Complexity Theory *

Steven Homer
Department of Computer Science
Boston University
Boston, MA 02215

Alan L. Selman
Department of Computer Science
State University of New York at Buffalo
226 Bell Hall
Buffalo, NY 14260

June 8, 1992

1 Introduction

The goal of complexity theory is to provide mechanisms for classifying combinatorial problems and measuring the computational resources necessary to solve them. Complexity theory provides an explanation of why certain problems have no practical solutions and provides a way of anticipating difficulties involved in solving problems of certain types. The classification is quantitative and is intended to investigate what resources are necessary (lower bounds) and what resources are sufficient (upper bounds) to solve various problems.

*Draft of an article for Encyclopedia of Computer Science and Technology. The authors wish to acknowledge support by the National Security Agency under grants MDA904-87-H-2003 and MDA-87-H-2020 and by the National Science Foundation under grants CCR-8814339 and CCR-9002292.

This classification should not depend on a particular computational model but rather should measure the intrinsic difficulty of a problem. Fortunately, such a measurement mechanism is possible as, up to the exactitude required, there are many general models of computation which give rise to the same classification. Thus the theory we provide is a robust one whose definition does not depend on a single machine or its technical features.

A robust theory of computable functions dates back to the work of Church and Turing, and provides models of computation and methods for complexity theory. Much of that work predated digital computers and was without forethought of modern technology. However, the most difficult and interesting facets of complexity theory appear exactly where the older mathematical theories lend no guiding hand and where new methods, particular to this study, must be developed.

Complexity theory today addresses issues of contemporary concern: parallel computation, circuit design, computations that depend on random number generators, development of efficient algorithms. Moreover, complexity theory is interested not in the merely computable but in problems that are *efficiently* computable. Algorithms whose running times are n^2 in the size of their inputs can be implemented to execute efficiently even for fairly large values of n, but algorithms that require an exponential running time can be executed only for small values of n. It is common today to identify efficiently computable problems with those that have polynomial time algorithms. (These ideas will be made mathematically precise.)

A complexity measure quantifies the use of a particular computational resource during execution of a computation. The two most important measures, and the two most common measures, are *time*, the time it takes a program to execute, and *space*, the amount of store used during a computation. However, other measures are considered as well, and we will introduce other resources as we proceed through this exposition.

The basic model of computation for our study is the multitape Turing machine. As mentioned above, complexity measures should not depend too much on the choice of computational model. To this end, we introduce an expanded version of Church's thesis (cf. [Chu36] and [Tur 7]). Church's thesis states that every computational device can be simulated by a Turing machine. Our expanded Church's thesis is even stronger. It asserts that every computational device can be simulated by a multitape Turing machine with the simulation taking at most polynomial time. (That is, if M is some computational device, then there is a Turing machine T_M that simulates M such that programs run at most a polynomial number of additional steps on T_M than on M.) Evidence for this thesis comes from the fact that it has withstood twenty years of intensive research, and it is has been proven for all known reasonable models of sequential computation including random access machines (RAMS). This thesis is particularly fortunate because of another assertion known as Cobham's thesis [Cob64]. Cobham's thesis asserts that computational problems can be feasibly computed on some computational device only if they can be computed in polynomial time. Truth be told, an n^{100} time algorithm is not a useful algorithm. It is a remarkable phenomenon though, that problems for which polynomial algorithms are found have such algorithms with small exponents and with small coefficients. Thus, combining the two theses, a problem can be feasibly computed only if it can be computed in polynomial time on some multitape Turing machine.

Complexity theory forms a basis for the classification and analysis of combinatorial algorithms. To illustrate this, consider the problem of determining whether an arbitrary graph possesses a Hamiltonian Circuit (i.e., a simple cycle that contains every vertex.) Currently it is not known whether this problem has a feasible solution, and all known solutions are equivalent to a sequential search of all paths through the graph, testing each in turn for the Hamiltonian property. Since complexity will be measured by time and/or space in Turing machine computations, we ask the reader to recall that input to a Turing machine is a word over some finite input alphabet Σ. (We will not define Turing machines in this article.) In order for the theory to be useful it must be possible to encode data structures such as graphs into words over Σ so that what is intuitively the size of the graph differs from the length of the input by no more than a polynomial. In fact, it is easy to do so. Then, we must demand that the theory is capable of classifying the intrinsic complexity of this problem in a precise way, and is capable of elucidating the difficulty in finding an efficient solution to this problem.

To summarize the discussion so far, we conclude that the use of the multitape Turing machine as basic model of computation is robust over all possible models of computation, and is robust via standard encodings for all standard data types.

2 Modes of Computation

A computation of a Turing machine is a sequence of moves, as determined by the transition function. Ordinarily, the transition function is single-valued. Such Turing machines are called *deterministic* and their computations are sequential.

A *nondeterministic* Turing machine is one that allows for a choice of next moves; in this case the transition function is multivalued. If M is a nondeterministic Turing machine and x is an input word, then M and x specify a *computation tree*. The root of the tree is the *initial configuration* of M. The children of a node are the configurations that may follow in one move. A path from the root to a leaf is a computation, and a computation is accepting if the leaf is an accepting configuration.

Given a Turing machine M, the *language* $L(M)$ *accepted by* M is the set of words such that some computation of M on x is accepting. (Observe that this definition is meaningful for both the deterministic and the nondeterministic modes.)

Nondeterminism is frequently a problematic concept for beginners. One should not expect nondeterministic machines to be designed to be executed, as are ordinary computing devices. Rather, one should understand nondeterministic Turing machines to be a very useful mode for classification of computational problems. For example, whereas it is not known whether there is a deterministic polynomial time-bounded Turing machine to solve (an encoding of) the Hamiltonian Circuit problem, it is easy to design a nondeterministic polynomial time-bounded Turing machine that solves this problem. This is what makes it possible to give an exact classification of the Hamiltonian Circuit problem. Indeed, this problem is known to be NP-complete, which places it among hundreds of other important computational problems whose deterministic complexity is still open.

We will return to nondeterminism and to a precise definition of NP-complete in a later section.

3 Complexity Classes and Complexity Measures

In order to define time complexity, we consider on-line Turing machines. An *on-line* Turing machine is a multitape Turing machine whose input is written on one of the work tapes which can be rewritten and used an as ordinary work tape. The machine may be either deterministic or nondeterministic. Let M be an on-line Turing machine, and let T be a function defined on the set of natural numbers. M is a $T(n)$ *time-bounded* Turing machine if for every input of length n, M makes at most $T(n)$ moves before halting. If M is nondeterministic, then every computation of M on words of length n must take at most $T(n)$ steps. The language $L(M)$ that is accepted by a deterministic $T(n)$ time-bounded M has *time complexity* $T(n)$.

By convention, the time it takes to read the input is counted, and every machine is entitled to read its input. This takes $n+1$ steps. So, when we say a computation has time complexity $T(n)$, we really mean $\max(n+1, \lceil T(n) \rceil)$.

Denote the length of a word x by $|x|$. We might be tempted to say that a nondeterministic Turing machine is $T(n)$ time-bounded if for every input word $x \in L(M)$, the number of steps of the shortest accepting computation of M on x is at most $T(|x|)$. (It turns out that the formulations are equivalent for the specific time bounds that we will write about. But, they are not equivalent for arbitrary time bounds.)

A complexity class is a collection of sets that can be accepted by Turing machines with the same resources. Now we define the time-bounded complexity classes: Define DTIME($T(n)$) to be the set of all languages having time-complexity $T(n)$. Define NTIME($T(n)$) to be the set of all languages accepted by nondeterministic $T(n)$ time-bounded Turing machines.

In order to define space complexity, we need to use off-line Turing machines. An *off-line* Turing machine is a multitape Turing machine with a separate read-only input tape. The Turing machine can read the input, but cannot write over the input. Let M be an off-line multitape Turing machine and let S be a function defined on the set of natural numbers. M is an $S(n)$ *space-bounded* Turing machine if for every word of length n, M scans at most $S(n)$ cells on any storage tape. If M is nondeterministic, then every computation must scan no more than $S(n)$ cells on any storage tape. The language $L(M)$ that is accepted by an $S(n)$ deterministic space-bounded Turing machine has *space-complexity* $S(n)$.

Observe that the space taken by the input is not counted. Every Turing machine is permitted to use at least one work cell, so when we say that a problem has space complexity $S(n)$, we always mean $\max(1, \lceil S(n) \rceil)$. For example, space complexity $\log(n)$ can never be 0.

One might be tempted to say that a nondeterministic Turing machine is $S(n)$ space-bounded if for every word of length n that belongs to $L(M)$, there is an accepting computation that uses no more than $S(n)$ work cells on any work tape. (The parenthetical comment made for time-bounds applies here as well.)

Now, we define the space-bounded complexity classes: Define DSPACE($S(n)$) to be the set of all languages having space- complexity $S(n)$. Define NSPACE($S(n)$) to be the set of all languages accepted by nondeterministic $S(n)$ time-bounded Turing machines.

The study of time complexity begins with the paper [HS65] of Hartmanis and Stearns. The title of this paper contains the first usage of the phrase "computational complexity." The study of space complexity begins with the paper [HLS65]. These seminal papers introduced some of the issues that remain of concern even today. These include time/space trade-offs, inclusion relations, hierarchy results, and efficient simulation of nondeterministic computations.

We will be primarily concerned with classes defined by logarithmic, polynomial and exponential functions. As we proceed to relate and discuss various facts about complexity classes in general, we will see what impact they have on the following list of *standard* complexity classes. These classes are well-studied in the literature and each contains important computational problems. The classes are introduced with their common notations.

1. L = DSPACE($\log(n)$);
2. NL = NSPACE($\log(n)$);
3. POLYLOGSPACE = $\bigcup\{\text{DSPACE}(\log(n)^k) \mid k \geq 1\}$;
4. DLBA = $\bigcup\{\text{DSPACE}(kn) \mid k \geq 1\}$;
5. LBA = $\bigcup\{\text{NSPACE}(kn) \mid k \geq 1\}$;
6. P = $\bigcup\{\text{DTIME}(n^k) \mid k \geq 1\}$;

7. $\mathrm{NP} = \bigcup\{\mathrm{NTIME}(n^k) \mid k \geq 1\}$;

8. $\mathrm{E} = \bigcup\{\mathrm{DTIME}(k^n) \mid k \geq 1\}$;

9. $\mathrm{NE} = \bigcup\{\mathrm{NTIME}(k^n) \mid k \geq 1\}$;

10. $\mathrm{PSPACE} = \bigcup\{\mathrm{DSPACE}(n^k) \mid k \geq 1\}$;

11. $\mathrm{EXP} = \bigcup\{\mathrm{DTIME}(2^{p(n)}) \mid p$ is a polynomial $\}$;

12. $\mathrm{NEXP} = \bigcup\{\mathrm{NTIME}(2^{p(n)}) \mid p$ is a polynomial $\}$;

The study of the computational complexity of specific computational problems is called *concrete complexity*. The concrete complexity literature provides a rich source of examples of problems in our standard list. The following is merely an introduction to a vast subject.

NL contains the problem of determining for arbitrary directed graphs G and vertices u and v, whether there is a path from u to v. This problem is not known to belong to L. The latter class contains the restriction of this problem to the case that no vertex has more than one directed edge leading from it. [Jon73, Jon75]. P is identified with the class of feasibly computed problems. The corresponding nondeterministic class NP will be discussed in a later section.

The class $\bigcup\{\mathrm{NSPACE}(kn) \mid k \geq 1\}$ is denoted LBA because it is known to be identical to the class of languages accepted by linear-bounded automata, otherwise known as the context-sensitive languages [Myh60, Kur64]. This explains the notation for the corresponding deterministic class as well.

E characterizes the complexity of languages accepted by writing push-down automata [Mag69], and NE characterizes the complexity of a well-known computational problem in finite model theory [JS74]. PSPACE contains many computational games, such as HEX [ET76].

4 Basic Results

Several different types of results apply to all complexity classes. They show that the definitions of these classes are invariant under small changes and prove a variety of relationships between these classes.

4.1 Linear Compression and Speedup

The first results are of the form: if a language can be accepted with resource $f(n)$, then it can be accepted with resource $cf(n)$, for any $c > 0$. These results justify use of "big-oh" notation for complexity functions.

The Space Compression Theorem [HLS65] asserts that if L is accepted by a k-tape $S(n)$ space-bounded Turing machine, then for any $c \geq 0$, L is accepted by a k-tape $cS(n)$ space-bounded Turing machine. If the $S(n)$ space-bounded Turing machine is nondeterministic, then so is the $cS(n)$ space-bounded Turing machine. A simple machine simulation proves the result. As a corollary, it follows that DSPACE($S(n)$) = DSPACE($cS(n)$) and NSPACE($S(n)$) = NSPACE($cS(n)$), for all $c \geq 0$.

Linear speedup of time is possible too, but not quite as readily as is linear compression of space. The Linear Speedup Theorem [HS65] asserts that if L is accepted by a k-tape $T(n)$ time-bounded Turing machine, $k \geq 1$, and if

$$\inf_{n\to\infty} T(n)/n = \infty, \tag{1}$$

then for any $c \geq 0$, L is accepted by a k-tape $cT(n)$ time-bounded Turing machine. Thus, if $\inf_{n\to\infty} T(n)/n = \infty$ and $c \geq 0$, then DTIME($T(n)$) = DTIME($cT(n)$).

Condition (1) stipulates that $T(n)$ grows faster than every linear function. The Linear Speedup Theorem does not apply if $T(n) = cn$, for some constant c. Instead, we have the result that for all $\epsilon \geq 0$, DTIME($O(n)$) = DTIME($(1+\epsilon)n$), and the proof actually follows from the proof of the Linear Speedup Theorem. This result cannot be improved, for Rosenberg [Ros67] showed that DTIME(n) $\neq$ DTIME($2n$).

The two linear speedup theorems also hold for nondeterministic machines. Thus, if $\inf_{n\to\infty} T(n)/n = \infty$ and $c \geq 0$, then NTIME($T(n)$) = NTIME($cT(n)$). And, for all $\epsilon \geq 0$, NTIME($O(n)$) = NTIME($(1+\epsilon)n$).

However, a stronger result is known for nondeterministic linear-time complexity classes. A Turing machine that accepts inputs of length n in time $n+1$ (the time it takes to read the input) is called *real-time*. Nondeterministic Turing machines that accept in time

$n+1$ are called *quasi-realtime*. The class of quasi-realtime languages is NTIME$(n+1)$. Book and Greibach [BG70] showed that NTIME$(n+1) = \bigcup\{\text{NTIME}(cn) \mid c \geq 1\}$.

Since DTIME(n) is a proper subset of DTIME$(2n)$, and DTIME$(2n) \subseteq$ NTIME$(2n)$ $=$ NTIME(n), it follows that DTIME$(n) \neq$ NTIME(n). In 1983, Paul, Pippenger, Szemeredi, and Trotter [PPST83] obtained the striking and deep result that DTIME$(O(n))$ $\neq$ NTIME$(O(n))$.

4.2 Inclusion Relationships

Now we survey the known inclusion relationships between time-bounded and space-bounded, deterministic, and nondeterministic classes.

First of all, it is trivial that for every function f, DTIME$(f) \subseteq$ DSPACE(f).

Obviously a Turing machine might enter an infinite loop and still use only bounded space. Nevertheless, if a language L is accepted by an $S(n)$ space-bounded Turing machine, where $S(n) \geq \log(n)$, then L is accepted by an $S(n)$ space-bounded Turing machine that halts on every input. The proof depends on the observation that within space $S(n)$ a Turing machine can enter at most an exponential in $S(n)$ possible distinct configurations. A machine enters an infinite loop by repeating one of these configurations, thus making loop detection possible. Analysis of this result yields the following theorem.

Theorem 1 DSPACE$(S(n)) \subseteq \bigcup\{\text{DTIME}(c^{S(n)}) \mid c \geq 0\}$ *, for* $S(n) \geq \log(n)$*.*

Theorem 2 NTIME$(T(n)) \subseteq \bigcup\{\text{DTIME}(c^{T(n)}) \mid c \geq 0\}$

Recall that a nondeterministic $T(n)$ time-bounded Turing machine M and an input x of length n determine a computation tree of depth at most $T(n)$, and that M accepts n only if one of the paths of the tree is an accepting path. Theorem 2 is simply a formal expression of the time it takes to execute a depth-first search of this tree.

The theorems just presented are proved by rather straightforward simulations. The next theorems involve deep recursions in their simulations. First we need a couple of technical definitions.

A function $S(n)$ is *space-constructible* if there is some Turing machine M that is $S(n)$ space-bounded, and for each n, there is some input of length n which uses exactly $S(n)$

cells. A function $S(n)$ is *fully space-constructible* if every input of length n uses $S(n)$ cells.

A function $T(n)$ is *time-constructible* if there is some Turing machine M that is $T(n)$ time-bounded, and for each n, there is some input of length n on which M runs for exactly $T(n)$ steps. A function $T(n)$ is *fully time-constructible* if for every input of length n, M runs for exactly $T(n)$ steps.

It is merely an exercise to see that ordinary arithmetic functions such as $\log(n)$, n^c, and c^n, are fully time-constructible and fully space-constructible.

Theorem 3 (Savitch [Sav70]) *If S is fully space-constructible and $S(n) \geq \log(n)$, then* $\text{NSPACE}(S(n)) \subseteq \text{DSPACE}(S^2(n))$.

This is a very important result. Observe that a standard depth-first search simulation, as in the previous theorem, would only provide an exponential upper-bound.

Corollary 1 $\text{PSPACE} = \bigcup\{\text{DSPACE}(n^c) \mid c \geq 0\} = \bigcup\{\text{NSPACE}(n^c) \mid c \geq 0\}$ *and* $\text{POLYLOGSPACE} = \bigcup\{\text{DSPACE}(\log(n)^c) \mid c \geq 0\} = \bigcup\{\text{NSPACE}(\log(n)^c) \mid c \geq 0\}$

For this reason, nondeterministic versions of PSPACE and POLYLOGSPACE were not defined as standard complexity classes.

Corollary 2 $\text{NSPACE}(n) \subseteq \text{DSPACE}(n^2)$ *and* $\text{NL} \subseteq \text{POLYLOGSPACE}$.

Theorem 4 ([Coo71a]) *If S is fully space-constructible and $S(n) \geq \log(n)$, then* $\text{NSPACE}(S(n)) \subseteq \bigcup\{\text{DTIME}(c^{S(n)} \mid c \geq 0\}$.

4.3 Relations between the Standard Classes

Figures 1 and 2 show the inclusion relations that emerge by application of the results just presented. In these figures, a complexity class C is included in complexity class D if there is a path from C to D reading upward.

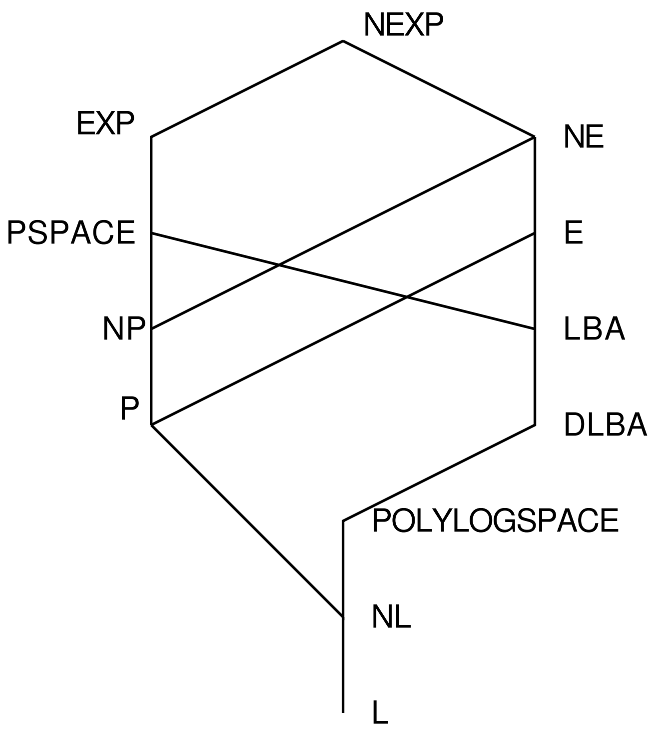

figure 1

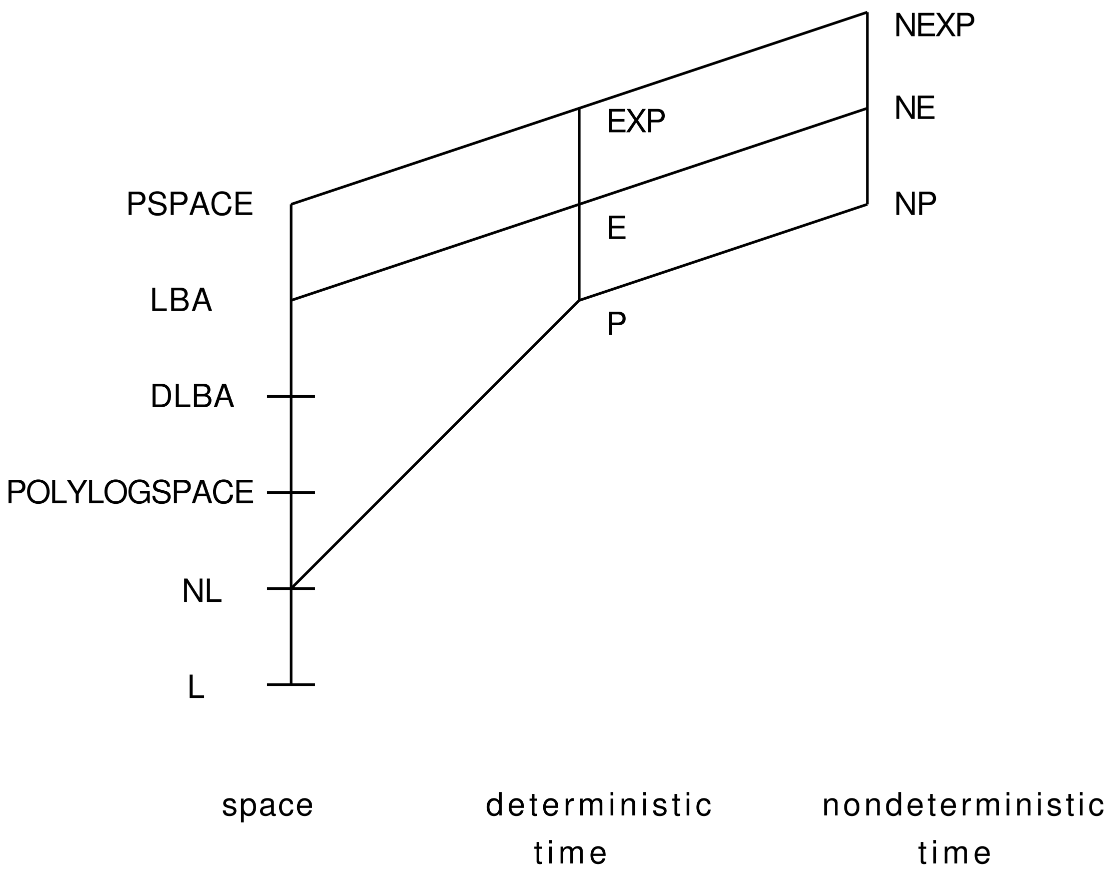

figure 2

The following consequences follow from Theorem 1: L $\subseteq$ P, PSPACE $\subseteq$ EXP, and DLBA $\subseteq$ E. By Theorem 2, we know that NP $\subseteq$ PSPACE. By Savitch's Theorem, LBA $\subseteq$ PSPACE, and (Corollary 2) NL $\subseteq$ POLYLOGSPACE. Theorem 4 is used to conclude that NL $\subseteq$ P and LBA $\subseteq$ E. All of the other inclusions in the figures are straightforward.

4.4 Separation Results

Now we consider which of these classes are the same and which are not equal. The following theorems are proved by diagonalization. Diagonalization is the technique invented by Cantor in order to show that the cardinality of the real numbers is greater than the cardinality of the integers.

The first theorem asserts that if two space bounds differ by even a small amount, then the corresponding complexity classes differ. The second theorem gives a separation result for time, but it requires that the time functions differ by a logarithmic factor. There are technical reasons for this. The intuition that makes results for time harder to obtain than for space is straightforward though. Time marches relentlessly forward, but space can be reused.

Theorem 5 ([HLS65]) *If S_2 is space-constructible, and*

$$\inf_{n\to\infty} S_1(n)/S_2(n) = 0,$$

then there is a language in $\mathrm{DSPACE}(S_2(n))$ *that is not in* $\mathrm{DSPACE}(S_1(n))$.

(This theorem was originally proved with the additional assumptions that $S_1(n) \geq \log(n)$ and $S_2(n) \geq \log(n)$. Sipser [Sip78] showed that the additional assumptions are unnecessary.) As consequences of this theorem, $\mathrm{L} \neq \mathrm{POLYLOGSPACE}$, $\mathrm{POLYLOGSPACE} \neq \mathrm{DLBA}$, and $\mathrm{DLBA} \neq \mathrm{PSPACE}$.

There is a very difficult to prove separation theorem for nondeterministic space from which it follows that LBA is a proper subset of PSPACE.

Theorem 6 ([HS65]) *If T_2 is fully time-constructible and*

$$\inf_{n\to\infty} T_1(n)\log(T_1(n))/T_2(n) = 0,$$

then there is a language in $\mathrm{DTIME}(T_2(n))$ *that is not in* $\mathrm{DTIME}(T_1(n))$.

Thus, $\mathrm{P} \neq \mathrm{EXP}$. Equality (or inequality) of all other inclusion relationships given in the figures is unknown. Amazingly, proper inclusion of each inclusion in the chain

$$\mathrm{L} \subseteq \mathrm{NL} \subseteq \mathrm{P} \subseteq \mathrm{NP} \subseteq \mathrm{PSPACE}$$

is an open question even though the ends of the chain are distinct (L $\neq$ PSPACE).

Similarly, equality of each inclusion in the chain

$$\mathrm{P} \subseteq \mathrm{NP} \subseteq \mathrm{PSPACE} \subseteq \mathrm{EXP}$$

is open, yet P $\neq$ EXP.

Also, it is not known whether any of the inclusions in the chain

$$\mathrm{DLBA} \subseteq \mathrm{LBA} \subseteq \mathrm{E} \subseteq \mathrm{NE}$$

are proper.

In addition to the results explicitly given here, there is a nondeterministic space hierarchy theorem due to Ibarra [Iba72] and a difficult nondeterministic time hierarchy theorem due to Cook [Coo73].

Book [Boo72, Boo76] has shown that none of the complexity classes POLYLOGSPACE, DLBA, and LBA is equal to either P or NP. He does this by analyzing the closure properties of these classes, and shows that they are different from one another. To this date, it is not known which of these classes contains a language that does not belong to the other.

5 Nondeterminism and NP-Completeness

Several different additions to the basic deterministic Turing machine model are often considered. These additions add computational power to the model and so allow us to compute certain problems more efficiently. Often these are important problems with seemingly no efficient solution in the basic model. The question then becomes whether the efficiency provided by the additional power is really due to the new model or whether the added efficiency could have been attained without the additional resources.

The original and most important example of this type of consideration is nondeterminism. For each of the standard nondeterministic complexity classes we have been considering, it is an open question whether the class is distinct from its deterministic counterpart.

Recall that a nondeterministic Turing machine is one with a multivalued transition function, and recall that such a machine M accepts an input word x if there some computation path of M on x that terminates in an accepting state. Let us assume, for each configuration of M, that there are exactly two possible next configurations, say c_0 and c_1. Then, M can be simulated in the following two-stage manner: (1) Write an arbitrary binary string on a work-tape of a deterministic Turing machine M'. (2) Given an input word x, M' deterministically executes the computation path of M on x that is determined by the binary-string written on its work-tape. That is, M' simultaneously simulates M and reads its binary word from left to right; if the symbol currently scanned is 0, then M' continues its simulation in configuration c_0, and if the symbol currently scanned is 1, then M' continues its simulation in configuration c_1. Clearly, there is a binary string r such that M' accepts x when r is written on its work-tape if and only if there is an accepting computation of M on x. Informally, stage (1) comprises a "guess" of a witness r to the fact that $x \in L(M)$, and stage (2) comprises a deterministic "verification" that r is a correct guess.

This "guessing" and "verifying" completely characterizes nondeterminism. Consider the problem (SAT) of determining, given a formula F of propositional logic, whether F is satisfiable. If F has n variables, there are 2^n different truth assignments. All known deterministic algorithms are equivalent to a sequential search of each of the assignments to see if one of them leads to satisfaction (i.e., evaluates to the truth value TRUE). Clearly, an exhaustive search algorithm for checking satisfaction takes 2^n steps for a formula of size n, thereby placing SAT into the complexity class E. However, the following nondeterministic algorithm for SAT reduces its complexity to polynomial time: Given an input formula F, (1) guess an assignment to the Boolean variables of F. This takes $O(n)$ steps. (2) Verify that the assignment evaluates to True. This takes $O(n \log(n))$ steps. Thus, SAT belongs to the class NP.

NP (the class of problems accepted in polynomial time by nondeterministic Turing machines) is the most important nondeterministic complexity class. NP plays a central role in complexity theory as many important problems from computer science and mathematics which are not known to be solvable deterministically in polynomial time are in NP. These include SAT, the graph Hamiltonian Circuit problem discussed earlier, various scheduling problems, packing problems, nonlinear programming, and hundreds of others. The most central and well-known open problem in complexity theory is whether P = NP.

The concept of NP-completeness plays a crucial role here as it gives a method of defining the most difficult NP problems. Intuitively, a problem A in NP is NP-complete if any problem in NP could be efficiently computed using an efficient algorithm for A as a subroutine.

Formally, A is NP-complete if (1) $A \in \mathrm{NP}$, and (2) for every $B \in \mathrm{NP}$, there is a function f that can be computed in polynomial time such that $x \in B$ if and only if $f(x) \in A$. (When property (2) holds for any sets B and A, we way B is *many-one reducible to A in polynomial time*, and we write $B \leq_m^{\mathrm{P}} A$.) This definition captures the intuitive notion: An efficient procedure for determining whether x belongs to B is to compute $f(x)$ and then input $f(x)$ to a subroutine for determining membership in A.

That NP-complete sets exist is not obvious and is due to Cook [Coo71b] who proved that SAT is NP-complete, and to Levin [Lev73], who independently obtained a variant of this result. Karp [Kar72] discovered a wide variety of NP-complete problems, and clearly demonstrated the importance of this new notion. These works opened the floodgates to the discovery of many more NP-complete problems in all areas of computer science and other computational disciplines.

The salient fact about NP-complete problems is that NP = P if and only if P contains an NP-complete problem. Thus, a single problem captures the complexity of the entire class.

6 Relative Computability

In this section we expand more broadly on the idea of using a subroutine for one problem in order to efficiently solve another problem. By doing so, we make precise the notion that the complexity of a problem B is related to the complexity of A – that there is an algorithm to efficiently accept B *relative to* an algorithm to efficiently accept A. Most generally, this should mean that an acceptor for B can be written as a program that contains subroutine calls of the form "$x \in A$," which returns TRUE if the Boolean test is true, and which returns FALSE, otherwise. The algorithm for accepting B is called a *reduction procedure* and the set A is called an *oracle*. The reduction procedure is *polynomial time-bounded* if the algorithm runs in polynomial time when we stipulate that only one unit of time is to be charged for the execution of each subroutine call. Placing faith in our modified Church's thesis and in Cobham's thesis, these ideas are made precise via the oracle Turing machine.

An *oracle* Turing machine is a Turing machine with a distinguished work tape called the oracle tape, and three special states Q, YES, and NO. When the Turing machine enters state Q the next state is YES or NO depending on whether or not the word currently written on the oracle tape belongs to the oracle set. In this way, if the machine is provided with an oracle A, it receives an answer to a Boolean test of the form "$x \in A$" in one move. Let M be an oracle Turing machine, let A be an oracle, and let T be a time complexity function. Oracle Turing machine M with oracle A is $T(n)$ *time-bounded* if for every input of length n, M makes at most $T(n)$ moves before halting. If M is a nondeterministic oracle Turing machine, then every computation of M with A on words of length n must make at most $T(n)$ moves before halting. The language accepted by M with oracle A is denoted $L(M, A)$.

6.1 The Polynomial Hierarchy

If B is accepted by a polynomial time-bounded oracle Turing machine with oracle A, then we write $B \in \mathrm{P}^A$. P^A is the class of sets acceptable in polynomial time relative to the set A. The class NP^A is defined analogously. P^{NP} is the class of sets accepted in polynomial time using oracles in NP. $\mathrm{NP}^{\mathrm{NP}}$ is the class of sets accepted in nondeterministic polynomial time using oracles in NP.

There is a hierarchy of complexity classes that lies between P and PSPACE that is generated by using more and more powerful oracles. This is called the *polynomial hierarchy* (cf. [Sto76, Wra76]). The class P forms the bottom of the hierarchy and NP lies on the first level. The second level, denoted Σ_2^{P}, is defined to be $\mathrm{NP}^{\mathrm{NP}}$. The third level, Σ_3^{P}, is $\mathrm{NP}^{\Sigma_2^{\mathrm{P}}}$. In this manner, the classes of the polynomial hierarchy are defined inductively. It is not known whether the polynomial hierarchy is a strict hierarchy, just as it is not known whether $\mathrm{P} \neq \mathrm{NP}$. However, researchers believe the polynomial hierarchy forms an infinite hierarchy of distinct complexity classes, and it serves as a useful classification scheme for combinatorial problems.

6.2 NP-Hardness

A set A is NP-*hard* if for all sets $B \in$ NP, $B \in \mathrm{P}^A$. Note that NP-hard sets are not required to belong to NP. NP-hard problems are as "hard" as the entire class of problems in NP, for if A is NP-hard and $A \in$ P, then P = NP.

Define a set B to be *Turing reducible to A in polynomial time* ($A \leq_{\mathrm{T}}^{\mathrm{P}} B$) if $B \in \mathrm{P}^A$. Then, it is interesting to compare $\leq_{\mathrm{T}}^{\mathrm{P}}$ with $\leq_m^{\mathrm{P}}$. (Recall that we needed the latter in order to define NP-complete sets.) The relations $\leq_{\mathrm{T}}^{\mathrm{P}}$ and $\leq_m^{\mathrm{P}}$ are called *reducibilities*, for they reduce the complexity of one problem to that of another. The reducibilities "$B \leq_m^{\mathrm{P}} A$" and "$B \leq_{\mathrm{T}}^{\mathrm{P}} A$" are not the same. They are known to differ on sets within the complexity class E [LLS75]. Define a set A to be $\leq_m^{\mathrm{P}}$-*hard* for NP if for all sets $B \in$ NP, $B \leq_m^{\mathrm{P}} A$. Under the assumption P $\neq$ NP, one can prove the existence of an NP-hard set A that is not $\leq_m^{\mathrm{P}}$-hard for NP [SG77]. Furthermore, NP-hard problems are known that (1) seem not to belong to NP, and (2) no $\leq_m^{\mathrm{P}}$-reduction from problems in NP is known.

6.3 Complete Problems for Other Classes

Completeness is not a phenomenon that applies only to the class NP. For each of the complexity classes in our list of standard classes, it is possible to define appropriate reducibilities and to find complete problems in the class with respect to those reducibilities. That is, there are problems that belong to the class such that every other problem in the class is reducible to them. A problem that is complete for a class belongs to a subclass (that is closed under the reducibility) if and only if the entire classes are the same. Interestingly, each of the sample problems given at the end of Section 3 are complete for the classes to which they belong. So, for example, the graph accessibility problem is complete for NL, and this problem belongs to L if and only if L = NL.

7 Parallelism

It should be clear to the reader by now that there is a well-developed complexity theory of sequential computation, and in subsequent sections we will return to this development. It is clear that the multitape Turing machine is the correct robust model and that time and space are the main resources. Here we consider the theory of synchronous highly parallel computations.

It is becoming evident that the powerful machines of the future will be parallel. VLSI chip technology will surely make it possible to connect together large numbers of processors to operate together synchronously. The question, of course, is what can be done with the result.

Several models of parallel computation have been proposed, and even though each model can simulate the other without much loss, the issue of "correct" model is not quite as settled as it is with sequential computation. The vector machine, which is considered by Pratt and Stockmeyer [PS78] is like a random access machine, but it can operate on vectors in one step. Other variations of random access machines have been considered by Hartmanis and Simon [HS74], Goldschlager [Gol78] (who considers the SIMDAG, a single instruction stream, multiple data stream, machine, which is based on a RAM that allows concurrent reads and writes in shared random access memory), and others. Fortunately, all of these devices can simulate one another with no more than a cubic increase in computation time.

The complexity measures that have emerged as most important for parallel computation are *parallel time* and *amount of hardware* (roughly speaking, a measure of the number of processors and amount of circuitry). The important discovery made by the researchers listed above, and further supported by results for yet other models of parallel computation by Borodin [Bor77], Chandra and Stockmeyer [CS76], and Kozen [Koz76], is that parallel time $T^{O(1)}$ is equivalent to sequential space $T^{O(1)}$. This phenomenon is called the "parallel computation thesis." It asserts that this relationship between parallel time and sequential space should hold for any reasonable model of parallel computation. As a particular consequence, the class of sets recognizable in polynomial time by parallel computers is PSPACE. Since PSPACE is thought to be a much larger class of problems than P, theorems that support the parallel computation thesis quantify the effective improvement made possible by parallelism.

7.1 Circuits

Uniform families of circuits provide a very simple and very effective model of parallel computation. In fact, since real computers are built from circuits, this is reason enough to consider circuit complexity.

A combinational (Boolean) circuit is a labeled, acyclic, directed graph. Nodes with in-degree $= 0$ are called *input nodes*. There is one node with out-degree $= 0$ called the *output node*. Interior nodes (i.e. noninput nodes) represent logical gates; they are labeled with AND, inclusive OR, or NOT. Arbitrary fan-out is allowed. The constants "true" and "false" are the allowed inputs. By associating true with 1 and false with 0, every Boolean circuit with n input nodes realizes a function $f : \{0,1\}^n \mapsto \{0,1\}$. When the input nodes receive their values, every interior value receives the value 0 or 1 in accordance with the logical gate that the interior node represents. If $x = x_1 x_2 \dots x_n$ is a string in $\{0,1\}^n$ and C is a circuit with n input nodes, we say C *accepts* x if C outputs $1 =$ true when x is the input.

Let $A \subseteq \{0,1\}^*$. Let A^n denote the subset of A consisting of all strings in A of length n. We say A^n is *realized by* C if for all strings x of length n, C accepts x if and only if x belongs to A.

In the realm of complexity theory as we have been describing it, one machine is expected to serve for inputs of all lengths. This machine (namely, a Turing machine) is "uniform;" it is finitely describable, and can compute only a computable function. A circuit however serves only for inputs of one length. Therefore, it is not a single circuit that corresponds to a machine, but a family of circuits, one for each length. We say that a family of circuits $\{C_n\}$ accepts a set A, $A \subseteq \{0,1\}^*$ if for each n, the circuit C_n accepts A^n. Thus, a family of circuits is "nonuniform;" it requires an infinite description, and nonrecursive sets can be accepted by families of circuits. This disparity will be addressed shortly.

The size of a circuit C, $s(C)$, is the number of interior nodes or gates, and the depth of C, $d(C)$, is the length of the longest path in C. For functions S and T, define

$$\begin{aligned}\mathrm{SIZE}(T(n)) &= \{A \mid \text{there is a family of circuits } \{C_n\} \text{ that accepts } A \\ &\quad\ \text{and } s(C_n) = O(T(n))\} \\ \textit{and} & \\ \mathrm{DEPTH}(S(n)) &= \{A \mid \text{there is a family of circuits } \{C_n\} \text{ that accepts } A \\ &\quad\ \text{and } d(C_n) = O(S(n))\}.\end{aligned}$$

Depth corresponds to parallel time. To wit, a proof that $A \notin \mathrm{DEPTH}(S(n))$ means that no parallel computer can compute A in time $O(S)$. This is because the parallel computation could be unraveled to form a circuit with constant delay at each gate. Thus, circuit depth is a lower bound on parallel time.

Since circuits can recognize nonrecursive sets, circuit depth cannot provide an upper bound for parallel time until uniformity is introduced. Let $\mathbf{C}$ denote any reasonable encoding of a circuit C as a binary string. Borodin [Bor77] defines a family $\{C_n\}$ of circuits to be *uniform* if some deterministic Turing machine can compute the transformation $1^n \mapsto \mathbf{C_n}$ in space $O(\log\ s(C_n))$. Other definitions of uniform have been proposed, but this is the simplest to exposit, is robust, and in most cases is very useful (cf. [Ruz81]). Now we define the uniform complexity classes.

$$\begin{aligned}\mathrm{USIZE}(T(n)) &= \{A \mid \text{there is a uniform family of circuits } \{C_n\} \text{ that accepts } A \\ &\quad\ \text{and } s(C_n) = O(T(n))\} \\ \textit{and} & \\ \mathrm{UDEPTH}(S(n)) &= \{A \mid \text{there is a uniform family of circuits } \{C_n\} \text{ that accepts } A \\ &\quad\ \text{and } d(C_n) = O(S(n))\}.\end{aligned}$$

Borodin [Bor77] proves the following relationships between USIZE and UDEPTH and Turing machine (i.e. sequential) time and space.

Theorem 7 ([Bor77]) *If $\lfloor \log\ T \rfloor$ is fully space constructible, then*

$$\mathrm{USIZE}(T^{O(1)}) = \mathrm{DTIME}(T^{O(1)}).$$

If S is fully space constructible, then

$$\mathrm{UDEPTH}(S^{O(1)}) = \mathrm{DSPACE}(S^{O(1)}).$$

The second assertion is an instance of the parallel computation thesis. As a consequence of the first assertion, the complexity class P can be characterized as USIZE($n^{O(1)}$). It has been known that P $\subseteq$ SIZE($n^{O(1)}$) ([Sav76, FP74]). The proofs of the assertions in one direction require careful circuit constructions of bounded Turing machine computations. In the other direction, on an input of length n, a Turing machine simulation of circuits uses uniformity to efficiently find the circuit that serves for inputs of length n.

7.2 Simultaneous Resource Bounds

Consider multitape Turing machines that simultaneously are T time-bounded and S space-bounded. [This should not be confused with the intersection of DTIME($T(n)$) and DSPACE($S(n)$). A set in the intersection is accepted by a T time-bounded Turing machine and by an S space-bounded Turing machine. In the current discussion, however, we insist that the same Turing machine satisfy both bounds.] In accordance with the previous theorem, Borodin [Bor77] asked whether simultaneous Turing machine time and space bounds correspond to uniform families of circuits that are simultaneously size and depth bounded. For example, we have already noted that P = USIZE($n^{O(1)}$). Similarly, as a consequence of the second assertion in this theorem, POLYLOGSPACE = UDEPTH($n^{O(1)}$). Are the sets that are accepted by Turing machines that simultaneously satisfy the constraints on the left hand side of these equations (polynomial time-bounded, and $(log\ n)^k$, for some constant k, space- bounded) the same as those accepted by uniform families of circuits that satisfy both conditions on the right hand side (polynomial size-bounded, and $(log\ n)^k$, for some constant k, depth-bounded)? Here we digress to relate a human touch: Steve Cook named the latter class (polynomial size and poly-log depth-bounded uniform families of circuits) NC in honor of Pippenger, who obtained an important characterization of this class. The former class (Turing machine polynomial time and poly-log space) was studied by Cook in [Coo79] and subsequently named SC for "Steve's Class" by Nick Pippenger.

Well, it is not known whether SC = NC. Both classes contain important computational problems. For example, both classes contain all deterministic context-free languages and all context-free languages are in NC. The conjecture is that context-free languages are not in SC. [Coo79, Ruz81].

NC remains an object of intense study. This class seems to encompass those problems for which there are very rapid parallel algorithms that can be efficiently constructed. Furthermore, NC is robust under changes of model and under different definitions of circuit uniformity [Pip79, Ruz81].

The most important open question in this area is whether P = NC. Can every feasibly computable problem be parallelized to advantage? The answer is believed to be no. Linear programming is a typical problem that is P-complete, and for this reason is not parallelizable unless P = NC. The articles [Coo83, Coo85] by Cook give many examples of useful problems in NC and examples of problems that are unlikely to be in NC.

8 Probabilistic Complexity Classes

Randomness is another resource that can be added to computations. Probabilistic algorithms allow computations to depend on the outcome of an ideal random generator (i.e., on unbiased coin tosses). Such algorithms can be classified depending on what languages the algorithms are defined to accept. Important computational problems that seem to be infeasible by ordinary deterministic computations have efficient solutions using probabilistic algorithms. Such algorithms can be easily implemented, and fast and reliable solutions to otherwise difficult problems are then possible.

In order to precisely define probabilistic complexity classes, we need the concept of a probabilistic Turing machine. A *probabilistic* Turing machine is just a nondeterministic Turing machine, but acceptance is defined differently. Each nondeterministic choice is considered as a random experiment in which each outcome has equal probability. We may assume that each nondeterministic branch has exactly two possible outcomes, so that each has possibility 1/2. A probabilistic Turing machine has three kinds of final states, accepting or a-states, rejecting or r-states, and undetermined or ?-states. The outcome of the machine on an input is now a random variable whose range is $\{a, r, ?\}$.

We let $Pr[M(x) = y]$ denote the probability that machine M on input x halts in a y-state. Note that the probability of a given nondeterministic path is obtained by raising 1/2 to a power equal to the number of nondeterministic choices along it. The probability that M accepts an input x, $Pr[M(x) = a]$, is the sum of the probabilities of all accepting paths.

We can now define several different probabilistic complexity classes. (These classes were originally defined by Gill [Gil77] and by Adleman and Manders [AM77].) Each consists of languages accepted by restricting the machines to polynomial time. The first class, PP, for probabilistic time, is the easiest to define, but is the least useful. Let χ_A denote the characteristic function of A. PP is the class of all languages for which there is a probabilistic, polynomial time-bounded Turing machine M such that for all x,

$$Pr[M(x) = \chi_A(x)] > 1/2.$$

That is,

$$x \in A \rightarrow Pr[M(x) = a] > 1/2,$$

and

$$x \notin A \rightarrow Pr[M(x) = r] > 1/2.$$

These Turing machines are of the "Monte Carlo" type. They are allowed to lie (i.e. terminate in an r-state when $x \in L$, or terminate in an a-state, when $x \notin L$) with small probability. On the other hand, "Las Vegas" algorithms may terminate with "?" with small probability, but they are not allowed to lie.

One would like to increase the reliability of a probabilistic Turing machine by repeating its computations a large number of times and giving as output the majority result. This is not possible with PP, and this is why there is no practical interest in this class. On a technical note, NP $\subseteq$ PP, and PP $\subseteq$ PSPACE. (All the probabilistic complexity classes described in this section are subsets of PSPACE, for we know already that in PSPACE we can simulate all computations of any nondeterministic polynomial time-bounded Turing machine.) It is not known whether PP is contained in the polynomial hierarchy.

The error probability is the probability that a probabilistic machine gives the wrong answer on a given input. The next class we define bounds the error probability away from 1/2, and that restriction makes it possible to increase reliability. The class BPP, for bounded error probabilistic polynomial time, is the class of all languages for which there is a probabilistic, polynomial time-bounded Turing machine M and a number $\epsilon, 0 < \epsilon < 1/2$, such that for all x,

$$Pr[M(x) \neq \chi_A(x)] < \epsilon.$$

This class is still of the Monte Carlo type. Obviously, P $\subseteq$ BPP $\subseteq$ PP. It is known that BPP $\subseteq \Sigma_2^{\mathrm{P}}$ [Sip83, Lau83]. Rabin [Rab76] and Solovay and Strassen [SS77] have shown that testing integers for primality belongs to BPP. (We state the result this way for expository ease. In fact, they showed that the complement of primality belongs to R, from which membership in BPP follows because the complement of every set in BPP also belongs to BPP.)

Next is an interesting one-sided version of BPP. The class R, for random polynomial time, is the class of all languages for which there is a probabilistic, polynomial time-bounded Turing machine M such that for all x,

$$x \in A \rightarrow Pr[M(x) = a] > 1/2,$$

and

$$x \notin A \rightarrow Pr[M(x) = a] = 0.$$

Let $A = L(M) \in$ R, and consider a Turing machine M_k that iterates the random experiment $M(x)$ for k independent trials, and that halts in an a-state if $M(x) = a$ in at least one trial. Then,

$$x \in A \rightarrow Pr[M_k(x) = a] > 1 - (1/2)^k,$$

and

$$x \notin A \rightarrow Pr[M_k(x) = a] = 0.$$

Select any number ϵ, $0 < \epsilon < 1/2$. Then, there is a k such that $1 - (1/2)^k > 1/2 + \epsilon$. *Voilá*, for this choice of k,

$$x \in A \rightarrow Pr[M_k(x) = a] > 1 - \epsilon,$$

and

$$x \notin A \to Pr[M_k(x) = a] = 0.$$

Thus, the error probability for languages in R can be made arbitrarily small; or, the reliability can be made arbitrarily large. Indeed, we just proved that R $\subseteq$ BPP. Also, it is obvious that P $\subseteq$ R $\subseteq$ NP.

For any class of sets $\mathcal{C}$, co-$\mathcal{C}$ denotes the collection of all set whose complements belong to $\mathcal{C}$. The class ZPP, for zero error probabilistic polynomial time, is defined to be R $\cap$ co-R. Explicitly, A is in ZPP if and only if there is a probabilistic, polynomial time-bounded Turing machine M such that for all x,

$$x \in A \to Pr[M(x) = r] = 0 \text{ and } Pr[M(x) = a] > 1/2,$$

and

$$x \notin A \to Pr[M(x) = a] = 0 \text{ and } Pr[M(x) = r] > 1/2, .$$

ZPP-algorithms are of the Las Vegas type. Obviously, P $\subseteq$ ZPP $\subseteq$ R. None of the inclusion relationships expressed in this section are known to be proper. These remain open questions.

BPP is nearly as feasible a class as is P. Since the error can be made arbitrarily small by iterating a large number of times (we showed this for R, and a similar result holds for BPP), BPP-algorithms can be used in practice with great reliability. A lovely result, whose proof uses this fact, makes the same point. Namely, BPP $\subseteq$ SIZE($n^{O(1)}$) [Adl78, BG81, Sch86].

8.1 Public-Key Cryptography

Though a thorough description of recent work in cryptography would require as much space as does this entire article, we would be remiss not to mention this exciting application of complexity theory. Cryptography is the art of providing secure communication over insecure channels. Twentieth century cryptography has been based on Shannon's pioneering work in Information Theory [Sha48]. The general idea has been to design cryptosystems that are secure because cryptanalysts reading coded messages have not enough *information* to decode them. A draw-back of such systems is that communication is possible only after two users agree on a common encoding and decoding key that first must be exchanged over a secure system. Modern world-wide communications systems and global economy make it desirable for parties to be able to communicate securely (for example, to arrive at a contractual agreement) without costly prior exchange of secret information. In 1976, Diffie and Hellman defined the notion of public-key cryptography, which makes secure communication over insecure channels conceivable for totally unacquainted parties. Each user A of a public-key cryptosystem generates a pair of keys, (E_A, D_A). E_A is an encoding key that is published in a directory for other users to use for encoding of messages intended for user A, and D_A is kept private. Several such systems have been proposed. They depend on the existence of so-called "trap-door" functions. These are functions that can be computed efficiently (in polynomial time), that can be inverted efficiently with the assistance of the the proper decoding key D_A, but that require exponential time to invert otherwise. In this manner, security of public-key cryptography is based on complexity theory. To repeat, the general idea is to design cryptosystems that are secure because cryptanalysts reading coded messages have not enough *computational resources* to decode them. Such systems are completely insecure from the Shannon point of view.

The most successful implementation of a public-key cryptosystem is the RSA system [RSA78] due to Rivest, Shamir, and Adleman. This system is inspired by the expectation that integer factorization is infeasible. Rabin [Rab79] has invented a variant of RSA whose security is equivalent to the complexity of the integer factorization problem.

Newer implementations of public-key cryptosystems [GM84, BG85] use probabilistic algorithms to encode single bits of a message. (The coded message is the concatenation of the encoded bits.) In these systems, the same cleartext message can give rise to many different coded messages, so a cryptanalyst, having a coded message and a candidate for the correct cleartext message, could not verify that his guess is correct by enciphering it. Security of these cryptosystems also depends on intractability of certain number theoretic problems.

Security of each of the public-key cryptosystems that have been proposed depends on assumed intractability of a specific computational (usually, number-theoretic) problem. Intractability of these problems remains open. Independent of these assumptions, public-key cryptosystems cannot be proved to exist until several still open problems in complexity theory are resolved. (The deep relations between cryptography and complexity theory can be found in [GS88], and in the tutorial [Sel89].)

9 Structural Complexity Theory

Structural complexity theory is concerned with many of the issues that have already come into focus, but attempts to gain deeper insight by delving further into underlying mathematical properties and relations. We conclude this article with an exposition that by example will provide an introduction to topics in this area.

9.1 The Isomorphism Conjecture

It is a remarkable phenomenon that hundreds upon hundreds of important combinatorial problems fall into just a few (seemingly) distinct equivalence classes of complexity: P-complete, NP-complete, PSPACE-complete, and a few others. In 1977, Berman and Hartmanis conjectured that an even tighter relationship may hold for many of these problems. They suggested that all NP-complete sets are essentially different encodings of the very same set. To make this precise, define a function f to be a *p-isomorphism* if it is one-one, onto, and f and its inverse f^{-1} are both computable in polynomial time. Then, two sets A and B are said to be *p-isomorphic* if there is a p-isomorphism f such that for all x, $x \in A \leftrightarrow f(x) \in B$. Berman and Hartmanis [BH77] showed that all NP-complete problems that were known at the time are p-isomorphic. They conjectured that *all* NP-complete sets are p-isomorphic. To date, no counter-example has been shown. There is no known time or space complexity class for which the question of whether the $\leq_m^{\mathrm{P}}$-complete sets of the class are all p-isomorphic has been answered. Nevertheless, this conjecture has stimulated very deep work. For example, Kurtz, Mahaney, and Royer [KMR88] show the existence of sets with exponential time complexity that are $\leq_m^{\mathrm{P}}$-reducible to one-another, but that are not p-isomorphic, and use a very intricate diagonalization to show the existence of a set A with exponential complexity such that all sets $\leq_m^{\mathrm{P}}$-equivalent to A are also p-isomorphic to A.

If all NP-complete sets are isomorphic, then $\mathrm{P} \neq \mathrm{NP}$. To see this, recall that if $\mathrm{P} =$

NP, then all sets in P are NP-complete. Thus, under this supposition, there exist finite NP-complete sets and there exist infinite NP-complete sets. But, finite sets cannot be p-isomorphic to infinite sets, so we conclude that not all NP-complete sets are isomorphic. Starting with this observation, Hartmanis and Berman considered the *density* of sets. For any set A, let c_A be the function defined on natural numbers so that for every n, $c_A(n)$ is the number of strings in A of length n. (c_A is called the *census* function of A.) A set A is called *sparse* if there is a polynomial p such that for all n, $c_A(n) \leq p(n)$. In contrast, a set A is *dense* if its census function grows faster than every polynomial. It is easy to see that typical NP-complete sets such as the satisfiability problem are dense, and it is easy to see that sparse sets cannot be p-isomorphic to dense sets. Thus, it is natural to inquire whether there exist sparse NP-complete sets. This question was solved by Mahaney [Mah82] who showed that sparse NP-complete sets exist if and only if P = NP.

9.2 Complements of Complexity Classes

It is always important to know whether complexity classes are closed under natural operations. (Recall the results listed at the end of Section 4 — several complexity classes are distinguished by showing that their closure properties differ.) Here we discuss closure under complements. The complement of a language L over the finite alphabet Σ is the language $\overline{L} = \Sigma^* - L$. For any complexity class $\mathcal{C}$, $co - \mathcal{C} = \{\overline{L} \mid L \in \mathcal{C}\}$.

It is easy to see that DTIME($T(n)$) = co-DTIME($T(n)$), for any time-bound $T(n)$. If M is a deterministic $T(n)$ time-bounded Turing machine, we can easily define a deterministic $T(n)$ time-bounded N that accepts $\overline{L(M)}$. N simply accepts if and only if M halts in a nonaccepting state, but in every other respect N behaves exactly as M.

The above trick of reversing the roles of accepting and nonaccepting states is insufficient for showing that deterministic space-bounded complexity classes are closed under complements, because space- bounded Turing machines may enter infinite loops. In section 4.2, we noted that every $S(n)$ space-bounded Turing machine, where $S(n) \geq \log(n)$, is equivalent to one that halts on every input. Sipser [Sip78] has shown this to be true even for smaller space bounds. Thus, it follows that DSPACE($S(n)$) = co-DSPACE($S(n)$), for all S.

It is not known whether nondeterministic time-bounded complexity classes are closed under complements. Let us consider NP as a typical instance.

Theorem 8 *Co*-NP $=$ NP *if and only if the complement of some* NP*-complete problem is in* NP*.*

In one direction the proof is trivial. To prove the converse, we begin with two key observations, whose proofs we leave as exercises for the reader: (1) for all sets L and A, if $L \leq_m^{\mathrm{P}} A$, then $\overline{L} \leq_m^{\mathrm{P}} \overline{A}$; (2) If $L \leq_m^{\mathrm{P}} A$ and $A \in \mathrm{NP}$, then $L \in \mathrm{NP}$. Let A be an NP-complete set whose complement is in NP, and let L be an arbitrary language in NP. We need to show that $\overline{L} \in \mathrm{NP}$. By definition, $L \leq_m^{\mathrm{P}} A$, so by (1), $\overline{L} \leq_m^{\mathrm{P}} \overline{A}$. Since we are assuming that $\overline{A} \in \mathrm{NP}$, the conclusion follows from (2).

In particular, co-NP $=$ NP if and only if the complement of SAT is in NP. A formula F is not satisfiable if and only if the negation of F is a tautology. Thus, we conclude that NP is closed under complements if and only if testing for tautologies is in NP. Of course, this remains an open problem.

Some long-standing important open questions do get solved. In 1987, Immerman [Imm88] and Szelepcsényi [Sze87] independently proved that nondeterministic $S(n)$ space-bounded complexity classes are closed under complements, for $S(n) \geq \log(n)$. It immediately follows that context-sensitive languages are closed under complements, thus settling a question raised by Kuroda in 1964 [Kur64]. The proof is quite sophisticated. One component of Immerman's proof is to nondeterministically count the number of accepting configurations that are reachable from an initial configuration. To this extent, his proof technique is similar to the proof of Mahaney's result mentioned above, for the latter involves showing that it is possible to nondeterministically counts the census function of a sparse set. In both proofs nondeterministic procedures for counting are key ingredients.

9.3 Oracles

In Section 6 of this article we introduced oracle Turing machines and defined, for any oracle A, the relativized classes P^A and NP^A. Baker, Gill, and Solovay in 1975 [BGS75] constructed specific oracles A and B with the properties that $\mathrm{P}^A = \mathrm{NP}^A$ and $\mathrm{P}^B \neq \mathrm{NP}^B$. Their results neither prove that $\mathrm{P} = \mathrm{NP}$ nor that $\mathrm{P} \neq \mathrm{NP}$. But, their results do have an interesting albeit informal interpretation. Proof techniques for proving that two classes differ usually relativize for all oracles, and simulation techniques that are used to prove two classes identical tend to relativize for all oracles. Thus, we might conclude on the basis of the Baker, Gill, and Solovay results, that new techniques will have to be invented before the P =? NP question can be settled.

These first oracle results led to a flurry of activity in the construction of oracles. Whenever an hypothesis can be relativized one way to make it true and another way to make it false, the informal argument that current proof techniques are unable to settle the question remains quite forceful. Oracle techniques have been applied to nearly all the issues raised in this article in order to assist in delineating the methods which might lead toward their resolution. For example, for each of the inclusion relations that hold between the standard complexity classes for which equality is not known, oracles have been constructed which make the classes identical and which make them different. Construction of such oracles can be a technically difficult enterprise. Among the most intricate oracle constructions, we cite those of Yao [Yao85] and Hastad [Has86] who obtained oracles relative to which the polynomial hierarchy is infinite.

Some questions concerning oracles remain open. For example, no oracle is known relative to which all NP-complete sets are isomorphic. [HS89] contains an oracle relative to which all complete sets for Σ_2^{P}, the second level of the polynomial hierarchy, are p-isomorphic and relative to which it not all complete sets for Σ_2^{P} are isomorphic, thus making Σ_2^{P} the only complexity class for which this problem is known to relativize both positively and negatively.

Limitations to this methodology are known also. For example, Long and Selman [LS86] showed that the polynomial hierarchy collapses if and only if there exists a sparse oracle relative to which the polynomial hierarchy collapses. In particular, they show that $\mathrm{P} = \mathrm{NP}$ if and only if there is an oracle T over a one letter alphabet (a tally language) such that $\mathrm{P}^T = \mathrm{NP}^T$. Thus, settling the oracle question for tally languages would settle the P =? NP question.

10 Bibliographic Remarks

The text book by Hopcroft and Ullman [HU79] contains proofs of most of the results surveyed in the first six sections, and so provides a fine introduction to complexity theory as well as to other areas of theoretical computer science. The recent text by Balcázar, Diaz, and Gabarró [BDG88] covers many topics in structural complexity theory. Garey and Johnson have authored the classic source on the theory of NP-completeness, and the appendix to this book lists hundreds of NP-complete problems. We highly recommend the expository article on synchronous parallel computation by Cook [Coo81]. Brassard has written a tutorial monograph on modern cryptography [Bra88]. Detailed expository articles on areas of complexity theory can be found in [Har89] and in the forthcoming [Sel90]. Research papers are found in the several scientific journals of computer science, and frequently appear first in preliminary form in the proceedings of the major conferences on theory of computing.

11 Acknowledgements

The authors are most grateful to Allan Borodin, Ken Regan, Jeff Shallit, and Peter van Emde Boas for their comments and their corrections. Errors that remain are entirely the author's doing.

References

[Adl78] L. Adleman. Two theorems on random polynomial time. In *Proc. 19th IEEE Symp. on Foundations of Computer Science*, pages 75–83, 1978.

[AM77] L. Adleman and K. Manders. Reducibility, randomness, and intractability. In *Proc. 9th ACM Symp. Theory of Computing*, pages 151–163, 1977.

[BDG88] J. Balcázar, J. Diaz, and J. Gabarró. *Structural Complexity I*. Springer-Verlag, Berlin, 1988.

[BG70] R. Book and S. Greibach. Quasi-realtime languages. *Mathematical Systems Theory*, 4:97–111, 1970.

[BG81] C. Bennett and J. Gill. Relative to a random oracle A, $\mathrm{P}^A \neq \mathrm{NP}^A \neq \text{Co-NP}^A$ with probability 1. *SIAM J. Comput.*, 10(1):96–113, February 1981.

[BG85] M. Blum and S. Goldwasser. An efficient probabilistic public-key encryption scheme which hides all partial information. In *Proceedings of CRYPTO 84*, pages 289–299. Springer-Verlag, 1985.

[BGS75] T. Baker, J. Gill, and R. Solovay. Relativizations of the P =? NP question. *SIAM J. Comput.*, 4(4):431–441, Dec. 1975.

[BH77] L. Berman and H. Hartmanis. On isomorphisms and density of NP and other complete sets. *SIAM J. Comput.*, 6:305–322, 1977.

[Boo72] R. Book. On languages accepted in polynomial time. *SIAM J. Comput.*, 1(4):281–287, 1972.

[Boo76] R. Book. Translational lemmas, polynomial time, and $(logn)^j$-space. *Theor. Comput. Sci.*, 1:215–226, 1976.

[Bor77] A. Borodin. On relating time and space to size and depth. *SIAM J. Comput.*, 6:733–744, 1977.

[Bra88] G. Brassard. *Modern Cryptography*, volume 325 of *Lecture Notes in Computer Science*. Springer-Verlag, New York, 1988.

[Chu36] A. Church. An unsolvable problem of elementary number theory. *Amer. J. of Math.*, 58:345–363, 1936.

[Cob64] A. Cobham. The intrinsic computational difficulty of functions. In *Proc. 1964 International Congress for Logic, Methodology, and the Philosophy of Science*, pages 24–30. North Holland, 1964.

[Coo71a] S. Cook. Characterizations of pushdown machines in terms of time-bounded computers. *J. ACM*, 19:175–183, 1971.

[Coo71b] S. Cook. The complexity of theorem-proving procedures. In *Proc. 3rd ACM Symp. Theory of Computing*, pages 151–158, 1971.

[Coo73] S. Cook. A hierarchy for nondeterministic time complexity. *J. Computer and Systems Sciences*, 7(4):343–353, 1973.

[Coo79] S. Cook. Deterministic CFL's are accepted simultaneously in polynomial time and log squared space. In *Proc. 11th Annual ACM Symp. on Theory of Comput.*, pages 338–345, 1979.

[Coo81] S. Cook. Towards a complexity theory of synchronous parallel computation. *L'Enseignement Mathematique*, XXVII:99–124, 1981.

[Coo83] S. Cook. The classification of problems which have fast parallel algorithms. In *Foundations of Computation Theory, Proc., 1983, Lecture Notes in Computer Science*, volume 158, pages 78 –93. Springer Verlag, 1983.

[Coo85] S. Cook. A taxonomy of problems with fast parallel algorithms. *Information and Control*, 64:2–22, 1985.

[CS76] A. Chandra and L. Stockmeyer. Alternation. In *Proc. 17th IEEE Symp. on Foundations of Computer Science*, pages 98–108, 1976.

[ET76] S. Even and R. Tarjan. A combinatorial problem which is complete in polynomial space. *J. Assoc. Comput. Mach.*, 23:710–719, 1976.

[FP74] M. Fischer and N. Pippenger. M. J. Fischer lectures on network complexity. Universität Frankfurt, 1974.

[Gil77] J. Gill. Computational complexity of probabilistic Turing machines. *SIAM J. Comput.*, 6(4):675–695, Dec. 1977.

[GM84] S. Goldwasser and S. Micali. Probabilistic encryption. *J. Comp. System Sci.*, 28:270–299, 1984.

[Gol78] L. Goldschlager. A unified approach to models of synchronous parallel machines. In *Proc. 11th Annual ACM Symp. on Theory of Comput.*, pages 89–94, 1978.

[GS88] J. Grollmann and A. Selman. Complexity measures for public-key cryptosystems. *SIAM J. Comput.*, 17(2), April 1988.

[Har89] J. Hartmanis, editor. *Computational Complexity Theory*, volume 38 of *Procs. of Symposia in Applied Mathematics.* Amer. Math. Society, 1989.

[Has86] J. Hastad. Almost optimal lower bounds for small depth circuits. In *Proc. 18th Annual ACM Symp. on Theory of Computing*, pages 6–20, 1986.

[HLS65] J. Hartmanis, P. Lewis, and R. Stearns. Hierarchies of memory limited computations. In *Proc. Sixth Annual IEEE Symp. on Switching Circuit Theory and Logical Design*, pages 179–190, 1965.

[HS65] J. Hartmanis and R. Stearns. On the computational complexity of algorithms. *Trans. Amer. Math. Soc.*, 117:285–306, 1965.

[HS74] J. Hartmanis and J. Simon. On the power of multiplication in random access machines. In *Proc. 15th Annual IEEE Symp. on Switching and Automata Theory*, pages 13–23, 1974.

[HS89] S. Homer and A. Selman. Oracles for structural properties: The isomorphism problem and public-key cryptography. In *Proc. Structure in Complexity Theory Fourth Annual Conference*, pages 3–14. IEEE Computer Society Press, 1989.

[HU79] J. Hopcroft and J. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, Reading, Massachusetts, 1979.

[Iba72] O. Ibarra. A note concerning nondeterministic tape complexities. *J. Comput. and Systems Sciences*, 19(4):609–612, 1972.

[Imm88] N. Immerman. Nondeterministic space is closed under complement. In *Proc. 3rd IEEE Struct. Complexity Theory Conf.*, pages 112–115, 1988.

[Jon73] N. Jones. Reducibility among combinatorial problems in log n space. In *Proc. 7th Ann. Princeton Conf. on Information Sciences and Systems*, pages 547–551, Princeton, NJ, 1973. Department of Electrical Engineering, Princeton University.

[Jon75] N. Jones. Space-bounded reducibility among combinatorial problems. *J. Comput. System Sci.*, 11:68–85, 1975.

[JS74] N. Jones and A. Selman. Turing machines and the spectra of first-order formulas. *J. Symbolic Logic*, 29:139–150, 1974.

[Kar72] R. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, pages 85–104. Plenum Press, New York, 1972.

[KMR88] S. Kurtz, S. Mahaney, and J. Royer. Collapsing degrees. *J. of Computer and System Sciences*, 37:247–268, 1988.

[Koz76] D. Kozen. On parallelism in Turing machines. In *Proc. 17th Annual IEEE Symp. on Foundations of Computer Science*, pages 89–97, 1976.

[Kur64] S. Kuroda. Classes of languages and linear bounded automata. *Information and Control*, 7(2):207–223, 1964.

[Lau83] C. Lautemann. BPP and the polynomial hierarchy. *Information Processing Letters*, 17:215–217, November 1983.

[Lev73] L. Levin. Universal sorting problems. *Problems of Information Transmission*, 9:265–266, 1973. English translation of original in *Problemy Peredaci Informacii*.

[LLS75] R. Ladner, N. Lynch, and A. Selman. A comparison of polynomial time reducibilities. *Theor. Comput. Sci.*, 1:103–123, 1975.

[LS86] T. Long and A. Selman. Relativizing complexity classes with sparse oracles. *J. ACM*, 33(3):618–627, 1986.

[Mag69] G. Mager. Writing pushdown acceptors. *J. Comput system Sci.*, 3(3):276–319, 1969.

[Mah82] S. Mahaney. Sparse complete sets for NP: solution of a conjecture of Berman and Hartmanis. *J. Comput. System Sci.*, 25:130–143, 1982.

[Myh60] J. Myhill. Linear bounded automata. WADD 60-165, Wright Patterson AFB, Ohio, 1960.

[Pip79] N. Pippenger. On simultaneous resource bounds. In *Proc. 20th IEEE Symp. on Foundations of Computer Science*, 1979.

[PPST83] W. Paul, N. Pippenger, E. Szemerédi, and W. Trotter. On determinism and non-determinism and related problems. In *Proc. 24th ACM Symp. on Theory of Comput.*, pages 429–438, 1983.

[PS78] V. Pratt and L. Stockmeyer. A characterization of the power of vector machines. *J. Comput. and Systems Science*, 12:198–221, 1978.

[Rab76] M. Rabin. Probabilistic automata. In J. Traub, editor, *Algorithms and Complexity: New directions and Recent Results*, pages 21–39. Academic Press, New York, 1976.

[Rab79] M. Rabin. Digitalized signatures and public-key functions as intractable as factorization. Technical Report MIT/LCS/TR-212, MIT Lab. For Comp. Sci., 1979.

[Ros67] A. Rosenberg. Real-time definable languages. *J. Assoc. Comput. Mach.*, 14:645–662, 1967.

[RSA78] R. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Comm. ACM*, 21:120–126, 1978.

[Ruz81] W. Ruzzo. On uniform circuit complexity. *J. Comput. and Systems Science*, 22:365–383, 1981.

[Sav70] W. Savitch. Relationships between nondeterministic and deterministic time complexities. *J. Comput. and Systems Sciences*, 4(2):177–192, 1970.

[Sav76] J. Savage. *The complexity of computing.* Wiley, New York, 1976.

[Sch86] U. Schöning. *Complexity and Structure.* Lecture Notes in Computer Science Vol. 211. Springer-Verlag, 1986.

[Sel89] A. Selman. Complexity issues in cryptography. In J. Hartmanis, editor, *Computational Complexity Theory*, volume 38 of *Procs. of Symposia in Applied Mathematics*, pages 92–107. Amer. Math. Society, 1989.

[Sel90] A. Selman, editor. *Complexity Theory Retrospective.* Springer-Verlag, 1990.

[SG77] I. Simon and J. Gill. Polynomial reducibilities and upward diagonalizations. In *Proc. 9th Annual ACM Symp. on Theory of Comput.*, pages 186–194, 1977.

[Sha48] C. Shannon. The mathematical theory of communication. *Bell System J.*, 27:379–423, 623–656, July/Oct. 1948.

[Sip78] M. Sipser. Halting space-bounded computations. In *Proc. 19th Annual Proc. IEEE Symp. on Foundations of Computer Science*, pages 73–74, 1978.

[Sip83] M. Sipser. A complexity theoretic approach to randomness. In *Proc. 15th ACM Symposium on Theory of Computing*, pages 330–335, 1983.

[SS77] R. Solovay and V. Strassen. A fast monte-carlo test for primality. *SIAM J. Comput.*, 6:84–85, 1977.

[Sto76] L. Stockmeyer. The polynomial-time hierarchy. *Theor. Comput. Sci.*, 3:1–22, 1976.

[Sze87] R. Szelepcsényi. The method of forcing for nondeterministic space. In *Bulletin of the EATCS*, volume 33, pages 96–99. 1987.

[Tur 7] A. Turing. On computable numbers with an application to the entscheidungsproblem. *Proc. London Math. Soc.*, 42:230–365, 1936-7.

[Wra76] C. Wrathall. Complete sets and the polynomial hierarchy. *Theor. Comput. Sci*, 3:23–33, 1976.

[Yao85] A. Yao. Separating the polynomial-time hierarchy by oracles. In *Proc. 26th IEEE Symp. on Foundations of Computer Science*, pages 1–10, 1985.