

Definition 2.1 *A real number x is* recursive *(or,* computable*) if there exist recursive functions $\phi : \mathbf{N} \to \mathbf{Q}$ and $t : \mathbf{N} \to \mathbf{N}$ such that $\lim_{n\to\infty} t(n) = \infty$ and $|\phi(n) - x| \leq 1/t(n)$.*

In other words, x is recursive if it has a recursive representation ϕ that converges to x recursively. It turns out that the converging rate could be required to be faster without changing the class of the recursive real numbers. In addition, the other definitions based on the notion of Dedekind cuts and binary expansions are also equivalent [Robinson, 1951].

Proposition 2.2 *The following are equivalent:*

(a) x is a recursive real number.

(b) There exists a recursive function $\phi : \mathbf{N} \to \mathbf{Q}$ such that for all $n \geq 0$, $|\phi(n)-x| \leq 2^{-n}$.

(c) The set of rational numbers r that are less than x is a recursive set.

(d) There exists a recursive function $\phi : \mathbf{N} \cup \{-1\} \to \mathbf{N}$, with $\phi(-1) \in \{1,-1\}$ and $\phi(i) \in \{0,1\}$ for all $i > 0$, such that $x = \phi(-1) \cdot (\sum_{i=0}^{\infty} \phi(i) \cdot 2^{-i})$.

Although these representations of real numbers define the same class of recursive real numbers, they do not define same classes of subrecursive real numbers. Through careful studies, it has been established that the Cauchy sequence representation is the most comprehensive representation system [Mostowski, 1957; Ko, 1983a; Weihrauch, 1997a]. From the above equivalent definition (b), we say a function ϕ *binary converges* to x if $|\phi(n)-x| \leq 2^{-n}$.

Since a real number is represented by a function mapping integers to rationals, a real function is a *functional* mapping integer functions to integer functions, or, as commonly called in recursive function theory, a *type-2 function*. Computable functionals have been studied thoroughly in recursive function theory. A standard machine model for computable functionals is the function-oracle Turing machine.

A *function-oracle Turing machine* is an ordinary Turing machine M equipped with an additional query tape and two additional states: the query state and the answer state. When the machine enters the query state, the oracle, a function ϕ, replaces the current

related to recursive analysis; see the discussions in Metakids, Nerode and Shore [1985].

[2]For the definition of Turing machines, refer to any standard textbook on theory of computation, e.g., Hopcroft and Ullman [1979].

string s in the query tape by the string $\phi(s)$, moves the tape head back to the first cell of the query tape, and puts the machine M in the answer state. When the time complexity is considered, the whole process of querying for the value $\phi(s)$, i.e., the move from the query state to the answer state, costs only one time unit to the machine. The computation of the function-oracle machine M on input s with oracle ϕ is written as $M^{\phi}(s)$.

Definition 2.3 *A real function $f : \mathbf{R} \to \mathbf{R}$ is* recursive *(or,* computable*) if there is a function-oracle TM M such that for each $x \in \mathbf{R}$ and each ϕ that binary converges to x, the function ψ computed by M with oracle ϕ (i.e., $\psi(n) = M^{\phi}(n)$) binary converges to $f(x)$. We say the function f is* computable on interval $[a, b]$ *if the above condition holds for all $x \in [a, b]$.*

Similar to computable real numbers, computable real functions have several different but equivalent formulations [Grzegorczyk, 1957, 1959; Lacombe, 1955; Mostowski, 1959]. It is easy to check that our definition using oracle TMs is also equivalent to Grzegorczyk's original definition.

One of the most important properties of a computable real function is that it must be continuous on its domain and the modulus of continuity is computable. Our definition of modulus functions is a little different from that commonly used in numerical analysis.

Definition 2.4 *Let $f : [a, b] \to \mathbf{R}$ be a continuous function on $[a, b]$. Then, a function $m : \mathbf{N} \to \mathbf{N}$ is said to be a* modulus function *of f on $[a, b]$ if $|x - y| \leq 2^{-m(n)}$ implies $|f(x) - f(y)| \leq 2^{-n}$, for all $x, y \in [a, b]$ and for all $n \geq 0$.*

Proposition 2.5 *If $f : [a, b] \to \mathbf{R}$ is computable on $[a, b]$ then f is continuous on $[a, b]$; furthermore, f has a recursive modulus function m on $[a, b]$.*

The above proposition allows us to give an equivalent definition of computable real functions without using the notion of oracle Turing machines.

Proposition 2.6 *A real function $f : [a, b] \to \mathbf{R}$ is computable if and only if f has a recursive modulus function m on $[a, b]$ and there exists a recursive function $\psi : (\mathbf{Q} \cap [a, b]) \times \mathbf{N} \to \mathbf{Q}$ such that for all $r \in \mathbf{Q} \cap [a, b]$ and all $n \geq 0$, $|\psi(r, n) - f(r)| \leq 2^{-n}$.*

For simplicity, we only state the above characterization for functions defined on a closed interval. It is, though, easy to extend to functions defined on the whole real line $\mathbf{R}$. By Proposition 2.6, a real function can be represented by two discrete functions: the modulus function m and the approximation function ψ. Using this representation, we may compute a functional F that maps real functions to real numbers by an oracle TM.

Definition 2.7 *A numerical functional F on $D \subseteq C[a, b]$ is* computable *if there exists a two-oracle TM M such that for any function $f \in D$, any oracle functions m and ϕ, and any input $n \in \mathbf{N}$,*

$$|M^{m,\phi}(n) - F(f)| \leq 2^{-n}$$

provided that the functions m and ϕ represent f as in Proposition 2.6.

Finally, we note that the notion of recursive real functions can be extended in a straightforward way to functions on $\mathbf{R}^n$ for $n > 1$. We omit the formal definitions.

2.2 Roots

In this section, we consider the computability of roots of recursive real functions f defined on a closed interval $[0,1]$. If f is a one-to-one function, and $f(0) \cdot f(1) < 0$ then by the intermediate value theorem, there must exist a root $x \in [0,1]$ such that $f(x) = 0$. The strong form of the recursive version of the intermediate value theorem holds so that the functional mapping a recursive, one-to-one real function to its root is recursive.

Theorem 2.8 *Let f be a recursive, one-to-one function on $[0,1]$ with $f(0)\cdot f(1) < 0$. Then, there exists a unique recursive real number $x \in [0,1]$ such that $f(x) = 0$. Furthermore, there exists a two-oracle TM M such that for any oracles m and ψ that represent a one-to-one real function f on $[0,1]$ (in the sense of Definition 2.7), with $f(0) \cdot f(1) < 0$, and for any input $n > 0$, M outputs a rational number r such that $|r - f^{-1}(0)| \leq 2^{-n}$.*

Note that in the above, the output r is an approximation to the unique root of f, instead of an *approximate root* r' satisfying $|f(r')| \leq 2^{-n}$.

If f is not known to be one-to-one, then the roots of f may not be computable. This result was first proved by Specker [1959]. The following characterization result was from Nerode and Hwang [1985]. For simplicity, we only state the results on recursive real functions defined on the closed interval $[0,1]$.

Definition 2.9 *A set $S \subseteq \mathbf{R}$ is* recursively open *if it is empty or if there is a recursive function $\phi : \mathbf{N} \to \mathbf{Q}$ such that for each $n \in \mathbf{N}$, $\phi(2n) < \phi(2n+1)$, and $S = \bigcup_{n=0}^{\infty}(\phi(2n), \phi(2n+1))$. A set S is* recursively closed *if the set $\mathbf{R} - S$ is recursively open.*

Theorem 2.10 *A set $S \subseteq [0,1]$ is recursively closed if and only if there is a recursive function $f : [0,1] \to \mathbf{R}$ such that S contains exactly the roots of f in $[0,1]$.*

Corollary 2.11 *(a) There exists a recursive real function f on $[0,1]$ which has an uncountable number of roots but none of them is computable.*

(b) If x is an isolated root of a recursive function f in $[0,1]$, then x is recursive.

(c) If a recursive real function f has only finitely many roots in $[0,1]$, then all roots of f are recursive.

(d) If a recursive real function f has a countably infinite number of roots in $[0,1]$, then it has an infinite number of recursive roots.

2.3 Ordinary Differential Equations

In this section, we consider the first-order ordinary differential equation with an initial condition

$$y'(x) = f(x, y(x)), \qquad y(0) = 0 \tag{1}$$

that is defined by a recursive real function f on the rectangle $[0,1] \times [-1,1]$. Assume that f is recursive and that the above equation has a solution y on $[0,1]$. Is the solution y computable? The answer, similar to the roots of a recursive real function, depends on the number of solutions y of the equation.

Theorem 2.12 [Pour-El and Richards, 1979]. *There exists a recursive real function $f : [0,1] \times [-1,1] \to \mathbf{R}$ such that the equation* (1) *defined by f does not have a computable solution y on $[0,\delta]$ for any $\delta > 0$.*

The equation (1) defined by the function f of the above theorem has uncountably many solutions y on $[0,1]$, but none of them is computable. On the other hand, if the solution is unique then it must be computable.

Theorem 2.13 [Pour-El and Richards, 1979]. *Assume that f is computable on $[0,1] \times [-1,1]$ and the equation* (1) *defined by f has a unique solution on $[0,b]$, $0 < b \leq 1$. Then, the solution y of equation* (1) *is computable on $[0,b]$.*

A function f satisfies the *Lipschitz condition* on a set $E \subseteq \mathbf{R}^2$ if there exists a constant L such that for all $(x,y_1),(x,y_2) \in E$, $|f(x,y_1)-f(x,y_2)| \leq L \cdot |y_1-y_2|$. The Lipschitz condition on f on the domain $[0,1] \times [-1,1]$ is one of the sufficient condition for the existence of a unique solution y for equation (1). The recursive version of this existence theorem follows from the above theorem.

Corollary 2.14 *Assume that f is computable and satisfies the Lipschitz condition on $[0,1] \times [-1,1]$. Then, the solution y of equation* (1) *is computable on $[0,1]$.*

2.4 Computability of Linear Operators

Many computability questions in recursive analysis may be formulated as the computability question of a linear operator on the Banach space $C[0,1]$. For instance, both the integration operator that maps a real function f to $g(x) = \int_0^x f(t)dt$ and the differentiation operator that maps a real function f to $g(x) = f'(x)$ are linear operators. Pour-El and Richards [1983b] showed a very interesting characterization of the computability of a linear operator that states as follows: a linear operator is computable if and only if it is bounded. In this section, we discuss this result and some of its applications.

In their original paper, Pour-El and Richards [1983b] proved this characterization results on linear operators in a very general form; that is, the result holds on linear operators defined on any Banach space with a computability theory. Here, for the sake of simplicity, we will consider only the space $C[0,1]$. First we state the recursive version of the Weierstrass approximation theorem. In the following, a sequence $\{\varphi_n\}$ of real-valued polynomials is *computable* if there is a computable function g such that $g(n,m)$ is a finite sequence of rational numbers $(b_0,\ldots,b_k)$ such that $|a_i - b_i| \leq 2^{-m}$ for all $i \leq k$, where $\varphi_n(x) = \sum_{i=0}^k a_i x^i$.

Theorem 2.15 [Pour-El and Caldwell, 1975]. *For every computable real function f defined on $[0,1]$, there is a computable sequence $\{\varphi_n\}$ of real-valued polynomials such that $|\varphi_n(x) - f(x)| \le 2^{-n}$, for all $n \ge 0$ and $x \in [0,1]$.*

From the above recursive version of the Weierstrass approximation theorem, we can assume that each real function f is represented as the limit of a sequence of polynomial functions; or, the sequence $\{x^0, x^1, \ldots\}$ is an *effective generation sequence* for functions in $C[0,1]$. Thus, if a linear operator T on $C[0,1]$ maps this generating sequence to a computable sequence of real functions, then it is potentially computable. A linear operator T from $C[0,1]$ to $C[0,1]$ is *closed* if for any sequence $\{f_n\}$ of functions in $C[0,1]$ that converges to f, the sequence $\{T(f_n)\}$ converges to $T(f)$. A linear operator T from $C[0,1]$ to $C[0,1]$ is *bounded* if there exists a constant $c > 0$ such that for all f in the domain of T, $\|T(f)\| \le c\|f\|$, where $\|f\|$ is the maximum of f on $[0,1]$.

Theorem 2.16 *Let $T : C[0,1] \to C[0,1]$ be a closed linear operator whose domain includes all functions x^k, $k \ge 0$, such that the sequence $\{T(x^k)\}$ is computable. Then, T maps computable real functions to computable real functions if and only if T is bounded.*

This result has many interesting applications. We list some of them below.

Corollary 2.17 *(a) For any recursive real function $f : [0,1] \to \mathbf{R}$, the integration function $g(x) = \int_0^x f(t)dt$ is recursive.*

(b) [Myhill, 1971]. *There exists a recursive real function $f : [0,1] \to \mathbf{R}$ that has a continuous derivative on $[0,1]$ but its derivative $f'(x)$ is not computable.*

About the derivatives, a positive result exists that state that if f has a continuous second derivative, then the first derivative f' is computable [Pour-El and Richards, 1978]. Indeed, this follows from the following characterization of the computability of derivatives [Ko and Friedman, 1982].

Theorem 2.18 *Let $f : [0,1] \to \mathbf{R}$ be a recursive real function having a continuous derivative on $[0,1]$. Then, f' on $[0,1]$ is computable if and only if f' has a recursive modulus function on $[0,1]$.*

2.5 Computability of a Two-Dimensional Region

For any Jordan curve Γ on the plane $\mathbf{R}^2$, we say that Γ is computable if there is a recursive function $f : [0,1] \to \mathbf{R}^2$ that is one-to-one except $f(0) = f(1)$ such that $\Gamma = f([0,1])$. In this section, we consider the question of whether the interior $Int(\Gamma)$ of a computable Jordan curve Γ is computable and whether its measure is a recursive real number. The answer depends on the two-dimension Lebesgue measure of the curve Γ. Namely, if the two-dimensional measure of the curve Γ itself is zero, then the interior is computable and its measure is recursive; and if the measure of Γ is positive, then the interior is not necessarily computable and its measure could be nonrecursive.

First, we must define the notion of computability of a two-dimensional region S. There are a number of different formulations of this concept, and we will only consider a very general concept called *recursive approximability* which can be used to study the notion of computability of any subset of $\mathbf{R}^2$. Basically, a set $S \subseteq \mathbf{R}^2$ is recursively approximable if we can determine whether a given point $\langle x, y\rangle$ of $\mathbf{R}^2$ belongs to S or not with a predefined error probability. In the following, we write $(\phi, \psi) \in CF(\langle x, y\rangle)$ to denote the fact that ϕ and ψ binary converge to x and y, respectively.

Definition 2.19 *A set $S \subseteq \mathbf{R}^2$ is* recursively approximable *if there exists an oracle TM M such that, for each $n \geq 1$, the two-dimensional Lebesgue measure of the set*

$$E_M(n) = \{\langle x, y\rangle \in \mathbf{R}^2 : (\exists(\phi, \psi) \in CF(\langle x, y\rangle)) M^{\phi,\psi}(n) \neq \chi_S(\langle x, y\rangle)\}$$

is at most 2^{-n}.

In general, if a set $S \subseteq \mathbf{R}^2$ is recursively approximable, then its two-dimensional measure $\mu(S)$ is recursive.

In the following, we say a real number x is a *left r.e.* real number if the set $\{r \in \mathbf{Q} : r < x\}$ is an r.e. set. The following results are from Ko [1995a] and Ko and Weihrauch [1996].

Theorem 2.20 *Assume that Γ is a computable Jordan curve on the plane $\mathbf{R}^2$.*

(a) If the two-dimensional Lebesgue measure of the curve Γ itself is zero, then its interior $\mathrm{Int}(\Gamma)$ *must be recursively approximable.*

(b) If the two-dimensional Lebesgue measure of the curve Γ itself is recursive, then the measure of its interior $\mathrm{Int}(\Gamma)$ *must also be recursive.*

Theorem 2.21 *(a) Assume that Γ is a computable Jordan curve on the plane $\mathbf{R}^2$. Then, the measure of the interior* $\mathrm{Int}(\Gamma)$ *of the curve Γ is a left r.e. real number.*

(b) For any left r.e. real number $x > 0$, there is a computable Jordan curve Γ such that its interior Int(Γ) has the measure x.

Note that the Jordan curve Γ of Theorem 2.21(b) must have a positive, nonrecursive two-dimensional Lebesgue measure. We remark that that if a computable Jordan curve Γ has a positive two-dimensional Lebesgue measure $\mu(\Gamma)$ then its Hausdorff dimension must be equal to two and hence the curve Γ is a fractal. (For the notions of fractals and the Hausdorff dimension, see, e.g., Falconer [1991].) On the other hand, when $\mu(\Gamma)$ is equal to zero, the Hausdorff dimension of $\mu(\Gamma)$ is between one and two. In general, it is not necessarily recursive.

Theorem 2.22 [Ko, 1997] *There exists a computable Jordan curve Γ on the plane $\mathbf{R}^2$ whose Hausdorff dimension is a nonrecursive real number between* 1 *and* 2.

3 Polynomial-Time Complexity Theory of Real Functions

In recursive analysis, we consider only the notion of computability of a numerical problem. Now we add complexity measures to the computation of real functions and consider the complexity of solving a numerical problem. There are two major differences between the results of recursive analysis and polynomial-time analysis. First, the notion of computability in recursive analysis is a universal criterion. Although the degrees of noncomputability may be of interests for some specific problems, the major issue in recursive analysis is to determine whether a problem is computable or not computable. In polynomial-time analysis, however, the situation is different. First, for a given numerical problem, we are not only interested in the polynomial-time computability or noncomputability of the solution, but also interested in identifying precisely its inherent complexity. Furthermore, for many problems, its polynomial-time computability is hard to prove or disprove absolutely, and the best we can do is to classify the problem into some well-known complexity classes in discrete complexity theory. Different numerical problems may require different proof techniques to classify their inherent complexity, and yet these different results allow us to compare their complexity in a single, uniform model. Indeed, as we will see, the complexity of different numerical operations forms a hierarchy that is parallel to the hierarchy of the discrete complexity classes.

Second, it is important to point out that certain critical analytic properties seem to affect greatly the inherent complexity of a numerical problem. In the last section, we have seen some results indicating that the computability of a numerical solution often depends not only on the computability of the input function defining the problem but also on some analytic properties of the function. For instance, in Section 2.3, it was shown that the Lipschitz condition on a function f is sufficient to guarantee the computability of the solution of the ordinary differential equation defined by f. In the complexity analysis of these problems, the analytic properties appear to be even more important. Many elementary numerical operations that are easily seen to be computable from the point of view of recursive analysis turn out to have very high complexity unless certain analytic properties are assumed on the input functions. Furthermore, we cannot expect in polynomial-time analysis a simple, clear-cut result like Theorem 2.16 that uses a single analytic property to characterize the computability of many different operations. Instead, it is one of the main issues in polynomial-time analysis to identify the critical analytic properties of the input functions that determine the complexity of the solutions.

One of the similarities between our approach to polynomial-time analysis and recursive analysis is that we do not study the polynomial-time computability of a numerical functional. Instead, we ask a weaker type of questions of whether a numerical functional maps a polynomial-time computable real function to a polynomial-time computable real number (or real function). This weaker approach is necessary since under the worst-case complexity measure, almost all nontrivial numerical functionals are easily provable to be not polynomial-time solvable, based on the simple adversary argument. Such general lower bound results fail to provide insight into the inherent complexity of the underlying numerical functionals. Our weaker approach restricts the domain of the numerical functionals to the set of polynomial-time computable functions. Since the class of polynomial-time computable functions is a rich, nontrivial class, this approach is not too restrictive and yet allows us to classify the complexity of the numerical functionals more accurately.

3.1 Complexity of Real Functions

In Section 2.1, we have defined computable real numbers and real functions. In order to define the complexity measures of real numbers and real functions, we need to further fix the representation system of real numbers. Instead of using rational numbers $\mathbf{Q}$ as the base set for approximating real numbers, we choose the set $\mathbf{D}$ of dyadic rational numbers, or, rational numbers with finite binary expansions, as the base set. Set $\mathbf{D}$ has a simple representation system: each dyadic rational number d is naturally represented by a binary string $s = \pm s_n s_{n-1} \cdots s_0 . t_1 t_2 \cdots t_m$ satisfying

$$d = \pm \sum_{i=0}^{n} s_i \cdot 2^i \pm \sum_{j=1}^{m} t_j \cdot 2^{-j}.$$

We say that a representation s for a dyadic rational d has the precision m, and write $prec(s) = m$, if it has m bits to the right of the binary point. We let $\mathbf{D}_n$ denote the class of dyadic rationals d which have a representation of precision $\leq n$; i.e., $\mathbf{D}_n = \{m \cdot 2^{-n} : m \in \mathbf{Z}\}$. The set $\mathbf{D}$ is preferred to the set $\mathbf{Q}$ because its members have simple binary representations and, more importantly, because it is *uniformly dense* in $\mathbf{R}$: the numbers in $\mathbf{D}_n$ are uniformly distributed over the real line.

Based on this representation system, we define the complexity of a real number x to be the complexity of computing a dyadic rational d of precision n that approximates x with an error $\leq 2^{-n}$; i.e., the first n bits of d, to the right of the binary point, are the effective bits of x.

Definition 3.1 *(a) Let t be an integer function. We say that the* time *(or,* space*)* complexity *of a computable real number x is bounded by t if there exists a TM which computes, on each input $n \in \mathbf{N}$, a dyadic rational number d in $t(n)$ moves (or, respectively, using $t(n)$ cells) such that $|d - x| \leq 2^{-n}$.*

(b) A real number x is polynomial-time computable *if its time complexity is bounded by a polynomial function p.*

It is known that all algebraic numbers and many well-known transcendental numbers, including e and π, are polynomial-time computable. Furthermore, the class of polynomial-time computable real numbers forms a real closed field.

We have defined computable real functions using the function-oracle TMs in which input and output real numbers are represented by rational-valued functions that binary converge to them. This definition allows us to define the complexity of real functions naturally.[3] For simplicity, we only consider real functions defined on a closed interval $[0, 1]$.

[3] As we pointed out in Section 2.1, the time complexity of an oracle TM is defined in such a way that the cost of making a query is only one unit of time.

Definition 3.2 *(a) Let $f : [0,1] \to \mathbf{R}$ be a computable function. We say that the* time *(or,* space*)* complexity *of f on $[0,1]$ is bounded by a function $t : \mathbf{N} \to \mathbf{N}$ if there exists an oracle TM M which computes f (as defined in Definition 2.3 except that the output of $M^\phi(n)$ is a dyadic rational) such that for all ϕ that binary converge to a real number $x \in [0,1]$ and for all $n > 0$, $M^\phi(n)$ halts in time $t(n)$ (or, respectively, $M^\phi(n)$ uses at most $t(n)$ cells of workspace[4]).*

(b) A real function $f : [0,1] \to \mathbf{R}$ is polynomial-time computable *if its time complexity is bounded by a polynomial function p.*

We have seen in Proposition 2.5 that a computable real function must have a computable modulus function. The following proposition gives a more precise relation between the time complexity and the modulus of continuity of a real function.

Proposition 3.3 *(a) Assume that the time complexity of $f : [0,1] \to \mathbf{R}$ is bounded by $t : \mathbf{N} \to \mathbf{N}$. Then, the function $t(n+2)$ is a modulus function for f on $[0,1]$.*

(b) Each polynomial-time computable real function $f : [0,1] \to \mathbf{R}$ has a polynomial modulus of continuity.

This observation gives us a characterization of polynomial-time computable real functions like that of Proposition 2.6.

Corollary 3.4 *A function $f : [0,1] \to \mathbf{R}$ is polynomial-time computable if and only if there exist polynomial functions m and q and a function $\psi : (\mathbf{D} \cap [0,1]) \times \mathbf{N} \to \mathbf{D}$ such that*

(i) m is a modulus function for f on $[0,1]$,

(ii) for any $d \in \mathbf{D} \cap [0,1]$ and all $n \in \mathbf{N}$, $|\psi(d,n) - f(d)| \le 2^{-n}$, and

(iii) $\psi(d,n)$ is computable in time $q(prec(d) + n)$.

3.2 Discrete Complexity Theory

To prepare for classifying the complexity of numerical problems in terms of discrete complexity classes, we give a short summary of some important notions in discrete complexity theory.

3.2.1 Complexity Classes

We first review some important complexity classes. We will only give informal definitions to these complexity classes. More careful discussions of the properties and relations between these complexity classes can be found in, for instance, Garey and Johnson [1979], Balcazar et al. [1988] and Ko [1991].

[4]When we consider the space complexity of computing a real function, we do not include the space of query tape in the space measure. This follows from the convention in discrete complexity theory that the input/output space is not included in the space measure.

First, we consider *language recognition problems* (or, *decision problems*), that is, the problems asking a yes/no answer. Such a problem is represented as a set of strings. The most important complexity classes of language recognition problems include *LOGSPACE*, *P*, *NP*, *PSPACE* and *EXPTIME*.

LOGSPACE: the class of sets accepted by deterministic TMs in logarithmic space (i.e., in space $c \log n$ for some constant $c > 0$).

P: the class of sets accepted by deterministic TMs in polynomial time (i.e., in time n^k for some constant $k > 0$).

NP: the class of sets accepted by nondeterministic TMs in polynomial time.

PSPACE: the class of sets accepted by deterministic TMs in polynomial space.

EXPTIME: the class of sets accepted by deterministic TMs in exponential time (i.e., in time 2^{n^k} for some constant $k > 0$).

It is known that $LOGSPACE \subseteq P \subseteq NP \subseteq PSPACE \subseteq EXPTIME$. It is easy to prove by diagonalization that $LOGSPACE \neq PSPACE$ and $P \neq EXPTIME$. Other than these easy separation results, none of the above inclusions is known to be proper. Indeed, these are the major open questions in discrete complexity theory. (It is known that nondeterministic polynomial space defines the same class *PSPACE* as deterministic polynomial space [Savitch, 1970].)

There are many other important complexity classes defined in discrete complexity theory. We will particularly use the following complexity classes:

UP: the class of sets accepted in polynomial time by nondeterministic TMs that have, for each input, at most one accepting computation path.

BPP: the class of sets accepted in polynomial time by probabilistic TMs with the error probability bounded by 1/4.

The class *UP* is closely related to the notion of one-way functions; namely, weak one-way functions exist if and only if $P \neq UP$. The class *BPP* denotes the class of problems that are feasibly solvable by randomized algorithms. It is known that $P \subseteq UP \subseteq NP$ and $P \subseteq BPP \subseteq PSPACE$. None of these inclusions is known to be proper.

In addition to the language recognition problems, we will also deal with *function evaluation problems* that are represented as functions mapping strings to strings. The main complexity class we are concerned with are the counting class $\#P$.

FP: the class of functions computed by deterministic TMs in polynomial time.

$\#P$: the class of functions that enumerate the number of accepting computations of polynomial-time nondeterministic TMs.

FPSPACE: the class of functions computed by deterministic TMs in polynomial space.

It is known that $FP \subseteq \#P \subseteq FPSPACE$, but the inclusions are not known to be proper.

3.2.2 Reducibility and Completeness

The class P is commonly identified with feasibly computable problems. From a complexity point of view, we would like to prove a problem not in P to demonstrate that it is not feasibly computable. However, a great number of problems arising from different areas of discrete computation turn out to belong to *NP* or *PSPACE* and are not known to be in P. In order to give a convincing evidence that these problems are not feasible, the notions of reducibility and completeness are applied to these problems.

First, we restrict ourselves to only language recognition problems. A set A is *polynomial-time reducible* to a set B if there exists a polynomial-time computable function ϕ such that for any input x, $x \in A$ if and only if $\phi(x) \in B$. A set B is *complete* for a complexity class $\mathcal{C}$ (with respect to the polynomial-time reducibility) if (a) $B \in \mathcal{C}$ and (b) for any $A \in \mathcal{C}$, A is polynomial-time reducible to B. It is easy to see that if A is polynomial-time reducible to B and if B is in P, then A is also in P; it follows that if A is complete for *NP* (or *PSPACE*), then $A \in P$ if and only if $P = NP$ (or, respectively, $P = PSPACE$). Garey and Johnson [1979] have collected hundreds of problems arising from different areas of research in discrete computation that are proved to be *NP*- or *PSPACE*-complete. Proving a new problem A to be *NP*-complete means that problem A is *equivalent* to *all* these difficult problems as far as polynomial-time computability is concerned. It also means practically getting a superpolynomial-time *lower bound* for the complexity of problem A in the sense that it is not in P unless $P = NP$.

The notion of completeness can also be extended to problems in P and *EXPTIME*. Since it is known that $P \neq EXPTIME$, an *EXPTIME*-complete problem A is *provably* not feasibly computable. In addition, an *EXPTIME*-complete problem A is not computable in time *NP* or space *PSPACE*, unless $EXPTIME = NP$ or $EXPTIME = PSPACE$, respectively, which relations are commonly conjectured to be false.

To define completeness for P, we require that the reducibility is computable in log space. More precisely, a set A is *log-space reducible* to a set B if there exists a function ϕ that is computable in log space[5] such that for any input x, $x \in A$ if and only if $\phi(x) \in B$. A set $B \in P$ is *P-complete* if for every set $A \in P$, A is log-space reducible to B.

The complexity classes below P, including *LOGSPACE*, *NLOGSPACE* and *NC*, are usually identified as the classes of problems that are feasibly solvable by parallel machines.[6] Therefore, a problem A being P-complete implies that A requires polynomially long sequential computation that is not parallelizable unless the class P collapses to one of the feasible parallel complexity classes.

[5] Following the convention in discrete complexity theory, the space complexity of a function only measures the workspace used by the machine computing the function, not including the input/output space.

[6] We omit the definitions of the classes *NLOGSPACE* and *NC* and their relation to parallel computation. The interested reader is referred to, for instance, Cook [1985].

3.2.3 Complete Real Functions

In our results on the complexity of numerical problems, we often reduce a discrete complete problem to a numerical problem, using a little weaker type of reducibility. The weaker type of reducibility is necessary as we are dealing with continuous problems and the reducibility is not as straightforward as that for discrete problems; yet the effect is essentially the same.

Definition 3.5 *A set $A \subseteq \{0,1\}^*$ is* polynomial-time reducible *to a real function $f : [0,1] \to \mathbf{R}$ if there exist two polynomial-time computable functions $g : \{0,1\}^* \to \mathbf{D}$ and $h : \{0,1\}^* \times \mathbf{D} \to \{0,1\}$ and a polynomial function p such that for every $w \in \{0,1\}^*$, $w \in A \iff h(w,e) = 1$, where e is any dyadic rational with the property $|e - f(g(w))| \leq 2^{-p(n)}$.*

In other words, to determine whether $w \in A$, we can first use g to transfer the instance w for problem A to an instance $d = g(w)$ for problem f, then from an approximate value $e \approx f(d)$, the answer to the question "$w \in ?A$" can be found by $h(w,e)$.

Definition 3.6 *Let $\mathcal{C}$ be a discrete complexity class. A real function $f : [0,1] \to \mathbf{R}$ is* complete *for the class $\mathcal{C}$ if f has a representation (m, ψ), as defined in Proposition 2.6, such that m is a polynomial function and ψ is computable in $\mathcal{C}$, and if every set $A \in \mathcal{C}$ is polynomial-time reducible to f.*

Note that if f is complete for the class $\mathcal{C}$, where $P \subseteq \mathcal{C}$, then f is polynomial-time computable if and only if $P = \mathcal{C}$.

3.3 Maximization

In the next five subsections, we classify the computational complexity of some basic numerical problems in terms of relations among discrete complexity classes. These problems include the maximization problem, the root-finding problem, the integration problem, ordinary differential equations, and integral equations. Additional results can be found in Ko [1991], Hoover [1987] and Papadimitriou and Tsitsiklis [1986]. In the following, we will write $\ell(s)$ to denote the length of a finite string $s \in \{0,1\}^*$.

3.3.1 Maximization and Nondeterminism

The maximization problem in recursive analysis is quite simply solved. Let $f : [0,1] \to \mathbf{R}$ be a recursive function. Then the maximum value $m_f = \max\{f(y) : 0 \leq y \leq x\}$ is also recursive; however, the maximum point x_0 such that $f(x_0) = m_f$ is not necessarily recursive (follows from Corollary 2.11).

The maximization problem in polynomial-time analysis is more interesting. First let us review the maximization problem in discrete complexity theory, where many optimization problems were proved to be NP-complete. Consider the following abstract maximization problem. For any function $\phi : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$, define $\max_\phi(x) = \max\{\phi(x,y) : \ell(y) = \ell(x)\}$. If we know that ϕ is polynomial-time computable, what is the complexity of computing $\max_\phi$? It is easy to show that $\max_\phi$ is computable in polynomial time if $P = NP$ and, furthermore, there exists a polynomial-time computable ϕ such that $\max_\phi$ is NP-complete.

The above simple example suggests naturally that we should try to use the notion of nondeterministic computation to characterize the complexity of the maximization problems—even for continuous problems. Indeed, the notion of NP can be used to give a precise characterization of the complexity of the maximization of one- or two-dimensional polynomial-time computable functions. In addition, differentiability of the function does not help reducing the complexity of maximization.

Theorem 3.7 [Friedman, 1984]. *There exists a polynomial-time computable real function $f : [0,1] \to \mathbf{R}$ that is infinitely differentiable (i.e., $f \in C^\infty[0,1]$) such that the function $g(x) = \max\{f(y) : 0 \le y \le x\}$ is complete for the class NP.*

Corollary 3.8 *The following are equivalent:*

(a) $P = NP$.

(b) For each polynomial-time computable $f : [0,1]^2 \to \mathbf{R}$, the function $g(x) = \max\{f(x,y) : 0 \le y \le 1\}$ is polynomial-time computable.

(c) For each polynomial-time computable $f : [0,1] \to \mathbf{R}$, the function $h(x) = \max\{f(y) : 0 \le y \le x\}$ is polynomial-time computable.

(d) For each polynomial-time computable $f : [0,1] \to \mathbf{R}$ that is infinitely differentiable (i.e., $f \in C^\infty[0,1]$), the function $k(x) = \max\{f(y) : 0 \le y \le x\}$ is polynomial-time computable.

3.3.2 Maximum Values and Left NP Real Numbers

Corollary 3.8 shows that computing the maximization function of a polynomial-time computable function is, in the worst case, as difficult as an NP-complete problem. Is the problem of computing the single maximum value $m_f = \max\{f(x) : 0 \le x \le 1\}$ as difficult as the problem of computing the maximization function? It turns out to be an interesting question depending on some subtle relations between deterministic and nondeterministic computation.

First we give a characterization of the complexity of the maximum values of polynomial-time computable real functions. Recall that a real number x is represented by a function ϕ that binary converges to x. For $x \in [0,1)$, such a function ϕ could also be represented by a set of strings $L_\phi = \{s \in \{0,1\}^* : 0.s \le \phi(\ell(s))\}$, where $0.s$ denotes the dyadic rational d whose binary expansion is equal to $0.s$. Function ϕ and set L_ϕ are polynomially equivalent in the sense that one is computable from the other in polynomial time. Note that L_ϕ is not equal to the left cut $L = \{d \in \mathbf{D} : d \le x\}$ unless $\phi(n) \le x$ for all $n \ge 1$, but L_ϕ is very close to the left cut. Such a set L_ϕ is called a *general left cut* of x.

Definition 3.9 *A real number $x \in [0,1)$ is a left NP real number if it has a general left cut L_ϕ that is in NP. A real number x is a left NP real number if $x' = x - \lfloor x \rfloor$ is a left NP real number.*

Theorem 3.10 [Ko, 1982]. *A real number x is a left NP real number if and only if there exists a polynomial-time computable real function $f : [0,1] \to \mathbf{R}$ such that $x = \max\{f(y) : 0 \le y \le 1\}$.*

From the above characterization result, the complexity of maximum values of polynomial-time computable real functions is the same as the complexity of left *NP* real numbers. To analyze the precise complexity of a left *NP* real number x, we need to define some new discrete complexity classes. Recall that *EXPTIME* denotes the class of sets computable in deterministic time 2^{n^k} for some constant k. We define another exponential time complexity classes *EXP* that contains all sets computable in deterministic time $2^{O(n)}$. We let *NEXP* denote the class of sets computable in nondeterministic time $2^{O(n)}$. For any complexity class $\mathcal{C}$, we let $\mathcal{C}_1$ denote the class of *tally* sets in $\mathcal{C}$, where a set T is called a tally set if it is a set over a single alphabet $\{0\}$, i.e., if $T \subseteq \{0\}^*$. It is easy to see that $EXP = NEXP$ if and only if $P_1 = NP_1$ [Book, 1974].

We note that the structure of a general left cut L_ϕ of a real number is closely related to the structure of a tally set. The following relations between these classes thus give the best classification of the complexity of left *NP* real numbers. It implies that if deterministic exponential time is different from nondeterministic exponential time then left *NP* real numbers are not always polynomial-time computable. Together with Corollary 3.8, it shows that computing a single maximum value has a *lower* complexity than computing the maximum function (of infinitely many maximum values).

Theorem 3.11 [Ko, 1982]. *In the following, (a) $\Rightarrow$ (b) $\Rightarrow$ (c) $\iff$ (d).*

(a) $P = NP$.
(b) Every left NP real number is polynomial-time computable.
(c) $P_1 = NP_1$.
(d) $EXP = NEXP$.

3.4 Roots and Inverse Functions

In this section, we consider the complexity of computing the root and the inverse function of a one-dimensional, one-to-one real function. We will consider only one-to-one real functions. The reason is simple: for functions that are not one-to-one, the roots could be nonrecursive even if the underlying function is polynomial-time computable. The following theorem can be proved from a construction similar to Corollary 2.11.

Theorem 3.12 *There exists a polynomial-time computable real function $f : [0,1] \to \mathbf{R}$ such that it has an uncountable number of roots but none of them is computable.*

If we add extra restrictions of analytic properties to the function f, the complexity of its roots becomes reasonably low. However, the conditions are too complicated to be studied here. The interested reader is referred to Ko [1991] for the study of the complexity of roots of functions that are not one-to-one.

3.4.1 Inverse Modulus of Continuity

In Theorem 2.8, we have seen that if a one-to-one function $f : [0,1] \to \mathbf{R}$ is recursive, then its inverse function is also recursive. However, the polynomial-time analog of this result does not hold, even if f is infinitely differentiable.

Theorem 3.13 *For any recursive real number $x \in [0,1]$, there exists a strictly increasing, polynomial-time computable function $f : [0,1] \to \mathbf{R}$ that is in $C^\infty[0,1]$ such that x is the unique root of f in $[0,1]$.*

Theorem 2.8 was proved by a simple binary search algorithm to find the root. Why does this algorithm fail to find the root in polynomial time? We note that this binary search algorithm can only find an *approximate root* y (such that $|f(y)| \approx 0$) in polynomial time which does not have to be close to the real, unique root x. A sufficient condition for the approximate root to be a good approximation to the real root is that the inverse function f^{-1} has a polynomial modulus function around the root. We say that a real function $g : [0,1] \to \mathbf{R}$ has a *polynomially-bounded local modulus* at $z \in [0,1]$ if g has a polynomial modulus on $[z-\epsilon, z+\epsilon]$ for some $\epsilon > 0$.

Theorem 3.14 [Ko and Friedman, 1982]. *Assume that f is one-to-one, polynomial-time computable on $[0,1]$ with the range $[a,b]$, $a < 0 < b$.*

(a) If f^{-1} has a polynomially-bounded local modulus at 0 then the root of f in $[0,1]$ is polynomial-time computable.

(b) If f^{-1} has a polynomial modulus function on $[a,b]$ then f^{-1} is polynomial-time computable on $[a,b]$.

3.4.2 Fundamental Theorem of Algebra

It is not hard to verify that the inverse function f^{-1} of an analytic, one-to-one, polynomial-time computable function $f : [0,1] \to \mathbf{R}$ must have a polynomial modulus function on its domain. Thus the roots of f on $[0,1]$ are polynomial-time computable. This result can be extended to functions that are not necessarily one-to-one.

Theorem 3.15 [Ko and Friedman, 1982]. *All roots of an analytic, polynomial-time computable function f on $[0,1]$ are polynomial-time computable.*

Extending the notion of polynomial-time computable real functions to polynomial-time computable complex-valued functions, we get the following polynomial-time version of the fundamental theorem of algebra. A complex-valued polynomial function is polynomial-time computable if and only if its coefficients are polynomial-time computable complex numbers.

Corollary 3.16 [Polynomial-Time Version of the Fundamental Theorem of Algebra, the weak form]. *All roots of a polynomial-time computable complex-valued polynomial function are polynomial-time computable.*

The strong form of this theorem that requires that the mapping from coefficients of a complex-valued polynomial to its roots be polynomial-time computable, though claimed to be true by many numerical analysts, was first *formally* proved true by Schönhage [1982] and then improved by Neff [1990] showing that this mapping actually is in NC.

Theorem 3.17 [Polynomial-Time Version of the Fundamental Theorem of Algebra, the strong form]. *There exists an algorithm in NC (hence in polynomial time) that for any given complex numbers $a_0, a_1, \ldots, a_n$, with $a_n = 1$, computes complex numbers $z_1, \ldots, z_n$ such that $\sum_{i=0}^{n} a_i z_j^i = 0$, for $j = 1, \ldots, n$.*

3.4.3 Log-Space Computability of Roots

In discrete complexity theory, we often identify polynomial-time computability with sequentially feasible computability, and identify *log-space computability*, or *NC computability* with *parallelly feasible computability* [Cook, 1985]. In Theorem 3.14, we have established that if the inverse f^{-1} of a one-to-one, polynomial-time computable function f has a polynomial modulus, then f^{-1} is sequentially feasibly computable. In this section, we investigate the question of whether the inverse functions f^{-1} having such nice properties are parallelly feasibly computable, or, log-space computable.

A real function $f : [0,1] \to \mathbf{R}$ is *log-space computable* if it is computable by an oracle TM that uses $O(\log n)$ workspace in its computation on input n, relative to all oracles ϕ, equivalently, if the functions m and ψ in Proposition 2.6 are log-space computable. A real function $g : [0,1] \to \mathbf{R}$ is *complete* for P if every set $A \in P$ is *log-space* reducible to g in the sense of Definition 3.5 with the reduction functions g and h computable in log space. The following result shows that even if a function f is log-space computable and satisfies the conditions of Theorem 3.14, the inverse function f^{-1} is not necessarily log-space computable.

Theorem 3.18 [Ko, 1990]. *There is a log-space computable, one-to-one function f : $[0,1] \to \mathbf{R}$ such that f^{-1} has a polynomial modulus but f^{-1} is complete for P.*

Corollary 3.19 *The following are equivalent:*

(a) $P = LOGSPACE$.

(b) For each log-space computable, one-to-one function $f : [0,1] \to \mathbf{R}$ whose inverse f^{-1} has a polynomial modulus, the inverse f^{-1} is log-space computable.

For the complexity of a single root, we identify its complexity in terms of P_1 versus $LOGSPACE_1$. Recall that $\mathcal{C}_1$ is the class of tally sets in $\mathcal{C}$. The relation between P_1 and $LOGSPACE_1$ is similar to the relation between exponential time EXP and linear space (cf. discussion before Theorem 3.11).

Theorem 3.20 [Ko, 1990]. *The following are equivalent:*

(a) $P_1 = LOGSPACE_1$.

(b) For each log-space computable, one-to-one function $f : [0,1] \to \mathbf{R}$ such that $f(0) < 0 < f(1)$ and that f^{-1} has a polynomial modulus, $x = f^{-1}(0)$ is log-space computable.

3.5 Integration

We have seen in Corollary 2.17 that integration is a recursive operation. In this section, we show however that the integration of a polynomial-time computable real function f is not necessarily polynomial-time computable, unless the discrete complexity class $\#P$ collapses to FP.

The class $\#P$ is a counting class containing functions that count the number of accepting computations of polynomial-time nondeterministic TMs. To see how this class characterizes the inherent complexity of integration, consider the following discrete counting problem: Assume that a set $A \subseteq \{0,1\}^* \times \{0,1\}^*$ is polynomial-time computable. What is the complexity of computing the function $c_A : \{0,1\}^* \to \mathbf{N}$ defined by $c_A(x)$ = the number of y such that $\ell(y) = \ell(x)$ and $(x,y) \in A$? It is not hard to see that $FP = \#P$ if and only if for all polynomial-time computable A, the function c_A is polynomial-time computable. We can then convert the function c_A into a continuous integration problem to obtain the following result.

Theorem 3.21 [Friedman, 1984; Ko, 1986]. *There exists a polynomial-time computable function $f : [0,1] \to \mathbf{R}$ such that the function $g(x) = \int_0^x f(t)\ dt$ is complete for $\#P$.*

Corollary 3.22 *The following are equivalent:*

(a) $FP = \#P$.

(b) For all polynomial-time computable functions $f : [0,1] \to \mathbf{R}$, the function $g(x) = \int_0^x f(t)\ dt$ is polynomial-time computable.

Assume that $\mathcal{F}$ is a complexity class of functions. Let $\mathcal{F}_1$ denote the class of functions f in $\mathcal{F}$ whose domain is exactly $\{0\}^*$. The complexity of computing the integral value over a fixed interval of a polynomial-time computable real function is, similar to that for the maximum values, dependent on the complexity of the class $\#P_1$.

Theorem 3.23 *The following are equivalent:*

(a) $FP_1 = \#P_1$

(b) For all polynomial-time computable functions $f : [0,1] \to \mathbf{R}$, the value $\int_0^1 f(t)\ dt$ is a polynomial-time computable real number.

3.6 Ordinary Differential Equations

We discussed the computability of the solution of an ordinary differential equation (1)

$$y'(x) = f(x, y(x)), \qquad y(0) = 0$$

in Section 2.3. It was proved that if function f is computable on $[0,1] \times [-1,1]$ and if the equation has a unique solution y, then the solution y is computable. The following result, due to Miller [1970], however shows that even if the solution y is unique and even if f is polynomial-time computable, the solution y could have arbitrarily high complexity.

Theorem 3.24 *Let a be an arbitrary recursive real number between 0 and 1. Then, there is a polynomial-time computable function f defined on $[0,1]\times[-1,1]$ such that $y(x)=ax^2$ is the unique solution of equation (1) defined by f.*

The above result suggests that in order to have a feasible solution, some extra analytic condition on the equation (1) must be assumed. One of the most commonly used condition is the *Lipschitz condition* on f: there exists a constant L such that for all $x\in[0,1]$ and all $z_1,z_2\in[-1,1]$,

$$|f(x,z_1)-f(x,z_2)|\leq L\cdot|z_1-z_2|.$$

It is known that if f satisfies the Lipschitz condition on $[0,1]\times[-1,1]$ then the solution y to equation (1) is unique. Furthermore, for such an equation, a standard successive approximation algorithm such as Euler's method works in polynomial space. The following result follows immediately from this observation and from Corollary 3.22.

Theorem 3.25 *In the following, (a) $\iff$ (b) $\Rightarrow$ (c) $\Rightarrow$ (d).*

(a) $P = PSPACE$.

(b) $FP = FPSPACE$.

(c) For all polynomial-time computable functions $f:[0,1]\times[-1,1]\to\mathbf{R}$ that satisfy the Lipschitz condition, the solution y of equation (1) is polynomial-time computable.

(d) $FP = \#P$.

The above result does not precisely characterize the complexity of the ordinary differential equation (1). In the following, we relax the Lipschitz condition a little to give a more precise characterization of the inherent complexity of equation (1). We say that a function f on $[0,1]\times[-1,1]$ satisfies the *right Lipschitz condition* on a region $E\subseteq[0,1]\times[-1,1]$ if there exists a constant $L>0$ such that for all $x\in[0,1]$ and all $z_1,z_2\in[-1,1]$, $z_1<z_2$,

$$f(x,z_2)-f(x,z_1)\leq L\cdot(z_2-z_1),$$

whenever the line segment connecting (x,z_1) and (x,z_2) lies entirely in region E. We also say that a region $E\subseteq[0,1]\times[-1,1]$ *polynomially covers (the graph of) a function* $y:[0,1]\to[-1,1]$ if there is a polynomial q such that for every $k\geq 1$, the set $\{(x,z):\ 0\leq x\leq 1-2^{-k}, |z-y(x)|\leq 2^{-q(k)}\}$ is contained in E.

Theorem 3.26 [Ko, 1983b]. *There exists a polynomial-time computable function f : $[0,1]\times[-1,1]\to\mathbf{R}$ such that*

(i) equation (1) defined by f has a unique solution y on $[0,1]$,

(ii) f satisfies the right Lipschitz condition on a region $E\subseteq[0,1]\times[-1,1]$ that polynomially covers the solution y, and

(iii) the solution y is complete for PSPACE.

To prove the above theorem, we first define a discrete initial value problem of the following form. Let $M_k=\{m\in\mathbf{N}:|m|\leq k\}$, and p a polynomial function. We say a function $\phi : \{0,1\}^n \times M_{p(n)} \to M_{p(n)}$ is a *derivative* function if for any string s of length n and integers $m_1, m_2 \in M_{p(n)}$, $\phi(s, m_1) \le \phi(s, m_2)$ whenever $m_1 \le m_2$. A function $v : \{0,1\}^n \to \mathbf{N}$ is a *solution* for the derivative function ϕ if for all $s \in \{0,1\}^n - \{1\}^n$, $\phi(s, v(s)) = v(succ(s))$, where $succ(s)$ is the successor of the string s in the lexicographic order on $\{0,1\}^*$. The discrete initial value problem is, for any given derivative function ϕ and a given initial value $\phi(0^n, 0) = m_0$, to compute the *solution* function $v : \{0,1\}^* \to \mathbf{N}$ such that $v(0^n) = 0$ and v is a solution of ϕ. Once such a discrete initial value problem is proved to be complete for *PSPACE*, we can encode it into a continuous function f to obtain the polynomial space lower bound for equation (1).

Corollary 3.27 *The following are equivalent:*

(a) Assume that equation (1) *defined by a polynomial-time computable function* $f : [0,1] \times [-1,1] \to \mathbf{R}$ *has a unique solution* y *on* $[0,1]$*. If* f *satisfies the right Lipschitz condition on a region* $E \subseteq [0,1] \times [-1,1]$ *that polynomially covers the solution* y*, then the solution* y *is polynomial-time computable.*

(b) $P = PSPACE$*.*

Similar to the problems about maximization and integration, a single value of the solution y of equation (1) can be characterized by the complexity class $PSPACE_1$.

Theorem 3.28 [Ko, 1983b]. *The following are equivalent:*

(a) Assume that equation (1) *defined by a polynomial-time computable function* $f : [0,1] \times [-1,1] \to \mathbf{R}$ *has a unique solution* y *on* $[0,1]$*. If* f *satisfies the right Lipschitz condition on a region* $E \subseteq [0,1] \times [-1,1]$ *that polynomially covers the solution* y*, then the value* $y(1)$ *is a polynomial-time computable real number.*

(b) $P_1 = PSPACE_1$*.*

3.7 Integral Equations

In this section, we consider the complexity of the solutions of Volterra integral equations of the first kind

$$\int_0^y K(y, s, \phi(s))\, ds = f(y), \qquad 0 \le y \le 1, \tag{2}$$

and of the second kind

$$\phi(y) = f(y) + \int_0^y K(y, s, \phi(s))\, ds, \qquad 0 \le y \le 1, \tag{3}$$

where function $\phi : [0,1] \to \mathbf{R}$ is to be solved from given functions $K : [0,1]^2 \times [-1,1] \to \mathbf{R}$ and $f : [0,1] \to \mathbf{R}$. While many numerical algorithms for Volterra equations of the second kind have been proposed and studied (see, e.g., Tricomi [1957], Miller [1971] and Baker [1977]), very few general algorithms for Volterra equations of the first kind are known. It is generally recognized that Volterra equations of the first kind are much more difficult to solve than Volterra equations of the second kind. Our results confirm this intuition.

We first consider Volterra equations (3) of the second kind. Since an ordinary differential equation is easily convertible to a Volterra integral equation of the second kind, it is only interesting if we require that the function K satisfy the Lipschitz condition. We say K satisfies the *Lipschitz condition* if there exists a constant $L > 0$ such that for all $y, s \in [0,1]$ and all $t_1, t_2 \in [-1,1]$

$$|K(y,s,t_1) - K(y,s,t_2)| \leq L \cdot |t_1 - t_2|.$$

If K satisfies the Lipschitz condition, then it is known, through Picard's successive approximation algorithm, that equation (3) has a unique solution ϕ that is polynomial-space computable. Thus, the complexity of solving equation (3) is precisely the same as the complexity of solving differential equation (1).

Theorem 3.29 *In the following, (a) $\Rightarrow$ (b) $\Rightarrow$ (c).*

(a) $P = PSPACE$.

(b) If functions K and f of equation (3) *are polynomial-time computable and if K satisfies the Lipschitz condition, then the solution ϕ of equation* (3) *is polynomial-time computable.*

(c) $FP = \#P$.

In addition, following from Theorem 3.26, if K only satisfies the Lipschitz condition on a region that polynomially covers the unique solution ϕ, then we know that *PSPACE* is also a lower bound for the solution ϕ.

Next, we consider Volterra equations (3) of the second kind for which the kernel function K only satisfies a *local Lipschitz condition*: for any $y \in [0,1]$, there exists a constant L_y such that

(i) $|K(y,s,t_1) - K(y,s,t_2)| \leq L_y \cdot |t_1 - t_2|$ for all $s \in [0,1]$ and all $t_1, t_2 \in [-1,1]$, and

(ii) there exists a polynomial function q such that for all $n \geq 0$, $0 \leq y \leq 1 - 2^{-n} \Rightarrow L_y \leq 2^{q(n)}$.

There are two reasons to consider this type of Volterra equations: first, it demonstrates quantitatively the importance of the Lipschitz condition of the kernel K on the complexity of the solution ϕ; and second, this study could be later applied to give a lower bound for Volterra equations of the first kind.

A careful analysis of Picard's successive approximation algorithm for Volterra equations of the second kind (see, for instance, Miller [1971]) shows that the equation (3) with K satisfying the above local Lipschitz condition is solvable in exponential space. In the other direction, an exponential time lower bound has been obtained for such equations through a discrete problem of systems of quadratic equations.

Theorem 3.30 [Ko, 1992]. *(a) If functions $K : [0,1]^2 \times [-1,1] \to \mathbf{R}$ and $f : [0,1] \to \mathbf{R}$ are polynomial-time computable and if K satisfies the local Lipschitz condition, then equation* (3) *has a unique solution ϕ that is computable in exponential space (i.e., in space 2^{n^k} for some constant $k > 0$).*

(b) There exist polynomial-time computable functions $K : [0,1]^2 \times [-1,1] \to \mathbf{R}$ and $f : [0,1] \to \mathbf{R}$ such that K satisfies the local Lipschitz condition and the solution ϕ of equation (3) defined by K and f is unique and has a polynomial modulus, but is complete for EXPTIME.

It is not known whether the gap between the exponential space upper bound and the exponential time lower bound can be narrowed.

Now we consider Volterra equations (2) of the first kind. As we pointed out above, no numerical algorithm is known to work for the general case, even if the function K satisfies the Lipschitz condition. Most algorithms attempt to convert a Volterra equation of the first kind into a Volterra equation of the second kind and then solve it by the known algorithms for Volterra equations of the second kind. There are certainly some limitation to this idea. First, more analytical properties on the kernel K, such as differentiability, are required in the conversion. Second, the Lipschitz condition on kernel K is often not preserved by the conversion. Thus the complexity for Volterra equations of the first kind could be much higher than polynomial space even if the kernel K satisfies the Lipschitz condition.

A specifically interesting conversion is the following. First we assume that functions K and f are differentiable with respect to variable y. We may then differentiate both sides of equation (2) to obtain

$$K(y, y, \phi(y)) + \int_0^y \frac{\partial}{\partial y} K(y, s, \phi(s)) ds = f'(y).$$

Further assume that $K(y, y, \phi(y)) = K_1(y)\phi(y)$ with $K_1(y)$ nonvanishing. Then, equation (2) is equivalent to the following equation of the second kind:

$$\phi(y) = \frac{f'(y)}{K_1(y)} - \int_0^y H(y, s, \phi(s)) ds, \tag{4}$$

where $H(y, s, t) = \frac{\partial}{\partial y} K(y, s, t)/K_1(y)$. We now may be able to apply, for instance, Picard's successive approximation algorithm to solve this equation. As we pointed out above, however, the new kernel H does not satisfy the Lipschitz condition, even if we assume that the original kernel K does. What we can prove is that if K satisfies the Lipschitz condition then the function H satisfies the local Lipschitz condition. Therefore, by Theorem 3.30, equation (4) has an exponential time lower bound.

Theorem 3.31 [Ko, 1992]. *(a) Assume that functions $K : [0,1]^2 \times [-1,1] \to \mathbf{R}$ and $f : [0,1] \to \mathbf{R}$ satisfy the following properties:*

(i) K and f are polynomial-time computable;

(ii) K satisfies the Lipschitz condition;

(iii) $K(y, y, t) = K_1(y, y) \cdot t$ and $K_1(y, y) \neq 0$ a.e.[7];

(iv) $\frac{\partial}{\partial y} K(y, s, t)$ and $f'(y)$ exist and are polynomial-time computable.

[7] By a.e., we mean that for any $n > 0$, all but finitely many $y < 1 - 2^{-n}$ satisfies $K_1(y, y) \neq 0$.

Then, equation (2) *has a unique solution* ϕ *on* $[0, 1]$ *that is exponential-space computable.*

(b) There exist functions K *and* f *satisfying conditions (i)—(iv) above, and the unique solution* ϕ *of equation* (2) *has a polynomial modulus but is complete for EXPTIME.*

3.8 Two-Dimensional Regions

In this section, we study the notion of polynomial-time computability of a two-dimensional bounded, simply-connected region (i.e., a bounded, connected open set with no holes). There are a number of different representations for a two-dimensional region S, each having different applications. Here, we only consider three possible representations.

First, we extend the the notion of recursive approximability to the notion of polynomial-time approximability. Recall from Definition 2.19 that $E_M(n)$ is the set of points in $\mathbf{R}^2$ at which errors might occur for M on input n.

Definition 3.32 *A set* $S \subseteq \mathbf{R}^2$ *is* polynomial-time approximable *if there exists a polynomial-time oracle TM* M *such that, for each* $n \geq 1$*, the two-dimensional Lebesgue measure of* $E_M(n)$ *is bounded by* 2^{-n}*.*

The notion of polynomial-time approximability allows the machine which computes the characteristic function of the set S to have errors but requires that the error probability must be within a predefined bound. Next, we consider the notion of *polynomial-time recognizability* which allows the machine which computes the characteristic function of the set S to have errors only around the boundary of the set S, and hence is useful for regions whose boundaries are Jordan curves. For any set $S \subseteq \mathbf{R}^2$ and any point $\langle x, y\rangle \in \mathbf{R}^2$, we let $\delta(\langle x, y\rangle, S)$ be the distance between $\langle x, y\rangle$ and S: $\delta(\langle x, y\rangle, S) = \min\{|\langle x, y\rangle - \langle u, v\rangle| : \langle u, v\rangle \in S\}$.

Definition 3.33 *A set* $S \subseteq \mathbf{R}^2$ *is* polynomial-time recognizable *if there exists a polynomial-time oracle TM* M *such that, for any integer* $n \geq 1$, $E_M(n) \subseteq \{\langle x, y\rangle : \delta(\langle x, y\rangle, S) \leq 2^{-n}$*.*

The third notion of polynomial-time computability is strictly limited to sets with a polynomial-time computable Jordan curve as a boundary. (We defined the notion of computability of a Jordan curve in Section 2.5. Its extension to polynomial-time computability is straitforward.) We now compare these three notions.

First, is a polynomial-time recognizable set always polynomial-time approximable? The answer is a negative one, and the proof is a simple extension of that of Theorem 2.21.

Theorem 3.34 *There exists a polynomial-time computable Jordan curve* Γ *such that its interior* $\mathrm{Int}(\Gamma)$ *is polynomial-time recognizable but not recursively approximable.*

Next, we study whether a polynomial-time approximable set is always polynomial-time recognizable. To answer this question, we need a new concept of probabilistic computation. In Section 3.2.1, we have defined *BPP* as the class of sets computable in polynomial time by probabilistic TMs. We say two sets A and B are a *BP-pair* if $A \cap B = \emptyset$ and there exists a polynomial-time probabilistic TM M that accepts each $x \in A$ with probability $\geq 3/4$ and rejects each $y \in B$ with probability $\geq 3/4$. Two disjoint sets A and B are called *P-separable* if there exsits a set $C \in P$ such that $A \subseteq C$ and $B \subseteq \overline{C}$. It is not known whether $BPP = P$ implies that all BP-pairs are P-separable.

Theorem 3.35 [Chou and Ko, 1995]. *In the following, (a) $\Rightarrow$ (b) $\Rightarrow$ (c) $\Rightarrow$ (d).*

(a) $FP = \#P$.

(b) All BP-pairs are P-seperable.

(c) All polynomial-time approximable subsets of $[0,1]^2$ are polynomial-time recognizable.

(d) $BPP = P$.

The third result is about the question of whether the interior of a polynomial-time computable Jordan curve is polynomial-time recognizable.

Theorem 3.36 [Chou and Ko, 1995]. *In the following, (a) $\Rightarrow$ (b) $\Rightarrow$ (c).*

(a) $FP = \#P$.

(b) For every polynomial-time computable Jordan curve Γ, its interior $\mathrm{Int}(\Gamma)$ is polynomial-time recognizable.

(c) $UP = P$; i.e., one-way functions do not exist.

Finally we apply the above results to the problem about the measure of the interior of a polynomial-time computable Jordan curve.

Corollary 3.37 *The following are equivalent:*

(a) $FP = \#P$.

(b) For every polynomial-time approximable set $S \subseteq [0,1]^2$, its two-dimensional Lebesgue measure is polynomial-time computable.

(c) For every polynomial-time computable Jordan curve Γ that is rectifiable (i.e., having a finite length), the measure of its interior $\mathrm{Int}(\Gamma)$ is polynomial-time computable.

Corollary 3.38 *There exists a polynomial-time computable Jordan curve Γ whose interior is polynomial-time recognizable, but its measure is not recursive.*

For other complexity issues about two-dimensional regions, such as the distance between a point and a region and the complexity of computing the winding numbers of a given curve, see Chou and Ko [1995].

4 Other Approaches

As we pointed out in Section 1, many different approaches have been proposed to study the computational complexity of continuous problems. Each approach seems to aim at some specific domain of problems. We believe that the approach discussed in Section 3 is best for a general complexity theory for real functions, particularly for proving lower bounds based on discrete polynomial-time complexity theory. However, some specific problems and some upper bound results may be better studied in different computational models. In this section, we give a brief introduction to a few other interesting approaches. We limit ourselves to presenting an informal computational model and an example of results in each approach. The interested readers are referred to the technical papers in the bibliography.

4.1 Real Random Access Machines

In computational linear algebra and computational geometry, as well as in mathematical programming, one often uses the real random access machine (real RAM) as the computational model (cf. Borodin and Munro [1975]). A real RAM is a random access machine in which a real number, with the infinite precision, can be stored in a single address. The machine is able to perform certain basic operations on real numbers, such as addition, multiplication, division by nonzeros and comparison, where real numbers can be randomly accessed by their symbolic names. The complexity of a real RAM is usually measured by the number of operations performed on a specific input. A typical question studied in this model is the matrix multiplication problem: given two $n \times n$ matrices A and B, compute their product $C = A \times B$. (For recent progress on this problem, see, for instance, Coppersmith and Winograd [1990].)

As we pointed out earlier, this model of computation assumes the availability of the infinite-precision operations that is completely incompatible with the approach in Section 3. A computational theory of real functions based on the real RAM model is closer to *symbolic computation* than to numerical computation. However, in Sections 4.3 and 4.4, we will present two complexity theories of real functions based on this model.

4.2 Representing Real Functions by Programs

Since our interests are focused on polynomial-time computable real functions, a numerical operator which maps such a function to a real number may be considered as a function that maps a program, in a fixed computational model, to a real number. This idea is close to the approach in symbolic computation, except that here we allow hybrid numerical and symbolic computation in a single computational model.

4.2.1 Turing Machine Representation

In Corollary 3.4 we have seen that each polynomial-time computable real function f could be represented by a modulus function m and a discrete function ϕ mapping rationals to rationals. Thus, such a function could be represented by a TM, with the modulus functions predefined. For instance, the integration problem in this approach is formulated in the following form:

> *Integration Problem.* Given a TM M and two integers $i, n > 0$, find a rational number r such that if M computes a real function $f : [0,1] \to \mathbf{R}$ with a modulus function $m(k) = k^i$ then $|r - \int_0^1 f| \leq 2^{-n}$.

Note that this is not a purely symbolic computation problem, since we ask for a real number as the output. Also, this problem is stronger than the polynomial-time computability approach of Section 3: we ask here for a general integration algorithm for all polynomial-time computable functions with a modulus function $m(k) = k^i$, but in Section 3, we allowed different integration algorithms for different functions.

One of the interesting results obtained in this approach is the constructive fixed-point theorem. The classical d-dimensional Brouwer fixed-point theorem states that a continuous function f from Δ_d to Δ_d must have a fixed point $x \in \Delta_d$ in the sense that $f(x) = x$, where Δ_d is the d-dimensional unit cube. Hirsh et al. [1989] have shown that in the real RAM model, the functional mapping a real function to one of its fixed points is not polynomial-time computable. As we pointed out in Section 3, such a lower bound result on functionals does not reveal much of its inherent complexity. Papadimitriou [1990] considered the fixed-point theorem in the model of representing real functions by TMs.

> *Fixed-Point Problem.* Let d be an integer $d \geq 1$. Given a TM M computing a real function f from Δ_d to Δ_d, with a linear modulus function, and an integer n, find a point $x \in \Delta_d$ such that $|f(x) - x| \leq 2^{-n}$.

Note that the above question asks for an *approximate* fixed point for each input n, and these approximate fixed points are not required to converge to a single fixed point. Thus, this question is weaker than the question of computing the functional of mapping real functions to their fixed points. To characterize the complexity of this problem, Papadimitriou [1990] defined a new discrete complexity class *PDLF* which contains all functions that are defined by a local search strategy similar to that of Sperner's lemma for the fixed-point theorem. We omit here the formal definition of *PDLF*. It is known that $FP \subseteq PDLF \subseteq TFNP$, where *TFNP* denotes the class of all total functions that are computable by nondeterministic TMs. Papadimitriou [1990] showed some evidence that this class *PDLF* is probably not equal to *FP* nor to *TFNP*.

Theorem 4.1 *The above fixed-point problem is complete for PDLF, when $d \geq 3$.*

It is obvious that the fixed-point problem in the case $d = 1$ is polynomial-time solvable. It is however not known whether the fixed-point problem in the case $d = 2$ is polynomial-time solvable or is complete for *PDLF*.

It is worth pointing out that this approach is very closely related to the approach presented in Section 3. For instance, the ideas of the proof of the above theorem can also be formulated in the complexity theory of Section 3. In the following, we say a function $f : \Delta_d \to \Delta_d$ is *well-conditioned* to mean that its approximate fixed points x satisfying $|f(x) - x| \leq \epsilon$ are actually close to some real fixed points y satisfying $f(y) = y$.

Corollary 4.2 [Ko, 1995b] *Let $d \geq 3$. The following are equivalent:*

(a) PDLF = FP.

(b) For every well-conditioned $f : \Delta_d \to \Delta_d$ that is computable in polynomial-time and has a unique fixed point x in Δ_d, x is polynomial-time computable.

A weaker result for the case $d = 2$ is also obtained in Ko [1995b].

4.2.2 Real-Valued Circuits

In addition to the Turing machine representation, a real function could also be represented by real-valued circuits. Since real-valued circuits are just an alternative representation to the real RAM model, this approach is more closely related to symbolic computation than to numerical computation. Indeed, the functions that could easily be represented by real-valued circuits are just rational functions f_1/f_2, where f_1 and f_2 are polynomial functions.

Hoover [1990], however, found a very interesting relation between the real-valued circuit representation and the boolean-valued circuit representation for real functions. Namely, he showed that the class of polynomial-time computable real functions, in the sense of Definition 3.2, coincides with the class of functions that are *sup-approximable* by a uniform family of real-valued circuits with polynomial-size and polynomial-magnitude. In the above, a family $\{c_{n,j}\}$ of real-valued circuits is of polynomial-size if the number of gates in $c_{n,j}$ is bounded by a polynomial in $n+j$; it is of polynomial-magnitude if for any input $x \in [-2^j, 2^j]$, each gate in $c_{n,j}$ outputs a real number in $[-2^{p(n+j)}, 2^{p(n+j)}]$ for some fixed polynomial p; it is a uniform family if there is a polynomial-time TM computing $c_{n,j}$ from (n, j). Also, a real function f is sup-approximated by a family $\{c_{n,j}\}$ of real-valued circuits if for all $x \in [-2^j, 2^j]$, $|f(x) - c_{n,j}(x)| \leq 2^{-n}$.

The above characterization of Hoover gives us a weak form of the polynomial-time version of Wierestrass approximation theorem (cf. Theorem 2.15 and see Ko [1991]), and, more importantly, provides a transformation between symbolic computation and numerical computation. For instance, Kaltofen [1987] asked whether the integration of polynomial functions is computable in polynomial time, provided that polynomial functions are given by straight-line programs. Using Hoover's characterization, it is immediate that this question is as difficult as the numerical integration problem and hence is $\#P$-complete.

4.3 Information-Based Complexity Theory

Traub et al. [1988] established the information-based complexity theory as a general theory to study approximately solved numerical problems (see also Werschulz [1991] and Traub and Woźniakowski [1993]). The main feature of the theory is to distinguish the notion of *information complexity* from the notion of *computational* (or, sometimes called *combinatorial*) *complexity* of a problem. Consider, for instance, the integration problem. In order to compute an approximate value of $\int_0^1 f(t)\ dt$ with an error bounded by ϵ, we must first evaluate f at some points $x_1, \ldots, x_n \in [0, 1]$, and then compute a real number y from the values $f(x_1), \ldots, f(x_n)$ such that $|y - \int_0^1 f(t)\ dt| < \epsilon$. We will call the amount of function evaluations of $f(x_i)$ necessary to obtain the approximation y as the *information complexity* of the integration problem, and call the amount of time required to compute y from the given values $f(x_1), \ldots, f(x_n)$ as the computational complexity. Traub et al. [1988] demonstrated that information complexity by itself is often a dominating complexity measure for many numerical problems.

Quantitatively, the information complexity of a problem is measured by the *radius of information* required to solve the problem. Informally speaking, for any given problem, we can only collect a finite amount of information $N = N(f)$ about the given input function f. From this set N of information, there are potentially infinitely many functions g in the domain of the input functions that have the same set of information $N(g) = N$. The radius $r(N)$ of information N is defined to be the radius of the minimum *ball* that contains all these functions g, where the measure for the radius depends on the problem. A nice general theorem in this theory states that the information N is strong enough to solve the problem with an error bounded by ϵ with respect to all functions in the domain (disregarding the computational complexity) if and only if the radius $r(N)$ of information N is less than ϵ. Using this theorem, we can compute the *optimal information* for a numerical problem, and the *optimal algorithm* with respect to the optimal information. Consider again the integration problem. Assume that the domain consists of all functions f such that $|f'(x)| \leq 1$ for all $x \in [0, 1]$. Then, it can be proved that the optimal information for computing $\int_0^1 f(t)\, dt$ within an error $1/4n$ is taking $f(x_i)$ at equi-distant points $1/2n$, $3/2n$, ..., $(2n-1)/2n$, and the radius of this set of information is $1/4n$. The worst-case error $1/4n$ could be achieved simply by the Riemann sum of the sample values $f((2i-1)/2n)$, $i = 1, \ldots, n$.

Comparing this complexity theory with the theory of polynomial-time analysis of Section 3, we notice the following differences. First, as we pointed out above, the information-based complexity emphasizes information complexity and generally disregards computational complexity. In the theory of polynomial-time analysis, both complexity measures are included in the consideration, and therefore the domain of the input functions is often restricted to polynomial-time computable functions. Second, the computational model used in the information-based complexity theory is close to the real RAM model with some new, non-conventional operations. Traub et al. [1988] pointed out that it is important to distinguish between different types of information such as the function evaluation $f(x)$, the derivative evaluation $f'(x)$, or sometimes even the integral evaluation $\int_0^1 f$. For different problems, one may assume weak or strong analytic properties of the input functions, and may decide to use inexpensive or costly information to solve the problem. Thus, we may imagine an extended real RAM model with oracles, in which one may assume the availability of different types of information given in the form of oracles, and the cost of collecting different types of information is measured separately (cf. Novak [1995]). Third, the lower bound achieved in the information-based complexity theory is basically by the adversary argument, and in the theory of Section 3, the technique of NP-completeness theory is applied.

Among the similarities between the two theories, we would like to point out that many notions and techniques of discrete algorithm theory could be applied to the information-based complexity theory. Among them, the notions of average-case complexity, parallel computation and probabilistic algorithms have been studied in the theory. Furthermore, if the real RAM model uses the logarithmic cost measure instead of the unit cost measure, then the two models are even closer (see Brattka and Hertling [1996] and Weihrauch [1997b]).

4.4 A Theory of NP-Completeness for Algebraic Computation

Blum et al. [1989] recently introduced a new theory of *NP*-completeness based on an algebraic computational model that is very close to the real RAM model. Since the only operations allowed in such a machine are arithmetic operations on reals, this theory is naturally more suitable for algebraic computatioinal problems. There are two novel features of this theory, however, that make it different from the classical algebraic complexity theory. First, the machine is allowed to have some fixed real numbers as its parameters; that is, a finite number of fixed real numbers are allowed to be prestored in the machine. As a consequence, there exist an uncountable number of machines (unlike any other computational models used in discrete or continuous computational theory). Second, nondeterministic machines are defined in such a way that a nondeterministic node of a machine may guess a real number with infinite precision (instead of being able to guess only a single bit in nondeterministic TMs). This means that the nondeterministic node has an uncountable number of computational successors and, in general, cannot be simulated by deterministic machines.

To be more precise, a (deterministic) machine M in this theory is a real RAM with arithmetic operations and the compare-and-branch operation that compares two real numbers and branches to new instructions according to the comparison result. Let $\mathbf{R}^\infty$ denote the set of all infinite sequences of real numbers with only a finite number of nonzeros. A sequence $\vec{x} = (x_1, \ldots, x_n, 0, \ldots)$ in $\mathbf{R}^\infty$ is of *size* n, and we write $\ell(\vec{x}) = n$, if n is the largest integer such that $x_n \neq 0$. A *decision problem* A is a subset of $\mathbf{R}^\infty$. We say a decision problem A is *polynomial-time computable*, and write $A \in P_{\mathbf{R}}$, if there is a machine M that solves an instance $\vec{x}$ of A such that $\vec{x} \in A$ if and only if $M(\vec{x})$ outputs 1 and that $M(\vec{x})$ halts in time $O(\ell(\vec{x})^k)$ for some constant $k > 0$. A decision problem A is in $NP_{\mathbf{R}}$ if there exists a polynomial-time machine M such that for each instance $\vec{x} \in \mathbf{R}^\infty$, $\vec{x} \in A$ if and only if there exists a sequence $\vec{y} \in \mathbf{R}^\infty$, $\ell(\vec{y}) \leq O(\ell(\vec{x})^k)$ for some constant $k > 0$, such that $M(\vec{x}, \vec{y})$ outputs 1. A decision problem A is $NP_{\mathbf{R}}$*-complete* if $A \in NP_{\mathbf{R}}$ and for all $B \in NP_{\mathbf{R}}$ there exists a polynomial-time machine M such that for each $\vec{x} \in \mathbf{R}^\infty$, $M(\vec{x})$ outputs a $\vec{y} \in \mathbf{R}^\infty$ such that $\vec{x} \in A$ if and only if $\vec{y} \in B$. It can be seen that these definitions are motivated by the discrete *NP*-completeness theory.

Blum et al. [1989] demonstrated a few problems that are $NP_{\mathbf{R}}$-complete. Among them, the following problem is the most fundamental one.

> *Degree-4 Algebraic Variety Problem.* Given a degree-4 polynomial ϕ over n variables $x_1, \ldots, x_n$, determine whether there exists an $\vec{x} \in \mathbf{R}^\infty$, $\ell(\vec{x}) \leq n$, such that $\phi(\vec{x}) = 0$.

Theorem 4.3 *The degree-4 algebraic variety problem is* $NP_{\mathbf{R}}$*-complete.*

From these $NP_{\mathbf{R}}$-complete problems, it is established that whether $P_{\mathbf{R}}$ is equal to $NP_{\mathbf{R}}$ is the fundamental question in this theory. Although this question is based on a theory using the real RAM model, we may treat the theory as a theory of symbolic computation and hence a subtheory of the classical discrete complexity theory. Since it is not known whether $P_{\mathbf{R}} = NP_{\mathbf{R}}$, it would be interesting to know whether it is equivalent to some well-known open questions in discrete complexity theory. At present, this question is not known to be equivalent to the question $P = NP$ in the classical *NP*-completeness theory. The best known relation between this question and questions in discrete complexity theory is that if $P = PSPACE$ then $P_{\mathbf{R}} = NP_{\mathbf{R}}$. For further development of this theory, see the forth coming book [Blum et al., 1997].

5 References

Aberth, O. [1980], *Computable Analysis*, McGraw-Hill, New York.

Baker, C. [1977], *The Numerical Treatment of Integral Equations*, Oxford University Press.

Balcázar, J., Diaz, J. and Gabarró, J. [1988], *Structural Complexity I*, Springer-Verlag, Berlin.

Bishop, E. [1967], *Foundations of Constructive Analysis*, McGraw-Hill, New York.

Bishop, E. and Bridges, D. [1985], *Constructive Analysis*, Springer-Verlag, Berlin.

Blum, L., Cucker, F., Shub, M. and Smale, S. [1997], *Complexity and Real Computation: A Manifesto*, Springer, (to appear).

Blum, L., Shub, M. and Smale, S. [1989], On a theory of computation and complexity over the real numbers; *NP* completeness, recursive functions and universal machines, *Bullet. Amer. Math. Soc. (new series)* **21**, 1–46.

Book, R. [1974], Tally languages and complexity classes, *Inform. Contr.* **26**, 186–193.

Borodin, A. and Munro, I. [1975], *The Computational Complexity of Algebraic and Numeric Problems*, American Elsevier, New York.

Brattka, V. and Hertling, P. [1996], Feasible real random access machines, in *SOFSEM'96: Theory and Practice of Informatics*, Jeffrey, K., Král, J. and Bartošek, M., eds., Lecture Notes in Computer Science, **1175**, Springer, 335–342.

Chou, A. W. and Ko, K. [1995], Computational complexity of two-dimensional regions, *SIAM J. Comput.* **24**, 923–947.

Cook, S. [1985], A taxonomy of problems with fast parallel algorithms, *Inform. Contr.* **64**, 2–22.

Coppersmith, D. and Winograd, S. [1990], Matrix multiplication via arithmetic progressions, *J. Symbolic Comput.* **9**, 251–280.

Falconer, K. [1991], *Fractal Geometry, Mathematical Foundations and Applications*, John Wiley & Sons, New York.

Friedman, H. [1984], On the computational complexity of maximization and integration, *Advances in Math.* **53**, 80–98.

Garey, M. R. and Johnson, D. [1979], *Computers and Intractability, a Guide to the Theory of NP-Completeness*, W.H. Freeman, San Francisco.

Grzegorczyk, A. [1957], On the definitions of computable real continuous functions, *Fund. Math.* **44**, 61–71.

Grzegorczyk, A. [1959], Some approaches to constructive analysis, in *Constructivity in Mathematics*, A. Heyting, ed., North-Holland, Amsterdam, 43–61.

Hirsh, M., Papadimitriou, C.H. and Vavasis, S. [1989], Exponential lower bounds for finding Brouwer fixpoints, *J. Complexity* **5**, 379–416.

Hoover, H.J. [1987], *Feasibly Constructive Analysis*, Ph.D. Dissertation, Department of Computer Science, University of Toronto, Toronto, Canada.

Hoover, H.J. [1990], Feasible real functions and arithmetic circuits, *SIAM J. Comput.* **19**, 182–204.

Hopcroft, J. and Ullman, J. [1979], *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley, Reading, Mass.

Kaltofen, E. [1987], Single-factor Hensel lifting and its application to the straght-line complexity of certain polynomials, *Proceedings, 19th ACM Symposium on Theory of Computing*, 443–452.

Ko, K. [1982], The maximum value problem and NP real numbers, *J. Comput. System Sci.* **24**, 15–31.

Ko, K. [1983a], On the definitions of some complexity classes of real numbers, *Math. Systems Theory* **16**, 95–109.

Ko, K. [1983b], On the computational complexity of ordinary differential equations, *Inform. Contr.* **58**, 157–194.

Ko, K. [1986], Approximation to measurable functions and its relation to probabilistic computation, *Annals of Pure and Applied Logic* **30**, 173–200.

Ko, K. [1990], Inverting a one-to-one real function is inherently sequential, in *Feasible Mathematics*, Eds., Buss, S. and Scott, P., Birkhäuser, Boston, 239–257.

Ko, K. [1991], *Complexity Theory of Real Functions*, Birkhäuser, Boston.

Ko, K. [1992], On the computational complexity of integral equations, *Annals of Pure Appl. Logic*, **58**, 201–228.

Ko, K. [1995a], A polynomial-time computable curve whose interior has a nonrecursive measure, *Theoret. Comput. Sci.* **145**, 241–270.

Ko, K. [1995b], Computational complexity of fixed points and intersection points, *J. Complexity*, **11**, 265–292.

Ko, K. [1997], On the computatbility of fractal dimensions and Julia sets, *Annals of Pure and Applied Logic* (to appear).

Ko, K. and Friedman, H. [1982], Computational complexity of real functions, *Theoret. Comput. Sci.* **20**, 323–352.

Ko, K. and Weihrauch, K. [1996], On the measure of two-dimensional regions with polynomial-time computable boundaries, *Proceedings, 11th IEEE Conference on Computational Complexity*, IEEE, 150–159.

Lacombe, D. [1955], Extension de la notion de fonction recursive aux fonctions d'une ow plusiers variables reelles, and other notes, *Comptes Rendus* **240**, 2478–2480; **241**, 13–14, 151–153, 1250–1252.

Metakids, G. and Nerode, A. [1982], The introduction of non-recursive methods into mathematics, *The L. E. J. Brouwer Centerary Symposium Volume*, Studies in Logic and Foundations, Vol. **110**, North Holland, 319–336.

Metakids, G., Nerode, A. and Shore, R.A. [1985], Recursive limits on the Hahn-Banach theorem, *Errett Bishop: Reflection on Him and His Research*, Contemporary Mathematics, Vol. **39**, Amer. Math. Assoc., 85–91.

Miller, R. K. [1971], *Nonlinear Volterra Integral Equations*, W. A. Benjamin, Menlo Park, California.

Miller, W. [1970], Recursive function theory and numerical analysis, *J. Assoc. Comput. Mach.* **4**, 465–472.

Mostowski, A. [1957], On computable sequences, *Fund. Math.* **44**, 37–51.

Mostowski, A. [1959], On various degrees of constructivism, in *Constructivity in Mathematics*, A. Heyting, ed., North-Holland, Amsterdam, 178–194.

Myhill, J. [1971], A recursive function defined on a compact interval and having a continuous derivative that is not recursive, *Michigan Math. J.* **18**, 97–98.

Neff, C.A. [1990], Specified precision polynomial root isolation is in *NC*, *Proceedings, 31st IEEE Symposium on Foundations of Computer Science*, 152–162.

Nerode, A. and Hwang, W. [1985], Applications of pure recursion theory in recursive analysis, *Acta Mathematica Sinica* (in Chinese) **28**, 625–636.

Novak, E. [1995], The real number model in numerical analysis, *J. Complexity* **11**, 57–73.

Papadimitriou, C.H. and Tsitsiklis, J. [1986], Intractable problems in control theory, *SIAM J. Contr. Optim.*, **24**, 639–654.

Papadimitriou, C.H. [1990], On graph-theoretic lemmata and complexity classes, *Proceedings, 31st IEEE Symposium on Foundations of Computer Science*, 794–801.

Pour-El, M. and Caldwell, J. [1975], On a simple definition of computable functions of a real variable—with applications to functions of a complex variable, *Z. Math. Logik Grundlag. Math.* **21**, 1–19.

Pour-El, M. and Richards, I. [1978], Differentiability properties of computable functions—a summary, *Acta Cybernetica* **4**, 123–125.

Pour-El, M. and Richards, I. [1979], A computable ordinary differential equation which possesses no computable solution, *Annals Math. Logic* **17**, 61–90.

Pour-El, M. and Richards, I. [1983a], Computability and noncomputability in classical analysis, *Trans. Amer. Math. Soc.* **275**, 539–560.

Pour-El, M. and Richards, I. [1983b], Noncomputability in analysis and physics: a complete determination of the class of noncomputable linear operators, *Advances in Math.* **48**, 44–74.

Pour-El, M. and Richards, I. [1988], *Computability in Analysis and Physics*, Springer, Berlin.

Robinson, R. [1951], Review of "Rekursive Funktionen" by R. Péter, *J. Symbolic Logic* **16**, 280–282.

Šanin, N. [1968], *Constructive Real Numbers and Function Spaces*, Translations of Mathematical Monographs **21**, transl. by E. Mendelson, AMS, Providence, Rhode Island.

Savitch, W. [1970], Relationships between nondeterministic and deterministic tape complexities, *J. Comput. Systems Sci.* **4**, 177–192.

Schönhage, A. [1982], The fundamental theorem of algebra in terms of computational complexity, preprint, Mathematisches Institut der Universität Tübingen, West Germany.

Specker, E. [1959], Der satz vom maximum in der rekursiven analysis, in *Constructivity in Mathematics*, A. Heyting, ed., North-Holland, Amsterdam, 254–265.

Traub, J., Wasilkowski, G. and Woźniakowski, H. [1988], *Information-Based Complexity*, Academic Press, New York.

Traub, J. and Woźniakowski, H. [1993], Recent pregress in information-based complexity, *Bullet. European Assoc. Theoret. Comput. Sci.* **51**, 141–154.

Tricomi, F. G. [1957], *Integral Equations*, Interscience, New York.

Weihrauch, K. [1995], *A Simple Introduction to Computable Analysis*, Informatik Berichte **171**, FernUniversität Hagen, Hagen, Germany.

Weihrauch, K. [1997a], A foundation for computable analysis, in *Combinatorics, Complexity and Logic*, Bridges, D. S., Calude, C. S., Gibbons, J., Reeves, S. and Witten, I. H., eds., Springer, 66–89.

Weihrauch, K. [1997b], A refined model of computation for IBC, Preprint.

Werschulz, A. G. [1991], *The computational Complexity of Differential and Integral Equations: An Information-Based Approach*, Oxford University Press, Oxford.