SCHOOL OF OPERATIONS RESEARCH
AND INDUSTRIAL ENGINEERING
COLLEGE OF ENGINEERING
CORNELL UNIVERSITY
ITHACA, NY 14853-7501

TECHNICAL REPORT NO. 918

August 1990

COMPUTATIONAL COMPLEXITY

by

David B. Shmoys[1] & Éva Tardos[2]

This paper is a preliminary version of a chapter to appear in *The Handbook of Combinatorics*, edited by R.L. Graham, M. Grötschel and L. Lovász, (North-Holland, Amsterdam).

[1]Research was supported in part by an NSF Presidential Young Investigator award CCR-89-96272 with matching support from IBM, UPS and Sun and by Air Force Contract AFOSR-86-0078.

[2]Research was supported in part by Air Force Contract AFOSR-86-0078.

Table of Contents

1. Introduction
2. Complexity of Computational Problems
   - Computational problems
   - Models of computation
   - Computational resources and complexity
   - Complexity classes
   - Other theoretical models of efficiency
3. Shades of Intractability
   - Undecidability
   - Evidence of intractability: $\mathcal{NP}$-completeness
   - The polynomial-time hierarchy
   - Evidence of intractability: $\#\mathcal{P}$-completeness
   - Evidence of intractability: $\mathcal{PSPACE}$-completeness
   - Proof of intractability
   - Extensions of $\mathcal{NP}$: short proofs via randomization
4. Living with Intractability
   - The complexity of approximate solutions
   - Probabilistic analysis of algorithms
   - Cryptography
5. Inside $\mathcal{P}$
   - Logarithmic space
   - The hardest problems in $\mathcal{P}$
   - Parallel computation
6. Attacks on the $\mathcal{P}$ versus $\mathcal{NP}$ Problem
   - Relativized complexity classes
   - Relating circuit complexity to Turing machine complexity
   - Constant-depth circuits
   - Monotone circuit complexity
   - Machine-based complexity classes

1 Introduction

Computational complexity theory attempts to explain the nature of computation, by providing insight into the question why certain computational problems appear to be more difficult than others. As a field, it lies on the interface between computer science and mathematics, having been motivated by practical questions of the former, while relying on the methods of the latter. Complexity theory has had a particularly close relationship with combinatorics, since most of its proof techniques are combinatorial in nature, and many important examples of computational problems have been drawn from combinatorics.

The interdependence of combinatorics and complexity theory is illustrated well by a celebrated result of Ajtai, Komlós and Szemerédi (1983), who provide an efficient parallel algorithm to sort numbers by using certain bipartite graphs, known as expanders. There is a beautiful probabilistic proof that these graphs exist: a graph is selected according to a certain probability distribution and then it is shown that the graph selected is an expander with non-zero probability. This proof merely shows the *existence* of such a graph, it does not show how to *construct* one.

In order for this existential theorem to be useful in an algorithmic setting, we must be able to construct an expander, which raises the following question: how do we compute the description of an expander of a specified order n? There is a trivial way to do the computation, which suggests that we have asked the wrong question: for each graph in the sample space, check by brute force whether it is an expander. Unfortunately, the sample space is large, containing $(n!)^2$ graphs, and to check each graph for the expander property is itself a nontrivial process, so for even modest values of n, this procedure is wildly impractical. Thus, a more relevant question is one of efficient computation: does there exist an efficient way to compute the description of an expander of a specified order? We shall see that complexity theory provides the necessary machinery to make this statement more precise.

As the previous example makes clear, computational issues have given an additional dimension to combinatorics. The probabilistic proof that expanders exist becomes just a first step in the search for the additional structure needed to provide an efficient construction. Indeed, one need only examine the table of contents of this volume to see the impact of computational issues on the present direction of combinatorics. Furthermore, complexity theory has often raised combinatorial questions that were previously unsolved, and thus has sparked research in combinatorics.

Complexity theory has also helped to understand the significance of certain combinatorial theorems. For example, consider the following two well-known theorems due to Hall and Camion, respectively: a bipartite graph has a perfect matching if and only if each subset of nodes in one part is adjacent to at least as many nodes in the other; a matrix with 0,+1, and -1 entries is totally unimodular if and only if each nonsingular submatrix has a row with an odd number of nonzero components. Each theorem provides an alternate characterization of the property in question, but, as Edmonds observed, there is a fundamental way in which only the first is a "good" characterization. The perfect matching itself is a concise proof that a particular graph has one and Hall's theorem provides a way to give an efficiently verifiable proof that a graph does not have one. On the other hand, both the definition and Camion's characterization of total unimodularity show only how to give a concise proof that a matrix is not totally unimodular. Just as complexity theory has provided a means of discussing efficient algorithms, it has also given an analogous formal notion of

a good characterization.

Complexity theory has been a rapidly expanding field over the past quarter century, and it would be impossible to include all of its advances. Much of complexity theory is directed towards the formulation of computational models that explain differences between the resources required to solve particular problems, and we will focus on those closely tied to combinatorial examples. For example, completeness results suggest that the problem of deciding who has a winning strategy in certain combinatorial games is computationally even more intractable than deciding if a graph is Hamiltonian.

Computational models provide a framework for discussing notions of both algorithms and proofs, and an important recent advance has been in understanding the power that tossing a coin adds in these settings. Suppose that if a certain theorem is false, then a random experiment will disprove it with probability one-half. By repeatedly performing the experiment, one can reach an overwhelmingly high level of confidence that the theorem is, in fact, true. Randomization has been applied to many different aspects of complexity theory, and has provided, for example, the only known efficient algorithms for several combinatorial problems. We will discuss the particularly notable impact that it has had on cryptography and parallel computation.

Parallel computers are rapidly becoming a reality, and traditional algorithmic techniques often appear ill-suited to take advantage of their power. As a result, there has been a great deal of work recently in studying algorithmic techniques for solving combinatorial problems in theoretical models of parallel computers.

Complexity theory has provided evidence that a wide variety of problems are computationally intractable, but evidence is not nearly as satisfying as proof. Although the fundamental problem of providing this proof remains open, significant steps toward settling this question have been made. Many of these results are based on beautiful combinatorial methods, and it is quite possible that combinatorial methods will play a role in the final solution.

Computational complexity is a theoretical discipline, and the results obtained do not always correspond precisely with practical considerations. When used to sort fewer than a million numbers, the algorithm mentioned above is inferior to algorithms that are theoretically inferior. Nonetheless, it is the practical importance of understanding the power of computation that forms the driving force behind complexity theory.

2 Complexity of Computational Problems

In this section, we will outline the essential machinery used to give formal meaning to the complexity of computational problems. This involves describing what is precisely meant by a computational problem, setting up a mathematical model of computation, and then formalizing the notion of the computational resources required for a problem with respect to that model. Unfortunately, there is no one standardized specification in which to discuss these questions. In view of this, one might become concerned that it is possible to prove apparently contradictory results under different specifications of these notions. Therefore, if this theory is to produce meaningful results, it is essential that the definitions be robust enough so that theorems proved with respect to them apply equally to all reasonable variants of this framework. Indeed, the particular definitions

that we will rely on will be accompanied by evidence that these notions are sufficiently flexible.

Computational problems

Suppose that there is a directed graph G and you wish to know whether it is Hamiltonian, that is, whether there is a directed circuit that traverses each of its nodes exactly once. If you handed G to your favorite graph theorist, he might cite an appropriate theorem to obtain the correct answer, "yes" or "no". The graph G is a single **instance** of the Hamiltonian circuit **problem**. Thus, for any **decision problem**, of which the Hamiltonian circuit problem is an example, we will be interested in the set of all instances for which the answer is "yes", which will be called the associated **language**. The computational problem is, given any instance, to decide whether or not this instance is in the language.

Making this more formal, we encode all of the instances in some arbitrary, but standardized way as strings of 0's and 1's. Consider the example above, where the instances to be encoded are directed graphs. There are two natural ways to represent the arcs of G, a directed graph of order n. In the adjacency matrix form, we simply give an $n \times n$ 0,1 matrix $A = (a_{ij})$ where $a_{ij} = 1$ if and only if (i, j) is an arc; the matrix can be represented by a string of n^2 0's and 1's, by simply listing the elements of each row, one after another. In the adjacency list form, we give a list of the nodes incident from node i, for each i; this can be encoded by using the binary representation for the names of the nodes from 1 to n and some additional symbols, such as '—' and '/', and representing the list for i by giving the encoding for i and the nodes incident from it separated by '—' symbols, and then concatenating the lists for all nodes, separated by '/' symbols. Notice that it is not necessary to expand the alphabet from $\{0, 1\}$ since any larger alphabet could be encoded using just those two symbols. Notice that the language for a given decision problem depends on the particular encoding selected, and when we refer to a decision problem, we will refer to the language L associated with some natural encoding.

Observe that this first way of encoding the graph may be less compact, since this encoding length is n^2, whereas the second might be significantly less if there are few arcs in G. Note, however, that this difference is limited in that the length of the input in one format is at most the square of the length in the other.

Consider now the following **directed reachability problem**: given a directed graph G, and two specified nodes s and t, does there exist a path from s to t? An instance of this problem consists of an encoding of G, followed by a suitable encoding of s and t. It is not hard to imagine that no algorithm for this problem that is given its input in the adjacency matrix form can "look at" significantly less than all n^2 bits for all graphs of order n, and still answer correctly for all of these instances. If the input were given in the more compact adjacency list format, it is still possible to design a procedure that always takes time proportional to the length of the input and always produces a correct answer. At first glance it seems that the list format can lead to more efficient algorithms, but, as was true for the length of the encodings, this difference can be easily bounded. Nonetheless, unless otherwise stated, this chapter will use the adjacency list encoding.

The previous example raises another interesting question, which is somewhat tangential to the main focus of this chapter. Is there a sense in which all "reasonable" decision problems on graphs encoded in adjacency matrix form require any algorithm to read some constant fraction of the input? This problem has a long history, both for directed and undirected graphs, and many attempts were made at giving sufficiently strong conditions before an accurate conjecture, due to Aanderaa and

Rosenberg, was proved by Rivest & Vuillemin (1976), and later strengthened by Kahn, Saks & Sturtevant (1984). Consider a decision problem L where the instances are undirected graphs, and L has two important properties: (1) **monotonicity** – if G is an instance in L and G' is formed by adding an edge, then G' is also in L ; (2) **invariance under isomorphism** – if G is an instance in L and its nodes are relabeled to form G', then G' is also in L. If we only count the number of questions of the form, "Is ij an edge?" then for any problem satisfying these two properties, any correct procedure uses essentially $n^2/4$ queries for some graph of order n. In fact, if n is a prime power, Kahn, Sturtevant & Saks have shown that all $n(n-1)/2$ queries must be asked.

Up to this point we have restricted our attention to decision problems, and this appears to be limiting. A more satisfying answer to an input in the language of the Hamiltonian circuit problem would be the Hamiltonian circuit itself. However, this information can be extracted by asking a series of questions about membership in L. Repeat the following for each arc in the graph: delete the arc, ask if the resulting graph is still Hamiltonian and replace the arc only in the case that the answer is "no". At the end of this procedure, the remaining graph is the circuit. Thus, we have shown that the **search problem** of finding the circuit is not much harder than the decision problem, and this relationship remains true for all well-formulated decision versions of search problems.

Another important type of computational problem is an **optimization problem**; for example, given G and two nodes s and t, we may wish to find the shortest path from s to t (or merely find the length). Problems of this type can be converted into decision problems by adding a bound b to the instance, and, for example, asking whether G has a path from s to t of length at most b. To see that this is reasonable, notice that the optimal value can be computed by **binary search**, that is, by asking a series of questions of this form by fixing b to be the midpoint of the current range of values for the optimum, and using the answer to cut the range in half for the next question. As above, we see that an optimization problem can be solved with only somewhat more work than the corresponding decision problem.

Throughout this chapter, we will be dealing with computational problems involving a variety of structures, and we will not be specifying the nature of the encodings used. We will operate on the premise that any reasonable encoding produces strings of length that can be bounded by a polynomial of the length produced by any other encoding. For problems that involve numbers we must be more careful in stating precisely what is meant by reasonable. First, we shall restrict attention to integral data. To highlight the remaining issues, consider the following problem, called the **subset sum problem**: given a set of numbers S and a target number t, does there exist a subset of S that sums to t? Typically, we will assume that the numbers are given in binary representation, although we will occasionally use a unary representation (*e.g.*, representing 5 by 11111). Notice that the latter could be exponentially bigger than the former, and thus size is a deceptive measure, since it makes instances of the problem larger than they need be. As we survey the complexity of computational problems that involve numbers, we will see that some are sensitive to this choice of encodings, such as the subset sum problem, whereas others are less affected.

Models of computation

We next turn our attention to defining a mathematical model of a computer. In fact, we will present three different models, and although their superficial characteristics make them appear quite different, they will turn out to be formally equivalent. The first of these is the Turing machine, which is an extremely primitive model, and as a result, it is easier to prove results about

what cannot be computed within this model. On the other hand, its extreme simplicity makes it ill-suited for algorithm design; one would never want to program a Turing machine. As a result, it will be convenient to have an alternative model, the random access machine (RAM), within which to discuss algorithms.

The name, Turing machine, is a slight misnomer, since a Turing machine is a mathematical formulation of an algorithm, rather than a machine. A Turing machine $M = (Q, \Gamma, \delta, q_0, A)$ is a machine that has a finite main memory represented by a finite set of states Q, a read-only input tape, a finite set of work tapes each of which contains a countably infinite number of cells (corresponding to the integers) to store a character from a finite alphabet Γ, which at least includes the input alphabet $\{0, 1\}$ and a blank symbol B. For each of the k work tapes and the input tape, there is a "head" that can read one cell of the tape at a given time, and will be able to move cell-by-cell across the tape as the computation proceeds. Throughout the computation, the heads will read the contents of cells, and depending on what was read and the current state of the main memory, rewrite the cells and then move each head by one cell, either left or right, as well as cause a change in the state of the main memory. The basic step of a Turing machine, a **transition**, is modeled by a partial function

$$\delta : Q \times \Gamma^{k+1} \mapsto Q \times \Gamma^k \times \{L, R\}^{k+1}$$

that selects the new state, the contents of the cells currently scanned on the work tapes, and indicates the direction in which each head moves one cell as a **deterministic** function of the current state and the contents of the input and work tape cells currently being read. One may view the transition function as the program hardwired into this primitive machine. The computation is begun in state q_0, with the input head at the left end of the input, and all of the work tapes and the rest of the input tape blank. The machine **halts** if δ is undefined for the current state and the symbols read. An input is **accepted** if it halts in a state in the accepting state set $A \subseteq Q$. A Turing machine M solves a decision problem L if L is the set of inputs accepted by M, and M halts on every input; such a language L is said to be **decidable**. This definition of a Turing machine is similar to the one given by Turing (1937), and the reader should note that many equivalent definitions are possible.

We have defined a Turing machine so that it can only solve decision problems, but this definition can be easily extended to model the computation of an arbitrary function by, for example, adding a write-only output tape, on which to print the output before halting. Although this appears to be a very primitive form of a computer, it has become routine to accept the following proposition.

Church's Thesis: Any function computed by an effective procedure can be computed by a Turing machine.

Although this is a *thesis*, in the sense that any attempt to characterize the inexplicit notion of effective procedure would destroy its intent, it is supported by a host of *theorems*, since for any known characterization of computable functions, it has been shown that these are Turing computable.

A random access machine (RAM) is a model of computation that is well-suited for specifying algorithms, since it uses an idealized, simplified programming language that closely resembles the assembly language of any modern-day digital computer. There is an infinite number of memory cells indexed by the integers, and there is no bound on the size of an integer that can be stored

in any cell. A program can directly specify a cell to be read from or written in, without moving a head into position. Furthermore, there is an indirect addressing option, that uses the contents of a cell as an index to another cell that is then (seemingly randomly) accessed. All basic arithmetic operations can be performed. For further details on RAM's the reader is referred to Aho, Hopcroft & Ullman (1974). Since the RAM is cast more in the mold of a traditional programming language, it is less natural to think of using a RAM to solve decision problems, but this is just a special case with a simple kind of output, 0 or 1.

Another model of computation closely tied to a practical setting is the logical circuit model. The simplicity of the circuit model makes it extremely attractive for proving lower bounds on the computational resources needed for particular functions, and research along these lines will be discussed in depth in Section 6. A circuit may be thought of as a directed acyclic graph, where the nodes of indegree 0 are the Boolean **input gates** (that can assume value 0 or 1), and the remainder correspond to **functional gates** of the circuit and are labeled with an operation, such as the logical **or**, negation, or logical **and** operations. The nodes of outdegree 0 are the **outputs**. A circuit with n input nodes and a single output node accepts the set of 0,1-strings of length n where the corresponding input generates an output of 1. In order to handle all inputs, we specify a circuit for each input length, and say that the family of circuits solves a decision problem L, if L is the set of strings accepted by some circuit in the family. Note that this model differs from the previous two, in that the circuit for inputs of length n can be tailored **nonuniformly** to the particular value of n, whereas a Turing machine or a RAM must run on inputs **uniformly**, independent of their length. To make the notion of a family of circuits equivalent to the other models of computation, we insist that there be a Turing machine that, on input n, computes the description of the circuit for inputs of length n.

In spite of the apparent differences in these three models, any particular choice is one of convenience, and not of substance.

(2.1) Theorem. *The following classes of decision problems are all identical:*

- *the class of decision problems solvable by a Turing machine;*
- *the class of decision problems solvable by a RAM;*
- *the class of decision problems solvable by a family of circuits that can be generated by a Turing machine.*

This result is complemented by the following theorem, which is a consequence of the fact that there are an uncountable number of decision problems, but only a countable number of Turing machines.

(2.2) Theorem. *Not all decision problems are solvable by a Turing machine.*

This theorem does not provide a particular problem that is not solvable; it merely proves the existence of such a problem. Particular problems from a wide variety of fields of mathematics are known to have this property. In the next section, we will provide a few problems that are not solved by any Turing machine. One famous example of such a problem is Hilbert's tenth problem.

Hilbert asked for an effective procedure to decide if a given multivariate polynomial has an integer valued root. Culminating years of progress in this area, Matijasevič (1970) proved that there does not exist a Turing machine that solves this decision problem, so that by Church's thesis, there is no such effective procedure.

One can think of a particular computation as proving (or disproving) the theorem "$x \in L$". We recognize that proving a theorem and verifying a proof of a theorem are very different acts, and the computational analogue of this difference is the driving force behind much of complexity theory. To model the ability to verify a proof that $x \in L$, imagine a machine that guesses one of the many possible ways in which the computation might proceed, and the accepted inputs are those for which there is a guess whose computation path shows that $x \in L$. Formally, in a **nondeterministic** Turing machine, the transition function δ is no longer a function, but rather a relation, in that at each step there is a finite set of possible next moves of which exactly one can be made. The notion of acceptance of a language L by a nondeterministic Turing machine is central to the definition: L is the set of inputs for which there exists a sequence of transitions of δ that cause the machine to halt in an accepting state. An equivalent formulation is to think of a nondeterministic Turing machine as a deterministic Turing machine with an additional **guess tape**, that is a read-only tape, where the head only moves to the right. The contents of the guess tape are magically constructed and presented to the machine as it begins the computation. The set of inputs in L is the set of inputs for which there is a guess that allows the machine to halt in an accepting state. For simplicity, we shall assume that the machine makes the same number of transitions for any guess. Note that a nondeterministic Turing machine accepting L can be converted into a deterministic machine for L by trying all of the (exponentially many) guesses.

Consider again the Hamiltonian circuit problem. A nondeterministic Turing machine for it might be constructed by letting the guess tape encode a sequence of n nodes of the graph. The Turing machine simply verifies that there is an arc between each consecutive pair of nodes in the guessed sequence. If the graph is Hamiltonian, then there is a correct guess, but otherwise, for any sequence of n nodes there will be some consecutive pair that is not adjacent. The correct guess, in essence the Hamiltonian circuit, is a **certificate** that the graph is Hamiltonian. Observe that this definition is one-sided, since the requirements for instances in L and not in L are quite different. What could serve as a "guess" to certify that a graph is not Hamiltonian?

Computational resources and complexity

Now that we have mathematical formulations of both problems and machines, we can describe what is meant by the computational resources required to solve a certain problem.

In considering the execution of a deterministic Turing machine, it is clear that the number of transitions before halting corresponds to the running time of the machine on a particular input. In discussing the running time of an algorithm, within any of the models, we do not want to simply speak of particular instances, and so we must make some characterization of the running time for all instances. The criterion that we will focus on for most of this chapter is the worst-case running time as a function of the input size. When we say that a Turing machine takes n^2 steps, this means that for any input of size n, the Turing machine always halts within n^2 transitions. Unless otherwise specified, the length of the input will be denoted by n.

A more practical alternative might have been to use the "average" running time, but in order to do so, we must characterize an "average" instance. That is, we must specify a probability distri-

bution from which the input is drawn and then analyze the expected running time when the input is drawn accordingly. By focusing on worst-case analysis, the results do not depend on any such restriction. On the other hand, a theory of algorithms that could handle this sort of probabilistic analysis and still be relatively insensitive to the particulars of the probability distribution would significantly enrich the scope of complexity theory.

We will not be interested in the precise count of the number of transitions, but rather in the order of the running time. A function $f(n)$ is $O(g(n))$ if there are constants N and c such that for all $n \geq N$, $f(n) \leq cg(n)$. Thus, rather than say that a Turing machine has worst-case running time $3n^2 + 5n - 17$, we say simply that it is $O(n^2)$. This simplification makes it possible to discuss complicated algorithms without being overwhelmed by details. Furthermore, for any Turing machine with superlinear running time and any constant c, there exists another Turing machine to solve the same problem that runs c times faster than the original, that can be constructed by using an expanded work tape alphabet.

We will also use a notation analogous to $O(\cdot)$ to indicate lower bounds. A function $f(n)$ is $\Omega(g(n))$ if there are constants N and c such that for all $n \geq N$, $f(n) \geq cg(n)$. A function $f(n)$ is $\Theta(g(n))$ if it is both $O(g(n))$ and $\Omega(g(n))$. It will also be useful to have a notion of one function being asymptotically smaller than another: $f(n)$ is $o(g(n))$ if $\lim_{n\to\infty} f(n)/g(n) = 0$.

For a nondeterministic Turing machine, we define the running time to be the number of transitions in any computation path generated by the input. (Recall that we added the restriction that all computation paths must have the same length.) For a circuit, we will want to characterize the number of operations performed as a measure of time, so that the relevant parameter is the size of the circuit, which is the order of the graph representing it. For a RAM, there are two standard ways in which to count the running time on a particular input. In each operation, a RAM can, for example, add two number of unbounded length. In the **unit-cost** model, this action takes one step, independent of the lengths of the numbers being added. In the **log-cost** model, this operation takes time proportional to the lengths of the numbers added. These measures can have radically different values. Take 17, and repeatedly square it k times. With each squaring, the number of bits in the binary representation essentially doubles. Thus, although we have taken only k steps in the unit-cost model, the time required according to the log-cost model is exponential in k. Technically, we will use the log-cost model, in order to ensure that the RAM is equivalent to the Turing machine in the desired ways. But, when speaking of the running time of an algorithm, it is traditional to state the running time in the unit-cost model, since for all standard algorithms one can prove that the pathological behavior of the above example does not come into play.

Time is not the only computational resource in which we will be interested; we will see that the space complexity of certain combinatorial problems gives more insight into the structure of these problems. In considering the space requirements, we will focus on the Turing machine model, and will only count the number of cells used on the work tapes of the machine. Furthermore, we will again be interested in the asymptotic worst-case analysis of the space used. As was true for time bounds, the space used by a Turing machine can be compressed by any constant factor. The space used by a nondeterministic Turing machine is the maximum space used on any computation path.

The notion of the **complexity** of a problem is the order of a given computational resource, such as time, that is necessary and sufficient to solve the problem. If we say that the complexity of the directed reachability problem for a graph with m arcs is $\Theta(m)$ this means that there is a (unit-cost

RAM) algorithm that has worst-case running time $O(m)$ and there is no algorithm with running time of lower order. Tight results of this kind are extremely rare, since the tremendous progress in the design of efficient algorithms has not been matched, or even approached, by the slow progress in techniques for proving lower bounds on the complexity of these problems in general models of computation. For example, consider the **3-colorability problem**: given an undirected graph G with m edges, can the nodes be colored with three colors so that no two adjacent nodes are given the same color; *i.e.*, is $\chi(G) \leq 3$? The best lower bound is only $\Omega(m)$, in spite of substantial evidence that it cannot be solved in time bounded by a polynomial.

Complexity classes

In order to study the relative power of particular computational resources, we introduce the notion of a **complexity class**, which is the set of problems that have a specified upper bound on their complexity. It will be convenient to define the complexity class $DTIME(T(n))$ to be the set of all languages L that can be recognized by a deterministic Turing machine within time $O(T(n))$. $NTIME(T(n))$ denotes the analogous class of languages for nondeterministic Turing machines. Throughout this chapter, it will be convenient to make certain assumptions about the sorts of time bounds that define complexity classes. A function $T(n)$ is called **fully time-constructible** if there exists a Turing machine that halts after exactly $T(n)$ steps on any input of length n. All common time bounds, such as $n \log n$ or n^2, are fully time-constructible. We will *implicitly* assume that any function $T(n)$ used to define a time-complexity class is fully time-constructible.

The single most important complexity class is $\mathcal{P}$, the class of problems solvable in polynomial time. Two of the most well-known algorithms, the Euclidean algorithm for finding the greatest common divisor of two integers and Gaussian elimination for solving a system of linear equations, are classical examples of polynomial-time algorithms. In fact, Lamé observed as early as 1844 that the Euclidean algorithm was a polynomial-time algorithm. In 1953, Von Neumann contrasted the running time for an algorithm for the assignment problem that "turn[ed] out [to be] a moderate power of n, *i.e.*, considerably smaller than the 'obvious' estimate $n!$" for a complete enumeration of the solutions. Edmonds (1965) and Cobham (1965) were the first to introduce $\mathcal{P}$ as an important complexity class, and it was through the pioneering work of Edmonds that polynomial solvability became recognized as a theoretical model of efficiency. With only a few exceptions, the discovery of a polynomial-time algorithm has proved to be an important first step in the direction of finding truly efficient algorithms. Polynomial time has proved to be very fruitful as a theoretical model of efficiency both in yielding a deep and interesting theory of algorithms and in designing efficient algorithms.

There has been substantial work over the last 25 years in finding polynomial-time algorithms for combinatorial problems. It is a testament to the importance of this development that much of this handbook is devoted to discussing these algorithms. This work includes algorithms for graph connectivity and network flow (see Chapter 2), for graph matchings (see Chapter 3), for matroid problems (see Chapter 11), for point lattice problems (see Chapter 19), for testing isomorphism (see Chapter 27), for finding disjoint paths in graphs (see Chapter 5) and for problems connected with linear programming (see Chapters 28 and 30).

Another, more technical reason for the acceptance of $\mathcal{P}$ as the theoretical notion of efficiency, is its mathematical robustness. Recall the discussion of encodings where we remarked that any reasonable encoding will have length bounded by a polynomial in the length of another. As a

result, any polynomial-time algorithm which expects its input in one form can be converted to a polynomial-time algorithm for the other. In particular, note that the previous discussion of the two different encodings of a graph can be swept aside and we can assume that the size of the input for a graph of order n is n. Notice further that the informal definition of $\mathcal{P}$ given above does not rely on any model of (deterministic) computation. This rash statement is justified by the following theorem.

(2.3) Theorem. *The following classes of decision problems are all equivalent:*

- *the class of decision problems solvable by a Turing machine in polynomial-time;*
- *the class of decision problems solvable by a RAM in polynomial-time under the log-cost measure;*
- *the class of decision problems solvable by a family of circuits of polynomial size, where the circuit for inputs of size n can be generated by a Turing machine with running time bounded by a polynomial in n.*

The importance of the class $\mathcal{NP} = \cup_k NTIME(n^k)$ is due to the wide range of important problems that are known to be in the class, and yet are not known to lie in $\mathcal{P}$. For example, the nondeterministic algorithm given for the Hamiltonian circuit problem is clearly polynomial, and so this problem lies in $\mathcal{NP}$. However, it is not known whether this, or any problem is in $\mathcal{NP} \setminus \mathcal{P}$, and this is, undoubtably the central question in complexity theory.

Open Problem Is $\mathcal{P} = \mathcal{NP}$?

For each decision problem L, there is a complementary problem, $\bar{L}$, such as the problem of recognizing non-Hamiltonian graphs. For any complexity class $\mathcal{S}$, let $co - \mathcal{S}$ denote the class of languages whose complement is in $\mathcal{S}$. The definition of $\mathcal{P}$ is symmetric with respect to membership and non-membership in L, so that $\mathcal{P} = co - \mathcal{P}$. $\mathcal{NP}$ is very different in this respect. In fact, it is unknown whether the Hamiltonian circuit problem is in $co - \mathcal{NP}$.

Open Problem Is $\mathcal{NP} = co - \mathcal{NP}$?

Edmonds (1965) brought attention to the class $\mathcal{NP} \cap co - \mathcal{NP}$, and called problems in this class **well-characterized**, since there is both a short certificate to show that the property holds, as well as a short certificate that it does not. Edmonds was working on algorithms for non-bipartite maximum matching at the time, and this problem serves as a good example of a problem in this class. If the instance consists of a graph G and a bound k and we wish to know if there is a matching of size at least k, the matching itself serves as a certificate for an instance in L, whereas an odd-set cover serves as a certificate for an instance not in L (see Chapter 3). Note that there is a min-max theorem, Tutte's theorem, characterizing the size of the maximum matching that is at the core of the fact that matching is in $\mathcal{NP} \cap co - \mathcal{NP}$, and indeed min-max theorems often serve this role. As mentioned above, matching is known to be in $\mathcal{P}$, and this raises the following question.

Open Problem Is $\mathcal{P} = \mathcal{NP} \cap co - \mathcal{NP}$?

We will also be concerned with complexity classes defined by the space complexity of problems. As for time, let $DSPACE(S(n))$ and $NSPACE(S(n))$ denote, respectively, the class of languages

accepted by deterministic and nondeterministic Turing machines within space $O(S(n))$. We will implicitly assume the following condition for all space bounds used to define complexity classes: a function $S(n)$ is **fully space-constructible** if there is a Turing machine that, on any input of length n delimits $S(n)$ tape cells and then halts. Three space complexity classes will receive the most prominent attention:

- $\mathcal{L} = DSPACE(\log n)$;
- $\mathcal{NL} = NSPACE(\log n)$; and
- $\mathcal{PSPACE} = \cup_k DSPACE(n^k)$.

One might be tempted to add a fourth class, $\mathcal{NPSPACE}$, but we shall see that nondeterminism does not add anything in this case. We will see that the chain of inclusions

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

holds, and the main thrust of complexity theory is to understand which of these inclusions is proper. At the extremes, a straightforward diagonalization argument due to Hartmanis, Lewis & Stearns (1965) shows that $\mathcal{L} \neq \mathcal{PSPACE}$, and a result of Savitch (1970) further implies that $\mathcal{NL} \neq \mathcal{PSPACE}$, but after nearly a quarter century more work, these are the only sets in this chain known to be distinct.

The lack of lower bounds with respect to a general model of computation for either space or time complexity has led to the search for other evidence that suggests that lower bounds hold. One such type of evidence might be the implication: if the Hamiltonian circuit problem is in $\mathcal{P}$, then $\mathcal{P} = \mathcal{NP}$. $\mathcal{NP}$ contains a tremendous variety of problems that are not known to be in $\mathcal{P}$, and so by proving such a claim, one shows that the Hamiltonian circuit problem is a hardest problem in $\mathcal{NP}$, in that any polynomial-time algorithm to solve it would in fact solve thousands of other problems.

The principal tool in providing evidence of this form is that of reduction. We will say that a problem L_1 **polynomial-time reduces** to L_2 if there exists a polynomial-time computable function f that maps instances of L_1 into instances of L_2 such that x is a "yes" instance of L_1 if and only if $f(x)$ is a "yes" of L_2. We shall denote this by $L_1 \propto_p L_2$. Notice that if there were a polynomial-time algorithm for L_2, we could then obtain a polynomial-time algorithm for L_1 by first computing $f(x)$ and then running the assumed algorithm on $f(x)$. This composite procedure is polynomial-time, since the composition of two polynomials is itself a polynomial.

Definition A problem L_1 is **$\mathcal{NP}$-complete** if

1. $L_1 \in \mathcal{NP}$;
2. for all $L \in \mathcal{NP}$, $L \propto_p L_1$.

The composition argument given above yields the following results.

(2.4) Theorem. *For any $\mathcal{NP}$-complete problem L, $L \in \mathcal{P}$ if and only if $\mathcal{P} = \mathcal{NP}$.*

(2.5) Theorem. *If L_1 is $\mathcal{NP}$-complete, $L_2 \in \mathcal{NP}$ and $L_1 \propto_p L_2$, then L_2 is $\mathcal{NP}$-complete.*

The first result says that any $\mathcal{NP}$-complete problem completely characterizes $\mathcal{NP}$ in its relationship to $\mathcal{P}$, and that we can focus on any such problem without loss of generality in trying to prove that the two classes are different. The Hamiltonian circuit problem is $\mathcal{NP}$-complete, and in the next section we will prove that several combinatorial problems are among the plethora having this property. The second result gives a strategy for proving that a problem is $\mathcal{NP}$-complete, provided that a "first" $\mathcal{NP}$-complete problem is known. Of course, to initiate this strategy, we must show that some natural problem has this property, and it was a landmark achievement in complexity theory when Cook (1971) showed that the problem of deciding the satisfiability of a formula in propositional logic is $\mathcal{NP}$-complete. The importance of this result was only fully recognized through the work of Karp (1972), whose seminal paper showed 21 natural combinatorial problems to be $\mathcal{NP}$-complete. Independently, Levin (1973) discovered a similar theory for studying the intractability of computational problems.

Cook's theorem, which will be given in the next section, was really just a first step in characterizing complexity classes. It is possible to use the same techniques to define notions of complete problems for $\mathcal{NL}$, $\mathcal{P}$ and $\mathcal{PSPACE}$. Although the technicalities of the kind of reductions used differ from example to example, the overall philosophy remains the same. Throughout Sections 3 and 5 we will be exploring these ideas in greater depth.

Other theoretical models of efficiency

From a practical perspective, polynomial time does not always serve as a good model of efficiency. Some of the polynomial-time algorithms are highly ineffective in practice, and compare unfavorably to non-polynomial-time alternatives. The most important such example is probably the **linear programming problem**, which is as follows: given an $m \times n$ matrix A, an m-vector b and an n-vector c, find a vector x that maximizes the objective function $c^T x$ subject to the conditions $Ax \leq b$. The simplex algorithm for linear programming is one of the most commonly used algorithms. Although this algorithm is known to take exponential time in the worst case, in practice it is almost always very efficient. The first polynomial-time algorithm for the problem, the ellipsoid method, was developed by Khachiyan (1979) based on ideas from convex programming by Shor, Yudin and Nemirovskii. The ellipsoid method and its implications are discussed in detail elsewhere in this volume (see Chapters 28 and 30). In practice, this theoretically efficient algorithm does not even come close to matching the efficiency of the theoretically inefficient simplex algorithm. Polynomial-time algorithms that are based on the ellipsoid method should be considered as indications of efficient solvability rather than as efficient algorithms. After the ellipsoid method, Karmarkar (1984) and Renegar (1988) used other convex programming techniques to yield polynomial-time algorithms for linear programming. Karmarkar's algorithm has received much publicity due to his claim that the algorithm is also efficient in practice. Indeed, on large problems with sparse constraint matrix A, a variant of the algorithm seems to compare favorably to the simplex algorithm.

One way to give a theoretical explanation of the surprising practical performance of the simplex method, a procedure that takes exponential time in the worst case, is to show that its expected running time on "reasonable" input distributions is polynomial. Results that show that the simplex algorithm runs in polynomial time on the average (for certain input distributions) will be discussed together with the probabilistic analysis of other algorithms in Section 4. In that section we will see that many problems, such as the Hamiltonian circuit problem, become significantly easier when studied from the viewpoint of average-case analysis. These results often do not correspond to

practical experience, so that it is hard to judge if such a result merely reflects that the particular distribution highlights easy instances.

We will present a few generalizations of $\mathcal{P}$ that exploit the power of randomization in the design and analysis of algorithms *without* making probabilistic assumptions about the inputs. A randomized algorithm is an algorithm that can flip coins during the computation; that is, we consider a fixed input, and study the algorithm's behavior as a random variable depending only on the coin flips used. For the algorithms discussed below, the algorithm is allowed to make mistakes, but for each input the probability of error must be very small.

The most well-known randomized algorithm is for testing primality. In the **primality testing problem**, we are given a natural number N, and we wish to decide if it is prime. It is not known whether primality testing is in $\mathcal{P}$. The input size of the number N is $\log N$, and therefore algorithms that simplistically search for divisors of N do not run in polynomial time. Fermat's theorem provides a way to conclude that a number is not prime without actually exhibiting a factor.

(2.6) Theorem. *If N is prime then a^N is congruent to a* modulo *N for every integer a.*

That is, if we find an integer a such that a^N is not congruent to a modulo N, denoted $a^N \not\equiv a \bmod N$, then we can conclude that N is not prime. (Note that $a^N \bmod N$ can be computed in polynomial time by repeatedly squaring modulo N.) Let such an a be called a **witness** for N's compositeness. The advantage of this kind of witness, compared to exhibiting a divisor, is the following easy lemma.

(2.7) Lemma. *If there exists an integer a such that $a^N \not\equiv a \bmod N$, then at least half of the integers in the range from 1 to N have this property.*

Unfortunately, there are composite numbers, the so-called **Carmichael numbers** that are not prime, but for which no witness exists. If we momentarily forget about the existence of these numbers, we get the following algorithm for testing primality: given an integer N, choose an integer a in the range 1 to N at random, and check if a is a witness for N. If a witness is found, then we know that N is not prime (and not even a Carmichael number). On the other hand, if N is not a prime (and also not a Carmichael number), then by Lemma (2.7), a random a is a witness with probability at least one half. Running this test k times with independent random choices, we either find a witness or can fairly safely conclude that no witness exists (with error probability 2^{-k}). This gives a randomized polynomial-time algorithm to recognize the language of all primes and Carmichael numbers. Rabin (1976) and Solovay & Strassen (1977), by using a somewhat more sophisticated variant of Fermat's theorem, gave randomized polynomial-time algorithms that accept the language of all primes.

The above idea actually gives a polynomial-time algorithm if the extended Reimann hypothesis is true. Miller (1976) has proved that if the extended Reimann hypothesis holds, then there exists a witness for N (in more or less the above sense) that is at most $O((\log N)^2)$. By trying all the integers up to this limit, we would get a polynomial-time deterministic algorithm for primality testing. For more details on this and other number-theoretic algorithms see the survey of Lenstra & Lenstra (1990).

The formal definition of a **randomized Turing machine** is similar to the definition of a nondeterministic Turing machine in the sense that at every point during the computation there could be several different next steps. Randomized Turing machines have a read-only **randomizing tape** similar to the guess tape of the nondeterministic Turing machine. We can think of this tape as providing the outcomes of the coin flips to be used by the algorithm. We shall assume that for a given input length n, the algorithm reads a fixed number of bits, $f(n)$, from the randomizing tape. The **probability** that the randomized Turing machine accepts an input x of length n is defined to be the fraction of all possible strings of length $f(n)$ that, when used as the initial segment of the randomizing tape, cause the Turing machine to accept x. For the randomized Turing machine, we take the simplifying approach that the running time for an input is the maximum number of transitions in some sequence of allowed transitions (*i.e.*, for some contents of the randomizing tape).

We define $\mathcal{BPP}$, the class of languages accepted by a randomized polynomial-time algorithm, as follows. A language L is in $\mathcal{BPP}$ if there exists a polynomial-time randomized Turing Machine that accepts each $x \in L$ with probability at least 2/3; and rejects each $x \notin L$ with probability at least 2/3. We can think of the outcomes of the computation as follows: if the Turing machine accepts x this means that "x is probably in L", whereas if it rejects, that means that "x is probably not in L". Note that the choice of the number 2/3 in the definition was rather arbitrary: if we run the algorithm k times independently, and take the majority decision, we can decrease the probability of error exponentially in k. If k is fairly large, then one can accept the answer given by the algorithm without any reasonable shadow of doubt. (The letters BPP stand for a Probabilistic Polynomial-time algorithm with probabilities Bounded away from 1/2.)

Other problems not known to be in $\mathcal{P}$ for which there is a randomized polynomial-time algorithm include computing the square root of an integer x modulo a prime p, and deciding whether the determinant of a matrix whose entries are multivariate polynomials is the zero polynomial. The algorithm for the latter problem assigns random values to the variables in the polynomials, and computes the resulting determinant (of numbers). If this determinant is non-zero, then certainly the determinant of variables is non-zero as a polynomial. On the other hand, if the determinant of variables is non-zero, then a random evaluation will yield a non-zero determinant with very high probability.

The previous idea yields a polynomial-time randomized algorithm for deciding whether a graph has a perfect matching. Tutte has shown that a graph on n nodes has a perfect matching if and only if a certain $n \times n$ matrix whose entries are indeterminates, has a non-zero determinant (see *e.g.*, the book by Lovász & Plummer (1986)). A technique of Lovász (1979) can be supplemented by a "variable-tagging" trick of Karp, Upfal & Wigderson (1986) to obtain a randomized polynomial-time algorithm for the **exact matching problem**, which is defined as follows: given a graph $G = (V, E)$, a subset of the edges $F \subseteq E$ and an integer k, decide if G has a perfect matching M with the property that $|M \cap F| = k$. No deterministic polynomial-time algorithm is known for this problem.

We shall give a sketch of the randomized algorithm for the case of bipartite graphs. Let $G = (U \cup W, E)$ be a bipartite graph with color-classes $U = \{u_1, \ldots, u_n\}$ and $W = \{w_1, \ldots, w_n\}$. Introduce an indeterminate x_{ij} corresponding to each edge $u_i w_j \in E$ and an additional indeterminate y. Now define an $n \times n$ matrix A with entries $a_{ij} = 0$ if $u_i w_j \notin E$; $a_{ij} = x_{ij} y$ if $u_i w_j \in F$; and $a_{ij} = x_{ij}$ otherwise. Clearly, the required perfect matching in G exists if and only if the coefficient of the term y^k in the determinant of A is non-zero. The randomized algorithm for deciding the

exact matching problem works as follows: choose values for the indeterminates x_{ij} at random and evaluate the resulting determinant, which has terms that are polynomials in the single variable y. Note that such a determinant can be evaluated using Gaussian elimination. If the resulting (univariate) polynomial has a y^k term then G certainly has an exact matching, and otherwise it probably does not.

Notice that the above randomized algorithms have a property stronger than required by the formal definition. In the randomized algorithm for primality testing, the conclusion that N is not a prime was certain; uncertainty arose only in the case of the conclusion, "N is probably prime". In the case of the exact matching algorithm, if the right term of the polynomial is non-zero, then we *know* that the required matching exists. The complexity class $\mathcal{RP}$ is defined to reflect this asymmetry. A language L is in $\mathcal{RP}$ if there exists a polynomial-time randomized Turing Machine RM such that each input that RM can accept (along any computation path) is in L and for each input $x \in L$, the probability that RM accepts x is at least $1/2$. Note again that the choice of the number $1/2$ is arbitrary.

The above mentioned algorithms show that primality testing is in $co - \mathcal{RP}$, and the exact matching problem is in $\mathcal{RP}$. There are very few problems known to be in $\mathcal{BPP}$ but not in $\mathcal{RP}$ or $co - \mathcal{RP}$. Bach, Miller & Shallit (1986) provided the first "natural" examples of problems in $\mathcal{BPP}$ that are not obviously in $\mathcal{RP}$ or $co - \mathcal{RP}$. They proved that the set of perfect numbers is in $\mathcal{BPP}$. (A natural number N is **perfect** if the sum of all its natural divisors is $2N$; for example, 6 is perfect.)

In some sense an $\mathcal{RP}$ algorithm is more satisfying than a $\mathcal{BPP}$ algorithm, since at least one of the two conclusions reached can be claimed with certainty. A randomized algorithm that *never makes mistakes* would be even more desirable. This can be defined in the following way: an algorithm is a **Las Vegas** algorithm accepting a language L, if given an input x, the algorithm either decides (correctly) that x is in L, or it decides (correctly) that x is not in L, or it halts without coming to a conclusion, and the probability of the third outcome is less than $1/2$ for every input x. It is easy to see that a language L is in $\mathcal{RP} \cap co - \mathcal{RP}$ if and only if there exists a polynomial-time Las Vegas algorithm to decide membership in L. Observe also that if we repeat any Las Vegas algorithm until it gives an answer, the resulting algorithm always gives the correct answer and for any input, it is expected to run in polynomial time. Recent results of Goldwasser & Kilian (1986) and Adleman & Huang (1987) yield a sophisticated Las Vegas algorithm for testing primality.

3 Shades of Intractability

In this section we will consider many computational problems, and see that the universe does not appear to be divided simply into tractable and intractable problems. Current evidence suggests that there are a variety of different classes of problems, each characterizing its own particular shade of intractability. Much of the work in complexity theory is aimed at understanding the correct framework in which to place these problems.

The subsections here reflect three types of approaches for characterizing the difficulty of these problems. The nicest sort of result places absolute limits on our ability to solve problems; for example, the most severe limit is to show that a problem is undecidable, and among decidable problems there are only a handful that can be proven intractable, in the sense that they require a certain

(non-trivial) amount of time or space to be solved. Much more common is to provide completeness results to show that particular problems are the most difficult problems within a given complexity class. If the class contains a great number of problems not known to be solvable with more modest resources, this provides evidence that the problem is intractable. We have already discussed the notion of $\mathcal{NP}$-completeness, and we will also describe several other variants of this approach. Finally, in order to better understand a problem it has frequently been useful to strengthen the basic Turing machine model in order define complexity classes that better characterize the problem. Such an alternative view has often made problems appear less intractable; the subsections on the polynomial-time hierarchy and on randomized proofs present results in this direction.

Undecidability

Not all decision problems can be solved by a Turing machine. The understanding of this inherent limitation on the power of computation was an outgrowth of results in mathematical logic. In particular, the first incompleteness theorem of Gödel (1931) contained the first undecidability result and provided many of the essential ideas that would be used by Church, Post and Turing in their groundbreaking work on the nature of computation. In particular, Turing (1937) proposed what we call a Turing machine, and this enabled the discussion to be conveniently directed towards computation.

At the core of all of these results is Gödel's notion of encoding theorems as strings in some uniform way. Analogously, a Turing machine can be encoded as a string by first specifying the number of states that the machine has, followed by a list of all of the allowed transitions. Each such string can also be interpreted as an integer by using a binary encoding. Thus, each integer i represents a Turing machine M_i. Turing showed that the language $L_{hp} = \{i \mid M_i \text{ halts on input } i\}$ is not solvable by a Turing machine. His proof that this **halting problem** is undecidable uses the following diagonalization argument. Suppose that L_{hp} were solvable by a Turing machine M. Build another machine M' that first uses M to decide if the input $i \in L_{hp}$; if it is, then M' enters an infinite loop, and if not, M' halts. Of course, M' must be M_k for some integer k. But M' and M_k cannot accept the same language, since M' halts on k if M_k does not, and vice versa.

A problem L_1 (many-one) **reduces** to a problem L_2 if there exists a function f computable by a Turing machine such that $x \in L_1$ if and only if $f(x) \in L_2$. Note that if L_1 is undecidable and L_1 reduces to L_2, then L_2 must also be undecidable. This provides a strategy for proving additional undecidable problems. As a simple example, consider the language $L_e = \{i \mid M_i \text{ accepts on an empty input}\}$. It is a simple exercise to convert the description of a given Turing machine M_i to the description of another (rather trivial) machine $M' = M_j$ that on every input, first runs M_i with input i and accepts if M_i halts on i. Clearly, M_j accepts an empty input if and only if M_i halts on i.

A similar strategy can be used to prove Gödel's undecidability theorem in the context of Turing computability, although the details of the reduction are more involved. The theory of arithmetic for non-negative integers with addition and multiplication can be defined as follows. Consider first-order formulas that can be constructed from variables and the constants 0 and 1 with the logical connectives $\neg$, $\vee$, $\wedge$, $\rightarrow$, $\exists$, and $\forall$, along with the operations $\cdot$,+ and the binary relations $=$ and $<$. A sentence is a formula in which all of the variables are bound. We consider the standard model of number theory (as defined by the Peano axioms). A sentence is **provable** if it can be deduced from these axioms. The theory of arithmetic $L_a = (\mathbf{Z}_+, +, \cdot, =, <, O, 1)$ is the collection of provable

sentences. A relatively straightforward construction shows that $L_e \propto L_a$, which yields the following fundamental result.

(3.1) Theorem. *L_a is undecidable.*

This theorem implies the incompleteness of this model of number theory, *i.e.*, there are sentences such that neither it nor its negation is provable. Every complete model is decidable, since a Turing machine can generate all possible deductions and stop if either the statement or its negation is proved.

These results were only the first steps in a rich area of research that can be viewed as the ancestor of modern-day complexity theory. One of its pinnacles of achievement is the solution of Hilbert's tenth problem, which asked for a procedure to decide if a given multivariate polynomial has an integer-valued root. The proof by Matijasevič (1970) that, in fact, this problem is undecidable was the final step in the long history of this problem. (For an introduction to this history and the many results on which this proof builds, the reader is referred to the survey of Davis (1977).)

Evidence of intractability: $\mathcal{NP}$-completeness

In the previous section, we described a way to provide evidence that a given problem in $\mathcal{NP}$ does not have an efficient algorithm. If a language L is shown to be $\mathcal{NP}$-complete, then it is a "hardest" problem in $\mathcal{NP}$, and since we believe that $\mathcal{P} \neq \mathcal{NP}$, this implies that there does not exist a polynomial-time algorithm to solve it. To show that some $\mathcal{NP}$ problem L is $\mathcal{NP}$-complete, we must show that each language $L' \in \mathcal{NP}$ polynomial-time reduces to L. We shall show that a natural problem from logic is $\mathcal{NP}$-complete.

A Boolean formula in conjunctive normal form is the conjunction (and) of clauses $C_1, \ldots, C_s$, each of which is a disjunction (or) of literals $x_1, \bar{x}_1, \ldots, x_t, \bar{x}_t$, where each x_i is a Boolean variable and $\bar{x}_i$ denotes its negation. In the **satisfiability problem** (SAT) we are given such a Boolean formula, and asked to decide if there exists a truth assignment for the variables such that the formula evaluates to true.

(3.2) Theorem. (Cook, Levin) *The satisfiability problem is $\mathcal{NP}$-complete.*

Proof: It is easy to see that the satisfiability problem is in $\mathcal{NP}$, since we interpret the first t cells of the guess tape as providing the assignment, and then it is a simple matter to evaluate the formula for that assignment in polynomial time.

Next, we must show that for any language $L \in \mathcal{NP}$ there exists a polynomial-time function f that maps each instance x of the original problem into a Boolean formula such that $x \in L$ if and only if $f(x)$ is satisfiable. In other words, for any polynomial-time nondeterministic Turing machine M with input x, we must construct a Boolean formula such that the existence of a good guess that leads to acceptance is simulated by the existence of a satisfying truth assignment for $f(x)$. Before giving the reduction, we first argue that M may be assumed to have a form somewhat simpler than the original definition. Imagine that the Turing machine has only one tape, which serves both as the input tape and the work tapes. A simple construction shows that if there exists a nondeterministic Turing machine M with running time $T(n)$, then there exists a nondeterministic machine of this simpler form that finishes within time $T^2(n)$ (by enlarging the alphabet so that each symbol denotes one character on all of the tapes of M). Furthermore, we can assume without

loss of generality that for some l the machine M runs for exactly time n^l on every input of length n.

Let M be such a simplified machine that runs on input x for T steps before halting. We can describe the configuration of the machine at any instant of the computation by giving the contents of the tape, the position of the head and the current state. This can all be encoded as a string by using the alphabet $\Gamma' = \Gamma \times (Q \cup \{!\})$ where the first coordinate gives the contents of a cell of the tape, and the second is '!' unless the head is reading that cell of the tape, when it is the current state. We can then encode the entire computation as a matrix, where each row of the matrix corresponds to one step of the computation, and each column corresponds to a cell of the tape. (Note that there are no more than T cells in either direction of the initial head position that could be reached during the computation.) Acceptance of x by M boils down to the following question: does there exist a guess that causes the matrix to be filled in so that in the last row, the configuration contains an accepting state?

It is straightforward to construct a Boolean formula that represents this question. Let $g_1, \ldots, g_T$ be variables that represent the binary values of the guess tape. Let a_{ijk} represent the contents of the ijth cell of the matrix in the sense that it is 1 if and only if it contains the kth character of Γ'. The formula will be a conjunction of pieces that correspond to the following conditions that we wish to impose: the variables represent some matrix, in that exactly one character is stored in each entry; the first row corresponds to the initial configuration for input x; the last row contains an accepting state; the computation proceeds in the deterministic way specified by the guesses g_i. We will not give each of these in detail, but only sketch the main ideas. The first is easy: for each of the $O(T^2)$ entries, check that at least one of the associated variables is 1 (by their or) and for each pair, check that not both are 1 (by the or of their negations). The second and third are equally routine. The last condition takes a bit more work and is based on a principle of locality: if locally the computation (*i.e.*, the matrix) appears correct, then the entire computation was performed correctly. In fact, it is sufficient to check that each 2×2 submatrix appears correct. Furthermore, it is an easy exercise to encode that some 2×2 submatrix in the ith and $i+1$st rows behaves according to the guess g_i. By taking the conjunction of all $O(T^2)$ such pieces of local information, we enforce that the computation is done correctly. It is now routine to verify that formula constructed is satisfiable if and only if there are guesses that lead M to accept x. □

Now that we have our initial $\mathcal{NP}$-complete problem, we proceed to give a number of reductions to show that several important combinatorial problems are $\mathcal{NP}$-complete. Literally thousands of problems are now known to be $\mathcal{NP}$-complete, so that we will only present a small handful of examples that serve to illustrate an important phenomenon in complexity theory, or relate to important combinatorial problems discussed elsewhere in this chapter, as well as in the rest of this volume. Most of the problems that we consider were shown to be $\mathcal{NP}$-complete in the pioneering work of Karp (1972).

Many restricted cases of the satisfiability problem are also $\mathcal{NP}$-complete. One that is often used in further $\mathcal{NP}$-completeness proofs is the 3-SAT problem, where each clause of the conjunctive normal form must contain exactly three literals. It is a simple task to show that by adding additional variables, longer clauses can be broken into clauses of length three, yielding a new formula that can be satisfied if and only if the original can.

For the **stable set problem**, we are given a graph G and a bound k, and asked to decide if

$\alpha(G) \geq k$; that is, do there exist k pairwise non-adjacent nodes in G? It is easy to see that this problem is in $\mathcal{NP}$, and to show that it is complete, we reduce 3-SAT to it and invoke Theorem (2.5). Given a 3-SAT instance ϕ, we construct a graph G as follows: for each clause in ϕ, let there be 3 nodes in G, each representing a literal in the clause, and let these 3 nodes induce a **clique** (*i.e.*, they are pairwise adjacent); complete the construction by making adjacent any pair of nodes that represent a literal and its negation, and set k to be the number of clauses in ϕ. If there is a satisfying assignment for ϕ, pick one literal from each clause that is given the assignment true; the corresponding nodes in G form a stable set of size k. If there is a stable set of size k, then it must have exactly one node in each clique corresponding to a clause. Furthermore, the nodes in the stable set cannot correspond to both a literal and its negation, so that we can form an assignment by setting to true all literals selected in the stable set, and extending this assignment to the remaining variables arbitrarily. This is a satisfying assignment. This is a characteristic reduction, in that we build **gadgets** to represent the variables and clause structure within the framework of the new problem. The $\mathcal{NP}$-completeness of two other problems follow immediately: the **clique problem**, given a graph G and a bound k, decide if there is a clique in G of size k; and the **node cover problem**, given a graph G and a bound k, decide if there exists a set of k nodes such that every edge is incident to a node in the set.

A very similar construction can be used to show that the **dominating set problem** is $\mathcal{NP}$-complete: given a graph G and an integer k, decide if there is a set S of k nodes such that every node in G is either in S or adjacent to a node in S. A somewhat more complicated reduction transforms 3-SAT into the Hamiltonian circuit problem to show that to be $\mathcal{NP}$-complete. A seemingly slight generalization of bipartite graph matching, the **3-dimensional matching problem**, can be shown to be $\mathcal{NP}$-complete: given disjoint node sets A, B and C, and a collection $\mathcal{F}$ of hyperedges of the form, (a, b, c) where $a \in A$, $b \in B$ and $c \in C$, does there exist a subset of $\mathcal{F}$ such that each node is contained in exactly one edge in the subset?

If we restrict the stable set problem to a particular constant value of k (*e.g.*, is $\alpha(G) \leq 100$?), then this problem can be solved in $\mathcal{P}$ by enumerating all possible sets of size 100. In contrast to this, Stockmeyer (1973) showed that the 3-colorability problem is $\mathcal{NP}$-complete by reducing 3-SAT to it. Let ϕ be a 3-SAT formula. We construct a graph G from it in the following way. First, construct a "reference" clique on three nodes, called true, false and undefined; these nodes will serve as a way of naming the colors in any 3-coloring of the graph. For each variable in ϕ, construct a pair of adjacent nodes, one representing the variable, and one representing its negation, and make them both adjacent to the undefined node. For each clause, $l_1 \vee l_2 \vee l_3$, construct the subgraph shown in Figure 1 below, where the nodes labeled with literals, as well as false (F) and undefined (U), are the nodes already described. It is easy to see that if ϕ has a satisfying assignment, we can get a proper 3-coloring of this graph as follows: color the nodes corresponding literals that are true in the assignment with the same color as is given node true; color analogously the nodes for false literals; and then extend this coloring to the remaining nodes in a straightforward manner. Furthermore, it involves only a little case-checking to see that if the graph is 3-colorable, then the colors can be interpreted as a satisfying assignment.

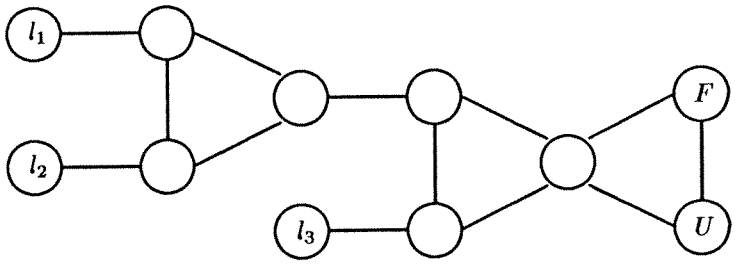

Figure 1:

The **integer programming problem**, defined as follows, is $\mathcal{NP}$-complete: given an $m \times n$ matrix A and an m-vector b, decide if there exists an integer n-vector x such that $Ax \leq b$. In this case, finding a reduction from 3-SAT is trivial: given a formula ϕ, represent each literal by an integer variable bounded between 0 and 1, and for each Boolean variable x, constrain the sum of the variables corresponding to x and its negation to be at most 1. The construction is completed by adding a constraint for each clause that forces the variables for the literals in the clause to sum to at least 1. On the other hand, to show that the problem is in $\mathcal{NP}$ requires more work, involving a calculation that bounds the length of a "smallest" solution satisfying the constraints (if one exists at all).

The subset sum problem defined previously is also $\mathcal{NP}$-complete. This is the first problem that we have encountered that is a "number problem" in the sense that it is not the combinatorial structure, but rather the numbers that make this problem hard. If the numbers are given in unary, there is a polynomial-time algorithm (such an algorithm is called **pseudo-polynomial**): keep a (large, but polynomially bounded) table of all possible sums obtainable using a subset of the first i numbers; this is trivial to do for $i = 1$, and it is not hard to efficiently find the table for $i + 1$ given the table for i; the table for $i = n$ gives us the answer. There are "number problems" that remain $\mathcal{NP}$-complete, even if the numbers are encoded in unary; such problems are called **strongly $\mathcal{NP}$-complete.** One example is the **3-partition problem**: given a set S of the $3n$ integers that sum to nB, does there exist a partition of S into n sets, $T_1, \ldots, T_n$, where each T_i has 3 elements that sum to B?

Consider a weighted generalization of the exact matching problem, where each edge of a bipartite graph is given a weight w_{ij} and we wish to decide if there is a perfect matching of total weight exactly W. It is easy to see that the randomized algorithm given for the unweighted case extends to the weighted case, if the weights are given in unary. (Generalize the previous construction to have a factor of y^w for each edge of weight w.) On the other hand, for weights in binary, there is a straightforward reduction from the subset sum problem: build a disjoint copy of $K_{2,2}$ for each number s in our subset sum instance, where the weights are such that one perfect matching of the subgraph has weight s, whereas the other has weight 0. It is easy to see that the union of these subgraphs has a perfect matching of weight B, if and only if some subset of the numbers of the instance of the subset set problem total exactly B.

The Complexity Class $\mathcal{NP} \setminus \mathcal{P}$?

If we are to believe the conjecture that $\mathcal{P} \neq \mathcal{NP}$, then there exists a non-empty complexity class $\mathcal{NP} \setminus \mathcal{P}$. One might ask the question: is it true that every problem in $\mathcal{NP}$ is either $\mathcal{NP}$-complete or in $\mathcal{P}$? If $\mathcal{P} = \mathcal{NP}$ this question has a (trivial) affirmative answer, but a negative answer to it (under the assumption the $\mathcal{P} \neq \mathcal{NP}$) might help explain the reluctance of certain problems to be placed in one of those two classes. In fact, Ladner (1975a) has shown, under the assumption that $\mathcal{P} \neq \mathcal{NP}$, that there is an extremely refined structure of equivalence between the two classes, $\mathcal{P}$ and $\mathcal{NP}$-complete.

(3.3) Theorem. *If L_1 is decidable and $L_1 \notin \mathcal{P}$, then there exists a decidable language L_2 such that $L_2 \notin \mathcal{P}$, $L_2 \propto_p L_1$ but $L_1 \not\propto_p L_2$.*

Note that if L_1 is $\mathcal{NP}$-complete, the language L_2 is "in between" the classes $\mathcal{P}$ and $\mathcal{NP}$-complete, if $\mathcal{P} \neq \mathcal{NP}$, and by repeatedly applying the result we see that there is a whole range of seemingly different complexity classes. Under the assumption that $\mathcal{P}$ is different from $\mathcal{NP}$, do we know of any candidate problems that may lie in this purgatory of complexity classes? The answer to this is "maybe". We will give four important problems that have not been shown to be either in $\mathcal{P}$ or $\mathcal{NP}$-complete. The problem for which there has been the greatest speculation along these lines is the **graph isomorphism problem**: given a pair of graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, decide if there is a bijection $\sigma : V_1 \mapsto V_2$ such that $ij \in E_1$ if and only if $\sigma(i)\sigma(j) \in E_2$. Later in this section we will provide evidence that it is not $\mathcal{NP}$-complete, and efforts to show that it is in $\mathcal{P}$ have so far fallen short (see Chapter 27 of this volume). A problem that mathematicians since the ancient Greeks have been trying to solve is that of **factoring** integers; a decision formulation that is polynomially equivalent to factoring is as follows: given an integer N and a bound k, does there exist a factor of N that is at most k? A problem that is no harder than factoring is the **discrete logarithm problem**: given a prime p, a generator g and an natural number $x < p$, find l such that $g^l \equiv x \bmod p$. Finally, there is the **shortest vector problem**, where we are given a collection of integer vectors, and we wish to find the shortest vector (in the Euclidean norm) that can be represented as a non-zero integral combination of these vectors. Here, current evidence makes it seem likely that this problem is really $\mathcal{NP}$-complete; related work is discussed elsewhere in this volume (see Chapter 19).

It is important to mention that there is an important subclass of $\mathcal{NP}$ which may also fall in this presumed gap. Edmonds' class of well-characterized problems, $\mathcal{NP} \cap co-\mathcal{NP}$, certainly contains $\mathcal{P}$ and is contained in $\mathcal{NP}$. Furthermore, unless $\mathcal{NP} = co-\mathcal{NP}$, it cannot contain any $\mathcal{NP}$-complete problem. On the other hand, the prevailing feeling is that showing a problem to be in this class is a giant step towards showing that the problem is in $\mathcal{P}$. A result of Pratt (1975a) shows that primality is in $\mathcal{NP}$, so it lies in $\mathcal{NP} \cap co - \mathcal{NP}$, though as discussed above, there is additional evidence that it lies in $\mathcal{P}$. The factoring problem, which appears to be significantly harder, also lies in $\mathcal{NP} \cap co-\mathcal{NP}$, since a prime factorization can be guessed along with certificate that each of the factors is indeed prime. One interesting open question connected with $\mathcal{NP} \cap co-\mathcal{NP}$ is concerned with the existence of a problem that is complete for this class. One might hope that there is some natural problem that completely characterizes the relationship of $\mathcal{NP} \cap co - \mathcal{NP}$ with $\mathcal{P}$ and $\mathcal{NP}$ (in the same manner that 3-SAT characterizes the $\mathcal{P} = \mathcal{NP}$ question).

One approach to shed light on the complexity of a problem that is not known to be either in $\mathcal{P}$ or

$\mathcal{NP}$-complete, has been to consider weaker forms of completeness for $\mathcal{NP}$. In fact, Cook's original notion of completeness, though technically a weaker definition of intractability, is no less damning. $L_1 \propto_p L_2$, can be thought of as solving L_1 by a restricted kind of subroutine call, where first some polynomial-time preprocessing is done, and then the subroutine for L_2 is called once. Cook (1971) proposed a notion of reducibility, where L_1 is solved by using a polynomial-time Turing machine that can, in one step, get an answer to any query of the form, "is $x \in L_2$?" Such a more general Turing machine is called an **oracle machine**, and in the next subsection, we will discuss Turing machines and complexity classes relative to an oracle A in more detail. Note that any complete language L with respect to this reducibility still has the property that $L \in \mathcal{P}$ if and only if $\mathcal{P} = \mathcal{NP}$. Karp (1972) focused attention on the notion $\propto_p$, and was able to show that $\mathcal{NP}$-completeness was powerful enough to capture a wide range of combinatorial problems. On the other hand, it remains an open question to show that Cook's notion of reducibility is stronger than Karp's; is there a natural problem in $\mathcal{NP}$ that is complete with respect to "Cook" reducibility, but not with respect to "Karp" reducibility?

Adleman & Manders (1977, 1979) showed that non-determinism and randomization can play a role in defining notions of reducibility. The first notion, γ-reducibility, used nondeterminism in order to provide a stronger notion of a complete problem that still is not in $\mathcal{NP} \cap co-\mathcal{NP}$ unless $\mathcal{NP} = co-\mathcal{NP}$. A randomized notion of completeness was introduced in order prove results of the form that L is in $\mathcal{RP}$ if and only if $\mathcal{RP} = \mathcal{NP}$. They also proved completeness results for several number theoretic problems in $\mathcal{NP}$ that are not known to be $\mathcal{NP}$-complete.

The polynomial-time hierarchy

It is important to realize that not all natural combinatorial problems lie in $\mathcal{NP}$ or $co-\mathcal{NP}$. For example, consider the following **generalized coloring problem**: given input graphs G and H, can we color the nodes of G with two colors so that the graph induced by each color does not contain H as a subgraph? Given the guess of a coloring, there is an exponential number of possible subgraphs that all must be checked, so at least the most obvious approach does not work. Is there a more sophisticated approach that puts this problem or its complement in $\mathcal{NP}$, or is this difficulty inherent?

One can view $\mathcal{NP}$ as the class of languages L that can be expressed in the following way: there exists a language $L' \in \mathcal{P}$ such that $x \in L \Leftrightarrow \exists y$ of length bounded by a polynomial in the length of x such that $(x, y) \in L'$. (We will denote this polynomially bounded quantification by $\exists_p$.) By writing $\mathcal{NP}$ in this way, we are led to a generalization that does contain the above problem. Merely allow an alternation of quantifiers: (G, H) has a coloring if $\exists_p$ a bipartition such that $\forall_p$ subgraphs G' of one part of G, G' is not equal to H. (Note that the specification of G' implicitly contains a bijection to the nodes H.) Of course, one need not stop with one alternation of quantification; this motivates a hierarchy of classes, called the **polynomial-time hierarchy**: $\Sigma_k^{\mathcal{P}} = \{L | \exists L' \in \mathcal{P}$ such that $x \in L$ if and only if $\exists_p y_1 \forall_p y_2 \cdots Q_k y_k(x, y_1, y_2, \ldots, y_k) \in L'\}$, where Q_k is $\exists_p$ if k is odd, and $\forall_p$ if k is even. Thus, generalized coloring is in $\Sigma_2^{\mathcal{P}}$. Furthermore, note that $\Sigma_0^{\mathcal{P}} = \mathcal{P}$ and $\Sigma_1^{\mathcal{P}} = \mathcal{NP}$. We also define $\Pi_k^{\mathcal{P}}$ to be $co-\Sigma_k^{\mathcal{P}}$. Clearly, $\Sigma_k^{\mathcal{P}} \cup \Pi_k^{\mathcal{P}} \subseteq \Sigma_{k+1}^{\mathcal{P}}$.

Alternatively, it is possible to define the polynomial-time hierarchy in terms of oracles, as was done in the original formulation by Meyer & Stockmeyer (1972). An **oracle** Turing machine has a special additional tape, called a query tape, and three special states, q_{ask}, q_{no} and q_{yes}. The oracle Turing machine M^A can ask questions of the form, "is x in A?" by writing x on the query tape,

and entering the state q_{ask}. The machine enters state q_{yes} or q_{no} depending on the outcome to the question. Let $\mathcal{NP}^A$ denote the class of languages recognized by nondeterministic polynomial-time oracle Turing machines with access to an oracle for the language A, and in general, any complexity class $\mathcal{C}$ has its relativized analogue $\mathcal{C}^A$. The notion of the space used by an oracle Turing machine is ambiguous, since it is not clear whether the space used on the query tape should be counted. We shall assume that it is, though this has the disadvantage that $\mathcal{L}^A$ might not contain A. Finally, let $\mathcal{NP}^{\mathcal{C}}$ denote the union of $\mathcal{NP}^A$ over all $A \in \mathcal{C}$. Consider again the generalized coloring problem; it is not hard to see that there is nondeterministic polynomial-time Turing machine to solve it, given an oracle for the following problem A: given G and H, decide if H is a subgraph of G. Since $A \in \mathcal{NP}$, we see that this coloring problem is in $\mathcal{NP}^{\mathcal{NP}}$. In fact, $\Sigma_2^{\mathcal{P}} = \mathcal{NP}^{\mathcal{NP}}$ and in general, $\Sigma_{k+1}^{\mathcal{P}} = \mathcal{NP}^{\Sigma_k^{\mathcal{P}}}$. Unfortunately, for each new complexity class, there is yet another host of unsettled questions.

Open Problem For each $k \geq 1$, is $\Sigma_k^{\mathcal{P}} = \Sigma_{k-1}^{\mathcal{P}}$?

Open Problem For each $k \geq 1$, is $\Sigma_k^{\mathcal{P}} = \Pi_k^{\mathcal{P}}$?

Contained in these, for $k = 1$, are the $\mathcal{P} = \mathcal{NP}$ and $\mathcal{NP} = co - \mathcal{NP}$ questions, and as was true for those questions, one might hope to find complete problems on which to focus attention in resolving these open problems. We define these notions of completeness with respect to polynomial-time reducibility, so that L is complete for $\mathcal{C}$ if and only if $L \in \mathcal{C}$ and all problems in $\mathcal{C}$ reduce ($\propto_p$) to L. As might be expected, analogues of the satisfiability problem which allow a particular number of alternations in the formulae can be used to provide a complete problem for each level of the hierarchy. On the other hand, it is more satisfying to have more natural complete problems, and Rutenberg showed that the generalized coloring problem is, in fact, complete for $\Sigma_2^{\mathcal{P}}$. Another problem of identical complexity is a similarly flavored node deletion problem: given graphs G and H and an integer k, decide if there is a subset of k nodes that can be deleted from G, so that the remaining graph no longer has H as a subgraph.

One piece of good news concerning this infinite supply of open problems, is that their answers may be related. There is a principle of upward inheritability that says that if $\Sigma_l^{\mathcal{P}} = \Pi_l^{\mathcal{P}}$ for some level l, then equality holds for all levels $k \geq l$. In fact, $\Sigma_l^{\mathcal{P}} = \Pi_l^{\mathcal{P}}$ implies that the entire hierarchy collapses to that level; *i.e.*, $\Sigma_k^{\mathcal{P}} = \Sigma_l^{\mathcal{P}}$ for all $k \geq l$. Note that $\mathcal{P} = \mathcal{NP}$ if and only if $\mathcal{P} = \mathcal{PH}$, where $\mathcal{PH} = \cup_{k \geq 0} \Sigma_k^{\mathcal{P}}$.

As we shall see, the polynomial-time hierarchy has helped to provide structure to a number of complexity classes. Perhaps the first result along these lines is due to Sipser (1983b), who used a beautiful "hashing function" technique to show that $\mathcal{BPP}$ is in the polynomial-time hierarchy, and in fact, can be placed within $\Sigma_2^{\mathcal{P}} \cap \Pi_2^{\mathcal{P}}$.

Evidence of intractability: $\#\mathcal{P}$-completeness

Consider the **counting problem** of computing the number of perfect matchings in a bipartite graph. If this is cast as a decision problem, it does not appear to be in $\mathcal{NP}$, since it seems that the number of solutions can only be certified by writing down possibly exponentially many matchings. However, consider modifying the definition of $\mathcal{NP}$ to focus on the *number* of good certificates, rather than the existence of one; let $\#\mathcal{P}$ be the class of problems for which there exists a language $L' \in \mathcal{P}$ and a polynomial $p(n)$ such that for any input x, the only acceptable output is $z = |\{y : |y| = p(|x|) \text{ and } (x, y) \in L'\}|$. Clearly, the problem of counting perfect matchings in a

bipartite graph is in $\#\mathcal{P}$. Any problem in $\#\mathcal{P}$ can be computed using polynomial space, since all possible certificates y may be tried in succession[1].

We can also define a notion of a complete problem for $\#\mathcal{P}$. To do this, we use a reducibility analogous to that used by Cook. Let $P_i, i = 1, 2$ denote a counting problem, where for an instance x, the number of solutions is $P_i(x)$. The problem P_1 reduces to P_2 if there exists a polynomial-time Turing machine that can compute P_1 if it has access to an oracle that, given x can compute $P_2(x)$ in one step. We define a counting problem to be **$\#\mathcal{P}$-complete** if it is in $\#\mathcal{P}$, and all problems in $\#\mathcal{P}$ reduce to it. A weaker notion of reducibility, analogous to the notion used by Karp, is called **parsimonious**: an instance x of P_1 is mapped in polynomial time to $f(x)$ for P_2, such that $P_1(x) = P_2(f(x))$. For either notion of reducibility, we see that any polynomial-time algorithm for P_2 yields a polynomial-time algorithm for P_1. It is not hard to see that the proof of Cook's theorem shows that the problem of computing the number of satisfying assignments of a Boolean formula in conjunctive normal form is $\#\mathcal{P}$-complete. Furthermore, by being only slightly more careful, it is easy to give parsimonious modifications of the reductions to the clique problem, the Hamiltonian circuit problem, and seemingly any $\mathcal{NP}$-complete problem, and so the counting versions of $\mathcal{NP}$-complete problems can be shown to be $\#\mathcal{P}$-complete.

Surprisingly, not all $\#\mathcal{P}$-complete counting problems need be associated with an $\mathcal{NP}$-complete problem. Computing the number the perfect matchings in a bipartite graph, (or equivalently, computing the **permanent** of a 0,1 matrix) is a counting version of a problem in $\mathcal{P}$, the perfect matching problem, and yet Valiant (1979) showed that this problem is complete. This made it possible to prove a variety of counting problems to be $\#\mathcal{P}$-complete that are related to polynomially solvable decision problems.

There are many problems that are essentially $\#\mathcal{P}$-complete, although they do not have the appearance of a counting problem. An important practical example is the problem of computing the reliability of a network: given a graph G with a source s and a sink t, and probabilities $p(e)$ that edge e will fail, what is the probability that s becomes disconnected from t? In practice, it may be more important to estimate this value, rather than determining it precisely, and work of Karp & Luby (1985) provides algorithms to do this.

Another problem that has been shown to be intimately connected with the estimation of the value of counting problems is that of uniformly generating combinatorial structures. As an example, suppose that a randomized algorithm requires a perfect matching in a bipartite graph G to be chosen uniformly from the set of all perfect matchings in G; how can this be done? Jerrum, Valiant & Vazirani (1986) have shown that a relaxed version of this problem, choosing perfect matchings that are selected with probability within an arbitrarily small constant factor of the uniform value, is equivalent to the problem of estimating the number of perfect matchings (using a randomized algorithm). In fact, their result carries through to most counting/generation problems related to problems in $\mathcal{NP}$, since it requires only a natural self-reducibility property, that says that an instance can be solved by handling some number of smaller instances of the same problem.

Stockmeyer (1985) has provided insight into estimating the value of $\#\mathcal{P}$ problems, by trying to place these problems within the polynomial-time hierarchy. Using Sipser's hashing function technique, Stockmeyer showed that for any problem in $\#\mathcal{P}$ and any fixed values $\epsilon, d > 0$, there exists a polynomial-time Turing machine with a $\Sigma_2^{\mathcal{P}}$-complete oracle that computes a value that is

[1] A recent result of Toda gives new evidence of the intractibility of this class by showing the $\mathcal{PH} \subseteq \mathcal{P}^{\#\mathcal{P}}$.

within a factor of $(1 + \epsilon n^{-d})$ of the correct answer.

Evidence of intractability: $\mathcal{PSPACE}$-completeness

In this subsection, we turn to the question of the space complexity of problems. When we discuss space complexity, we may assume that the Turing machine has only one work tape (and a separate input tape), since by expanding the tape alphabet, any number of tapes can be simulated by one tape without using more space.

It is not hard to see that $\mathcal{PSPACE}$ must contain the entire polynomial-time hierarchy. For any $L \in \Sigma_k^{\mathcal{P}}$, $\exists L' \in \mathcal{P}$ such that $x \in L$ if and only if $\exists_p y_1 \forall_p y_2 \cdots Q_k y_k (x, y_1, y_2, \ldots, y_k) \in L'$, and so all possible $y_1, \ldots, y_k$ can be tried in succession, to see if the condition specified by the alternating quantifiers is satisfied. Similarly, $\mathcal{PSPACE}$ contains $\#\mathcal{P}$, since all computation paths of a polynomial-time counting machine may be tried in succession. We remarked when introducing $\mathcal{PSPACE}$ that it was not an oversight that $\mathcal{NPSPACE}$ was not defined, since $\mathcal{PSPACE} = \mathcal{NPSPACE}$. This result is a special case of the following.

(3.4) Theorem. (Savitch) *If L is accepted by a nondeterministic Turing machine using space $S(n) \geq \log n$, then it is accepted by a deterministic Turing machine using space $S(n)^2$.*

Proof: The proof is based on the idea of bounding the number of configurations of the Turing machine and using a natural divide-and-conquer strategy. In any given computation, a nondeterministic Turing machine M can be completely described by specifying the input head position, the contents of the work tape, the work tape head position and the current state. If M uses $S(n)$ space, then there are no more than $d^{S(n)}$ configurations, for some constant d, and one can be written down in $S(n)$ space. Consequently, we can assume that M halts within $d^{S(n)}$ steps, since we can keep a counter of the number of steps taken and halt if that many are done.

To simulate M by a deterministic machine, we build a procedure that recursively calls itself to check for any configurations I and K, and a bound l, whether M, when started in I, can reach K within 2^l steps. There is such a computation if there exists a midpoint of the computation J such that I can lead to J and J can lead to K each within 2^{l-1} steps. The existential quantifier can be implemented by merely trying all possible J in some specified order. The basis of this recursion is the case $l = 0$, where we merely need to know if $I = K$, or if one transition leads from I to K, and this can be easily checked. In implementing this procedure, we need to keep track of the current midpoint at each level of the recursion, and so we need $lS(n)$ space to do this. Thus, by running the procedure with the initial configuration and every accepting configuration with $l = S(n) \log d$, we get a deterministic procedure that accepts L and uses space $O(S(n)^2)$. □

The idea of "guessing a midpoint" is also the main idea used to derive a $\mathcal{PSPACE}$-complete problem, where completeness is defined with respect to polynomial-time reductions. The problem of the validity of quantified Boolean formulae is as follows: given a formula in first-order logic in prenix form, $\exists x_1 \forall x_2 \cdots Q_k x_k \phi(x_1, \ldots x_k) = \mathsf{true}$, decide if it is valid. The proof is a mixture of Savitch's theorem and Cook's theorem. We use the alternation of existential and universal quantifiers to capture the notion of the existence of a midpoint such that both (for all) the first and second halves of the computation are legitimate. The basis of the recursion is now solved by building a Boolean formula to express that two configurations are either the same or one is the result of a single transition from the other.

An instance of the previous problem can be viewed as a game between an existential player and a universal player; the existential player gets to choose values for x_1, then the universal player chooses for x_2, and so on. The decision question amounts to whether the first player has a strategy so that ϕ must evaluate to **true**. There are many other $\mathcal{PSPACE}$-complete problems known, and most of these have a game-like flavor. An example of a more natural $\mathcal{PSPACE}$-complete game is the **Shannon switching game**: given a graph G with two nodes s and t, two players alternately color the edges of G, where the first player, coloring with red, tries to construct a red path from s to t, whereas the second player, coloring with blue, tries to construct a blue (s,t)-cut; does the first player have a winning strategy for G?

Another problem that has a simple $\mathcal{PSPACE}$-completeness proof deals with determining if two nodes are connected in a directed graph G (of order 2^n) that is given by a circuit with $2n$ inputs where the first n specify in binary a node i and the second n specify a node j, and the output of the circuit is 1 if and only if (i,j) is an arc of G. If L is accepted by M in polynomial space, L can be reduced to this **circuit-based directed reachability problem**, by letting the nodes of G correspond to configurations of M, where arcs (i,j) correspond to possible transitions of M. It is easy to construct in polynomial time a (polynomial-size) circuit that decides if one configuration follows another by one transition of M. Since it easy to massage M so that it only has one accepting configuration, we have reduced the acceptance of the input x by M to whether the initial configuration of x is connected to the accepting configuration.

The role of games in $\mathcal{PSPACE}$-completeness suggests a new type of Turing machine, called an **alternating Turing machine**, which was originally proposed by Chandra, Kozen & Stockmeyer (1981). Consider a computation to be a sequence of moves made by two players, an existential player and a universal player. The current state indicates whose move it is, and in each configuration the specified player has several moves from which to choose. Each computation path either accepts or rejects the input. The input is accepted if the existential player has a winning strategy, that is, if there is a choice of moves for the existential player, so that for any choice of moves by the universal player, the input is accepted. For simplicity, assume again that each computation path has the same number of moves. As before, the time to accept an input x is this number of moves, and the space needed to accept x is the maximum space used on any computation path. Observe that a nondeterministic machine is an alternating machine where only the existential player moves. It is easy to see that the lth level of the polynomial-time hierarchy can be defined in terms of polynomial-time alternating Turing machines where the number of "alternations" between existential and universal quantifiers is less than l.

The role of $\mathcal{PSPACE}$ in the definition of these machines suggests that alternating polynomial time is closely related to $\mathcal{PSPACE}$ and indeed this is just a special case of a general phenomenon. Let $ATIME(T(n))$ and $ASPACE(S(n))$ denote the classes accepted by an alternating Turing machine with $O(T(n))$ time and $O(S(n))$ space, respectively. Chandra, Kozen & Stockmeyer proved two fundamental results characterizing the relationship of alternating classes to deterministic and nondeterministic ones. Note that the first result implies Savitch's theorem, and in fact, the proof of the last inclusion of Theorem (3.5) is very similar to the proof of Savitch's theorem.

(3.5) Theorem. *If* $T(n) \geq n$, *then*

$$ATIME(T(n)) \subseteq DSPACE(T(n)) \subseteq NSPACE(T(n)) \subseteq ATIME(T(n)^2)$$

(3.6) Theorem. *If* $S(n) \geq \log n$, *then*

$$ASPACE(S(n)) = \cup_{c>0} DTIME(c^{S(n)})$$

Among the consequences of these results, we see that $\mathcal{AP} = \mathcal{PSPACE}$. In fact, one can view an alternating Turing machine as extremely idealized parallel computer, since it can branch off an unbounded number of parallel processes that can be used in determining the final outcome. Therefore, one can consider these results as a proven instance of the **parallel computation thesis:** parallel time equals sequential space (up to a polynomial function).

Proof of intractability

It is, of course, far preferable to *prove* that a problem is intractable, rather than merely giving evidence that supports this belief. Perhaps the first natural question is, are there *any* decidable languages that require more time than $T(n)$? The diagonalization techniques used to show that there are undecidable languages can be used to show that for any Turing computable $T(n)$, there must exist such a language; consider the language L of all i such that if i is run on the Turing machine M_i that i encodes, either it is rejected, or it runs for more than $T(n)$ steps. It is easy to see that L is decidable, and yet no Turing machine that always halts within $T(n)$ steps can accept L. Using our stronger assumption about $T(n)$ (full time-constructibility) we are able to define such a language that is not only decidable, but can be recognized within only slightly more time than $T(n)$. The additional logarithmic factor needed in Theorem (3.7), which combines results of Hartmanis & Stearns (1965) and Hennie & Stearns (1966), is due to the fact that the fastest known simulation of a (multitape) Turing machine by a machine with some constant number of tapes slows down the machine by a logarithmic factor.

(3.7) Theorem. *If* $\liminf_{n\to\infty} \dfrac{T_1(n)\log T_1(n)}{T_2(n)} = 0$, *then there exists* $L \in DTIME(T_2) \setminus DTIME(T_1)$.

Since any multitape machine can be simulated by a 1-tape machine without using any additional space, the corresponding space hierarchy theorem does not require the logarithmic factor. Seiferas, Fischer & Meyer (1978) have proved an analogous, but much more difficult, hierarchy theorem for nondeterministic time.

(3.8) Theorem. *If* $\liminf_{n\to\infty} \dfrac{T_1(n+1)}{T_2(n)} = 0$, *then there exists* $L \in NTIME(T_2) \setminus NTIME(T_1)$.

Although these theorems are non-constructive, Meyer & Stockmeyer (1972) developed the following strategy that makes use of completeness results in order to prove lower bounds on particular problems. Intuitively, a completeness result says that a problem is a "hardest" problem for some complexity class, and Theorems (3.7) and (3.8) can be used to show that certain complexity classes have provably hard problems. Consequently, these two pieces imply that the complete problem is provably hard.

As an example, consider the **circuit-based large clique problem**, L_{cc}, which is the problem analogous to the circuit-based directed reachability problem, that tests whether the (compactly represented) input graph on $N = 2^n$ nodes has a clique of size $N/2$. This problem is complete for the class $\mathcal{NEXPTIME} = \cup_k NTIME(2^{n^k})$ with respect to polynomial-time reducibility. This can be seen by introducing the circuit-based version of satisfiability; a formula is represented by a polynomial-size circuit that outputs 1 whenever the first set of inputs gives the binary representation of the index of a variable that occurs in the clause specified by the remainder of the inputs. The generic reduction of Cook's theorem translates exactly to show that circuit-based satisfiability is $\mathcal{NEXPTIME}$-complete, and the completeness of L_{cc} follows by using essentially the same reduction used to show the $\mathcal{NP}$-completeness of the ordinary clique problem. Given this completeness result, it is easy to prove the following theorem.

(3.9) Theorem. *There exists a constant $c > 0$ such that $L_{cc} \notin NTIME(2^{n^c})$.*

Proof: Let L be the language in $NTIME(2^{n^2}) - NTIME(2^n)$ specified by Theorem (3.8). Since $L \in \mathcal{NEXPTIME}$, $L \propto_p L_{cc}$. Let n^k be the time bound for this reduction. Choose $c = 1/k$, and now suppose that $L_{cc} \in NTIME(2^{n^c})$. We can decide if a string x of length n is in L by first applying the reduction to get a string of length at most n^k, and then using the Turing machine that decides L_{cc} within the assumed bound. This composite nondeterministic procedure runs in time $n^k + 2^{n^{kc}} = O(2^n)$, and this contradiction proves the theorem. □

One interpretation of this result is that there exist graphs specified by circuits such that proofs that these graphs have a large clique require an exponential number of steps (in terms of the size of the circuit). Observe that we would have been able to prove a stronger result if we would have had a better bound on the length of the string produced by the reduction. Lower bounds proved using this strategy are often based on completeness with respect to reducibilities that are further restricted to produce an output of length bounded by a linear function of the input length.

This strategy has been applied primarily to problems from logic and formal language theory. A result of Fischer & Rabin (1974) that contrasts nicely with Theorem (3.1) concerns L_{pa}, the language of all provable sentences in the theory of arithmetic for natural numbers without multiplication, which was shown to be decidable by Presburger (1929).

(3.10) Theorem. *There is a constant $c > 0$ such that $L_{pa} \notin NTIME(2^{2^{cn}})$.*

A representative sample of problems treated in this fashion is surveyed by Stockmeyer (1987).

Extensions of $\mathcal{NP}$: short proofs via randomization

In the same way that randomized algorithms give rise to an extended notion of efficiently computable, randomized proofs give rise to an extension of $\mathcal{NP}$, the class of languages for which membership is efficiently provable. Randomized proofs can be thought of as efficiently proving membership in a language by giving overwhelming statistical evidence, rather than proving with certainty. In contrast to the extensions of $\mathcal{NP}$ discussed previously, this notion will give rise to complexity classes much closer to the spirit of $\mathcal{NP}$, and can be thought of as being "just above $\mathcal{NP}$". In creating this branch of complexity theory, Goldwasser, Micali & Rackoff (1985) and Babai (1985) have given definitions to capture related notions of proof based on statistical evidence. We

will give a brief overview of this area, and the interested reader is directed to the surveys by Goldwasser (1989) and Johnson (1988) or the introduction to the paper by Babai & Moran (1988).

Although the graph isomorphism problem is in $\mathcal{NP}$, it is not known to be in $co-\mathcal{NP}$. Goldreich, Micali & Wigderson (1986) provide a way to "prove" that two graphs are not isomorphic, using randomization. Suppose that King Arthur has two graphs G_1 and G_2, and Merlin wants to convince him that the two graphs are not isomorphic. Merlin can do this in the following way: Merlin asks Arthur to choose one of the two graphs, randomly relabel the nodes, and show him this random isomorphic copy of the chosen graph; Merlin will tell which graph Arthur chose. If the two graphs are isomorphic, then Merlin has only a fifty percent chance of choosing the right graph, assuming that he cannot read Arthur's mind. If Merlin can successfully repeat this test several times, then Arthur can fairly safely conclude that Merlin can distinguish isomorphic copies of the two graphs; in particular, the two graphs are not isomorphic.

The above randomized proof is a prime example of the interactive proof defined by Goldwasser, Micali & Rackoff (1985). An **interactive protocol** consists of two Turing machines: the Prover (Merlin) and the Verifier (Arthur). The Prover is trying to convince the Verifier that a certain word x is in a language L. The Prover has unrestricted power; the Verifier is restricted to be a randomized polynomial-time Turing machine. The two machines have a common input tape, and each has its own private randomizing tape and work tapes. The two machines operate in turns; after each machine's turn, it writes its output on a special communication tape, and the other machine starts its turn by reading this message. At the end of the protocol, the Verifier announces whether or not it accepts x. A language L has an **interactive proof system** if there exists an interactive protocol such that

- if $x \in L$, then the Verifier accepts with probability at least $2/3$ (*i.e.*, if $x \in L$, then the Prover can "prove this");
- if $x \notin L$, then for any Turing machine used in place of the Prover, the Verifier rejects with probability at least $2/3$ (*i.e.*, if $x \notin L$, then no cheating Prover can convince the Verifier of the opposite).

Let $\mathcal{IP}$ denote the class of languages that have an interactive proof system with a polynomial number of turns. Note that the choice of probability $2/3$ is arbitrary, since repeating the protocol k times and choosing the majority decision decreases the probability of error exponentially in k.

There are two important new elements in this proof system. The first is randomization. Languages that can be accepted by a deterministic polynomial-time Verifier are exactly the languages in $\mathcal{NP}$. (The Prover provides the choices for the nondeterministic moves.) The second element is interaction. Standard proofs can be written down on a piece of paper, whereas the transcript of the above communication is not a proof. Interaction provides the framework in which randomization can be exploited.

Babai (1985) has proposed a similar, but seemingly weaker, version of a randomized proof system: **Arthur - Merlin games** can be defined in the same way as interactive proof systems with the additional assumption that Arthur's messages are restricted to be his random coin tosses. This is equivalent to assuming that Arthur's randomizing tape is not private, since if Merlin can see Arthur's random bits, he can do Arthur's computation as well. In fact, Arthur's role in an

Arthur-Merlin game can be restricted to providing the outcomes of the required coin tosses. In this case, we can assume that there is a deterministic polynomial-time Referee that decides whether to accept the input after the communication is over. This last perspective suggests the term "game": Merlin and Arthur play a game to debate if $x \in L$, and the Referee decides who won; Merlin can play optimally, whereas Arthur can make only random moves. For any input, each player has some probability (depending on Arthur's moves) of being selected as the winner. For a game to be an Arthur-Merlin game, it is required that the winning probabilities be bounded away from 1/2.

An Arthur-Merlin game can also be reformulated as a variant of a Turing machine similar to an alternating Turing machine. The state set of the corresponding Turing machine is partitioned into two sets, the existential and the randomizing states. The acceptance probability can be computed by evaluating the computation tree in the following way: the value of an existential node is the maximum value of its children (Merlin can choose the branch with the best probability of acceptance), and the value of a randomizing node is the average of the values of its children.

Earlier, Papadimitriou (1983) defined a similar game, the **game against nature**, without the assumption that the winning probabilities be bounded away from 1/2. In this setting, Arthur and the Referee (by themselves) accept the analogue of $\mathcal{BPP}$ where the probabilities are not bounded away from 1/2. This class is at least as hard as $\#\mathcal{P}$; this shows that relaxing the probability requirements in this way greatly increases the power of the game. Papadimitriou showed that the class of languages defined by games against nature with a polynomial number of moves is equal to $\mathcal{PSPACE}$.

Analogous to the polynomial-time hierarchy, one can define randomized proof hierarchies. For a polynomially bounded function $t(n)$, let $\mathcal{IP}(t(n))$ denote the class of languages accepted by an interactive proof system with less than $t(n)$ alternations of moves, where n is the length of the input. For Arthur-Merlin games, Babai introduced a finer distinction; $\mathcal{AM}(t(n))$ and $\mathcal{MA}(t(n))$ are, respectively, the classes of languages that can be accepted by Arthur-Merlin games with less than $t(n)$ alternations of moves when Arthur moves first, and when Merlin moves first (similar to $\Sigma^{\mathcal{P}}_{t(n)}$ and $\Pi^{\mathcal{P}}_{t(n)}$). Clearly, $\mathcal{AM}(1) = \mathcal{BPP}$ and $\mathcal{MA}(1) = \mathcal{NP}$. We shall use the notation $\mathcal{AM} = \mathcal{AM}(2)$, $\mathcal{MA} = \mathcal{MA}(2)$ and $\mathcal{AM}(poly) = \cup_{k>0}\mathcal{AM}(n^k)$.

One might expect a natural relationship between the levels of the Arthur-Merlin hierarchy and the polynomial-time hierarchy. The simplified proof that $\mathcal{BPP} \subseteq \Sigma^{\mathcal{P}}_2 \cap \Pi^{\mathcal{P}}_2$ due to Lautemann (1983) has been extended by Babai (in the case of constant $t(n)$) and Aiello, Goldwasser & Hastad (1986) (in general) to show that $\mathcal{AM}(t(n)) \subseteq \Pi^{\mathcal{P}}_{t(n)}$ for $t(n) \geq 2$.

It would be natural to conjecture that the hierarchies for both notions of randomized proofs are strict, and that $\mathcal{AM}(k)$ is strictly contained in $\mathcal{IP}(k)$ for every k. (For example, in the graph non-isomorphism proof system given above, note how important it is that Arthur's coins be private.) The first surprise in this area was Babai's proof that the finite levels of the Arthur-Merlin hierarchy collapse.

(3.11) Theorem. *$\mathcal{AM}(t(n)+1) = \mathcal{AM}(t(n))$ for every polynomially bounded $t(n) \geq 2$.*

To prove this result, we will decrease the number of alternations by interchanging a move of Arthur with a move of Merlin. Consider a move of Merlin followed by a move of Arthur. If we simply ask Arthur to move first, this gives Merlin an unfair advantage that may be sufficient to turn

around the odds of the game. To decrease this advantage, we shall use the standard trick of letting Arthur and Merlin play several games in parallel, and choosing the majority decision. Consider again the consecutive moves of Merlin and Arthur in the original game. At Arthur's move, we split the game into several parallel games by allowing Arthur to make several alternate (random) moves. It can be shown that by interchanging the (unsplit) move of Merlin with the (split) move of Arthur, we obtain a new game that defines the same language. In essence, if Merlin's winning probability was small enough, then even after seeing (ahead of time) the several next moves of Arthur, it is not too likely that he has a single move that is good for the majority of Arthur's choices.

Babai & Moran (1988) later proved the stronger result that, in fact, $\mathcal{AM}(2t(n)) = \mathcal{AM}(t(n))$ for every polynomially bounded function $t(n) \geq 2$.

It was an even bigger surprise when Goldwasser & Sipser (1986) proved that interactive proofs and Arthur-Merlin games define the same complexity classes.

(3.12) Theorem. *$\mathcal{IP}(t(n)) = \mathcal{AM}(t(n))$ for every polynomially bounded function $t(n) \geq 2$.*

The idea of the proof can be outlined as follows. Instead of playing the game given by the interactive protocol, Merlin can try to convince Arthur that he could win for a majority of Arthur's random choices. This approach requires a technique to prove a lower bound on the size of an exponentially large set. To give an idea of how to provide such a lower bound consider the following problem. Consider a language $L \in \mathcal{NP}$ that consists of pairs (x, y) of strings of equal length where, for example, x can be viewed as the common input, and y as Arthur's random coin tosses. Define $L(x) = \{y : (x, y) \in L\}$, and suppose that Arthur wants to convince Merlin that $|L(x)| \geq f$ for some integer f. This problem might be $\#\mathcal{P}$-complete. However, the fraction of random strings for which the Prover can win in an interactive proof is either at least $2/3$ or at most $1/3$. Therefore, it might be useful to achieve the following, more modest goal: for any fixed ϵ, prove that there exists an $\mathcal{AM}$ protocol in which Merlin can almost certainly win if $|L(x)| \geq (1 + \epsilon)f$ and has almost no chance to win if $|L(x)| < f$. Goldwasser & Sipser have given such a protocol by using a hashing function technique similar to the one used to prove that $\mathcal{BPP} \subseteq \Sigma_2^{\mathcal{P}} \cap \Pi_2^{\mathcal{P}}$.

To see how this lower bounding technique can be used to turn an interactive proof into an Arthur-Merlin game, we consider the graph non-isomorphism problem. The following Arthur-Merlin protocol is an adaptation of the Arthur-Merlin protocol given by the Goldwasser & Sipser proof when applied to the graph non-isomorphism protocol given in the beginning of this subsection. Let $Aut(X)$ denote the automorphism group of the graph X (see Chapter 27), and let (G_1, G_2) denote the input. We may assume that both graphs are connected. Furthermore, let G denote the graph that consists of node-disjoint copies of G_1 and G_2. If the two graphs are isomorphic then $|Aut(G)| = 2 \cdot |Aut(G_1)| \cdot |Aut(G_2)|$, and otherwise, $|Aut(G)| = |Aut(G_1)| \cdot |Aut(G_2)|$. The non-isomorphism of two graphs can be shown by proving fairly tight lower bounds for $|Aut(G_i)|$ for $i = 1, 2$, and an upper bound for $|Aut(G)|$. The upper bound can be shown by providing a lower bound on the number of different isomorphic copies of G. (In fact, the lower bounds for $|Aut(G_i)|$ can be proved without randomization by guessing generators for the automorphism groups and using a polynomial-time algorithm to compute the cardinality of the generated group.)

Although the result of Goldwasser and Sipser implies that we can restrict attention to one of these two sorts of randomized proofs and forget the other, it is quite useful to have both notions. If one wishes to design a protocol, clearly it is easier to use private coins. On the other hand, the

relative simplicity of an Arthur-Merlin game makes it well-suited for proving properties of these hierarchies.

Recall that for randomized polynomial-time algorithms we have defined, in addition to $\mathcal{BPP}$, a narrower complexity class of languages, $\mathcal{RP}$, that can be accepted by a polynomial-time randomized algorithm with one-sided error. Analogous to this, we consider interactive proofs with one-sided error. An interactive proof system has the **perfect completeness** property if the Verifier accepts each word in the language, and, with high probability, rejects each word not in the language. Surprisingly, every language in $\mathcal{IP}$ can be recognized by an Arthur-Merlin game with the perfect completeness property without even increasing the number of turns. This fact was proved by Zachos & Furer (1985) for the finite levels of the Arthur-Merlin hierarchy and by Goldreich, Mansour & Sipser (1987) for the general case. Similarly, one can consider proofs with the **perfect soundness** property, *i.e.*, interactive proofs in which the Verifier rejects each word not in the language, and accepts each word in the language with high probability. However, only languages in $\mathcal{NP}$ can have interactive proofs with the perfect soundness property; given such an interactive protocol, we can construct a nondeterministic Turing machine that guesses the Verifier's random coin tosses and the communication that led to acceptance, and then checks the validity of the communication.

The most important complexity class defined via randomized proofs seems to be $\mathcal{AM}$[2]. All known examples of languages that belong to $\mathcal{IP}$ actually belong to one of the finite levels, and as a consequence belong to $\mathcal{AM}$. The class $\mathcal{AM}$ is an extension of $\mathcal{NP}$ using randomization, but not too much interaction. This seems to suggest that randomization is the more important element of the new kind of proofs introduced. We now give an oracle-based generalization of $\mathcal{NP}$ using randomization, that turns out to be equivalent to $\mathcal{AM}$.

We define a random language A to be one that contains each word x independently with probability 1/2. For a complexity class $\mathcal{C}$, let us define $almost - \mathcal{C}$ to be the class of languages that are in $\mathcal{C}^A$ for almost every oracle A. In other words, a language is in $almost - \mathcal{C}$ if it is in $\mathcal{C}^A$ for a random oracle with probability 1. Gill (1977) has shown that $\mathcal{BPP} = almost - \mathcal{P}$. One can similarly show that $\mathcal{AM} \subseteq almost - \mathcal{NP}$, and one might expect that the converse is also true. However, this is not clear. The difficulty is that a nondeterministic Turing machine can use the power of nondeterminism to access an exponential number of random bits, whereas Arthur can generate only a polynomial number. However, a recent result of Nisan & Wigderson (1988) implies that a polynomial-time nondeterministic Turing machine cannot take advantage of the exponential number of random bits available, so that $\mathcal{AM} = almost - \mathcal{NP}$.

As is true for most complexity classes, there is little known about the limits of $\mathcal{AM}$. It appears that $\mathcal{AM}$ does not contain $co-\mathcal{NP}$, but to prove this would imply that $\mathcal{NP} \neq co-\mathcal{NP}$. Nonetheless, the following theorem, due to Boppana, Hastad & Zachos (1987), provides some evidence.

(3.13) Theorem. *If* $co - \mathcal{NP} \subseteq \mathcal{AM}$ *then* $\Sigma_2^{\mathcal{P}} = \Pi_2^{\mathcal{P}} = \mathcal{AM}$.

Proof. We describe a compact proof due to Babai. Assume that $co - \mathcal{NP} \subseteq \mathcal{AM}$. It suffices to prove that this implies that $\Sigma_2^{\mathcal{P}} \subseteq \mathcal{AM} \subseteq \Pi_2^{\mathcal{P}}$, since $co - \mathcal{C} \subseteq \mathcal{C}$ implies that $\mathcal{C} = co - \mathcal{C}$ for any

[2]Recent results imply that this intuition seems to be incorrect. Lund, Fortnow, Karloff & Nisan recently showed that the permanent could be computed using an interactive proof with a polynomial number of rounds. This implies that $\mathcal{PH} \subseteq \mathcal{IP}$, and Shamir extended their techniques to prove the even more astounding result that testing the validity of a quantified boolean formula is in $\mathcal{IP}$, which implies that $\mathcal{IP} = \mathcal{PSPACE}$!

class $\mathcal{C}$. Note that $\mathcal{AM} \subseteq \Pi_2^{\mathcal{P}}$ is true without the assumption, as we have already mentioned.

Now let L be a language in $\Sigma_2^{\mathcal{P}}$. There exists a language $L' \in \Pi_1^{\mathcal{P}} = co - \mathcal{NP}$ such that $L = \{x : \exists_p y \text{ such that } (x, y) \in L'\}$. By assumption, $L' \in \mathcal{AM}$ and therefore $L \in \mathcal{MAM} = \mathcal{MA}(3)$. Theorem (3.11) implies that $\mathcal{MA}(3) = \mathcal{AM}$. □

We have seen that the graph non-isomorphism problem is in $\mathcal{AM}$. Therefore, Theorem (3.13) provides new evidence that the graph isomorphism problem is not $\mathcal{NP}$-complete.

(3.14) Corollary. *If the graph isomorphism problem is $\mathcal{NP}$-complete, then $\Sigma_2^{\mathcal{P}} = \Pi_2^{\mathcal{P}} = \mathcal{AM}$.*

Goldwasser, Micali & Rackoff introduced the interactive proof system in order to characterize the minimum amount of "knowledge" that needs to be transferred in a proof. Interactive proofs make it possible to "prove", for example, that two graphs are isomorphic, without giving any further clue about the isomorphism between them. These aspects of interactive proofs shall be discussed at the end of the next section.

4 Living with Intractability

The knowledge that a problem is $\mathcal{NP}$-complete is little consolation for the algorithm designer that needs to solve it. Contrary to their theoretical equivalence, all $\mathcal{NP}$-complete problems are not equally hard from a practical perspective. In this section, we will examine two approaches to these intractable problems that, while not overcoming their inherent difficulty, make them appear more manageable. In this process, finer theoretical distinctions will appear among these problems that help to explain the empirical evidence. We conclude this section by considering work that uses intractability to an algorithm's advantage; much of modern cryptography relies on the simple idea that a code is secure if any algorithm to efficiently break it could also be used to solve an intractable problem.

The complexity of approximate solutions

Most of the 21 $\mathcal{NP}$-complete problems in Karp's original paper are decision versions of optimization problems; this is also true for a great majority of the problems catalogued by Garey & Johnson (1979). Although the combinatorial nature of these problems makes it natural to focus on optimal solutions, for most practical settings in which these problems arise it is nearly as satisfying to obtain solutions that are guaranteed to be nearly optimal. In this subsection we will first classify the sorts of results that one might obtain in this area, and then sketch several results that exemplify the techniques that have been used.

In order to simplify the discussion of terminology, we will focus on minimization problems where feasible solutions have integral objective function values. For more technical reasons, we will make the simplifying assumption that the value of the objective function depends on the input in a reasonable way; we assume that it is bounded by a polynomial in the length of the input and the value of largest number represented in the input. Consider an instance I of an optimization problem, and let $OPT(I)$ denote the value of an optimal solution to I. If an approximation algorithm A (or more simply, a heuristic) delivers a solution of value $A(I)$, $A(I)/OPT(I)$ is called its **performance ratio** for I. The **absolute performance guarantee** of A is the smallest ρ such

that the performance ratio is at most ρ for all instances. If the absolute performance guarantee of A is ρ, and A is polynomial-time, then it is called a **ρ-approximation algorithm**. For some problems, this absolute measure gives a distorted picture, since an algorithm may perform poorly only when $OPT(I)$ is small. As a result, we will often consider the **asymptotic performance guarantee** of A, which is the infimum of all ρ such that the performance ratio is at most ρ when restricted to instances where $OPT(I) \geq k$, for some choice of k.

For some problems we will provide evidence that there is no polynomial-time algorithm with a particular performance guarantee, in the sense that its existence implies that $P = \mathcal{NP}$. On the other hand, for an increasingly wide range of problems, we now know that for any fixed error ϵ, there is a $(1+\epsilon)$-approximation algorithm A_ϵ; such a family of algorithms is called a **polynomial approximation scheme**.

The definition of a polynomial approximation scheme allows the running time to depend even superexponentially on $1/\epsilon$, since ϵ is a constant for the sake of the analysis. However, if such a scheme is to be really useful in practice, the dependence on $1/\epsilon$ must be constrained. If the running time of A_ϵ increases as a polynomial function of $\log(1/\epsilon)$, then the assumption about the value $OPT(I)$ makes it possible to choose ϵ of polynomial length, so that $A_\epsilon - OPT(I) < 1$. By the integrality of feasible solutions, we see that this gives a polynomial-time algorithm for the problem. A more modest goal is to limit the dependence on ϵ to be a polynomial in $1/\epsilon$, and such a family of algorithms is called a **fully polynomial approximation scheme**. The previous argument shows that a fully polynomial approximation scheme can be used to obtain an algorithm that finds an optimal solution, where the running time is polynomial in the size of the unary encoding of the problem. This yields an important result, due to Garey & Johnson (1978); if a problem is strongly $\mathcal{NP}$-complete, then there is no fully polynomial approximation scheme for it unless $\mathcal{P} = \mathcal{NP}$. Notice that this argument does not exclude the existence of such a family of algorithms with asymptotic performance guarantee arbitrarily close to 1; such a family will be called a **fully polynomial asymptotic approximation scheme**.

An example of an algorithm with the strongest sort of performance guarantee predates complexity theory. In the **edge coloring problem**, we are given an undirected graph G and an integer k, and we wish to color the edges with as few colors as possible so that no two edges incident to the same node receive the same color. In a classical result, Vizing (1964) proved that the chromatic index is at most one more than maximum degree of G, and his proof immediately yields a polynomial-time algorithm that uses at most $OPT(I) + 1$ colors. In fact, this strong result was viewed as suggestive evidence that the edge coloring problem was not $\mathcal{NP}$-complete, but Holyer (1981) showed that deciding if a cubic graph requires either 3 or 4 colors is $\mathcal{NP}$-complete. This example also serves as a good illustration of the difference between the asymptotic performance guarantee, which is 1 in this case, and the absolute performance guarantee. Suppose that there exists a polynomial-time edge coloring ρ-approximation algorithm with $\rho < 4/3$. If this algorithm is run on a cubic graph that can be colored with three colors, then the algorithm must return a coloring that uses fewer than four colors; it returns an optimal coloring. Thus, by Holyer's result no such algorithm can exist unless $\mathcal{P} = \mathcal{NP}$.

The idea of studying the performance guarantees of heuristics for optimization problems was first proposed by Graham (1966), in the context of a simple **machine scheduling problem**: each of n jobs is to be scheduled on one of m machines, where job j requires processing time p_j to be scheduled on any machine; find the minimum deadline d so that there is an assignment of jobs to

machines where the jobs assigned to each machine have a total processing requirement at most d. The decision version of this problem is a generalization of the 3-partition problem, so that it is strongly $\mathcal{NP}$-complete. The most natural heuristic is to list the jobs, and then repeatedly assign the next listed job to the machine that currently has the smallest total processing load. Graham showed that this heuristic is a 2-approximation algorithm. To see this, consider the last job j assigned to the machine with the largest total processing time P assigned to it. Since j is assigned to this machine, all other machines must be assigned a total no less than $P - p_j$ and so $OPT(I) \geq P - p_j$. However, $OPT(I)$ is clearly at least p_j, and thus $P \leq 2OPT(I)$. For this problem, there is essentially no difference between the absolute and asymptotic performance guarantees, because the problem has a simple **scaling property**; by multiplying all of the processing times by some factor, we get an equivalent problem with a larger optimal value.

Perhaps the most notorious problem in combinatorial optimization is the **traveling salesman problem**: given a complete graph with edge weights $\{c_e\}$, find the Hamiltonian circuit of minimum total edge weight. The following reduction from the Hamiltonian circuit problem shows that any constant absolute performance ratio ρ is not achievable in polynomial time, unless $\mathcal{P} = \mathcal{NP}$; if $G = (V, E)$ is the Hamiltonian circuit instance, set $c_e = 1$ for $e \in E$ and $c_e = \rho|V|$ otherwise. Since this problem has the scaling property, an identical result holds for the asymptotic performance guarantee. It is natural to assume that the edge weights satisfy the triangle inequality; that is, for all nodes u, v, w, $c_{uv} + c_{vw} \leq c_{uw}$. The decision version of this special case is still $\mathcal{NP}$-complete (replace $\rho|V|$ by 2 in the previous construction) but with this restriction, it is possible to guarantee reasonably good solutions. Consider the following heuristic: start with a trivial circuit (consisting of a single node), and in each iteration find the cheapest edge between a node i in the current circuit and a node j not in it; extend the current circuit by inserting j between i and one of its neighbors in the circuit. To see that this is a 2-approximation algorithm, we observe that the sequence of selected edges is exactly the same as in Prim's minimum spanning tree algorithm (see Aho, Hopcroft & Ullman (1974)), and the total length of the Hamiltonian circuit constructed is no more than twice this tree. Since each Hamiltonian circuit contains a spanning tree, we get the claimed guarantee. Christofides (1976) improved this result by constructing a minimum spanning tree, and then adding a minimum weight matching on the nodes of odd degree, to form an Eulerian graph G_E. Since any Hamiltonian circuit on the nodes of odd degree can be decomposed into two disjoint matchings, the total weight of the added matching is no more than half the optimal circuit. Since the Hamiltonian circuit formed by traversing the nodes in the order that they are first encountered in an Eulerian tour of G_E has total weight no more than the total edge weight of G_E, we have obtained a 3/2-approximation algorithm. It is a challenging open problem to improve on this performance guarantee.

For each of the previous problems, we have either been able to provide an algorithm with constant performance guarantee, or been able to prove that such an algorithm would imply that $\mathcal{P} = \mathcal{NP}$. For the next few examples, we will present algorithms for which there is evidence that they cannot be improved, in terms of their performance guarantees.

In the **center problem**, the input is a complete graph with integer edge weights $\{c_e\}$ and an integer k, and the aim is to select k nodes that cover all remaining nodes within as small a radius as possible, where a node v is covered within radius r if there is a selected node u such that $c_{uv} \leq r$. As was done in the traveling salesman problem, an easy reduction (from the dominating set problem) shows that no fixed performance guarantee can be achieved. Unlike the traveling salesman

problem, a slight modification of this reduction shows that even with the triangle inequality, no ρ-approximation algorithm exists for $\rho < 2$ unless $\mathcal{P} = \mathcal{NP}$. Hochbaum & Shmoys (1985) give a 2-approximation algorithm that relies on a more general technique, called **dual approximation**. A dual approximation algorithm finds solutions that have objective function values that are at *most* $OPT(I)$, at the expense of obtaining solutions that are infeasible. The performance of a dual approximation algorithm is a bound on the infeasibility of the solutions generated.

There are two key parameters in the center problem: the number of centers k and the radius r. In a problem that might be viewed as a dual of the center problem, we want to minimize k, in order to achieve a specified radius of coverage r. A ρ-dual approximation algorithm for this problem is a polynomial-time algorithm that, given edge weights and a radius r, finds k nodes that cover all points within radius ρr, where k is no more than the optimal number of nodes needed to cover within radius r. It is easy to see that such a ρ-dual approximation algorithm can be used within a binary search procedure to obtain a ρ-approximation algorithm for the center problem. The following algorithm is a 2-dual approximation algorithm: choose any node, delete all nodes covered within radius $2r$, and repeat until all nodes are covered. To see that the number of nodes selected is not too large, observe that it is impossible to choose two nodes covered by the same node in the optimal solution for radius r. The same framework of using a dual approximation algorithm for a dual problem to obtain an ordinary approximation algorithm for the original problem can, of course, be applied to any problem where there is such a tradeoff between two resources.

One important technique that has been used in constructing approximation schemes is that of **rounding and scaling**. The technique is based on a simple idea: round the data sufficiently so that a pseudo-polynomial algorithm for the rounded problem runs in time polynomial in the original instance size; then interpret the solution in terms of the original problem, and if the rounding was small enough, the solution is still nearly optimal. Ibarra & Kim (1976) developed this technique to give a fully polynomial approximation scheme for the **knapsack problem**: there are n pieces to be packed in a knapsack of size B, where piece j has a size $s_j \leq B$ and a value v_j, and the aim is pack a subset of total size no more than B, with maximum total value. It is easy to see that the subset sum problem reduces to a decision version of this, and also that an approach similar to the one used for the subset sum problem gives a pseudo-polynomial algorithm for this problem (where only the v_j must be encoded in unary). Algorithm A_ϵ is as follows: round v_j down to its closest multiple of $\mu = \epsilon \max_j\{v_j\}/n$, and solve the rounded instance; if this solution is worse than $\max_j\{v_j\}$, return the most valuable piece instead. This solution differs from $OPT(I)$ by at most $n\mu = \epsilon \max_j\{v_j\} \leq \epsilon A_\epsilon$, and so $OPT(I)/A_\epsilon(I) \leq (1+\epsilon)$. Since the rounded problem (when rescaled by μ) has length in its unary encoding at most n^2/ϵ plus the size of the original instance (encoded in binary), we see that this gives a fully polynomial approximation scheme.

The **bin packing problem** is dual to the machine scheduling problem: given a specified capacity d, pack n pieces with sizes p_j into as few bins of capacity d as possible. Since it is easy to formulate the subset sum problem as a question of whether 2 bins suffice, we see that a ρ-approximation algorithm with $\rho < 3/2$ would imply that $\mathcal{P} = \mathcal{NP}$. On the other hand, the bin packing problem does not have the scaling property, and Johnson (1973) showed that many simple heuristics do quite well, with asymptotic guarantees as small as 11/9. It was a great surprise when Fernandez de la Vega & Lueker (1981) not only substantially improved this 11/9 bound, but gave a polynomial asymptotic approximation scheme, by showing that a rounding and scaling-like approach could be used for this problem. Karmarkar & Karp (1982) gave a fully polynomial

asymptotic approximation scheme for the bin packing problem, by using similar ideas in conjunction with a more efficient approach to solving a related underlying linear program. This result was also quite surprising, since no such approximation scheme was known for any strongly $\mathcal{NP}$-complete "number problem." Using techniques similar to those of Fernandez de la Vega & Lueker (1981),

Hochbaum & Shmoys (1987) give a $1+\epsilon$-dual approximation algorithm for the bin packing problem for any $\epsilon > 0$. This implies a polynomial approximation scheme for the machine scheduling problem. Unlike the bin packing problem, the machine scheduling problem *does* have the scaling property, so the existence of an asymptotic fully polynomial approximation scheme would imply that $\mathcal{P} = \mathcal{NP}$. The key observation is to separate the pieces to be packed into two classes: **big** pieces, with $p_j > \epsilon d$ and the remaining **small** pieces. Rounding and scaling can be used for the instance restricted to the big pieces, since if each piece is rounded down to the closest multiple of $\epsilon^2 d$, then there are at most $k = 1 + 1/\epsilon^2$ different piece sizes. For any constant k, there is polynomial-time algorithm to solve the bin packing problem with only k different sizes (although the degree of the polynomial depends on k). If the solution to the rounded problem is interpreted for the original, then since each bin contains at most $1/\epsilon$ pieces, the total difference between the contents of a bin with and without rounding is at most $(1/\epsilon)\epsilon^2 d$; that is, each bin contains at most $(1+\epsilon)d$. Clearly, the number of bins used is no more than the optimum with capacity d. To complete the packing, add the small pieces to any bin with no more than d in it already, and start a new bin only when all bins are overfilled. It is easy to see that this gives a $(1+\epsilon)$-dual approximation algorithm.

For the preceding problems, we have been able to find algorithms whose performance matches the limits implied by assuming that $\mathcal{P} \neq \mathcal{NP}$. Unfortunately, this is much more the exception than the rule. The interest in performance guarantees of algorithms for graph problems is largely due to the work of Johnson (1974a), but little improvement has been made in the intervening years.

For the maximum stable set problem, the algorithm with the best known performance guarantee is a simple greedy heuristic. A stable set I is constructed by placing a node v of smallest degree in I and deleting v and its neighbors and iterating until the entire graph is deleted. If $\alpha(G) \geq n/k$, then any node in I has degree at most $n-(n/k)$, and I has $\log_k n$ nodes. This implies that the algorithm's performance ratio is $O(n/\log n)$. (For maximization problems, we invert all performance ratios.)

Garey & Johnson (1976) developed a graph composition technique to show that if there exists a ρ-approximation algorithm for the maximum stable set problem for some constant ρ, then there exists a polynomial approximation scheme. Let G^2 be the graph formed by composing a graph G with itself, in the sense that each node of G is replaced by a copy of G, and there is an edge between any pair of nodes in copies of G that correspond to adjacent nodes. Note that $\alpha(G^2) = \alpha(G)^2$, and that given a stable set of size k^2 in G^2 one can efficiently find an stable set of size k in G. By applying a ρ^2-approximation algorithm to G^2, we get a set of size $\alpha(G)^2/\rho^2$, which can then be used to find an stable set in G of size $\alpha(G)/\rho$. By repeatedly applying this technique, we get the claimed result. Since this problem has the scaling property (simply take disjoint copies of the graph), an identical result holds for asymptotic performance guarantees.

The state of the art for the node coloring problem is not much better. For the node coloring problem, a corollary of the $\mathcal{NP}$-completeness of 3-colorability is that no ρ-approximation algorithm can exist with $\rho < 4/3$. The same result holds for asymptotic performance guarantees. To see this, note that for any graph G we can produce a graph G' with $\chi(G') = k\chi(G)$, by taking k disjoint

copies of G and adding an edge between any pair of nodes in different copies. This construction maintains the ratio of the chromatic number of those instances constructed from graphs G with $\chi(G) = 4$ and those constructed from graphs with $\chi(G) = 3$. Garey & Johnson (1976) use a more intricate composition technique to increase this ratio, and thereby prove that an asymptotic performance guarantee less than 2 would imply that $\mathcal{P} = \mathcal{NP}$.

There is a large gap between these negative results, and the known guarantees provided by heuristics. The first nontrivial result was proved by Johnson (1974b), who considered the following algorithm: until all nodes are deleted, repeatedly find a stable set (using the algorithm given above), color it with a new color, and delete it from the graph. In any k-colorable graph with n nodes, there is an stable set of size n/k, and so the bound for the stable set algorithm implies that $O(n/\log n)\chi(G)$ colors are used. Wigderson (1983) observed that a graph G with $\chi(G) = 3$ can be colored with $O(\sqrt{n})$ colors by the following simple algorithm: while the maximum degree node v has degree at least $\sqrt{n}$, color the (bipartite) neighborhood of v with 2 (unused) colors, delete the colored nodes and repeat; color the remaining graph with $\sqrt{n}$ additional colors by repeatedly coloring a node with some color not used at one of its neighbors. Since the first phase can only proceed for $\sqrt{n}$ iterations, only $3\sqrt{n}$ colors are used in total. By using the same idea to recursively color a k-colorable graph G, using the approximation procedure for $(k-1)$-colorable graphs until the maximum degree is sufficiently small, Wigderson improved Johnson's bound to $O(n(\log\log n)^2/(\log n)^2)\chi(G)$.

Probabilistic analysis of algorithms

One justified criticism of complexity theory is that it focuses on worst-case possibilities, and this may in fact be unrepresentative of the practical reality. In this section, we will examine results that are concerned with the probabilistic analysis of algorithms, where the inputs are selected according to a specified probability distribution, and the average behavior is studied. After a brief discussion of the sorts of results that might be obtained, we will illustrate this approach by discussing results concerning several of the combinatorial problems already mentioned. We shall also sketch the main ideas of an analogue of $\mathcal{NP}$-completeness, recently proposed by Levin, to provide evidence that a problem is hard to solve in even a probabilistic sense. For a more extensive survey of results in this area, the reader is referred to Karp, Lenstra, McDiarmid & Rinnooy Kan (1985), whereas Coffman, Lueker & Rinnooy Kan (1988) give a nice introduction to the techniques used.

In order to discuss the probabilistic performance of algorithms, we first introduce some terminology from probability. Let $\{r_n\}$ be a sequence of random variables, and let r be a random variable. Then $r_n \to r$ **in expectation**, if $\lim_{n\to\infty} E[|r_n - r|] = 0$. A stronger notion of convergence is that $r_n \to r$ **in probability**, which occurs if for every $\epsilon > 0$, $\lim_{n\to\infty} Pr[|r_n - r| > \epsilon] = 0$. The sequence $r_n \to r$ **almost surely**, if $Pr[\limsup_{n\to\infty} r_n = r = \liminf_{n\to\infty} r_n] = 1$.

These definitions are used to characterize different notions of an algorithm producing asymptotically optimal solutions. For example, if r_n denotes the performance ratio $A(I_n)/OPT(I_n)$ for a particular algorithm A when run on an input I_n of size n drawn from a specified distribution, then it is **optimal in expectation** if $r_n \to 1$ in expectation. Successively stronger results would prove that an algorithm is **optimal in probability** and **optimal almost surely**, depending on whether $r_n \to 1$ in probability or almost surely. These notions characterize approximation results, in the sense that the solutions produced by the algorithm may always be suboptimal, since merely the asymptotic error is bounded. It would, of course, be still more preferable to prove that the

algorithm generally yields the optimal solution. We will say that an algorithm **solves** the problem **in probability** if $\lim_{n\to\infty} Pr[A(I_n) = OPT(I_n)] = 1$. This definition can also be applied to decision problems in $\mathcal{NP}$, where the solution values are restricted to be 0 or 1, but in this case we will require that the algorithm also produce a certificate proving that the solution value is 1. Finally, an algorithm **solves** the problem in **expected polynomial time**, if it always finds an optimal solution, and the expected value of the running time is bounded by a polynomial.

As a first example, consider again the machine scheduling problem, and the list scheduling heuristic discussed in the context of worst-case analysis. Graham (1966) showed that the algorithm always produces a schedule that completes within $[1 + m(max_j p_j)/(\Sigma_j p_j)]OPT(I_n)$, where m and n denote the number of machines and jobs, respectively. If the processing times are independently and identically distributed (i.i.d.) uniformly over the interval [0,1], one can simply analyze the likely behavior of the error term given by the worst-case analysis. Bruno & Downey (1986) have shown that this implies that $\lim_{n\to\infty} Pr[A(I_n)/OPT(I_n) < 1 + 4m/n] = 1$. Thus, for any fixed value of m, this heuristic is optimal in probability.

In a paper that stimulated the area of probabilistic analysis of algorithms for $\mathcal{NP}$-complete problems, Karp (1976) proposed a natural partitioning scheme for solving the traveling salesman problem over instances where the n nodes are i.i.d. uniformly in the unit square $[0,1]^2$, and the intercity distances c_{ij} are given by the Euclidean distance between each pair of nodes. A classical theorem of Beardwood, Halton & Hammersley (1959) proves that there exists a constant c such that $OPT(I_n)/\sqrt{n} \to c$ almost surely for instances drawn from this distribution.

We now give the adaptive partitioning method of Karp. For simplicity, assume that no two nodes have a common vertical coordinate or horizontal coordinate. Find a vertical line through a node that splits the unit square into two parts so that each part contains essentially the same number of nodes. (Consider a node on the dividing line to be in both parts.) For each of the two parts, split them analogously by finding a horizontal line that ensures that the number of nodes in each new region is reduced by a factor of two. Repeat this procedure recursively on each region until each small part has $\log n$ nodes. A dynamic programming algorithm can find an optimal solution to any of these small instances in $O(n \log^2 n)$ time. The union of these tours is a spanning Eulerian graph, and as in Christofides' algorithm, the Eulerian tour yields an ordering of the nodes.

To analyze the performance of the algorithm, let R_i denote the ith region, and let T_i and T_i^* denote, respectively, the length of the part of the heuristically generated tour in R_i, and the length of the part of the optimal tour in R_i. We first show that T_i is at most T_i^* plus the 3/2 times the perimeter of R_i. Consider R_i, and add a node at each point where the optimal tour leaves this region. Construct a tour through all of the nodes in R_i in the following way: take the pieces of the optimal tour, and add the perimeter once, which makes the graph connected; then take alternate pieces of the perimeter, choosing à la Christofides, the matching of smaller length. This Eulerian graph can be shortcut to yield a tour, proving the claim. This implies that $A(I_n) \leq OPT(I_n)$+total perimeter length $\leq OPT(I_n) + O(\sqrt{n/\log n})$. The theorem of Beardwood, Halton & Hammersley (or even a simple neighborhood argument) shows that there exist constants c' and $d < 1$ such that $Pr[OPT(I_n) < c'\sqrt{n}] \leq d^n$. These results are sufficient to prove that the algorithm is optimal almost surely; it is not hard to see that for any $\epsilon > 0$,

$$\sum_{n=1}^{\infty} Pr[(A(I_n)/OPT(I_n)) \geq 1+\epsilon] \leq \sum_{n=1}^{\infty} (Pr[OPT(I_n) < c'\sqrt{n}] + Pr[|OPT(I_n) - A(I)| \geq \epsilon c'\sqrt{n}]) < \infty$$

and by the Borel-Cantelli lemma, this implies that the relative error converges to 0 almost surely.

Although worst-case analysis of the traveling salesman problem requires assumptions such as the triangle inequality to obtain reasonable results, the same is not true for probabilistic analysis. In fact, consider instances of the directed traveling salesman problem, where all n^2 entries of the distance matrix are i.i.d. uniform random variables over the interval [0,1]. Algorithms to find a minimum weight perfect matching in a bipartite graph yield a polynomial-time procedure to find the minimum length collection of directed circuits that cover all of the nodes. It remains merely to patch these circuits together to build a tour. A fairly natural algorithm is to take the two longest circuits, find the minimum cost way to delete two arcs and to add two arcs to join the two circuits, and repeat until the result is a single tour. Karp showed that the expected value of the performance ratio of this algorithm is $1 + O(n^{-1/2})$, and so it is optimal in expectation. (For a survey containing this and other results on the probabilistic analysis of the traveling salesman problem, the reader is referred to the survey of Karp and Steele (1985).)

Since the Hamiltonian circuit problem is a decision problem, we shall focus instead on probabilistic results to solve this problem, and so the analysis has a somewhat different flavor. The study of algorithms for this problem grew out of the theory of random graphs of Erdös & Rényi (1960), which is treated elsewhere in this volume (see Chapter 6). There are two standard distributions of random graphs of order n: the graph $G_{n,m}$ is chosen uniformly from all graphs with m edges, and the graph $G_{n,p}$ is selected by including each possible edge independently with probability p. Most results in one model can be translated to an equivalent result in the other. A major part of the study of random graphs deals with the evolution of the graph as the number of edges grows as a function of the number of nodes, and one of the fundamental results of this area has been to provide a threshold function that characterizes the density where the graph quickly changes from having an overwhelming chance of being non-Hamiltonian to having an overwhelming chance of being Hamiltonian. The ultimate result along these lines is due to Komlós & Szemerédi (1983), who proved that if $H(n,m)$ is the probability that $G_{n,m}$ is Hamiltonian, and $m(n) = (n \log n)/2 + (n \log\log n)/2 + nw(n)$, then $\lim_{n\to\infty} H(n, m(n)) = 0$ if $w(n) \to -\infty$; $= \exp(\exp(-2c))$ if $w(n) \to c$; and $=1$ if $w(n) \to \infty$. This result is surprising in view of the fact that an identical result was shown by Erdös & Rényi (1960) for the property of having minimum degree two, which is clearly necessary for a graph to be Hamiltonian.

Although the previous result is not algorithmic, the progress towards this result has included a number of results that yield polynomial-time algorithms to find a Hamiltonian circuit in the case that it is likely that one exists. The first result proving that a modest number of edges is sufficient for almost all graphs to have a Hamiltonian circuit is due to Perepeliča (1970), who gave an algorithm that solves the Hamiltonian circuit problem in probability for graphs with $cn^{3/2}\sqrt{\log n}$ edges. There have been a series of results that have gradually reduced the number of edges that are sufficient to be sure of efficiently finding a Hamiltonian circuit. Based on a nearly algorithmic result of Pósa (1976), Koršunov (1976) gave the first result of the correct order by reducing the bound to $3n \log n$. In a recent paper, Bollobás, Fenner & Frieze (1985) provide a polynomial-time algorithm HAM that yields an algorithmic proof of the result of Komlós & Szemerédi. By considering the special case $G_{n,p}$ with $p = 1/2$, they give a slight extension of HAM that solves the Hamiltonian circuit problem in expected polynomial time. For this distribution, the probability that HAM fails to find a Hamiltonian circuit in a graph that is connected and has minimum degree at least two is so small, that if an exponential-time dynamic programming algorithm to solve the Hamiltonian circuit

problem is run in this case, the expected running time remains polynomial. Furthermore, they use a variant of this algorithm to solve in probability the **bottleneck** version of the traveling salesman problem, where the aim is to minimize the maximum weight edge included in the Hamiltonian circuit, where the edges of the undirected complete graph are given weights i.i.d. uniformly in the interval [0,1]. The algorithm first computes the minimum value t such that the subgraph containing the t cheapest edges has minimum degree 2, and then applies HAM to find a Hamiltonian circuit in this subgraph.

For the graph coloring problem, we will initially consider the restricted input distribution $G_{n,p}$ with $p = 1/2$. Although abandoning worst-case analysis allows for much better results for this problem as well, the best polynomial-time algorithms are still expected to use twice the optimal number of colors. Grimmett & McDiarmid (1975) propose a simple greedy coloring algorithm, which can be viewed in the following way; find a maximal stable set by selecting the lowest numbered node, deleting it and its neighbors, and repeating. It is not hard to see that a set of size about $\log_2 n$ is selected. Since the algorithm does not "look" at the remaining graph, we see that the remaining graph is a random graph of smaller order selected according to the same distribution. This allows us to analyze the process of repeatedly choosing such a set until the entire graph has been colored. It is not hard to show that the size of the largest stable set in $G_{n,p}$ is, in probability, $2\log_2 n$. This implies that the algorithm uses at most $2\chi(G_{n,p})$ colors in probability. Bollobás (1988) has recently shown that the lower bound on $\chi(G_{n,p})$ given by the stable set computation is indeed tight, so that the greedy algorithm does indeed use twice as many colors as needed, and it remains an interesting open question to give an efficient algorithmic proof of Bollobás' result. Although we have focused on the case $p = 1/2$, these results have been extended to include a broader spectrum of distributions.

We next consider the subset sum problem. Initially, we will restrict attention to the following distribution of instances that are chosen to be "yes" instances, and so the problem is to find an appropriate subset: randomly select a vector x in $\{0,1\}^n$, and choose integers $s_1, \ldots, s_n$ i.i.d. uniformly in $[1, U]$, where $U = 2^{\delta n^2}$ for any fixed $\delta > 1/2$, and let the target $t = \Sigma_{i=1}^n x_i \cdot s_i$. Lagarias & Odlyzko (1983) showed how to use the lattice basis reduction algorithm of Lovász to find x, given the s_i and t. A **lattice** is the collection of vectors that can be represented as an integral linear combination of a given set of n linearly independent basis vectors in $\mathbf{Z}^n$. Given one basis, the basis reduction algorithm of Lovász produces a basis whose shortest vector has length at most $2^{(n-1)/2}$ times the length of the shortest non-zero vector in the lattice (see Chapter 19).

We shall give a short proof of the result of Lagarias & Odlyzko due to Frieze (1986). For the analysis, consider a fixed choice of x, and a random selection of the s_i. We may assume that $\Sigma_{i=1}^n s_i/2 \leq t$ since otherwise we can solve the problem of finding the complementary subset. Consider the $(n+1)$-dimensional lattice given by the basis $a_0 = (ct, 0, \ldots, 0)$, $a_i = (-cs_i, b_{i1}, b_{i2}, \ldots, b_{in})$, $i = 1, \ldots, n$, where c is an integer greater than $n2^{n/2}$ and the matrix $B = (b_{ij})$ is the identity matrix. Note that $\hat{x} = (0, x_1, \ldots, x_n)$ is in this lattice. Let $\lambda = (\lambda_0, \ldots, \lambda_n)$ be the shortest vector returned by the basis reduction algorithm, and let λ^* denote the shortest non-zero vector in the lattice. We will show that, with probability approaching 1, λ is $\hat{x}$ or $-\hat{x}$. Using the above bound for the basis reduction algorithm, we see that $||\lambda|| \leq 2^{n/2}||\lambda^*|| \leq 2^{n/2}||\hat{x}|| \leq \sqrt{n2^n} = m$, where $||\cdot||$ denotes the Euclidean norm. Our choice of c implies that $\lambda_0 = 0$. Next, consider the set A_0 of lattice points $\hat{a}$ with $a_0 = 0$ and $||\hat{a}|| \leq m$ that are not integer multiples of $\hat{x}$. We can bound the probability that the algorithm fails by the probability that A_0 is non-empty. It is not hard to see

that if $\hat{a} \in A_0$, then it must satisfy $\Sigma_{i=1}^{n} a_i \cdot s_i = dt$ for some integer d, $|d| \leq 2m$. Consider some fixed choice of d and $(a_1, \ldots, a_n)$ that is not an integer multiple of x. The probability that they satisfy this equation is at most $1/U$, and since there are at most $(4m+1)(2m+1)^n = o(2^{\delta n^2})$ such choices, we get the claimed result. By choosing t uniformly among the integers from 1 to $\lceil cnU \rceil$, where $c \leq 1$ is some positive constant, this technique can be extended to instances that may not have solutions.

Although the probabilistic analysis of algorithms has focused mainly on $\mathcal{NP}$-complete problems, it has often served as a useful tool to show that the average-case running time of certain algorithms is much better than the worst-case running time. The most important example of this is the simplex method for linear programming, which is a practically efficient algorithm that was shown to have exponential worst-case running time by Klee & Minty (1972). Borgwardt (1982) showed that if the rows of the constraint matrix are i.i.d. according to a spherically symmetric distribution (where each vector is viewed as a ray from the origin), then a variant of the simplex method requires only a polynomial number of iterations. Independently, Smale (1983) showed similar results under the assumption that the rows of the constraint matrix, the vector of right-hand sides, and the objective function are independently distributed with the symmetry assumption that the distribution is invariant under coordinate permutations. Smale analyzed a practical variant of the simplex algorithm, and unlike Borgwardt's result, his distribution includes both feasible and infeasible problems. There has been a great deal of work done in extending these results to more general probability distributions and in analyzing algorithms closer to the particular variants of the simplex method most commonly used in practice. For a thorough survey of results in this area, the reader is referred to Shamir (1987).

As we have seen, not all problems in $\mathcal{NP}$ have been solved efficiently even with probabilistic assumptions, and only the simplest sorts of distributions have been analyzed. This raises the specter of intractability: are there distributions for which certain $\mathcal{NP}$-complete problems remain hard to solve? We will sketch the main ideas behind a recent result of Levin (1986), which proposes a notion of completeness in a probabilistic setting. Once again, evidence for hardness is given by showing that if a particular problem in $\mathcal{NP}$ with a specified input distribution can be solved in expected polynomial time, then every such problem and distribution pair can also be solved so efficiently. For a more complete discussion, the reader is encouraged to read the column of Johnson (1984).

A bit of care must be given in formulating the precise notion of polynomial expected time, so that it is insensitive to both the particular choice of machine and encoding. If $\mu(x)$ denotes the probability that a randomly selected instance of size n is x, and $T(x)$ is the running time on x, then one would typically define expected polynomial time to require that the sum of $\mu(x)T(x)$ over all instances of size n is $O(n^k)$ for some constant k. Instead, we consider μ to be the density function over the set of all instances I, and require that $\sum_{x \in I} \mu(x) T(x)^{1/k} / |x| < \infty$ for some constant k. Levin's notion of **random** $\mathcal{NP}$ requires that the distribution function $M(x) = \sum_{i=1}^{x} \mu(i)$ can be computed in $\mathcal{P}$, where each instance is viewed as a natural number. This notion does not seem to be too restrictive, and includes all of the probability distributions discussed here. It remains only to define the notion of reducibility. As usual, "yes" instances must be mapped by a polynomial-time function f to "yes" instances, and analogously for "no" instances, but one must consider the density functions as well. The pair (L_1, μ_1) reduces to (L_2, μ_2) if in addition we require that $\mu_2(x)$ is at least a polynomial fraction of the total probability of elements that are mapped to x by f.

Levin showed that all of random $\mathcal{NP}$ reduces to a certain **random tiling problem**. Instances consist of integers $k \leq N$, a set of tile types, each of which is a unit square labeled with letters in its four corners, and a side-by-side sequence of k such tiles where consecutive tiles have matching labels in both adjoining corners. We wish to decide if there is a way of extending the sequence to fill out an $N \times N$ square, where adjacent tiles have matching labels in their corresponding corners, and the top row starts with the specified sequence of tiles. The instances are selected by first selecting $N = n$ proportionately to n^{-2}, choosing k uniformly between 1 and N; after choosing the tile types uniformly, the tiles in the sequence are selected in order uniformly among all possible extensions of the current sequence. More recently, Venkatesan & Levin (1988) have shown that a generalization of the problem of edge coloring digraphs (where for certain subgraphs, the number of edges given each color is specified) is also random $\mathcal{NP}$-complete.

Cryptography

The basic goal of cryptography is to provide a means for private communication in the presence of adversaries. Until fairly recently, cryptography has been more of an art than a science. Modern cryptography has suggested ways to use complexity theory for designing schemes with provable levels of security. We will not be able to discuss in detail, or even mention, most of the results here. We intend rather to give a flavor of the problems considered and the results proved. The interested reader is referred to the survey by Rivest (1990) or to the recent special issue of SIAM Journal on Computing (April 1988) on cryptography for more results, and for further references.

A traditional solution to the privacy problem uses secret key crypto-systems. Suppose two parties wish to communicate a message M (a string of 0's and 1's) of length n, and they agree in advance on a secret key K, a string of 0's and 1's of the same length. They send the encrypted message $C = M \oplus K$ instead of the real message M, where $\oplus$ denotes the componentwise addition over $GF(2)$. This scheme is provably secure, in the sense that (as long as the key K is kept secret) an adversary can learn nothing about the message by intercepting. Every string of length n is equally likely to be the message encoded by the string C. This crypto-system is provably secure, but it is very inconvenient to use since before each transmission the two parties must *agree* on a new secret key.

In the paper that founded the area of modern cryptography, Diffie & Hellman (1976) suggested the idea of public-key crypto-systems. They proposed that a transmission might be sufficiently secure if the task of decryption is computationally infeasible, rather than impossible in an information theoretic sense. Such a weaker assumption might make it possible to securely communicate without *agreeing* on a secret key. In a **public key crypto-system** the participants agree on a security parameter n. Messages to be sent are divided into pieces of length n.

- Each participant should randomly choose a public encryption key E and a corresponding secret decryption key D depending on the security parameter n.

- There must be a deterministic polynomial-time algorithm that, given a message M of length n and an encryption key E, produces the encrypted message $E(M)$.

- Similarly, there must be a deterministic polynomial-time algorithm that, given a message M of length n and an decryption key D, produces the decrypted message $D(M)$.

- It is required that for every message M, $D(E(M)) = M$.

- For every constant c and sufficiently large n, the probability that a polynomial-time algorithm using the public key E but not the private key D can decrypt a randomly chosen message $E(M)$ of length n is less than n^{-c}.

The users of this system should publish a directory of the public keys. When a user named Bob wants to send a message M to another user named Alice, he looks up Alice's public-key E_A in this directory, and sends her the encrypted message $E_A(M)$. By the assumptions, only Alice can decrypt this message using her private key D_A.

The basic ingredient of this system is a "trapdoor" function. The encryption function is assumed to be a function that is easy to compute, but hard to invert. There are several ways to formalize this notion. The one used most often is the **one-way function**, that is a 1 to 1 function f such that f can be computed in polynomial time, but for n sufficiently large, no polynomial-time algorithm can invert f on even a polynomial fraction of the instances of length n. In order for a one-way function to be useful in the above scheme, it has to be a **trapdoor function**, which is a one-way function with a key K, such that by knowing the key one can invert the function in polynomial time.

Given the present state of complexity theory, there is little hope to prove that any function is one-way, or that a public key crypto-system satisfies the last requirement above. Ideally, one would like to prove the security of a crypto-system based on the assumption that $\mathcal{P} \neq \mathcal{NP}$. However, this also seems to be quite difficult. Complexity theory is mainly concerned with worst-case analysis. For the purpose of cryptography, average-case analysis, and a corresponding notion of completeness (such as the one suggested by Levin) would be more appropriate. Furthermore, these inversion problems have the special property that the inverse is unique, whereas the nondeterministic algorithms for $\mathcal{NP}$-complete problems have several accepting paths for most instances.

Over the last ten years, several public-key crypto-systems have been suggested and analyzed. Some have been proven secure, based on assumptions about the computational difficulty of a particular problem (*e.g.*, factoring integers), or on the more general assumption that one-way functions exist.

Rivest, Shamir & Adleman (1978) were the first to suggest such a scheme. Their scheme is based on the assumed difficulty of factoring integers. Each user of this scheme has to select a number n that is the product of two random primes. Random primes can be selected using a randomized primality testing algorithm (since every $(\log n)$th number is expected to be prime). The public key consists of a pair of integers (n, e) such that e is relative prime to $\phi(n)$, where $\phi(n)$ denotes the number of integers less than n relatively prime to n. A message M is encrypted by $C = M^e \pmod{n}$. The private key consists of integers (n, d), where $d \cdot e = 1 \pmod{\phi(n)}$, and decryption is done by computing $C^d = M \pmod{n}$. Given the prime factorization of n, one can find the required private key d in polynomial time, but the task of finding the appropriate d given only n and e is equivalent to factoring.

Rabin (1979) suggested a variation on this scheme in which the task of decryption is equivalent to factoring, rather than just the task of finding the decryption key. The idea is to encrypt a message M by $C = M^2 \pmod{n}$. (A slight technical problem that has to be overcome before turning this into an encryption scheme is that squaring modulo a product of two primes is not a 1-1 function, but is rather 4-1.) An algorithm that can extract square roots modulo n can be used to factor: choose a random integer x and let the algorithm find a square root y of the integer $(x^2$

mod n); with probability 1/2, the greatest common divisor of $x - y$ and n is one of the two primes used to create n.

It is conceptually appealing to suggest schemes that are based on $\mathcal{NP}$-complete problems rather than number-theoretic ones. Merkle & Hellman (1978), and since then several others, have suggested schemes based on the subset sum problem. The public key of such systems is an integer vector $a = (a_1, \ldots, a_n)$. A message M, that is a 0-1 vector of length n, is encrypted by the inner product $C = (a \cdot M)$. One problem with this scheme is that there is no apparent private key that can make the task of decryption easier for the intended receiver. Crypto-systems based on this idea have built some additional trapdoor information into the structure of the vector a, and so the decryption problem is based on restricted variant of the subset sum problem. Furthermore, as discussed in the previous subsection, randomly chosen subset sum problems can be easy to solve. In an innovative paper, Shamir (1982) used Lenstra's integer programming algorithm to break the Merkle-Hellman scheme, *i.e.*, he gave a polynomial-time decryption algorithm that does not need the secret key. Since then, several other subset sum-based schemes have been broken by clever use of Lovász's basis reduction algorithm (see Chapter 19).

The schemes mentioned so far have encrypted messages of length n, for some given security parameter n. An apparent problem with this approach is that even if we prove that no polynomial-time algorithm can decrypt the messages, it still might be possible to deduce some relevant information about the message in polynomial time. Goldwasser & Micali (1984) suggested encrypting messages bit by bit to achieve provable levels of security. How does one encrypt a single bit? Deterministic encryption schemes cannot be used repeatedly when there are only two possible messages. A natural solution to encrypt bits more securely is to append a number of random bits to the one bit of information, and encrypt the resulting longer string. Goldwasser & Micali proposed a different randomized bit-encryption scheme for which they can prove security.

The Goldwasser-Micali scheme is based on the assumed difficulty of the quadratic residuousity problem. An integer x is a **quadratic residue** modulo n if $x = y^2 \pmod{n}$ for some integer y. The Jacobi symbol $(\frac{x}{n})$ is a 0, +1, -1 valued function, that gives some help in recognizing quadratic residues. It can be computed efficiently, and $(\frac{x}{n}) = 1$ for all quadratic residues. The **quadratic residuousity problem** is to decide for given integers n and x with $(\frac{x}{n}) = 1$ whether x is a quadratic residue modulo n. The Goldwasser-Micali encryption scheme works as follows: a public key consists of an integer n that is the product of two large primes, and a quadratic non-residue y with $(\frac{y}{n}) = 1$. The bit 0 is encrypted by a random quadratic residue $r^2 \pmod{n}$, the bit 1 is encrypted by a random quadratic non-residue of the form $r^2 y \pmod{n}$. The task of distinguishing encryptions of 0's from encryptions of 1's is exactly the quadratic residuousity problem. Decryption is easy for the intended receiver, who knows the prime factorization of n. Yao (1982) has extended this result by proving that a secure randomized bit-encryption scheme exists if a one-way function exists.

For a public key crypto-system, it is does not seem to be very restrictive to further assume that $E(D(M)) = M$ for every message M. Diffie & Hellman noticed that such a system can be used to solve the **signature problem**, so that the participants can electronically sign their messages. When Bob sends a message M to Alice he can use his private decryption key D_B to append the "signature" $D_B(M)$ after the message M. Given such a signature, Alice can use Bob's public key E_B to convince herself that the message came from Bob, exactly as she received it. The Rivest, Shamir & Adleman scheme has this additional property, and therefore can be used for signatures as well. Rabin's scheme is, in fact, more appropriate for signatures then for encryption, since it is

not a 1-1 function.

This raises a problem concerning Rabin's scheme. The signature can easily be forged (or the crypto-system broken) by adversaries that are more active and malicious than merely observing signed (or encrypted) messages. A **plain text message attack** against a signature scheme occurs when the user is willing to sign a message of an adversary's choice. Applying the same idea as was used to prove the security of the system, an adversary can break it merely by asking the user to sign a randomly chosen message ($x^2 \bmod n$).

The first signature scheme that is secure against chosen message attacks was given by Goldwasser, Micali & Rivest (1988). The proof of the security of the scheme is based on the assumed difficulty of certain number-theoretic functions. More recently, Bellare & Micali (1988) have given a signature scheme that is secure against chosen message attacks and is based on the existence of trapdoor functions.

A further related issue is that of generating pseudo-random numbers. When random numbers are used in algorithms and crypto-systems it is essential that the random bits used are unbiased and independent. The speed or the reliability of the algorithm, and the security of the crypto-system depend on the quality of the random numbers used. Natural sources of randomness (such as coins, or noise diodes) are fairly slow in generating random bits. On the other hand, for both of these applications, truly random bits could be replaced by a sequence of bits that no polynomial-time algorithm can distinguish from truly random bits. A **pseudo-random number generator** takes a truly random string x (a **seed**) of length n and expands it to a **pseudo-random string** y of length n^k for some constant k. The quality of the pseudo-random number generator is measured by certain statistical tests. A pseudo-random number generator passes the **next bit test** if after seeing the first i bits of y for some $i < n^k$ no polynomial-time algorithm can predict the next bit with probability $1/2+n^{-c}$ for any constant c. Most computers have built-in pseudo-random number generators, and some of the standard algorithms used have been proven to output inappropriate sequences by clever use of Lovász's basis reduction algorithm. Such examples include the linear congruential generator (where the seed consists of integers a, b, m and x_0, and the pseudo-random numbers are generated by the recurrence $x_{i+1} = ax_i + b \pmod m$), and the binary expansion of algebraic numbers (where the seed is some representation of the algebraic number).

The first provably secure pseudo-random bit generator was developed by Blum & Micali (1984). They proved that pseudo-random bit generators exist, based on an assumption related to the existence of one-way functions. They assume that there exist permutations f of the set of n-bit integers and functions B from the n-bit integers to $\{0,1\}$, such that the composite function $B(f(x))$ can be computed in polynomial time, but the probability that a polynomial-time algorithm computes $B(y)$ for a randomly chosen y of length n (for a sufficiently large n), is less than $1/2 + n^c$ for any constant c. The Blum-Micali pseudo-random number generator produces the sequence $b_0, \ldots, b_k$, defined by $b_i = B(f(x_{k-i}))$, and $x_{i+1} = f(x_i)$, where the random seed is used to select the functions used and the initial vector x_0. They also give particular suggestions for these functions based on the assumed difficulty of the discrete logarithm problem. For a prime p, an integer g is a **generator modulo** p if every non-zero integer x modulo p can be written as $x = g^e \pmod n$ for an appropriate integer exponent $e < \phi(n)$. We shall use the notation $e = index_{p,g}(x)$. The **discrete logarithm problem** is to compute $index_{p,g}(x)$ given a prime p, a generator g and an integer x. The seed of the Blum-Micali generator is used to randomly select a large prime p, a generator g and a starting point x_0. The pseudo-random numbers are defined using the above scheme with the

functions $f(x) = g^x \pmod{p}$ and $B(x) = 1$ if $index_{g,p}(x) \leq (p-1)/2$. The defined pseudo-random number generator can be proved to pass the next bit test based on the assumption that the discrete logarithm problem is difficult.

One might wonder whether certain pseudo-random number generators pass statistical tests other than the next bit test. However, Yao (1982) proved that if a pseudo-random number generator passes the next bit test then it passes any statistical test, *i.e.*, no randomized polynomial-time algorithm can distinguish the generated pseudo-random numbers from truly random numbers.

In some of the above cryptographic applications, one can imagine that it might be useful to be able to convince someone that a number is the product of two primes, without telling the two primes themselves. Interactive proof systems make this possible. In fact, Goldwasser, Micali & Rackoff (1985) invented their notion of interactive proof systems for exactly this application. Informally, an interactive proof of a statement is said to be a zero-knowledge proof if the verifier cannot learn anything from the proof except the validity of the statement. Imagine, for example, a proof that a graph has a Hamiltonian circuit that gives no help in finding the circuit. There are several ways to formalize this notion. The one we shall use is *computational* zero knowledge. We say that an interactive protocol is a **zero-knowledge protocol** if the verifier by himself (without the participation of the prover) can generate a sequence of communication that is indistinguishable in polynomial time from the true transcript of the conversation. Consider the example of the interactive proof for the graph non-isomorphism problem. At first sight, this seems to be a zero-knowledge protocol, since (so long as the verifier does not deviate from the protocol) he always knows the prover's next move, and hence could generate the conversation himself. There are zero-knowledge protocols for the graph non-isomorphism problem, but this protocol is not, in fact, zero-knowledge, since the verifier can use it to test if a third graph is isomorphic to one of the two in the input (*i.e.*, the verifier can gain extra information by deviating from the protocol). Goldreich, Micali & Wigderson (1986) and, subsequently but independently, Brassard & Crépeau (1986) proved that every language in $\mathcal{NP}$ has a zero-knowledge interactive proof. The Goldreich, Micali, & Wigderson protocol is based on the existence of one-way functions, whereas the Brassard & Crépeau protocol is based on the assumed difficulty of certain number-theoretic functions.

To prove that all languages in $\mathcal{NP}$ have zero-knowledge proofs, one merely has to provide a zero-knowledge proof for a single $\mathcal{NP}$-complete problem. The following protocol for the Hamiltonian circuit problem is due to Blum. Suppose that the prover, Alice, knows a Hamiltonian circuit C in a graph G, and she wants to convince the verifier, Bob, of this fact without showing him the circuit. The basic cryptographic tool needed for the protocol is a secure bit-encryption scheme. Alice must be able to encrypt a bit, so that Bob has no chance to figure out on his own what the bit is, but later Alice can prove which bit she encrypted, if she chooses to do so. To convince Bob that a graph has a Hamiltonian circuit, Alice chooses a random permutation P, and uses this permutation to obtain a random isomorphic copy G' of the graph G. She encodes the permutation, and the $n(n-1)/2$ bits representing the adjacency matrix of the graph G'. Bob can choose either to ask Alice for a proof that the encoded graph G' is isomorphic to the original, or to ask for a proof that G' has a Hamiltonian circuit. In the first case, Alice decrypts every encrypted bit, thereby providing the permutation P and the graph G'. In the second case, she decrypts only the bits corresponding to edges participating in the Hamiltonian circuit C. If Alice does not know a Hamiltonian circuit, then she will be caught cheating with probability at least $1/2$. Therefore, this protocol is an interactive proof that the graph is Hamiltonian. It is intuitively clear that this is a

zero-knowledge protocol (assuming that the encryption function is secure) since Bob can either ask to see a random isomorphic copy of the graph, or a random permutation of the sequence of nodes in a circuit, neither of which contains any information.

In the above zero-knowledge proof, cryptographic assumptions are needed to guarantee that the protocol is zero-knowledge. That is, the transcript of the communication contains the information that Alice wants to hide (*e.g.*, the permutation in the case that Bob chooses to see the circuit), but it is computationally infeasible to decrypt this information. Brassard & Crépeau devised a protocol that is, with high probability, zero-knowledge in the information theoretic sense. This is not an interactive proof in the usual sense, since cryptographic assumptions are needed to guarantee the convincing power of the protocol. The protocol is based on the assumed difficulty of the quadratic residuousity problem, and no similar protocol is known based on general cryptographic assumptions (such as the existence of one-way functions). The idea is to use the following encryption scheme in the previous protocol: Bob chooses an integer n that is the product of two large primes and a random square x modulo n, then tells x and n to Alice, and proves in zero-knowledge that x is a square modulo n; Alice will encrypt a 0 by a randomly chosen square $r^2 \pmod{n}$ and a 1 by a randomly chosen square $r^2x \pmod{n}$. Since Bob succeeded in proving that x is a square modulo n, then x is indeed a square with very high probability. But, if x is a square, then either 0 or 1 is encrypted by a random square, and Bob cannot distinguish them, no matter how much computational power he has. However, Alice can prove which of the two bits she was encrypting, by revealing r. For Alice to cheat (*e.g.*, to claim that an encryption of 1 is an encryption of 0) she has to be able to find the square root of x, which is equivalent to factoring. This protocol is dual of the one given by Blum, in the sense that here Alice can cheat if she has unlimited computational power, whereas in the previous one, Bob can. The latter protocol has the conceptual advantage that in order for Alice to cheat, she must be able to factor n on-line, whereas in Blum's protocol, Bob can go home and try to use all of his computational power to figure out Alice's secret after the protocol is over.

It is a quite powerful tool to have zero-knowledge protocols for all of $\mathcal{NP}$. As an example, consider the following general problem: suppose that n people each have their own input, and they want to jointly compute a function of these inputs, without giving unnecessary information about their private inputs to the others. Interesting special cases of this problem include the majority function (that is used for voting), the sum function (for a situation when the participants wish to compute their average income), or comparison (for the millionaire's problem, when two millionaires want to decide which one of them is richer). Special protocols to handle some of these problems have been suggested earlier. Goldreich, Micali & Wigderson (1987) used zero-knowledge proofs and other powerful techniques discovered in the last 10 years that we could not cover here, to design protocols to compute any polynomially computable function in zero-knowledge.

5 Inside $\mathcal{P}$

In this section we shall focus on the complexity of problems in $\mathcal{P}$. After proving that a problem is in $\mathcal{P}$, the most important next step undoubtedly is to find an algorithm that is *truly* efficient, and not merely efficient in this theoretical sense. However, we will not address that issue, since that is best deferred to the individual chapters that discuss polynomial-time algorithms for particular

problems. In this section, we shall address questions that relate to machine-independent complexity classes inside $\mathcal{P}$.

From a practical viewpoint, the most appealing complexity class inside $\mathcal{P}$ is the class of languages for which there are polynomial-time algorithms that can be speeded up significantly if several processors work simultaneously. We shall discuss parallel computation, and focus on the complexity class $\mathcal{NC}$, which serves as a theoretical model of efficient parallel computability, much as $\mathcal{P}$ serves as a theoretical model of efficient "sequential" computation.

We shall also consider the space complexity of problems in $\mathcal{P}$. Recall that the parallel computation thesis suggests that there is a close relationship between sequential space complexity and parallel time complexity. We will show another proven case of this thesis: every problem in $\mathcal{L}$ (and even in $\mathcal{NL}$) can be solved extremely efficiently in parallel. This is one source of interest in the complexity class $\mathcal{L}$. Another source is that this complexity class is the basis for the natural reduction that helps to distinguish among the problems in $\mathcal{P}$ in order to provide a notion of a hardest problem in $\mathcal{P}$.

Logarithmic space

The most general restriction on the space complexity of a language L that is known to imply that $L \in \mathcal{P}$ is logarithmic space. Observe that $\mathcal{L} \subseteq \mathcal{NL} \subseteq \mathcal{P}$, where the last inclusion follows, for example, from Theorem (3.6). The typical use of logarithmic space is to store a constant number of pointers, *e.g.*, the names of a constant number of nodes in the input graph, and in some sense, this restriction attempts to characterize such algorithms. Although $\mathcal{L}$ contains many combinatorial examples, we will postpone their presentation until later in this section, when we consider a narrower complexity class defined by parallel computation. Instead, we will focus on the nondeterministic and randomized analogues, for which there are languages that appear to be best characterized in terms of their space complexity.

To define the notion of a logarithmic-space reduction, we introduce a variant of logarithmic-space computation that can produce output of superlogarithmic size. We say that a function f can be **computed in logarithmic space** if there exists a Turing machine with a read-only input tape and a write-only output tape that, on input x, halts with $f(x)$ written on its output tape and uses at most logarithmic space on its work tapes. A problem L_1 **reduces in logarithmic space** to L_2 if there exists a function f computable in logarithmic space that maps instances of L_1 into instances of L_2 such that x is a "yes" instance of L_1 if and only if $f(x)$ is a "yes" instance of L_2. Let $L_1 \propto_{log} L_2$ denote such a logarithmic-space reduction (or log-space reduction, for short).

In order for this notion of reducibility to be useful, one has to prove that the $\propto_{log}$ relation is transitive. This is not as apparent as in the case of the $\propto_p$ reduction, but it can be shown fairly easily. As a corollary, we see that if $L_1 \propto_{log} L_2$ and there were a log-space algorithm for L_2, then we could obtain a log-space algorithm for L_1.

It is worth noting that all of the polynomial-time reductions mentioned in the discussion of $\mathcal{NP}$-completeness are based on local replacements, and therefore are log-space reductions. In fact, log-space reduction is sometimes used in place of polynomial-time reduction in the definition of $\mathcal{NP}$-completeness and $\mathcal{PSPACE}$-completeness. A problem L_1 is **$\mathcal{NL}$-complete** if $L_1 \in \mathcal{NL}$ and for all $L \in \mathcal{NL}$, $L \propto_{log} L_1$. The composition argument given above yields the following result.

(5.1) Theorem. *For any $\mathcal{NL}$-complete problem L, $L \in \mathcal{L}$ if and only if $\mathcal{L} = \mathcal{NL}$.*

The following theorem, due to Savitch (1970), shows that the complexity of the directed reachability problem can best be characterized by its space requirement in a nondeterministic computation.

(5.2) Theorem. *The directed graph reachability problem is $\mathcal{NL}$-complete.*

Proof. A nondeterministic log-space algorithm for the directed graph reachability problem can guess the nodes in an (s,t)-path one at a time, and at each step verify that the graph contains the arc that connects the nodes in the guessed path. To show that the problem is $\mathcal{NL}$-complete, we must give a log-space reduction from each problem in $\mathcal{NL}$ to the directed graph reachability problem. Let L be in $\mathcal{NL}$ and let M denote a nondeterministic log-space Turing machine that recognizes L. We can assume without loss of generality that M has a single accepting configuration (*e.g.*, it erases its work tapes and enters a special accepting state before halting). For a given input x, we define a directed graph whose nodes correspond to the configurations of the nondeterministic Turing machine M with input x. Note that the number of these configurations is bounded by a polynomial in the size of x. Two configurations are joined by an arc if one can follow the other via a single transition. The input x is in L if and only if this graph contains a path from the starting configuration to the accepting configuration. □

In view of the above result, it was very surprising when Alelunias, Karp, Lipton, Lovász and Rackoff (1979) showed that the **undirected graph reachability problem**, the analogue of the directed graph reachability problem for undirected graphs, can be solved by a randomized Turing machine using logarithmic space. We define the class $\mathcal{RL}$ to be the log-space analogue of $\mathcal{RP}$. A language L is in $\mathcal{RL}$ if there exists a randomized Turing Machine RM that works in logarithmic space, such that each input x that RM accepts along any computation path is in L and for every $x \in L$, the probability that RM accepts x is at least $1/2$.

(5.3) Theorem. *Undirected graph reachability is in $\mathcal{RL}$.*

Proof. Let $G = (V, E)$ denote the input graph, s and t the two specified nodes, n the number of nodes and m the number of edges.

The algorithm attempts to find the required path by following a random walk in the graph, starting from the node s. At each step, an edge incident to the current node is chosen uniformly at random and the walk is continued along this edge. The algorithm stops either if t is reached, or if $4nm$ steps have been made. We will show that if t and s are in the same connected component of the graph, then with probability at least $1/2$, a random walk will reach t in at most $4mn$ steps.

For the purpose of the proof, we may assume that the graph is connected. We first claim that the random walk will use each edge of the graph in each direction with the same frequency, $1/(2m)$. Let p_v denote the frequency that the random walk is at the node v, *i.e.*, the limit as n tends to infinity of the fraction of the first n steps that the random walk is at the node v. (The existence of this limit can be proved using the theory of Markov chains.) Let $deg(v)$ denote the degree of the node v. The frequency that the random walk uses a particular edge to leave v is $e_v = p_v/deg(v)$. Before the walk leaves v it has to enter v, and therefore $e_v = (\sum_{(w,v)\in E} e_w)/deg(v)$. This implies that e_v is the same for each node $v \in V$, and therefore $e_v = 1/(2m)$. Consequently, each edge

(and, in particular, an edge entering t) is going to be used, on the average, once every $2m$ steps. However, this does not mean that the expected time to reach t is at most $2m$, since this is only true as a limit, and might not be true at the beginning of the walk.

To get an idea about the beginning of the random walk, consider the expected time between two visits of the same node. After leaving a node v, the random walk is expected to return to v in $1/p_v = 2m/deg(v)$ steps. Therefore, the expected time to go from v to an adjacent node w is at most $2m$ steps. Using induction on the distance d between v and w, we see that the random walk starting in v is expected to visit w (and traverse each of the edges along the shortest path from v to w) in at most $2md$ steps. In particular, a random walk starting at s is expected to visit t in $2mn$ steps. Finally, if the expected number of steps that it takes to reach t is at most $2nm$, then the probability that t is reached in $4nm$ steps must be at least $1/2$. □

Why is this theorem about undirected graphs? One could conceivably run the same algorithm for directed graphs. The first problem with this approach is that the random walk might get stuck in a strongly connected component from which there is no path to the rest of the graph. This problem could be solved by repeatedly restarting the walk. However, there is a more fundamental aspect to this problem. Consider the graph that consists of an (s,t)-path and an arc from each node back to the source s. This graph is strongly connected, and yet the expected time to reach t from s is exponential in the length of the path. A log-space Turing machine cannot count an exponential number of steps, so it is not clear how one could correctly stop the computation if t is not reachable from s. There could be a way out; termination could be based on a random event that has an exponentially small chance of occurring. Since the directed path problem is $\mathcal{NL}$-complete, this would prove that $\mathcal{RL} = \mathcal{NL}$. However, our definition of a randomized Turing machine does not allow such an algorithm. We have assumed that each branch of the computation tree uses the same number of random bits, and in an algorithm based on the previous idea, some branches of the computation will never stop.

One can think of the classes $\mathcal{L}$ and $\mathcal{NL}$ as lower-level analogues of $\mathcal{P}$ and $\mathcal{NP}$. By studying the relationships of $\mathcal{L}$, $\mathcal{NL}$ and $co\text{–}\mathcal{NL}$, one hopes to better understand the relationship of deterministic and nondeterministic computation. It is this point of view that makes the following theorem of Immerman (1988) and Szelepcsényi (1987) one of the biggest surprises in recent developments in complexity theory.

(5.4) Theorem. $\mathcal{NL} = co - \mathcal{NL}$

The proof uses a definition of nondeterministically computing a *function*, similar to the notion of γ-reduction that has been mentioned at the end of the subsection on $\mathcal{NP}$-completeness. We say that a function $f(x)$ **can be computed in nondeterministic logarithmic space** if there is a nondeterministic log-space Turing machine that, on input x, outputs the value $f(x)$ on at least one branch of the computation and on every other branch either stops without an output or also outputs $f(x)$. If $f(x)$ is a Boolean function, then we say that the language L defined by $f(x) = 1$ is **decided in nondeterministic logarithmic space**, which is equivalent to L being in $\mathcal{NL} \cap co - \mathcal{NL}$.

We will prove Theorem (5.4) by showing that the $\mathcal{NL}$-complete directed graph reachability problem can be decided in nondeterministic logarithmic space. Given a directed graph $G = (V, A)$,

a source node s and an integer k, let $f(G,s,k)$ denote the number of nodes reachable from the node s along paths of length at most k.

(5.5) Lemma. *The directed graph reachability problem is decidable in nondeterministic logarithmic space if and only if the function $f(G,s,k)$ can be computed in nondeterministic logarithmic space.*

Proof. To prove the only if direction, we use the fact that the directed graph reachability is $\mathcal{NL}$-complete. If it is decidable in logarithmic space, then so is the problem to recognize if there is a path of length at most k. To compute $f(G,s,k)$, we use the assumed nondeterministic machine for each node v, to decide if v is reachable from s by a path of length at most k. By counting the number of reachable nodes, we obtain $f(G,S,k)$.

To prove the opposite direction, we use the following nondeterministic log-space computation. First compute $f(G,s,n)$. Then for each node v, (nondeterministically) try to guess a path from s to v. Count the number of nodes for which a path has been found. If a path has been found to t, we accept the input. If $f(G,s,n)$ nodes have been reached without finding a path to t, this proves that t is not reachable from s, so we reject. Finally, if the number of nodes that have been reached is less than $f(G,s,n)$, then this is an incorrect branch of the computation, and the computation stops without producing an output. □

(5.6) Lemma. *The function $f(G,s,k)$ can be computed in nondeterministic logarithmic space.*

Proof. The proof is by induction on k. The base case to compute $f(G,s,0)$ is trivial. Assume that a log-space nondeterministic Turing machine has already computed the value $f(G,s,k)$. The value $f(G,s,k+1)$ can be computed by checking the nodes of G one at a time, and counting those for which a path of length at most $k+1$ exists. We shall decide if there exists such a path from s to a particular node v by using $f(G,s,k)$ to decide if there is a path of length at most k to any of the predecessors of v; this can be done by using a variant of the algorithm given in the if part of Lemma (5.5).□

The theorem of Immerman and Szelepcsényi implies that $\mathcal{NSPACE}(S(n)) = co\text{-}\mathcal{NSPACE}(S(n))$ for every $S(n) \geq \log n$. This type of upward inheritability result concerning either space or time complexity classes is usually proved by a so-called **padding argument**. Let L be a language in $\mathcal{NSPACE}(S(n))$. Let the language L' consist of the words formed by appending sufficient \$'s to a word of length n from L to make the length equal to $2^{S(n)}$ (where we assume that \$ is not in the alphabet of L). It is easy to see that $L' \in \mathcal{NL} (= co-\mathcal{NL})$. The nondeterministic log-space Turing machine that proves that $L' \in co-\mathcal{NL}$ can be used to test non-membership in L rather than L'. (We simulate appending the correct number of \$'s to an input by keeping a counter that maintains a record of the input head position, when the simulated head would like to read one of imaginary \$'s.) This shows that $L \in co-\mathcal{NSPACE}(S(n))$.

The hardest problems in $\mathcal{P}$

One important application of log-space computation was introduced by Cook (1973), who used log-space reducibility to introduce a notion of a hardest problem in $\mathcal{P}$. A problem L_1 is **$\mathcal{P}$-complete** if $L_1 \in \mathcal{P}$ and for all $L \in \mathcal{P}$, $L \propto_{log} L_1$. The transitivity of the log-space reduction gives the following theorem.

(5.7) Theorem. *For any $\mathcal{P}$-complete problem L, $L \in \mathcal{L}$ if and only if $\mathcal{L} = \mathcal{P}$.*

Later in this section we shall see that $\mathcal{P}$-completeness also provides evidence that a problem cannot be efficiently solved in parallel. This fact has greatly increased the interest in $\mathcal{P}$-completeness and a variety of problems have been shown to be $\mathcal{P}$-complete. Perhaps the most natural example is the circuit value problem, which was proved $\mathcal{P}$-complete by Ladner (1975b). Given a circuit (with **and**, **or** and **not** gates) with truth values assigned to its input gates, the **circuit value problem** is to compute the output of the circuit.

(5.8) Theorem. *The circuit value problem is $\mathcal{P}$-complete.*

Proof. A computational model that uses circuits and is equivalent to Turing machines has already been discussed, and this equivalence suggests the theorem. The circuit value problem is clearly in $\mathcal{P}$. Given a polynomial-time Turing machine M and an input x, we shall build a circuit that has value 1 if and only if x is in the language accepted by the Turing machine M. The construction is done in a way analogous to the proof of Cook's theorem. We encode the entire computation as a matrix, but instead of constructing a Boolean formula, we build a circuit that computes one row of this matrix from the preceding one to simulate the computation of the Turing machine. □

When proving other problems in $\mathcal{P}$ to be $\mathcal{P}$-complete we shall use the same strategy as was used for $\mathcal{NP}$-completeness: merely reduce from problems that are already known to be $\mathcal{P}$-complete. To facilitate later reductions, note that restricted versions of the circuit value problem are $\mathcal{P}$-complete. For example, this is true for the **restricted monotone circuit value problem**, where the circuits use only **and** and **or** gates, and each node in the directed acyclic graph that defines the circuit has both indegree (**fan-in**) and outdegree (**fan-out**) at most 2. This can be shown by considering the inputs and their negations as separate input variables, and then converting the circuit into an equivalent one that computes the value of every gate and its negation separately.

Dobkin, Lipton and Reiss (1979) proved that each problem in $\mathcal{P}$ log-space reduces to the linear programming problem, and the celebrated result of Khachiyan (1979) showed that it is in $\mathcal{P}$. Valiant gave a straightforward reduction from a restricted circuit value problem that uses linear constraints to trace the value computed by the circuit.

The **maximum flow problem** is an important special case of the linear programming problem. In this problem, we are given a directed graph $G = (V, A)$ with two specified nodes, the source s and the sink t, and a non-negative capacity $u(a)$ assigned to each arc $a \in A$. A feasible flow is a vector $f \in \mathbf{R}^A$ that satisfies the capacity constraints, *i.e.*, $0 \leq f(a) \leq u(a)$ for each arc $a \in A$, and the flow conservation constraints, *i.e.*, the sum of the flow values on the arcs incident to a node $v \neq s, t$ is the same as the sum of the flow values on the arcs incident from v. The value of a flow is $\sum_{a=(v,t)\in A} f(a) - \sum_{a=(t,v)\in A} f(a)$. The maximum flow problem is to find a feasible flow of maximum value. Goldschlager, Shaw and Staples (1982) showed that computing the parity of the maximum flow is also $\mathcal{P}$-complete. We give a log-space reduction from the restricted monotone circuit value problem for circuits in which the output gate is an **or** gate and all gates have fan-in and fan-out at most 2. Number the gates of the circuit so that wires of the circuit go from higher numbered gates to lower numbered gates, and the output gate is numbered 0. The nodes of the graph will correspond to the gates of the circuit with two additional nodes, a source s and a sink

t. The source is connected to the nodes corresponding to inputs assigned the value **true**, where the arc (s, j) has capacity 2^j. If a wire of the circuit connects gates $j \leq i$, then we connect node i to node j with an arc of capacity 2^j. A wire that carries the value **true** will correspond to an arc that has flow value equal to its capacity. To simulate an **and** gate, we connect the corresponding node v to the sink with an arc whose capacity is equal to difference between the capacities of the arcs incident to and incident from v. For an **or** gate, we connect the corresponding node to the source with an arc of the same capacity. The circuit has value 1 if and only if the value of the maximum flow is odd.

There is a collection of $\mathcal{P}$-complete problems that are related to simple polynomial-time algorithms. The **maximal stable set problem**, in which the objective is to find a maximal (not maximum) stable set in an undirected graph, can clearly be solved by a simple greedy algorithm. Similarly, the depth-first search procedure greedily finds a **depth-first search tree**, which is a spanning tree of an undirected graph rooted at a given node r, such that for any edge (u, v) in the graph, the path from r to one of u and v passes through the other (see Aho, Hopcroft & Ullman (1974)). When using one of these procedures, we usually select the first available node in each step, and so we find a specific solution, the lexicographically-first one. Reif (1985) proved that the related problem of finding the lexicographically-first depth first search tree is $\mathcal{P}$-complete. Cook showed that finding the lexicographically-first maximal stable set is $\mathcal{P}$-complete. These results might be surprising since both of these problems are easy to solve in polynomial time. However, $\mathcal{P}$-completeness also provides evidence that the problem is not solvable efficiently in parallel. Consequently, these completeness results support the intuition that these particular procedures are inherently sequential.

Parallel Computation

Parallel computation gives us the potential of curbing the effects of the intractability of certain problems by solving them with a large number of processors simultaneously. In studying parallel algorithms, we shall not be concerned with the precise number of parallel processors used, but rather their order as a function of the input size. We say that a parallel algorithm using $O(p(n))$ processors achieves **optimal speedup**, if it runs in $O(t(n))$ time and the best sequential algorithm known for solving the same problem runs in $O(t(n)p(n))$ time. Efficient algorithms that reach optimal (or near optimal) speedup with a significant number of processors have been found for many of the basic combinatorial problems. Another aspect of parallel computation is concerned with the inherent limitations of using many processors to speed up a computation. To be somewhat realistic, we shall only be interested in algorithms that use a polynomial number of processors. Consequently, we will focus on the possible speedup of polynomial-time sequential computation by parallel processing. The interested reader is referred to the survey of Karp and Ramachandran (1990) for more details and references.

First we define a model of parallel computation. Although many such models have been proposed, one that seems to be the most convenient for designing algorithms is the **parallel random access machine (PRAM)**. The PRAM is the parallel analogue of the random access machine; it consists of a sequence of random access machines called processors, each with its own infinite local random access memory, in addition to an infinite shared random access memory where each memory cell can store any integer, and the input is stored in the shared memory. Each processor knows the input size and its identification number, although otherwise the processors are identical

(*i.e.*, they run the same program). Different variants of the basic PRAM model are distinguished by the manner in which they handle read and write conflicts. In an **EREW PRAM** (exclusive-read exclusive-write PRAM), for example, it is assumed that each cell of the shared memory is only read from and written into by at most one processor at a time. At the other extreme, in a **CRCW PRAM** (concurrent-read, concurrent-write PRAM), each cell of the memory can be read from and written into by more than one processor at a time. If different processors attempt to write different things in the same cell, then the lowest numbered processor succeeds.

To illustrate the power of parallel computation, we give parallel algorithms for two of the problems that we have already discussed. Although finding the lexicographically-first maximal stable set is $\mathcal{P}$-complete, Karp and Wigderson (1985) have proved, surprisingly, that a maximal stable set can be found efficiently in parallel. Similar, much simpler and more efficient randomized algorithms have subsequently been independently discovered by Luby (1986) and by Alon, Babai and Itai (1986).

Let us first review a sequential algorithm for the problem: select a node v and include it in the stable set, delete v and all of its neighbors from the graph; repeat this procedure until all nodes have been deleted. Note that this algorithm requires n iterations for a path of length $2n$. A similar approach can still be used for a parallel algorithm. To make the algorithm fast, one selects a stable set in each iteration (rather than a single node), where the set is deleted along with its neighborhood. The following simple way to choose a stable set is due to Luby. A processor is assigned to each node and each edge of the graph. For a graph of order n, the processor assigned to node v picks a random integer $c(v)$ from 1 to n^4. Next, each processor assigned to an edge compares the values at the two nodes of the edge. The stable set selected consists of those nodes v for which $c(v)$ is strictly larger than the values assigned to any of its neighbors.

This algorithm clearly finds a maximal stable set, but it is less clear that the algorithm runs fast. It can be shown that each iteration is expected to remove a constant fraction of the edges, and consequently, the expected number of iterations is $O(\log n)$. The algorithm can be implemented on a randomized CRCW PRAM in $O(\log n)$ time (if we assume that a processor can choose a random number of size $O(\log n)$ in one step).

Luby gave an elegant version of a technique of Karp and Wigderson that is used to convert certain randomized algorithms into deterministic ones. The technique can be used if the analysis of the randomized algorithm depends only on pairwise, rather than fully, independent random choices. The idea is that pairwise independent random integers between 1 and n^4 can be chosen from a sample space of polynomial size. Each iteration can then be run for each point in the sample space simultaneously, and this ensures that a good sample point is used. This method can be used to convert the above randomized algorithm into a deterministic one.

A simple, but important example of a deterministic parallel algorithm is for the directed graph reachability problem. One way to solve this problem is by repeatedly squaring the adjacency matrix of the graph. More precisely, after the ith iteration, the variable $a_{v,w}$ will indicate whether there exists a path of length at most 2^i from v to w. To update this information we assign a processor to every triplet of the nodes. In the ith iteration, the processor assigned to the triplet (v, u, w) checks whether there exists a path from v to w through u, that has fewer than 2^{i-1} nodes both before and after u. This algorithm, when implemented on a CRCW PRAM uses $O(n^3)$ processors, and runs in $O(\log n)$ time.

Parallel computation can also be considered in the framework of other computational models. By using many processors, one can evaluate a circuit in time proportional to its depth. Hence, the depth of a circuit is the natural analogue of parallel time. When using families of circuits as a model of computation, one must make certain uniformity assumptions. In this case, the usual uniformity assumption is **log(space)-uniformity**; that is, for inputs of size n, the circuit must be generated by a Turing machine using space $O(\log n)$. This allows us to define parallel complexity classes. For $i \geq 0$, the class $\mathcal{AC}^i$ is defined to be the class of languages that are decidable by a family of logspace-uniform circuits with **and**, **or**, and **not** gates of depth $O(\log^i n)$ and size polynomial in n. The class $\mathcal{NC}^i$ is defined in the same way with the additional restriction that each gate in the circuit has constant fan-in. Clearly, $\mathcal{NC}^i \subseteq \mathcal{AC}^i$. Furthermore, an **and** gate or an **or** gate with a polynomial number of inputs can be computed by a constant fan-in circuit in depth $O(\log n)$. Consequently, $\mathcal{AC}^i \subseteq \mathcal{NC}^{i+1}$. Notice that any function that depends on all of its inputs requires depth $\Omega(\log n)$ to be computed on a constant fan-in circuit, and an analogous bound holds in any reasonable model of parallel computation. Analogous to $\mathcal{P}$, membership in the class $\mathcal{NC} = \cup_{i \geq 0} \mathcal{NC}^i = \cup_{i \geq 0} \mathcal{AC}^i$ has become accepted as the theoretical model of efficiently computable in parallel.

Similar complexity classes can be defined using the PRAM model. Stockmeyer and Vishkin (1984) considered the CRCW PRAM without the multiplication, and with additions and subtractions limited to polynomial-size numbers. They showed the following equivalence.

(5.9) Theorem. *For $i \geq 1$, $\mathcal{AC}^i$ is the class of languages recognizable by the above variant of the CRCW PRAM with a polynomial number of processors in $O(\log^i n)$ time.*

Proof. We first show how to simulate a family of circuits by a PRAM. Once we have the appropriate circuit from the family, this is easy to do. The only difficulty lies in constructing the circuit. By our uniformity assumption, the circuit for inputs of size n can be generated by a Turing machine in space $O(\log n)$, and so the problem of deciding if (i, j) is an arc of the circuit is in $\mathcal{L} \subseteq \mathcal{NL}$. We have seen an $O(\log n)$-time CRCW PRAM algorithm for the directed graph reachability problem. To finish this proof, we combine the *proof* of the $\mathcal{NL}$-completeness of the directed graph reachability problem with the above algorithm, to prove that any language in $\mathcal{NL}$ can be recognized in time $O(\log n)$ by a CRCW PRAM.

To prove the other direction we have to simulate a CRCW PRAM by a logspace-uniform family of circuits. The only difficulty is in simulating indirect addressing. It is a good exercise to show that indirect addressing can be simulated by unbounded fan-in circuits in constant depth. □

The maximal stable set algorithm of Luby discussed earlier uses a randomized version of the CRCW PRAM. In a randomized CRCW PRAM algorithm, every processor can choose a logarithmic-size random number in one step. We define the class $\mathcal{RNC}$ to consist of all languages L for which there exists a randomized CRCW PRAM algorithm that runs in polylogarithmic time using a polynomial number of processors and both accepts each word in L and rejects each word not in L with probability at least 2/3.

The restriction in the definition of $\mathcal{NC}^i$ is similar to the exclusive-write restriction in the PRAM model. However, this similarity is somewhat misleading. Hoover, Klawe and Pippenger (1984) proved that a constant fan-in circuit can be converted into an equivalent circuit with both constant fan-in and constant fan-out, while increasing both the size and depth by only a constant factor.

Consequently, for $i \geq 2$, any language in $\mathcal{NC}^i$ can be recognized by an EREW PRAM in $O(\log^i n)$ time using a polynomial number of processors.

Ruzzo (1981) has discovered a characterization of the class $\mathcal{NC}^i$ using alternating Turing machines. He proved that for $i \geq 2$, the class $\mathcal{NC}^i$ is the class of languages that can be recognized by an alternating Turing machine simultaneously using $O(\log n)$ space and $O(\log^i n)$ time. Ruzzo gave a more sophisticated definition for uniformity of circuit families, that is more restrictive for circuits of depth $O(\log n)$. Under his definition, the above equation also holds true for $i = 1$.

The proof of Theorem (5.9) also shows that $\mathcal{NL} \subseteq \mathcal{AC}^1$. As a consequence, a log-space reduction can be simulated efficiently in parallel, and therefore $\mathcal{P}$-completeness provides evidence that a problem is not efficiently solvable in parallel.

(5.10) Theorem. *For any $\mathcal{P}$-complete problem L, $L \in \mathcal{NC}$ if and only if $\mathcal{NC} = \mathcal{P}$.*

On the other hand, $\mathcal{NC}^1 \subseteq \mathcal{L}$. More generally, any logspace-uniform family of constant fan-in circuits of depth at least $\log n$, can be simulated by a Turing machine in space proportional to the depth of the circuit, that is, $\mathcal{NC}^i \subseteq \mathcal{DSPACE}(\log^i n)$. As a corollary, we obtain the following chain of containments:

$$\mathcal{NC}^0 \subseteq \mathcal{AC}^0 \subseteq \mathcal{NC}^1 \subseteq \mathcal{L} \subseteq \mathcal{NL} \subseteq \mathcal{AC}^1 \subseteq \mathcal{NC}^2 \subseteq \ldots \subseteq \mathcal{NC} \subseteq \mathcal{P} \subseteq \mathcal{NP} \subseteq \mathcal{PSPACE}.$$

We have seen that $\mathcal{NC}$ is contained in $\cup_{i \geq 1} \mathcal{DSPACE}(\log^i n)$. By the analogue of Theorem (3.7) for space complexity, this implies that $\mathcal{NC} \neq \mathcal{PSPACE}$. It is fairly easy to argue that $\mathcal{NC}^0$ does not contain the or function, and therefore $\mathcal{NC}^0 \neq \mathcal{AC}^0$. The only further separation result known is that $\mathcal{AC}^0 \neq \mathcal{NC}^1$. This is a landmark result of Furst, Saxe and Sipser (1981) and Ajtai (1983), and will be discussed in the next section.

On a CRCW PRAM one can compute the or of n variables in one step. This suggests that the CRCW PRAM is not a reasonable model of parallel computation. In a more realistic model, computing any function that depends on all of its inputs requires $\Omega(\log n)$ time. The **addition problem** takes two n-bit binary numbers as inputs (where each bit is stored in a separate memory location) and computes their $(n+1)$-bit sum. Surprisingly, even the addition problem can be solved in constant time on a CRCW PRAM. It is a good exercise to figure out how to add two numbers using constant-depth, unbounded fan-in circuits.

Cook, Dwork and Reischuk (1986) have considered the limitations of the CREW PRAM (concurrent-read exclusive-write PRAM). They considered the or function, and showed that it required $\Omega(\log n)$ time. This result is subtler than it seems since with clever use of the concurrent read option one can actually do better than $\log_2 n$.

Next we shall give examples of simple parallel algorithms. First we consider the arithmetic operations. We have just mentioned that addition (and similarly subtraction) can be done in $\mathcal{AC}^0$. The standard shift-and-add algorithm reduces multiplication to the addition of n numbers. Consequently, multiplication can be done in $\mathcal{AC}^1$. In fact, multiplication is in $\mathcal{NC}^1$. This fact can be established by noticing that the addition of three numbers can be reduced to the addition of two in $\mathcal{NC}^0$ (by separately maintaining the carries). Beame, Cook and Hoover (1986) showed that integer division can also be computed by depth $O(\log n)$ constant fan-in circuits. However, the construction is not log-uniform. The best log-uniform family of circuits for integer division is due

to Reif (1986) and has depth $O(\log n \log\log n)$. Evaluating arithmetic expressions with addition and multiplication operations has been considered by Brent (1974) who gave an $O(\log^2 n)$-time parallel algorithm. Miller and Reif (1985) have developed a general tree contraction method, that can manage the binary tree associated with the parenthesis structure of the arithmetic expression. The resulting algorithm runs in $O(\log n)$ time and uses $O(n)$ processors.

Another important area of efficient parallel algorithms deals with the sorting problem. One can easily sort in $\mathcal{AC}^0$, and consequently in $\mathcal{NC}^1$. However, the trivial algorithm uses $O(n^2)$ processors. The first deterministic EREW PRAM algorithm that runs in $O(\log n)$ time using $O(n)$ processors was given by Ajtai, Komlós and Szemerédi (1983). However, the constant in the running time is too large for the method to have any practical use. Cole (1988) later gave a simpler method that achieved the same asymptotic bounds.

The Ajtai, Komlós, Szemerédi algorithm is based on their $O(\log n)$-time sorting network. Consider the following restricted version of the sorting problem. Suppose that n numbers are given in memory locations $1, \ldots, n$. The basic operation for the algorithm is to compare the numbers in two memory locations and rearrange them so that the smaller number is in the location with the smaller index. A **comparison network** can compare and possibly swap the numbers in several disjoint pairs of memory locations at the same time. The network has to be oblivious, in the sense that the pairs to be compared cannot depend on the outcomes of previous comparisons. A comparison network is a **sorting network** if it is guaranteed to rearrange the elements in non-decreasing order. The construction of the Ajtai, Komlós and Szemerédi sorting network is based on constant-degree expander graphs. A bipartite graph with color classes A and B such that $|A| = |B|$ is an **ϵ-expander** if for any $S \subset A$ satisfying $|S| \leq |A|/2$ it follows that the number of neighbors of S in B is at least $(1+\epsilon)|S|$. For every constant $\epsilon < 1$ there exists a constant d such that the union of d randomly chosen perfect matchings is an ϵ-expander with non-zero probability. This establishes that constant-degree ϵ-expanders with $\epsilon > 0$ exist. The first deterministic construction of constant degree expanders was given by Margulis (1973). Since then, several simpler constructions have been given that yield better parameters.

A classical (and much more practical) sorting network, that takes $O(\log^2 n)$-time, is the sorting network of Batcher (1968). It repeatedly uses an $O(\log n)$-time **merging network**, that is a comparison network guaranteed to rearrange the elements in non-decreasing order if the first $n/2$ and the second $n/2$ elements of the input are separately in non-decreasing order. The Batcher merging network can be defined recursively. Assume for simplicity that n is a power of 2. To merge two lists of length $n/2$, we first merge the lists of length $n/4$ consisting of the elements in the odd and even numbered memory cells of the two lists separately. We then compare and possibly swap the pairs in locations $2i$ and $2i+1$ for $i = 1, \ldots, n/2 - 1$ in parallel. It is not difficult to see that this defines an $O(\log n)$-time merging network.

Interestingly, many of the simple parallel graph algorithms are based on matrix operations. The parallel directed graph reachability algorithm presented earlier was in fact based on Boolean matrix multiplication. We shall discuss matrix operations before the more combinatorial elementary graph problems. General matrix multiplication can easily be done on a CRCW PRAM in $O(\log n)$ time using $O(n^3)$ processors. In fact, using the ideas from the more efficient matrix multiplication techniques, one can save somewhat on the number of processors used.

Csanky (1976) has given an $\mathcal{NC}$ algorithm to compute the determinant of a matrix in $O(\log^2 n)$

time. The method actually computes the characteristic polynomial of a matrix A (*i.e.*, the polynomial $\det(A - xI)$, whose constant term is $\det A$). In fact, Csanky's method can be used to compute the characteristic polynomial of a matrix over any field of characteristic 0, where the field operations are assumed to take unit time. Berkowitz (1984) and Chistov have independently extended this result to matrices over arbitrary fields. Over fields of characteristic 0, the last non-zero term of the characteristic polynomial gives the rank of the matrix. This is not true over arbitrary fields. Recently, deterministic parallel algorithms to compute the rank of a matrix over arbitrary fields have been independently discovered by Chistov and Mulmuley (1987), improving on the randomized algorithm of Borodin, von zur Gathen and Hopcroft (1982). As a corollary, we get a parallel algorithm to solve systems of linear equations.

Next consider the minimum cost spanning tree problem. The first $\mathcal{NC}$ algorithm for the problem was given by Bentley in 1980. The following simple algorithm was discovered by Maggs and Plotkin (1988). It uses matrix multiplication over the semiring defined by the operations minimum and maximum instead of addition and multiplication, respectively. For simplicity, we assume that all of the edge costs $c(e)$ are distinct. This implies that the minimum cost spanning tree is unique. An edge (v, w) belongs to the minimum cost spanning tree if and only if the minimum over all (v, w)-paths of the maximum cost of an edge in the path is $c(v, w)$. Let C be the n by n matrix defined by $c_{ij} = c(i, j)$ if $(i, j) \in E$, $c_{ij} = 0$ if $i = j$ and $c_{ij} = +\infty$ otherwise. An edge (v, w) is in the unique minimum cost spanning tree if and only if the nth power of the matrix C with the semiring operations has $c(i, j)$ as its (i, j)th entry.

Consider the problem of testing if a graph is planar. Ja'Ja' and Simon (1982) have observed that the planarity of a graph can be tested efficiently in parallel using Tutte's characterization that states that a straight line embedding of a 3-connected graph can be obtained by solving a set of linear equations. One drawback of this approach is that Tutte's theorem only works for 3-connected graphs. Ja'Ja' and Simon extended this approach to check whether an arbitrary graph is planar, but the algorithm only constructs the plane embedding for 3-connected graphs. Miller and Reif (1985) have used the tree contraction method mentioned earlier to extend this algorithm to arbitrary graphs. Klein and Reif (1988) have recently given an efficient planarity testing algorithm that does not rely on matrix inversion, and is also more efficient in terms of the number of processors used.

We next turn our attention to elementary graph searching techniques. A parallel algorithm for the directed graph reachability problem based on Boolean matrix multiplication has been discussed earlier. This algorithm can be used to construct a breadth-first search tree of the graph. It is an important open problem to find a polylogarithmic-time parallel algorithm for breadth-first search that uses fewer processors than are needed for matrix multiplication. The undirected graph reachability problem can be solved more efficiently. Shiloach and Vishkin (1982) have given an implementation of an algorithm by Hirschberg, Chandra and Sarwate (1979) that takes $O(\log n)$ time on a CRCW PRAM using $O(m + n)$ processors. Although this algorithm is fairly efficient, it is still not optimal in the sense that the product of the time and the number of processors is more than the running time of the best sequential algorithm known for the problem. Gazit (1986) gave an $O(\log n)$-time randomized CRCW PRAM algorithm that only uses $O((m+n)/\log n)$ processors. No deterministic $O(\log n)$-time optimal algorithm is known. These algorithm can be used to construct a spanning forest of the graph, but not necessarily the breadth-first search tree.

Another elementary search technique is depth-first search. No efficient deterministic parallel algorithm is known for finding a depth-first search tree. Aggarwal and Anderson (1988) have given

a randomized parallel algorithm for finding a depth-first search tree (first for undirected graphs, and later with Kao (1989), they extended the result to directed graphs). The algorithm is based on a randomized algorithm to find a perfect matching in parallel that, in turn, is based on matrix inversion. As a consequence, the algorithm uses a large number of processors, and is not efficient in practice.

Lovász (1985) has given an $\mathcal{NC}$ algorithm to find the ear-decomposition of a directed graph. The **ear decomposition** $D = [P_0, \ldots, P_k]$ of a directed graph $D = (V, E)$ is a partition of the edge set of D into simple directed paths and circuits called **ears**, such that P_0 is a simple circuits, and for each $i \geq 1$, P_i is a simple path or circuit, with both ends in a previous ear and none of the internal nodes in any of the previous ears. An ear-decomposition is said to be **open** if the only circuit in the decomposition is P_0. Lovász has used the ear-decomposition algorithm to find a minimum cost branching of a directed graph. (A branching is a directed tree where each node except a specified root has indegree 1.) In undirected graphs, an ear-decomposition can be found more efficiently. Lovász's algorithm for undirected graphs runs in $O(\log n)$ time and uses a linear number of processors. The ear-decomposition algorithm does not in general give an open ear-decomposition (even if one exists). Maon, Schieber and Vishkin (1986) modified the algorithm to find an open ear-decomposition with the same complexity. Ear-decomposition techniques can be used to check whether a graph is 2-edge or 2-node connected, though a parallel algorithm using $O(\log n)$ time and a linear number of processors had been given earlier by Tarjan and Vishkin (1985).

Miller and Ramachandran (1987) used the ear-decomposition technique to design an $\mathcal{NC}$ algorithm for finding 3-connected components that uses only a linear number of processors. The first $\mathcal{NC}$ algorithm to decide whether a graph is 3-connected is due to Ja'Ja' and Simon (1982); however, they were unable to find the 3-connected components of a graph. Miller and Reif (1985) used tree contraction methods to find 3-connected components in parallel.

One of the most important open problems in parallel computation is to give an efficient parallel algorithm that finds a maximum matching in a graph. Recall the polynomial time randomized algorithm of Lovász (1979) that was discussed in introducing the idea of randomized computation. The algorithm decides whether a graph has a perfect matching by computing the determinant of a single randomly chosen matrix. Determinants can be computed efficiently in parallel. This establishes that deciding whether a graph has a perfect matching is in $\mathcal{RNC}$. The same idea shows that the exact matching problem is in $\mathcal{RNC}$. This is an interesting case of a problem in $\mathcal{RNC}$ that is not known to be in $\mathcal{P}$.

It is important to notice that decision and search problems are not equivalent for parallel computation. There does not appear to be a way to turn the $\mathcal{RNC}$ algorithm of Lovász that decides whether a perfect matching exists, into an $\mathcal{RNC}$ algorithm that finds a perfect matching if one exists. The difficulty seems to lie in identifying a particular matching. The graph might have several matchings, and the different processors working in parallel have to coordinate which perfect matching to choose. An $\mathcal{RNC}$ algorithm to find a perfect matching in a graph (if one exists) has been given by Karp, Upfal and Wigderson (1986). Later Mulmuley, Vazirani and Vazirani (1987) gave a very elegant and more efficient algorithm. Here we briefly outline the later algorithm in the case of bipartite graphs. For a bipartite graph $G = (U \cup W, E)$ with n nodes in each of the color classes U and W, independently assign to each edge e an integral cost $c(e)$ selected uniformly at random between 1 and $2|E|$. The algorithm is based on a lemma which states that with probability

at least 1/2, the minimum cost perfect matching is unique. The algorithm proceeds as follows: define an n by n matrix A with entries $a_{ij} = 2^{c(e)}$ if $e = (u_i, v_j) \in E$ and 0 otherwise. The 2^c terms in the expansion of the determinant of the matrix A correspond to matchings of cost c. Now assume that the minimum cost perfect matching is unique, and has cost c. The determinant expansion has exactly one term that is not divisible by 2^{c+1} and consequently, $\det A \neq 0$. Let b_{ij} denote the (i,j)th entry of A^{-1}. An edge $e = (u_i, v_j)$ belongs to the unique minimum cost matching if and only if the highest power of 2 dividing b_{ij} is $2^{c-c(e)}$.

As a corollary of the $\mathcal{RNC}$ perfect matching algorithm, one immediately has an $\mathcal{RNC}$ algorithm to find a maximum matching (or to find a matching of a given size). The resulting algorithm will output a matching that is a maximum-size matching with high probability. Karloff (1986) has noticed that the Gallai-Edmonds structure theorem (see Chapter 3) implies that a maximum matching algorithm can be used to produce a minimum odd-set cover. This gives a Las Vegas-type randomized maximum matching algorithm, that is, an algorithm that is likely to find a matching proven to be maximum. Further consequences of the perfect matching algorithm are $\mathcal{RNC}$ algorithms for finding a minimum cost perfect matching if the costs are given in unary, and for the maximum flow problem with capacities given in unary. Recall that the maximum flow problem with capacities given in binary is $\mathcal{P}$-complete. It is an interesting open problem whether a minimum cost matching problem (with costs given in binary) is in $\mathcal{RNC}$.

There are several important problems that are known to be in $\mathcal{P}$, but are not known to be in $\mathcal{NC}$ (and even $\mathcal{RNC}$), and yet are not known to be $\mathcal{P}$-complete. In addition to the ones already mentioned, perhaps the most prominent is the problem of computing the greatest common divisor of two integers. A fast parallel algorithm for this would have implications for a variety of number-theoretic problems, including primality testing, which have so far resisted attempts to classify their parallel complexity. One major element of Luks' (1982) polynomial-time algorithm to test if two constant-degree graphs are isomorphic, is an algorithm of Furst, Hopcroft and Luks (1980) to compute the order of a permutation group, and a recent result of Babai, Luks and Seress (1988) provides a parallel analogue of that result. Nonetheless, the problem a providing an efficient parallel algorithm for testing constant-degree graph isomorphism remains an important open problem.

6 Attacks on the $\mathcal{P}$ versus $\mathcal{NP}$ Problem

In previous sections, we have discussed techniques for providing evidence of the intractability of concrete problems (*e.g.*, $\mathcal{NP}$-completeness). However, this evidence is not a proof, as long as the fundamental problems remain open. In this section, we will be concerned with approaches to settling the $\mathcal{P}$ versus $\mathcal{NP}$ question.

First we consider the generalization of these unresolved questions to oracle Turing machines. For example, we will consider questions such as whether $\mathcal{P}^A = \mathcal{NP}^A$ for an oracle A. These problems were introduced in order to gain insight into the techniques that might be used to settle the fundamental open problems.

Recent results, most importantly those of Furst, Saxe and Sipser (1981) and Razborov(1985a), have provided a major breakthrough in the development of techniques for proving lower bounds on the computational complexity of certain problems. These methods have succeeded by considering computational models with serious limitations on some of the resources. The main lines of such

research are for polynomial-size, constant-depth circuits and polynomial-size monotone circuits (*i.e.*, circuits without negations that use only **and** and **or** gates). The techniques developed for these results are combinatorial in nature.

The Turing machine model has also been used to prove lower bounds on the complexity of certain problems. We have mentioned lower bounds based on diagonalization arguments in Section 3. We will discuss three combinatorial techniques. We discuss the classical crossing sequence technique, where the lower bound on time is based on the amount of information that can pass from one end of the tape to the other. Next we discuss a result of Hopcroft, Paul and Valiant (1977) that relates the time and space complexity of a problem. We conclude with the landmark result of Paul, Pippenger, Szemerédi and Trotter (1983), that proves that $DTIME(n) \neq NTIME(n)$.

Relativized complexity classes

In the previous sections we have introduced the notion of an oracle Turing machine and complexity classes defined by such Turing machines. Several of the theorems and proof techniques discussed in the previous sections easily extend to these complexity classes (*i.e.*, they **relativize**). By generalizing diagonalization and simulation results, we get theorems such as $\mathcal{AP}^A = \mathcal{PSPACE}^A = \mathcal{NPSPACE}^A$ for every oracle A.

The main result concerning oracle complexity classes, due to Baker, Gill and Solovay (1975), is that the answer to the relativized versions of the fundamental open problems depends on the oracle.

(6.1) Theorem. *There exist languages A and B such that $\mathcal{P}^A \neq \mathcal{NP}^A \neq \text{co}-\mathcal{NP}^A$, and $\mathcal{P}^B = \mathcal{NP}^B = \text{co}-\mathcal{NP}^B$.*

The existence of an appropriate language A for the first alternative is proved by diagonalization. Any $\mathcal{PSPACE}$-complete language B will provide an example of the second alternative. As a corollary, we can assert that techniques that relativize cannot settle the $\mathcal{P}$ versus $\mathcal{NP}$ problem.

One might wonder which one of the two alternatives provided by the theorem of Baker, Gill, and Solovay is more typical. Recall the definition of a random oracle. We say that a statement **holds for a random oracle** if the probability that the statement holds for a random oracle is 1. The Kolmogorov 0-1 law of probability theory states that if an event $\mathcal{A}$ is determined by the independent events, $\mathcal{B}_1, \mathcal{B}_2, \ldots$ and $\mathcal{A}$ is independent of any event that is a finite combination of the events $\mathcal{B}_i$, then the probability of $\mathcal{A}$ is either 0 or 1. Applying this to the event $\mathcal{A}$ that $\mathcal{P}^A = \mathcal{NP}^A$ and the events $\mathcal{B}_i$ that the ith word is in the language A, we see that the probability of $\mathcal{P}^A = \mathcal{NP}^A$ for a random oracle A is either 0 or 1. Bennett and Gill (1981) have provided the final answer to this question.

(6.2) Theorem. *$\mathcal{P}^A \neq \mathcal{NP}^A$ and $\mathcal{NP}^A \neq co-\mathcal{NP}^A$ for a random oracle A.*

Furst, Saxe and Sipser (1981) discovered the following connection between constant-depth circuit lower bounds and separating relativized complexity classes. This result provides further motivation for studying restricted models of computation. The **parity function** indicates the sum modulo 2 of the bits of the input.

(6.3) Theorem. *If for any constant c, any family of constant-depth circuits computing the parity function has $\Omega(n^{\log^c n})$ gates, then there exists an oracle A that separates the polynomial-time hierarchy from $\mathcal{PSPACE}$; that is, $\cup_{k\geq 1}\Sigma_k^{\mathcal{P},A} \neq \mathcal{PSPACE}^A$.*

The idea of the proof is as follows. For any oracle A, we define the language $L(A) = \{1^n | A$ contains an odd number of strings of length $n\}$. $L(A)$ is in $\mathcal{PSPACE}^A$ for any oracle A. Using diagonalization and the lower bound assumed in the theorem, one can construct an oracle A such that $L(A)$ is not in $\cup_{k\geq 1}\Sigma_k^{\mathcal{P},A}$. First one shows that it is sufficient to consider alternating Turing machines in which every branch of the computation has a single oracle question at the end of the branch. The computation tree of such an alternating Turing machine with k levels of alternation corresponds to a depth-k circuit where the oracle answers are the inputs.

Another related problem is to separate the finite levels of the polynomial-time hierarchy using oracles. Baker and Selman (1979) proved the existence of an oracle A such that $\Sigma_2^{\mathcal{P},A} \neq \Pi_2^{\mathcal{P},A}$ (and consequently, $\Sigma_2^{\mathcal{P},A} \neq \Sigma_3^{\mathcal{P},A}$). Sipser (1983a) showed a result similar to Theorem (6.3) for the finite levels of the polynomial-time hierarchy. It replaces parity by a certain function F_k, that is in some sense, a generic function that is computable in depth k.

(6.4) Theorem. *If for every k and c, any family of circuits of depth $k-1$ computing F_k has $\Omega(n^{\log^c n})$ gates, then for every k there exists an oracle A_k, such that $\Sigma_k^{\mathcal{P},A_k} \neq \Sigma_{k+1}^{\mathcal{P},A_k}$.*

The circuit complexity results that are needed in these theorems have been proved in a series of papers by Furst, Saxe and Sipser (1981), Sipser (1983a), Yao (1985) and Hastad (1986). These results will be discussed in the next subsection. Based on stronger circuit complexity results, Babai (private communication) and Cai (1986) separated $\mathcal{PSPACE}$ from the finite levels of the hierarchy by random oracles. The question whether random oracles separate the finite levels of the polynomial-time hierarchy remains open. Another open question concerning random oracles is whether $\mathcal{P}^A = \mathcal{NP}^A \cap \text{co} - \mathcal{NP}^A$ for a random oracle A.

Similar oracle separation results have also been proved for classes of languages defined by randomized proof systems. For example, Santha (1989) provided an oracle B for which $\mathcal{MA}^B \neq \mathcal{AM}^B$. Aiello, Goldwasser and Hastad (1986) provided an oracle under which $\mathcal{IP}$ is not contained in the polynomial-time hierarchy. In contrast with the speedup theorem of Babai and Moran mentioned earlier, Aiello, Goldwasser and Hastad (1986) proved that for any functions f and g such that $g(n) = o(f(n))$, there exists an oracle B such that $\mathcal{AM}^B(f(n)) \neq \mathcal{AM}^B(g(n))$[3] .

Relating circuit complexity to Turing machine complexity

Among the three models of computations introduced in Section 2, circuits have the simplest, most combinatorial structure, and therefore seem to be most amenable for proving lower bounds. This is especially true if we consider nonuniform families of circuits; that is, we forget about the

[3]In considering of the results of Lund, Fortnow, Karloff & Nisan and Shamir, it is significant to note that these techniques do not relativize, as is evident from the oracle results known for these classes.

uniformity requirement. This simple combinatorial structure has made it possible to use combinatorial techniques for proving lower bounds. Boppana and Sipser (1990) give an excellent survey of circuit complexity lower bounds.

Before discussing lower bounds in detail, we would like to understand better the relationship between uniform and nonuniform circuit families. Nonuniform families of circuits are clearly stronger than uniform ones. It is easy to give examples of nonrecursive functions that can be computed by small (nonuniform) circuits. It is conceivable that $\mathcal{P} \neq \mathcal{NP}$, and yet all $\mathcal{NP}$-complete problems have polynomial-size circuits.

To measure the amount of nonuniformity in a family of circuits, we introduce a nonuniform version of the Turing machine. A **nonuniform Turing machine**, in addition to the usual features of a deterministic Turing machine, has an associated sequence of advice strings $a_1, a_2, \ldots$, (to be written onto an additional **advice tape**). The nonuniform Turing machine **accepts** an input of length n if with a_n written on its advice tape, the Turing machine accepts the input. Note that there is an important difference between the acceptance rule for a nonuniform and a nondeterministic Turing machine. A nonuniform Turing machine has to use the same advice for every input of length n. On the other hand, the definition does not restrict the behavior of a nonuniform Turing machine with incorrect advice.

We can measure the nonuniformity of a Turing machine by the length of the advice. A language is in $\mathcal{P}/log$ if and only if there exists a nonuniform Turing machine that accepts L, runs in polynomial time, and whose family of advice strings $\{a_n\}$ has length $O(\log n)$. Similarly, $\mathcal{P}/poly$ is the set of languages that can be accepted by a polynomial-time nonuniform Turing machine. It is easy to see that a language L can be recognized by a family of polynomial-size (nonuniform) circuits if and only if it is in $\mathcal{P}/poly$. (The right advice is the description of the circuit.) Karp and Lipton (1982) used simple self-reducibility properties to prove following theorem, that showed that uniform and nonuniform complexity classes are not too far from each other.

(6.5) Theorem. *The following relationships exist between non-uniform and uniform complexity classes: (1) if $\mathcal{NP} \subseteq \mathcal{P}/log$ then $\mathcal{P} = \mathcal{NP}$; (2) if $\mathcal{NP} \subseteq \mathcal{P}/poly$ then $\Sigma_2^{\mathcal{P}} = \Pi_2^{\mathcal{P}}$; (3) if $\mathcal{PSPACE} \subseteq \mathcal{P}/poly$ then $\mathcal{PSPACE} = \Sigma_2^{\mathcal{P}}$.*

The following theorem which is based on an idea of Adleman (1978) shows a natural way in which nonuniform families of circuits are more powerful that their uniform analogues.

(6.6) Theorem. *$\mathcal{BPP} \subseteq \mathcal{P}/poly$.*

Proof. Consider a randomized Turing machine RM that recognizes a language L. If we run the algorithm k times and take the majority decision, the error probability decreases exponentially in k. Therefore, we can assume without loss of generality that RM accepts every word $x \in L$ with probability $1 - 2^{-n-1}$ and rejects every word $x \notin L$ with probability $1 - 2^{-n-1}$, where n denotes the length of x.

To prove the theorem, we can concentrate on inputs of a given length n. Let $p(n)$ denote the maximum number of random bits used along a computation path for inputs of length n. We say that a string r of length $p(n)$ is **good for an input** x if the Turing machine RM with r on its

randomizing tape accepts x if and only if $x \in L$. A string r of length $p(n)$ is **good** if it is good for every input of length n. We claim that good strings exist. This proves the theorem, since a good string can serve as the advice string.

For a given input x of length n, a random string r of length $p(n)$ is good with probability at least $1 - 2^{-n-1}$ by assumption. Consequently, a random string r of length $p(n)$ is good with probability at least $1 - 2^n \cdot 2^{-n-1} = 1/2$. Hence, good strings exist. □

One can similarly define the notion of a **nonuniform nondeterministic Turing machine** that has both a guess tape and an advice tape. A language L belongs to $\mathcal{NP}/poly$ if there exists a nonuniform nondeterministic Turing machine that recognizes L, and runs in polynomial time. Analogously to Theorem (6.6) one can prove that $\mathcal{AM} \subseteq \mathcal{NP}/poly$. This gives another reason to view $\mathcal{AM}$ as a complexity class just above $\mathcal{NP}$.

Using an easy counting argument, Shannon (1949) showed that almost all Boolean functions of n variables require exponential-size circuits. For problems in $\mathcal{NP}$, the best lower bound known is linear; Blum (1984) has proved a $3n - o(n)$ bound.

Constant-depth circuits

The first superpolynomial lower bound in a restricted model of computation was given by Furst, Saxe and Sipser (1981) and Ajtai (1983) for constant-depth circuits. They proved independently, that every constant-depth circuit computing the parity function has superpolynomial size. Yao (1985) established a truly exponential lower bound by extending the techniques of Furst, Saxe and Sipser. Hastad (1986) has further strengthened the bound and greatly simplified the proof.

All of these proofs are based on probabilistic combinatorial arguments (see Chapter 33 for an overview of this method). We will give a brief outline of Hastad's version of the proof of Furst, Saxe, and Sipser. First note that we can assume without loss of generality that the circuit has alternating levels of **and** and **or** gates and uses negations only at the input level. (Every constant-depth circuit can be converted into one that satisfies these restrictions and has depth and size at most twice the original.) We shall refer to the levels by their distance from the inputs, so that the inputs are at level 0. The proof is by induction on the depth of the circuit. The base case, that any depth-2 circuit computing the parity function has exponential size, is trivial. The idea of the induction is the following. Consider a gate at the second level. Suppose, for example, that it is an **and** gate. We will try to express the function computed by this gate as an **or** of **and**s. By substituting such a description for the original one at each gate at the second level, we get a constant-depth circuit computing the same function whose second and third levels are **or** levels. By collapsing these two levels into one, we get a circuit of smaller depth that computes the same function. If this change did not exponentially increase the size of the circuit, the resulting circuit would contradict the induction hypothesis.

The main idea is to use random restrictions. In a **random restriction**, we select a partial assignment for the variables according to the following probability distribution R_p: every input variable is independently assigned the value 0 and 1, each with probability $(1 - p)/2$, and kept unassigned with probability p. Every restriction of the parity function is a parity function of the remaining variables. Furst, Saxe and Sipser showed that one can choose p so that the number of input variables does not decrease too much, and yet the gates at the first and second levels

in the resulting circuit can be swapped without an exponential blowup. This yields the desired contradiction. The following lemma, due to Hastad (1986), is the essence of the proof.

(6.7) Lemma. *Let f be a function that is given in the form of an* and *of* or *gates where each* or *gate has fan-in at most t. A random restriction of f from R_p can be written as the* or *of* and *gates where each* and *gate has fan-in at most s, with probability at least α^s where $\alpha = 2pt/\log\phi$, and ϕ is the golden ratio ($= (1+\sqrt{5})/2$).*

By using these arguments, Hastad gave a simple proof of Yao's exponential lower bound on the size of constant-depth circuits for the parity function.

(6.8) Theorem. *If $\{C_n\}$ is a family of depth-d circuits that computes the parity function, then C_n, the circuit for inputs of size n, has at least $\Omega(2^{\frac{1}{10}n^{1/(d-1)}})$ gates.*

We have seen that parity is in $\mathcal{NC}_1$, and so we get the following theorem as a corollary.

(6.9) Theorem. $\mathcal{AC}_0 \neq \mathcal{NC}_1$.

There is a parallel series of results aimed at separating the finite levels of the polynomial-time hierarchy by oracles. The related problem in circuit complexity is whether circuits of depth k are more powerful than circuits of depth $k-1$. Sipser introduced the function F_k, mentioned earlier, that is in some sense a generic function computable in depth k, and proved a superpolynomial lower bound on the size of any circuit of depth $k-1$ computing F_k. An exponential bound was claimed by Yao (1985) and proved by Hastad using the technique outlined above.

In a series of results, Razborov (1987) and Smolenski (1987) have given simpler proofs of Theorem (6.9), and in fact, have proved stronger results. Razborov considered circuits that may have parity gates, in addition to the usual and, or and not gates, where a parity gate computes the parity of the number of 1's in the input. Razborov has proved that every constant-depth circuit that computes the majority function using these gates has exponential size; the **majority function** has value 1 if at least half of the input variables are 1. Smolenski simplified the proof, and generalized it to prove that every constant-depth circuit that decides whether the sum of the inputs is 0 modulo p using and, or and modulo-q gates has exponential size, where p and q are powers of different primes.

The proof uses the **approximation technique** invented by Razborov (1985a) in his innovative paper on monotone circuit complexity (that will be discussed later). In fact, this application of the technique is much simpler than the original one. To give an idea of the approximation technique, we describe the proof that a constant-depth circuit computing the majority function has exponential size.

Consider a circuit that computes the majority function. We can assume without loss of generality that the circuit uses only parity and or gates , since they can be used to simulate both and and not gates within constant depth. The idea of the proof is to introduce "approximations" of the gates used during the computation. Using the approximate gates, instead of the real gates, one computes an approximation of the majority function. The quality of the approximation will be

measured in terms of the number of inputs for which the modified circuit differs from the original; an approximation is considered good if this number is small when only a single gate of the original circuit is modified. The main point of the approximation is to keep the computed function "simple" in some sense. We will show that every simple function differs from the majority function on a significant fraction of the inputs. The approximation of the majority function computed by the approximated gates is simple and therefore "far" from the real majority. From this we can conclude that many approximations had to occur. The technique used to approximate the gates and the definition of "simplicity" used is due to Razborov. We will present Smolenski's shorter proof that the majority function cannot be approximated closely with simple functions.

In fact, the approximation technique is applied not to the majority function, but to a closely related function, the **exactly-k function**, f_k. This function is 1 when exactly k of the inputs are 1. It is easy to see that if there is an $s(n)$-size family of circuits that computes the majority function in depth d, then, for every k, there is an $O(s(2n))$-size family of circuits of depth $d+1$, that computes the exactly-k function. Therefore, any exponential lower bound for the exactly-k function implies a similar bound for the majority function.

Every Boolean function can be expressed as a polynomial over the two-element field $GF(2)$. If p_1 and p_2 are polynomials representing two functions, then $p_1 + p_2$ is the polynomial corresponding to the parity of two functions. The polynomial $p_1 p_2$ corresponds to their **and**, which makes it easy to see that $(p_1 + 1)(p_2 + 1) + 1$ corresponds to their **or**. The measure of simplicity for this proof is the degree of the polynomial representing the output of the circuit. Note that the polynomials corresponding to the inputs have degree 1; *i.e.*, they are very simple. Since the degree of the corresponding polynomial is not increased by computing the parity, the parity gates do not have to be approximated. On the other hand, using unbounded fan-in **or** gates can greatly increase the degree of the computed polynomials. We will approximate the **or** gates so that the approximated function will have fairly low degree. As a consequence, the approximation of the exactly-k function will have degree at most $\sqrt{n}$.

Before showing how to approximate, we show that polynomials of degree at most $\sqrt{n}$ have to differ from the exactly-k functions on a significant fraction of the inputs.

(6.10) Lemma. *If $p_1, \ldots p_n$ are polynomials of degree at most $\sqrt{n}$, then for at least one value of k, the polynomial p_k differs from f_k on at least $2^n/3n$ inputs.*

Proof. Let A be the set of inputs α for which $f_k(\alpha) = p_k(\alpha)$ for every k. Consider the set of all Boolean functions on A. These functions form a linear space of dimension $|A|$. On the other hand, we will show that each of these functions is equal to a polynomial of degree at most $n/2+\sqrt{n}$ on the set A. Actually, instead of arbitrary polynomials, it suffices to consider multi-linear polynomials since $x^2 = x$ over the field $GF(2)$. Polynomials of this low degree form a linear space whose dimension is equal to the number of different multi-linear monomials of degree at most $n/2 + \sqrt{n}$, which is

$$\sum_{i=0}^{n/2+\sqrt{n}} \binom{n}{i} \leq \frac{2}{3} 2^n.$$

Therefore, we can conclude that $|A| \leq \frac{2}{3}2^n$, so that at least one of the polynomials p_k differs from f_k on at least $\frac{1}{3n}$ of the inputs.

It remains to show that every Boolean function on A is equal to a polynomial of degree at most $n/2 + \sqrt{n}$. For every $\alpha \in A$ consider the function f_α that is 1 if and only if the input is equal to α. Let $I(\alpha)$ denote the set of indices i for which $\alpha_i = 1$, and let $k = |I(\alpha)|$. The function f_α can be expressed as

$$f_\alpha = (\Pi_{i \in I(\alpha)} x_i) f_k = (\Pi_{i \notin I(\alpha)} (1 + x_i)) f_k.$$

By replacing f_k with p_k in one of these two expressions, depending on whether k is less than or more than $n/2$, we get a polynomial of degree at most $n/2 + \sqrt{n}$ that is equal to f_α on A. Every Boolean function on A can be expressed as the sum of functions of the form f_α, and therefore is a polynomial of degree at most $n/2 + \sqrt{n}$. □

Now we are ready to consider the problem of approximating the or gates. The following lemma will serve as the basis for the approximation.

(6.11) Lemma. *For every Boolean function $f = \vee_{i=1}^{m} g_i$, there is a function f' that differs from f on at most 2^{n-l} inputs, and whose corresponding polynomial has degree at most l times the maximum degree of the polynomials representing the functions g_i, $i = 1, \ldots m$.*

Proof. Randomly select a subset $I \subseteq \{1, \ldots, m\}$, where each i is independently included in I with probability 1/2. Let $f(I)$ be the parity of the g_i indexed by I. In this way, independently construct l functions $f_1, \ldots, f_l$, and consider $f' = \vee_{i=1}^{l} f_i$. We claim that f' satisfies the requirements of the lemma with non-zero probability. It is clear that the degree of the polynomial for f' is at most l times the maximum degree of the polynomials for the g_i. Furthermore, consider an input α; we claim that the probability that $f'(\alpha) = f(\alpha)$ is at least $1 - 2^{-l}$. To see this, consider two cases. If $g_i(\alpha) = 0$ for every i, then both $f(\alpha) = 0$ and $f'(\alpha) = 0$ for any such choice of f'. On the other hand, if there exists an index i for which $g_i(\alpha) = 1$, then $f(\alpha) = 1$ and for each j, $f_j(\alpha) = 1$ independently with probability 1/2. Therefore, $f'(\alpha) = 1$ with probability $1 - 2^{-l}$, and the expected number of inputs on which $f' = f$ is at least $2^n - 2^{n-l}$. Random variables must sometimes achieve their expected value, so that there exists a polynomial f', constructed as above, that differs from f on at most 2^{n-l} inputs. □

To finish the proof, assume that for every k, there is an $s(n)$-size family of depth-d circuits to compute the exactly-k function. Now apply Lemma (6.11), with $l = n^{\frac{1}{2d}}$ to approximate the or gates in these circuits for inputs of size n. The functions computed by the gates at the ith level will be approximated by polynomials of degree at most l^i. Therefore, each resulting approximation p_k of the exactly-k function will have degree at most $\sqrt{n}$. Lemma (6.11) implies that for any index k, the polynomial p_k differs from the exactly-k function on at most $s(n)2^{n-l}$ inputs. By Lemma (6.10), this shows that $s(n)2^{n-l} \geq \frac{1}{3n} 2^n$. In other words, $s(n) \geq \frac{1}{3n} 2^l$, which establishes the desired exponential lower bound.

Monotone circuit complexity

Another restriction of the circuit model is the **monotone circuit**, where not gates are not allowed, leaving only and and or gates. Note that any function computed by such a circuit is **monotone**, in the sense that changing some of the 0's in the input into 1's cannot change the

function value from 1 to 0. Moreover, observe that any monotone function can be computed by a monotone circuit (for example, by using the disjunctive normal form). Examples of monotone functions include the majority function and the k-clique function (whose input is an undirected graph G where the input variables correspond to the potential edges of the graph and the function value is 1 if the graph contains a clique of size k). The **monotone circuit complexity** of a monotone function f is the minimum number of gates in a monotone circuit computing f. The notion of monotone circuit complexity was introduced with the hope that exponential lower bounds on the monotone circuit complexity of monotone problems in $\mathcal{NP}$ will eventually lead to exponential bounds for general circuits. Prior to the landmark paper of Razborov, the best lower bound on a monotone property in $\mathcal{NP}$ was linear.

In a groundbreaking paper, Razborov (1985a) gave a superpolynomial lower bound on the monotone circuit complexity of the clique problem. Shortly afterwards, Andreev (1985) used similar techniques to obtain an strictly exponential lower bound on a less natural $\mathcal{NP}$-complete problem. Alon and Boppana (1987), by strengthening the combinatorial arguments of Razborov, proved an exponential lower bound on the monotone circuit complexity of the k-clique function.

The proof uses a more elaborate application of the approximation technique. Recall the main idea of proving lower bounds by approximation: for every gate in the circuit, one introduces an approximation. The approximations are defined so that the function computed by the approximated circuit is "close" to the real function. The main point of the approximation is to keep the approximating functions "simple" in some sense. Finally, one has to prove that no simple function is close to the k-clique function. From this, one concludes that many approximations occurred.

In the constant-depth proof, we considered an approximation to be close if it differed from the original function on only a small number of inputs. In this proof, a more sophisticated definition of closeness is used. First, we will restrict attention to a specified subset of test inputs: the edge-minimal accepted inputs (the k-cliques), and the edge-maximal rejected inputs (the maximal $(k-1)$-colorable graphs). Furthermore, for each test input, we will only count errors in one direction: for any circuit, we will measure the distance of its approximation from the original by the additional number of k-cliques rejected and the additional number of maximal $(k-1)$-colorable graphs that are accepted. Razborov's proof shows that every small monotone circuit either accepts most maximal $(k-1)$-colorable graphs, or rejects most k-cliques.

The objective in approximating is to keep the functions computed simple in some sense. The measure of simplicity is also more involved than in the constant-depth case. It is defined in terms of properties of the system of subsets whose characteristic vectors are minimal members of the corresponding language. To show that the k-clique function cannot be approximated too closely with simple functions is easy. The difficulty lies in the proof that and and or gates can be approximated well. The main technique used in the approximation is the sunflower theorem of Erdös and Rado.

It is an interesting problem to study the power of negation; how different can the monotone and non-monotone circuit complexity of a monotone function be? Pratt (1975b) proved that the monotone circuit complexity of Boolean matrix multiplication is $\Theta(n^3)$. This, together with Strassen's $O(n^{\log_2 7})$ matrix multiplication technique, proves that these two notions are distinct. Razborov (1985b), using techniques similar those used for the clique lower bound, showed that the perfect matching problem (that is in $\mathcal{P}$) has superpolynomial monotone circuit complexity, thereby establishing a superpolynomial gap. Tardos (1988) showed that this could be increased to

an exponential separation, by combining the arguments of Razborov (1985a), Alon and Boppana (1987) and results of Grötschel, Lovász and Schrijver (1981) on the polynomial computability of a graph function θ that is closely related to the clique function. (See Chapter 28 for further details concerning the function θ.)

Karchmer and Wigderson (1988) considered monotone circuits in trying to establish super-logarithmic lower bounds on the depth required to compute certain functions with polynomial-size circuits. They considered the undirected reachability problem (whose inputs are the potential edges of a graph with two special nodes s and t). This problem is clearly in $\mathcal{P}$, and in fact, it can be decided by a family of polynomial-size monotone circuits that have depth $O(\log^2 n)$. Karchmer and Wigderson proved that any family of polynomial-size, monotone circuits with fan-in at most 2 that solves the undirected reachability problem has depth $\Omega(\log^2 n)$.

The intuition for the proof comes from a new characterization of the depth of circuits with fan-in at most 2 in terms of communication complexity. Consider the following game between two players. The game is given by two sets, $B_1, B_2 \subseteq \{0,1\}^n$ such that $B_1 \cap B_2 = \emptyset$. The first player gets $x \in B_1$ and the second player gets $y \in B_2$. The goal of the game is that the two players should agree on a coordinate i such that $x_i \neq y_i$. Let $C(B_1, B_2)$ denote the minimum number of bits that the two players must communicate to agree on such a coordinate. (For example, the first player could tell x to the second player, and then the second player can find an appropriate coordinate to tell to the first player.) Karchmer and Wigderson proved that the minimum depth in which a Boolean function f can be computed with a polynomial-size circuit is equal to $C(f^{-1}(0), f^{-1}(1))$.

A similar theorem holds for the monotone circuit complexity of monotone Boolean functions. In the case of the undirected reachability problem, the corresponding game is the following: the first player has an $[s,t]$-path and the second player has an $[s,t]$-cut, and the goal of the game is to find an edge in the intersection of the path and the cut. Consider the following protocol for this connectivity game: the first player sends the name of the midpoint on the path, and the second player responds by telling which side of the cut this node lies on. Note that the protocol requires $O(\log n)$ rounds, and in each round the first player sends $\log n$ bits and the second player sends 1. Karchmer and Wigderson prove that even if the second player were allowed to send $O(n^\epsilon)$ bits in each round (instead of just 1), the players would still need this many rounds. The claimed lower bound on the monotone circuit depth is proved as a consequence of this fact. They show that if a polynomial-size small depth circuit existed, then by simulating $\epsilon \log n$ layers of the circuit in one round of the protocol, one can find a protocol contradicting the previous claim.

The proof uses a random restriction technique that is a "top down" approach, as opposed to the "bottom up" approach of the parity lower bound proof of Furst, Saxe and Sipser. Assume for a contradiction that there exists a k-round protocol with $k \leq \log n/(40 \log\log n)$. The idea is to prove that there exists a large subset of all possible paths and cuts under which the two players send the same messages in the first round. Using a random restriction technique, Karchmer and Wigderson prove that after such a first round, one can give some extra information to both players to get smaller, and yet nicer, subsets of paths and cuts that allow a one-to-one correspondence with a sufficiently dense family of shorter paths and a sufficiently dense family of cuts in a graph on fewer nodes. The contradiction follows from the fact that these families have a protocol with one fewer round.

Machine-based complexity classes

In one of the first results in complexity theory, Rabin (1963) considered a machine-based notion of "real-time" computation, and gave a combinatorial argument to show that a certain language cannot be computed this efficiently. Hennie (1965) considered a related language in a more powerful Turing machine model. He generalized Rabin's proof by introducing a **crossing sequence technique** to prove lower bounds on computation time. This proof uses Turing machines that have only a single tape, that serves both as the input tape and as the work tape. The lower bound is based on the amount of information that must pass through the middle portion of the tape.

Consider the language of **palindromes**, *i.e.*, the language consisting of all words over the alphabet $\{0,1\}$ that read the same forwards as backwards. It is easy to see that palindromes can be recognized by such a one-tape Turing machine in $O(n^2)$ time by letting the machine repeatedly compare the two end letters, and then erasing them. Hennie proved that every algorithm takes at least $\Omega(n^2)$ time. To prove this, define the **crossing sequence** of a particular tape cell to be the sequence of states of the Turing machine in which that cell is read by the machine during the computation. Now consider the subset of palindromes of length $4n$ that have a block of $2n$ 0's in the middle. For each of the middle $2n$ positions, each of the 2^n palindromes must have a different crossing sequence; if there were two palindromes x and y with identical crossing sequences at a middle position, then the Turing machine must also accept the non-palindrome with its first n characters like x and its last n characters like y. Since this implies that the crossing sequences for the middle positions are long, we get the $\Omega(n^2)$ time bound.

Next we discuss a result of Hopcroft, Paul and Valiant (1977) relating the space and time complexity of a problem. Clearly, a computation that takes time $T(n)$ cannot take more then space $T(n)$. The question is whether one could save on this space requirement, possibly by using more time. Hopcroft, Paul and Valiant have shown that one can, and in fact, that $DTIME(T(n)) \subseteq DSPACE(T(n)/\log T(n))$.

The proof of Hopcroft, Paul and Valiant is based on properties of a **pebbling game** that abstracts the combinatorial essence of the problem. The input to the pebbling game is a directed acyclic graph, and the aim is to place a pebble on each node, by following these simple rules: a pebble may be placed on a node only if all of its predecessors have a pebble on them, and a pebble may be removed from any node at any time. The pebbling problem is to determine the minimum number of pebbles required. For example, the graph that consists of a single directed path can be pebbled with two pebbles. The pebbling problem can be considered as an abstract way to study computation, especially, in terms of space requirements. If one views the nodes of the graph as partial results of a computation and the edges as dependencies, an algorithm to pebble this computation graph corresponds to an algorithm that computes the result, where the pebbles indicate the partial results retained in memory. The space used by the algorithm is proportional to the number of pebbles used. This algorithm might make inefficient use of time, since it might have to recompute the same partial result several times.

Given a Turing machine that runs in time $T(n)$ and an input x, we will construct a corresponding graph $G(x)$ whose $T(n)$ nodes correspond to the steps of the Turing machine. Two nodes, t_1 and t_2 are connected by an arc if the configuration of the Turing machine at time t_1 directly influences its configuration at time t_2; that is, if either $t_2 = t_1 + 1$, or there exists a tape where the same cell

is read at times t_1 and t_2, and is not read at any time in between. If the Turing machine has k tapes, then every node of this graph has indegree at most $k+1$.

Let $f_k(n)$ denote the maximum number of pebbles needed to pebble a directed acyclic graph of order n, where each node has indegree at most k. Hopcroft, Paul and Valiant have shown that $f_k(n) \leq c_k n/\log n$.

Intuitively, one would like to apply the bound on $f_k(n)$ to the graph $G(x)$ to obtain the theorem of Hopcroft, Paul and Valiant. The problem is that in order to use the pebbling strategy whose existence is claimed by this bound, one would have to know both the graph $G(x)$ and the pebbling strategy. Hopcroft, Paul and Valiant overcome this difficulty with a sophisticated use of nondeterminism and Savitch's theorem. An additional idea that plays an essential role in this proof is the theoretical application of a programming idea known as the principle of locality: a program tends to access nearby memory cells so it is more efficient to divide the memory into blocks, and only keep in "main memory" some of the most recently accessed blocks. This idea is used by applying the bound on $f_k(n)$ to a graph similar $G(x)$ whose nodes correspond to time periods (rather than a single step) and whose arcs correspond to blocks of memory (rather than a single cell). By reducing the problem to a smaller computation graph, the entire graph can be guessed within the necessary bounds.

Throughout this survey, we have repeatedly raised questions related to the power of nondeterminism. Informally, one can think of the $\mathcal{NP}$ versus $\mathcal{P}$ question as asking whether nondeterminism can help speed up computation. In a celebrated result, Paul, Pippenger, Szemerédi and Trotter (1983) showed that it can. More precisely, they showed that there are problems solvable in $NTIME(n)$ that cannot be solved in $DTIME(n)$. However, the lower bound they establish is only very slightly more then linear; it is $O(n\sqrt[4]{\log^*(n)})$, where $\log^*(n)$ is the number of times one must repeatedly take the logarithm so that $\log(\log ...(\log(n))) \leq 1$. It is also important to note that padding techniques can be used to extend this result to show that for any superlinear time bound $T(n)$, nondeterminism provides additional power.

To prove this theorem, we show that any language accepted by a deterministic Turing machine in time $O(T(n))$, where $\lim T(n)/n \to \infty$, can be accepted by a nondeterministic Turing machine in slightly less time. Given this, the nondeterministic time hierarchy theorem (Theorem (3.8)) shows that nondeterministic linear time strictly contains its deterministic analogue. The framework for the proof, suggested by Paul and Reischuk (1980), consists of constructing a sublinear-time alternating Turing machine with four levels of alternation and then simulating this computation by a nondeterministic Turing machine.

The basis for the alternating Turing machine is a result concerning the structure of directed acyclic graphs. Consider again the computation graph $G(x)$; we wish to use limited alternation to perform the computation faster. The intuition behind the combinatorial core of the proof is that we would like to find a small set of nodes S that captures the key partial results in the computation, in the sense that the value of any node in the graph can be recomputed using the values for S in addition to computing $o(n)$ other partial results. If there is such a set S, we can guess the local configuration of the Turing machine at those points, and then simultaneously for each node in S, verify this guess by recomputing its value, using the guesses for the other nodes in S.

Motivated by this discussion, let a node set S be a **segregator** if $|S| = o(n)$ and for each node $v \notin S$, there are $o(n)$ nodes from which v is reachable when S is deleted. Paul and Reischuk asked

whether every constant-degree directed acyclic graph has a segregator. Unfortunately, Schnitger (1982) has shown that there are such graphs that do not have segregators. However, Pippenger observed that it might be possible to exploit the more refined structure of computation graph $G(x)$. A graph is called a **k-page graph** if there exists an ordering of its nodes so that if the nodes are embedded along the spine of a book in that order, the edges can be partitioned so that the edges in each part can be embedded in a single page of the book without crossing. (More formally, this can be viewed as decomposing the graph into the union of k outerplanar graphs, where the nodes appear on the outer face in the same order in each of the graphs.)

It is easy to see that the graph $G(x)$ is a $2k$-page graph, where k is the number of the tapes used by the Turing machine. (The ordering of the nodes is the one given by time. The edges that correspond to dependencies caused by one tape can be drawn on two pages, where one page is used for edges corresponding to moments when the head returns to the same cell from the left, and the other page is used when the head returns from the right.) At the heart of the proof of this theorem is the result of Szemerédi, showing that any $2k$-page graph has a segregator. As in the proof of Hopcroft, Paul and Valiant, we are still limited by the fact that neither $G(x)$ nor the segregator are known, but arguments along the same lines as were used there are sufficient to complete this proof.

7 Acknowledgments

In writing this survey, we have often been aided by the expertise of others. In particular, we would like to thank Laci Babai and Michael Sipser for sharing with us many of their insights into complexity theory. The surveys that we have cited in particular areas of complexity theory were extremely useful and we wish express our gratitude to their authors. In addition, we would like to thank Shafi Goldwasser, Leslie Hall, David Johnson, Jan Karel Lenstra, Laci Lovász, Albert Meyer, Carolyn Haibt Norton and Larry Stockmeyer for their many insightful comments on an earlier draft. We are also grateful to Helena Lourenco for her help in preparing the bibliography. The research of the first author was supported in part by an NSF Presidential Young Investigator award CCR-89-96272 with matching support from IBM, UPS and Sun and by Air Force Contract AFOSR-86-0078. The research of the second author was supported in part by Air Force Contract AFOSR-86-0078.

References

ADLEMAN, L.

[1978] Two theorems on random polynomial time, in: *Proceedings of the 19th Annual IEEE Symposium on Foundations of Computer Science* (IEEE Computer Society Press, Washington, D.C.) pp. 75-83.

ADLEMAN, L.M. and M.-D.A. HUANG

[1987] Recognizing primes in random polynomial time, in: *Proceedings of the 19th Annual ACM Symposium on Theory of Computing* (ACM Press, New York, NY) pp. 462-469.

ADLEMAN, L.M. and K. MANDERS

[1977] Reducibility, randomness and intractability, in: *Proceedings of the 9th Annual ACM Symposium on Theory of Computing* (ACM Press, New York, NY) pp. 151-163.

[1979] Reductions that lie, in: *Proceedings of the 20th Annual IEEE Symposium on Foundations of Computer Science* (IEEE Computer Society Press, Washington, D.C.) pp. 397-410.

AHO, A.V., J.E. HOPCROFT and J.D. ULLMAN

[1974] *The design and analysis of computer algorithms* (Addison-Wesley, Reading, Massachusetts).

AIELLO, W., S. GOLDWASSER and J. HASTAD

[1986] On the power of interaction, in: *Proceedings of the 27th Annual IEEE Symposium on Foundations of Computer Science* (IEEE Computer Society Press, Washington, D.C.) pp. 368-379.

AJTAI, M.

[1983] Σ_1^1-formulae on finite structures, *Ann. Pure Appl. Logic*, **24**, 1-48.

AJTAI, M., J. KOMLÓS and E. SZEMERÉDI

[1983] Sorting in $c \log n$ parallel steps, *Combinatorica*, **3**, 1-19.

AGGARWAL, A. and R.J. ANDERSON

[1988] A random $\mathcal{NC}$ algorithm for depth first search, *Combinatorica*, **8**, 1-12.

AGGARWAL, A., R.J. ANDERSON and M.-Y. KAO

[1989] Parallel depth-first search in general directed graphs, in: *Proceedings of the 21st Annual ACM Symposium on Theory of Computing* (ACM Press, New York, NY) pp. 297-308.

ALELUNIAS, R., R.M. KARP, R.J. LIPTON, L. LOVÁSZ and C. RACKOFF

[1979] Random walks, universal traversal sequences, and complexity of maze problems, in: *Proceedings of the 20th Annual IEEE Symposium on Foundations of Computer Science* (IEEE Computer Society Press, Washington, D.C.) pp. 218-223.

ALON, N., AND L. BABAI and A. ITAI

[1986] A fast and simple randomized parallel algorithm for the maximal independent set problem, *J. Algorithms*, **7**, 567-583.

ALON, N. and R.B. BOPPANA

[1987] The monotone circuit complexity of Boolean functions, *Combinatorica*, **7**, 1-22.

ANDREEV, A.E.

[1985] On a method for obtaining lower bounds for the complexity of individual monotone functions (in Russian), *Dokl. Akad. Nauk SSSR*, **282**, 1033-1037. Translation. *Soviet Math. Dokl.*, **31**, 530-534.

BABAI, L.
[1985] Trading group theory for randomness, in: *Proceedings of the 17th Annual ACM Symposium on Theory of Computing* (ACM Press, New York, NY) pp. 421-429.
BABAI, L., E.M. LUKS and A. SERESS
[1988] Fast management of permutation groups, in: *Proceedings of the 29th Annual IEEE Symposium on Foundations of Computer Science* (IEEE Computer Society Press, Washington, D.C.) pp. 272-282.
BABAI, L. and S. MORAN
[1988] Arthur-Merlin games: a randomized proof system, and a hierarchy of complexity classes, *J. Comput. System Sci.*, **36**, 254-276.
BACH, E., G. MILLER and J. SHALLIT
[1986] Sums of divisors, perfect numbers and factoring, *SIAM J. Comput.*, **15**, 1143-1154.
BAKER, T., J. GILL and R. SOLOVAY
[1975] Relativizations of the $\mathcal{P} =? \mathcal{NP}$ question, *SIAM J. Comput.*, **4**, 431-442.
BAKER, T.P. and A.L. SELMAN
[1979] A second step toward the polynomial hierarchy, *Theoret. Comput. Sci.*, **8**, 177-187.
BATCHER, K.E.
[1968] Sorting networks and their applications, in: *Proceedings of the AFIPS Spring Joint Computer Conference*, **32** (Thompson Book Company, Washington, D.C.), pp. 307-314.
BEAME, P.W., S.A. COOK and H.J. HOOVER
[1986] Log depth circuits for division and related problems, *SIAM J. Comput.*, **15**, 994-1003.
BEARDWOOD, J., J.H. HALTON and J.M. HAMMERSLEY
[1959] The shortest path through many points, *Proc. Cambridge Philos. Soc.*, **55**, 299-327.
BELLARE, M. and S. MICALI
[1988] How to sign given any trapdoor function, in: *Proceedings of the 20th Annual ACM Symposium on Theory of Computing* (ACM Press, New York, NY) pp. 32-42.
BENNETT, C.H. and J. GILL
[1981] Relative to a random oracle A, $P^A \neq NP^A \neq$ co-NP^A with probability 1, *SIAM J. Comput.*, **10**, 96-113.
BERKOWITZ, S.J.
[1984] On computing the determinant in small parallel time using a small number of processors, *Inform. Process. Lett.*, **18**, 147-150.
BLUM, M. AND S. MICALI
[1984] How to generate cryptographically strong sequences of pseudo-random bits, *SIAM J. Comput.*, **13**, 850-864.
BLUM, N.
[1984] A Boolean function requiring $3n$ network size, *Theoret. Comput. Sci.*, **28**, 337-345.
BOLLOBÁS, B.
[1988] The chromatic number of random graphs, *Combinatorica*, **8**, 49-55.
BOLLOBÁS, B., T.I. FENNER and A.M. FRIEZE
[1987] An algorithm for finding Hamilton cycles in a random graph, *Combinatorica*, **7**, 327-341.
BOPPANA, R.B., J. HASTAD and S. ZACHOS
[1987] Does co-NP have short interactive proofs?, *Inform. Process. Lett.*, **25**, 127-132.
BOPPANA, R.B. and M. SIPSER
[1990] The complexity of finite functions, in: *The Handbook of Theoretical Computer Science*, edited by J. van Leeuwen et al. (North-Holland, Amsterdam) to appear.

BORGWARDT, K.-H.
[1982] Some distribution-independent results about the asymptotic order of the average number of pivot steps of the simplex method, *Math. Oper. Res.*, **7**, 441-462.
BORODIN, A., J. VON ZUR GATHEN and J. HOPCROFT
[1982] Fast parallel matrix and GCD computations, *Inform. and Control*, **52**, 241-256.
BRASSARD, G. and C. CRÉPEAU
[1986] Non-transitive transfer of confidence: a *perfect* zero-knowledge interactive protocol for SAT and beyond, in: *Proceedings of the 27th Annual IEEE Symposium on Foundations of Computer Science* (IEEE Computer Society Press, Washington, D.C.) pp. 188-195.
BRENT, R.P.
[1974] The parallel evaluation of general arithmetic expressions, *J. Assoc. Comput. Mach.*, **21**, 201-208.
BRUNO, J.L. and P.J. DOWNEY
[1986] Probabilistic bounds on the performance of list scheduling, *SIAM J. Comput.*, **15**, 409-417.
CAI, J.-Y.
[1986] With probability one, a random oracle separates $PSPACE$ from the polynomial-time hierarchy, *J. Comput. System Sci.*, **38**, 68-85.
CHANDRA, A.K., D.C. KOZEN and L.J. STOCKMEYER
[1981] Alternation, *J. Assoc. Comput. Mach.*, **28**, 114-133.
CHRISTOFIDES, N.
[1976] *Worst-Case Analysis of a New Heuristic for the Travelling Salesman Problem*, Report 388 (Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh, PA).
COBHAM, A.
[1965] The intrinsic computational difficulty of functions, in: *Proceedings of the 1964 International Congress for Logic, Methodology and Philosophy of Science*, edited by Y. Bar-Hillel (North-Holland, Amsterdam) pp. 24-30.
COFFMAN, JR., E.G., G.S. LUEKER and A.H.G. RINNOOY KAN
[1988] Asymptotic methods in the probabilistic analysis of sequencing and packing heuristics, *Management Sci.*, **34**, 266-290.
COLE, R.
[1988] Parallel merge sort, *SIAM J. Comput.*, **17**, 770-785.
COOK, S.A.
[1971] The complexity of theorem-proving procedures, in: *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing* (ACM Press, New York, NY) pp. 151-158.
[1973] An observation on time-storage tradeoff, in: *Proceedings of the 5th Annual ACM Symposium on Theory of Computing* (ACM Press, New York, NY) pp. 29-33.
COOK, S., C. DWORK and R. REISCHUK
[1986] Upper and lower time bounds for parallel random access machines without simultaneous writes, *SIAM J. Comput.*, **15**, 87-97.
CSANKY, L.
[1976] Fast parallel matrix inversion algorithms, *SIAM J. Comput.*, **5**, 618-623.
DAVIS, M.
[1977] Unsolvable problems, in: *Handbook of Mathematical Logic*, edited by J. Barwise (North-Holland, Amsterdam) pp. 567-594.
DIFFIE, W. and M.E. HELLMAN
[1976] New directions in cryptography, *IEEE Trans. Inform. Theory*, **22**, 644-654.

DOBKIN, D., R.J. LIPTON and S. REISS
[1979] Linear programming is log-space hard for P, *Inform. Process. Lett.*, **8**, 96-97.
EDMONDS, J.
[1965] Paths, trees, and flowers, *Canad. J. Math.*, **17**, 449-467.
ERDÖS, P. and A. RÉNYI
[1960] On the evolution of random graphs, *Magyar Tud. Akad. Mat. Kut. Int. Kozl.*, **5**, 17-61.
FERNANDEZ DE LA VEGA, W. and G. S. LUEKER
[1981] Bin packing can be solved within $1+\epsilon$ in linear time, *Combinatorica*, **1**, 349-355.
FISCHER, M.J. and M.O. RABIN
[1974] Super-exponential complexity of Presburger arithmetic, in: *Complexity of Computation*, SIAM-AMS Proceedings, Vol. 7, edited by R. M. Karp (American Mathematical Society, Providence, Rhode Island) pp. 27-41.
FRIEZE, A.M.
[1986] On the Lagarias-Odlyzko algorithm for the subset sum problem, *SIAM J. Comput.*, **15**, 536-539.
FURST, M., J. HOPCROFT and E. LUKS
[1980] Polynomial time algorithms for permutation groups, in: *Proceedings of the 21st Annual IEEE Symposium on Foundations of Computer Science* (IEEE Computer Society Press, Washington, D.C.) pp. 36-41.
FURST, M., J.B. SAXE and M. SIPSER
[1981] Parity, circuits, and the polynomial-time hierarchy, *Math. Systems Theory*, **17**, 13-27.
GAREY, M.R. and D.S. JOHNSON
[1976] The complexity of near-optimal graph coloring, *J. Assoc. Comput. Mach.*, **23**, 43-49.
[1978] "Strong" NP-completeness results: motivation, examples, and implications, *J. Assoc. Comput. Mach.*, **25**, 499-508.
[1979] *Computers and Intractability: A Guide to the Theory of NP-Completeness* (W. H. Freeman and Co., New York).
GAZIT, H.
[1986] An optimal randomized parallel algorithm for finding connected components in a graph, in: *Proceedings of the 27th Annual IEEE Symposium on Foundations of Computer Science* (IEEE Computer Society Press, Washington, D.C.) pp. 492-501.
GILL, J.
[1977] Computational complexity of probabilistic Turing machines, *SIAM J. Comput.*, **6**, 675-695.
GÖDEL, K.
[1931] Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme, I., *Monatsh. Math. Phys.*, **38**, 173-198.
GOLDREICH, O., Y. MANSOUR and M. SIPSER
[1987] Interactive proof systems: provers that never fail and random selection, in: *Proceedings of the 28th Annual IEEE Symposium on Foundations of Computer Science* (IEEE Computer Society Press, Washington, D.C.) pp. 449-461.

GOLDREICH, O., S. MICALI and A. WIGDERSON
[1986] Proofs that yield nothing but their validity and a methodology of cryptographic protocol design, in: *Proceedings of the 27th Annual IEEE Symposium on Foundations of Computer Science* (IEEE Computer Society Press, Washington, D.C.) pp. 174-187.
[1987] How to play any mental game, or a completeness theorem for protocols with honest majority, in: *Proceedings of the 19th Annual ACM Symposium on Theory of Computing* (ACM Press, New York, NY) pp. 218-229.
GOLDSCHLAGER, L.M., R.A. SHAW and J. STAPLES
[1982] The maximum flow problem is log space complete for P, *Theoret. Comput. Sci.*, **21**, 105-111.
GOLDWASSER, S.
[1989] Interactive proof systems, in: *Computational Complexity Theory*, AMS Symposia in Applied Mathematics, **38**, edited by J. Hartmanis (AMS, Providence, Rhode Island), 108-128.
GOLDWASSER, S. and J. KILIAN
[1986] Almost all primes can be quickly certified, in: *Proceedings of the 18th Annual ACM Symposium on Theory of Computing* (ACM Press, New York, NY) pp. 316-329.
GOLDWASSER, S. AND S. MICALI
[1984] Probabilistic encryption, *J. Comput. System Sci.*, **28**, 270-299.
GOLDWASSER, S., S. MICALI and C. RACKOFF
[1989] The knowledge complexity of interactive proof systems, *SIAM J. Comp.*, **18**, pp. 186-208.
GOLDWASSER, S., S. MICALI and R. L. RIVEST
[1988] A digital signature scheme secure against adaptive chosen-message attacks, *SIAM J. Comput.*, **17**, 281-308.
GOLDWASSER, S. and M. SIPSER
[1986] Private coins versus public coins in interactive proof systems, in: *Proceedings of the 18th Annual ACM Symposium on Theory of Computing* (ACM Press, New York, NY) pp. 59-68.
GRAHAM, R.L.
[1966] Bounds for certain multiprocessing anomalies, *Bell System Tech. J.*, **45**, 1563-1581.
GRIMMETT, G.R. and C.J.H. MCDIARMID
[1975] On colouring random graphs, *Math. Proc. Cambridge Philos. Soc.*, **77**, 313-324.
GRÖTSCHEL, M., L. LOVÁSZ and A. SCHRIJVER
[1981] The ellipsoid method and its consequences in combinatorial optimization, *Combinatorica*, **1**, 169-197.
HARTMANIS, J., P.M. LEWIS II and R.E. STEARNS
[1965] Hierarchies of memory limited computations, in: *Proceedings of the 6th Annual IEEE Symposium on Switching Circuit Theory and Logical Design*, (IEEE, Inc., New York, NY) pp. 179-190.
HARTMANIS, J. and R.E. STEARNS
[1965] On the computational complexity of algorithms, *Trans. Amer. Math. Soc.*, **117**, 285-306.
HASTAD, J.
[1986] Almost optimal lower bounds for small depth circuits, in: *Proceedings of the 18th Annual ACM Symposium on Theory of Computing* (ACM Press, New York, NY) pp. 6-20.

HENNIE, F.C.
[1965] One-tape, off-line Turing machine computations, *Inform. and Control*, **8**, 553-578.
HENNIE, F.C. and R.E. STEARNS
[1966] Two-tape simulation of multitape Turing machines, *J. Assoc. Comput. Mach.*, **13**, 533-546.
HIRSCHBERG, D.S., A.K. CHANDRA and D.V. SARWATE
[1979] Computing connected components on parallel computers, *Comm. ACM.*, **22**, 461-464.
HOCHBAUM, D.S. and D.B. SHMOYS
[1985] A best possible heuristic for the k-center problem, *Math. Oper. Res.*, **10**, 180-184.
[1987] Using dual approximation algorithms for scheduling problems: theoretical and practical results, *J. Assoc. Comput. Mach.*, **34**, 144-162.
HOLYER, I.
[1981] The NP-completeness of edge-coloring, *SIAM J. Comput.*, **10**, 718-720.
HOOVER, H.J., M.M. KLAWE and N.J. PIPPENGER
[1984] Bounding fan-out in logical networks, *J. Assoc. Comput. Mach.*, **31**, 13-18.
HOPCROFT, J., W. PAUL and L. VALIANT
[1977] On time versus space, *J. Assoc. Comput. Mach.*, **24**, 332-337.
IBARRA, O.H. and C.E. KIM
[1976] Fast approximation algorithms for the knapsack and sum of subset problems, *J. Assoc. Comput. Mach.*, **22**, 463-468.
IMMERMAN, N.
[1988] Nondeterministic space is closed under complementation, *SIAM J. Comp.*, **17**, 935-938.
JA' JA', J. and J. SIMON
[1982] Parallel algorithms in graph theory: planarity testing, *SIAM J. Comput.*, **11**, 314-328.
JERRUM, M.R., L.G. VALIANT and V.V. VAZIRANI
[1986] Random generation of combinatorial structures from a uniform distribution, *Theoret. Comput. Sci.*, **43**, 169-188.
JOHNSON, D.S.
[1973] *Near-Optimal Bin Packing Algorithms*, Doctoral thesis (Massachusetts Institute of Technology, Cambridge, MA).
[1974a] Approximation algorithms for combinatorial problems, *J. Comput. System Sci.*, **9**, 256-278.
[1974b] Worst case behavior of graph coloring algorithms, in: *Proceedings of the 5th Southeastern Conference on Combinatorics, Graph Theory, and Computing* (Utilitas Mathematica Publishing, Winnipeg) , edited by F. Hoffman, R. C. Mullin, R.B. Levon, D. Roselle, R.G. Stanton and R. S. D. Thomas, pp. 513-527.
[1984] The NP-Completeness Column: An Ongoing Guide, *J. Algorithms*, **5**, 284-299.
[1988] The NP-Completeness Column: An Ongoing Guide, *J. Algorithms*, **9**, 426-444.
KAHN, J., M. SAKS and D. STURTEVANT
[1984] A topological approach to evasiveness, *Combinatorica*, **4**, 297-306.
KARCHMER, M. and A. WIGDERSON
[1988] Monotone circuits for connectivity require super-logarithmic depth, in: *Proceedings of the 20th Annual ACM Symposium on Theory of Computing* (ACM Press, New York, NY) pp. 539-550.
KARLOFF, H.J.
[1986] A Las Vegas RNC algorithm for maximum matching, *Combinatorica*, **6**, 387-391.

KARMARKAR, N.
[1984] A new polynomial-time algorithm for linear programming, *Combinatorica*, **4**, 373-395.
KARMARKAR, N. and R.M. KARP
[1982] An efficient approximation scheme for the one-dimensional bin-packing problem, in: *Proceedings of the 23rd Annual IEEE Symposium on Foundations of Computer Science* (IEEE Computer Society Press, Washington, D.C) pp. 312-320.
KARP, R.M.
[1972] Reducibility among combinatorial problems, in: *Complexity of Computer Computations*, edited by R. E. Miller and J. W. Thatcher (Plenum Press, New York) pp. 85-103.
[1976] The probabilistic analysis of some combinatorial search algorithms, in: *Algorithms and Complexity: New Directions and Recent Results*, edited by J. F. Traub (Academic Press, New York) pp. 1-19.
KARP, R. M., J.K. LENSTRA, C.J.H. MCDIARMID and A.H.G. RINNOOY KAN
[1985] Probabilistic analysis, in: *Combinatorial Optimization: Annotated Bibliographies*, edited by M. O'hEigeartaigh, J. K. Lenstra and A. H. G. Rinnooy Kan (Centre for Mathematics and Computer Science, Amsterdam and John Wiley & Sons, Chichester) pp. 52-88.
KARP, R.M. and R. LIPTON
[1982] Turing machines that take advice, *Enseign. Math. II*, **28**, 191-209.
KARP, R.M. and M. LUBY
[1985] Monte-Carlo algorithms for the planar multiterminal network reliability problem, *J. Complexity*, **1**, 45-64.
KARP, R.M. and V. RAMACHANDRAN
[1990] A survey of parallel algorithms for shared-memory machines, in: *The Handbook of Theoretical Computer Science*, edited by J. van Leeuwen, A. R. Meyer, M. Nivat, M. S. Paterson and D. Perrin (North-Holland, Amsterdam) to appear.
KARP, R.M. and J.M. STEELE
[1985] Probabilistic analysis of heuristics, in: *The Traveling Salesman Problem*, edited by E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan and D. B. Shmoys (John Wiley & Sons, Chichester) pp. 181-205.
KARP, R.M., E. UPFAL and A. WIGDERSON
[1986] Constructing a maximum matching is in random NC, *Combinatorica*, **6**, 35-48.
KARP, R.M. and A. WIGDERSON
[1985] A fast parallel algorithm for the maximal independent set problem, *J. Assoc. Comput. Mach.*, **32**, 762-773.
KHACHYIAN, L G.
[1979] A polynomial algorithm in linear programming (in Russian), *Dokl. Akad. Nauk SSSR*, **244**, 1093-1096. Translation. *Soviet Math. Dokl.*, **20**, 191-194.
KLEE, V. and G.J. MINTY
[1972] How good is the simplex algorithm?, in: *Inequalities, III*, edited by O. Shisha (Academic Press, New York, NY) pp. 159-174.
KLEIN, P.N. and J.H. REIF
[1988] An efficient algorithm for planarity, *J. Comput. System Sci.*, **37**, 190-246.
KOMLÓS, J. and E. SZEMERÉDI
[1983] Limit distribution for the existence of Hamiltonian cycles in random graphs, *Discrete Math.*, **43**, 55-63.

KORŠUNOV, A.D.
[1976] Solution of a problem of Erdös and Rényi on Hamiltonian cycles in nonoriented graphs, *Dokl. Akad. Nauk SSSR*, **228**, 529-532. Translation. *Soviet Math. Dokl.*, **17**, 760-764.

LADNER, R.E.
[1975a] On the structure of polynomial time reducibility, *J. Assoc. Comput. Mach.*, **22**, 155-171.
[1975b] The circuit value problem is log-space complete for P, *SIGACT News*, **7**:1, 18-20.

LAGARIAS, J.C. and A.M. ODLYZKO
[1983] Solving low-density subset sum problems, in: *Proceedings of the 24th Annual IEEE Symposium on Foundations of Computer Science* (IEEE Computer Society Press, Washington, D. C.) pp. 1-10.

LAUTEMANN, C.
[1983] BPP and the polynomial hierarchy, *Inform. Process. Lett.*, **17**, 215-217.

VAN LEEUWEN, J., A.R. MEYER, M. NIVAT, M.S. PATERSON AND D. PERRIN, editors
[1990] *Handbook of Theoretical Computer Science* (North-Holland, Amsterdam), to appear.

LENSTRA, A.K. and H.W. LENSTRA, JR.
[1990] Algorithms in number theory, in: *The Handbook of Theoretical Computer Science*, edited by J. van Leeuwen, A.R. Meyer, M. Nivat, M.S. Paterson and D. Perrin (North-Holland, Amsterdam) to appear.

LEVIN, L.A.
[1973] Universal Sorting Problems, *Problemy Peredachi Informatsii*, **3**, 265-266.
[1986] Average case complete problems, *SIAM J. Comput.*, **15**, 285-286.

LOVÁSZ, L.
[1979] On determinants, matchings, and random algorithms, in: *Fundamentals in Computation Theory, FCT '79, (Proc. Conf. Algebraic, Arithmetic, and Categorical Methods in Computation Theory, Berlin, Wendisch-Rietz 1979)* edited by L. Budach (Akademie-Verlag, Berlin), pp. 565-574.
[1985] Computing ears and branchings in parallel, in: *Proceedings of the 26th Annual IEEE Symposium on Foundations of Computer Science* (IEEE Computer Society Press, Washington, D. C.) pp. 464-467.

LOVÁSZ, L. and M. PLUMMER
[1986] *Matching Theory* (Akademiai Kiado, Budapest and North-Holland, Amsterdam).

LUBY, M.
[1986] A simple parallel algorithm for the maximal independent set problem, *SIAM J. Comput.*, **15**, 1036-1053.

LUKS, E.M.
[1982] Isomorphism of graphs of bounded valence can be tested in polynomial time, *J. Comput. System Sci.*, **25**, 42-65.

MAGGS, B.M. and S A. PLOTKIN
[1988] Minimum-cost spanning tree as a path-finding problem, *Inform. Process. Lett.*, **26**, 291-293.

MAON, Y., B. SCHIEBER and U. VISHKIN
[1986] Parallel ear decomposition search (EDS) and *st*-numbering in graphs, in: *Theoret. Comput. Sci.*, **47**, 277-298.

MARGULIS, G.A.
[1973] Explicit constructions of concentrator graphs (in Russian), *Problemy Peredachi Informatsii*, **9**:4, 71-80.
MATIJASEVIČ, YU. V.
[1970] Enumerable sets are diophantine (in Russian), *Dokl. Akad. Nauk SSSR*, **191**, 279-282. Translation. *Soviet Math. Dokl.*, **11**, 354-358.
MERKLE, R. and M. HELLMAN
[1978] Hiding information and signatures in trapdoor knapsacks, *IEEE Trans. Inform. Theory*, **24**, 525-530.
MEYER, A.R. and L.J. STOCKMEYER
[1972] The equivalence problem for regular expressions with squaring requires exponential time, in: *Proceedings of the 13th Annual IEEE Symposium on Switching and Automata Theory* (IEEE Computer Society Press, Washington, D.C.) pp. 125-129.
MILLER, G.L.
[1976] Riemann's hypothesis and tests for primality, *J. Comput. System Sci.*, **13**, 300-317.
MILLER, G.L. and V. RAMACHANDRAN
[1987] A new graph triconnecitivity algorithm and its paralellization, in: *Proceedings of the 19th Annual ACM Symposium on Theory of Computing* (ACM Press, New York, NY) pp. 335-344.
MILLER, G.L. and J.H. REIF
[1985] Parallel tree contraction and its applications, in: *Proceedings of the 26th Annual IEEE Symposium on Foundations of Computer Science* (IEEE Computer Society Press, Washington, D.C) pp. 478-489.
MULMULEY, K.
[1987] A fast parallel algorithm to compute the rank of a matrix over an arbitrary field, *Combinatorica*, **7**, 101-104.
MULMULEY, K., U.V. VAZIRANI and V.V. VAZIRANI
[1987] Matching is as easy as matrix inversion, *Combinatorica*, **7**, 105-113.
NISAN, N. and A. WIGDERSON
[1988] Hardness vs. randomness, in: *Proceedings of the 29th Annual IEEE Symposium on Foundations of Computer Science* (IEEE Computer Society Press, Washington, D.C.) pp. 2-11.
PAPADIMITRIOU, C.H.
[1983] Games against nature, in: *Proceedings of the 24th Annual IEEE Symposium on Foundations of Computer Science* (IEEE Computer Society Press, Washington, D.C.) pp. 446-450.
PAUL, W.J., N. PIPPENGER, E. SZEMERÉDI and W.T. TROTTER
[1983] On determinism versus nondeterminism and related problems, in: *Proceedings of the 24th IEEE Symposium on Foundations of Computer Science* (IEEE Computer Society Press, Washington, D.C.) pp. 429-438.
PAUL, W. and R. REISCHUK
[1980] On Alternation II, *Acta Inform.*, **14**, 391-403.
PEREPELIČA, V.A.
[1970] On two problems from the theory of graphs (in Russian), *Dokl. Akad. Nauk SSSR*, **194**, 1269-1272. Translation. *Soviet Math. Dokl.*, **11**, 1376-1379.

PÓSA, L.
[1976] Hamiltonian circuits in random graphs, *Discrete Math.*, **14**, 359-364.
PRATT, V.R.
[1975a] Every prime has a succinct certificate, *SIAM J. Comput.*, **4**, 214-220.
[1975b] The power of negative thinking in multiplying Boolean matrices, *SIAM J. Comput.*, **4**, 326-330.
PRESBURGER, M.
[1929] Über die Vollständigkeit eines gewissen Systems der Arithmetik ganzer Zahien, in weichen die addition ais einzige Operation hervortritt, *Comptes Rendus. I Congres des Math. des Pays Slaves*, 92-101.
RABIN, M.O.
[1963] Real time computation, *Israel J. Math.*, **1**, 203-211.
[1976] Probabilistic algorithms, in: *Algorithms and Complexity: New Directions and Recent Results*, edited by J. F. Traub (Academic Press, New York) pp. 21-39.
[1979] *Digitalized Signatures as Intractable as Factorization*, TR-212 (Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, MA).
RAZBOROV, A.A.
[1985a] Lower bounds for the monotone complexity of some Boolean functions (in Russian), *Dokl. Akad. Nauk SSSR*, **281**, 798-801. Translation. *Soviet Math. Dokl.*, **31**, 354-357.
[1985b] Lower bounds on monotone complexity of the logical permanent (in Russian), *Mathematischi Zametki*, **37**, 887-900. Translation. *Mathematical Notes of the Academy of Sciences of the USSR*, **37**, 485-493.
[1987] Lower bounds on the size of bounded depth circuits over a complete basis with logical addition (in Russian), *Matematischi Zametki*, **41**, 598-607. Translation. *Mathematical Notes of the Academy of Sciences of the USSR*, **41**, 333-338.
REIF, J.H.
[1985] Depth-first search is inherently sequential, *Inform. Process. Lett.*, **20**, 229-234.
[1986] Logarithmic depth circuits for algebraic functions, *SIAM J. Comput.*, **15**, 231-242.
RENEGAR, J.
[1988] A polynomial-time algorithm, based on Newton's method, for linear programming, *Math. Programming*, **40**, 59-93.
RIVEST, R.L.
[1990] Cryptography, in: *The Handbook of Theoretical Computer Science*, edited by J. van Leeuwen, A.R. Meyer, M. Nivat, M.S. Paterson and D. Perrin (North-Holland, Amsterdam) to appear.
RIVEST, R.L., A. SHAMIR and L. ADLEMAN
[1978] A method for obtaining digital signatures and public-key cryptosystems, *Comm. ACM*, **21**, 120-126.
RIVEST, R.L. and J. VUILLEMIN
[1976] On recognizing graph properties from adjacency matrices, *Theoret. Comput. Sci.*, **3**, 371-384.
RUZZO, W.L.
[1981] On uniform circuit complexity, *J. Comput. System Sci.*, **22**, 365-383.
SANTHA, M.
[1989] Relativized Arthur-Merlin versus Merlin-Arthur games, *Inform. and Comp.*, **80**, 44-49.

SAVITCH, W.J.
[1970] Relationships between nondeterministic and deterministic tape complexities, *J. Comput. System Sci.*, **4**, 177-192.
SCHNITGER, G.
[1982] A family of graphs with expensive depth-reduction, *Theoret. Comput. Sci.*, **18**, 89-93.
SEIFERAS, J.I., M.J. FISCHER and A.R. MEYER
[1978] Separating nondeterministic time complexity classes, *J. Assoc. Comput. Mach.*, **25**, 146-167.
SHAMIR, A.
[1982] A polynomial time algorithm for breaking the basic Merkle-Hellman cryptosystem, in: *Proceedings of the 23rd Annual IEEE Symposium on Foundations of Computer Science* (IEEE Computer Society Press, Washington, D. C.) pp. 145-152.
SHAMIR, R.
[1987] The efficiency of the simplex method: a survey, *Management Sci.*, **33**, 301-334.
SHANNON, C.E.
[1949] The synthesis of two-terminal switching circuits, *Bell System Tech. J.*, **28**, 59-98.
SHILOACH, Y. and U. VISHKIN
[1982] An $O(\log n)$ parallel connectivity algorithm, *J. Algorithms*, **3**, 57-67.
SIPSER, M.
[1983a] Borel sets and circuit complexity, in: *Proceedings of the 15th ACM Symposium on Theory of Computing* (ACM Press, New York, NY) pp. 330-335.
[1983b] A complexity theoretic approach to randomness, in: *Proceedings of the 15th ACM Symposium on Theory of Computing* (ACM Press, New York, NY) pp. 61-69.
SMALE, S.
[1983] On the average number of steps of the simplex method of linear programming, *Math. Programming*, **27**, 241-262.
SMOLENSKY, R.
[1987] Algebraic methods in the theory of lower bounds for Boolean circuit complexity, in: *Proceedings of the 19th ACM Symposium on Theory of Computing* (ACM Press, New York, NY) pp. 77-82.
SOLOVAY, R. and V. STRASSEN
[1977] A fast Monte-Carlo test for primality, *SIAM J. Comput.*, **6**, 84-85.
STOCKMEYER, L.
[1973] Planar 3-colorability is polynomial complete, *SIGACT News*, **5**:3, 19-25.
[1985] On approximation algorithms for #P, *SIAM J. Comput.*, **14**, 849-861.
[1987] Classifying the computational complexity of problems, *J. Symbolic Logic*, **52**, 1-43.
STOCKMEYER, L. and U. VISHKIN
[1984] Simulation of parallel random access machines by circuits, *SIAM J. Comput.*, **13**, 413-422.
SZELEPCSÉNYI, R.
[1987] The method of forcing for nondeterministic automata, *Bull. European Assoc. Theoret. Comput. Sci.*, 96-100.
TARDOS, É.
[1988] The gap between monotone and non-monotone circuit complexity is exponential, *Combinatorica*, **8**, 141-142.
TARJAN, R.E. and U. VISHKIN
[1985] An efficient parallel biconnectivity algorithm, *SIAM J. Comput.*, **14**, 862-874.

TURING, A.M.
[1937] On computable numbers, with an application to the Entscheidungsproblem, *Proc. London Math. Soc. II*, **42**, 230-265.
VALIANT, L.G.
[1979] The complexity of computing the permanent, *Theoret. Comput. Sci.*, **8**, 189-201.
VENKATESAN, R. and L.A. LEVIN
[1988] Random instances of a graph coloring problem are hard, in: *Proceedings of the 20th Annual ACM Symposium on Theory of Computing* (ACM Press, New York, NY) pp. 217-222.
VIZING, V.G.
[1964] On an estimate of the chromatic class of a p-graph, *Diskret. Analiz.*, **3**, 25-30.
WIGDERSON, A.
[1983] Improving the performance guarantee for approximate graph coloring, *J. Assoc. Comput. Mach.*, **30**, 729-735.
YAO, A.C.
[1982] Theory and applications of trapdoor functions, in: *Proceedings of the 23rd Annual IEEE Symposium on Foundations of Computer Science* (IEEE Computer Society Press, Washington, D. C.) pp. 80-91.
[1985] Separating the polynomial-time hierarchy by oracles, in: *Proceedings of the 26th Annual IEEE Symposium on Foundations of Computer Science* (IEEE Computer Society Press, Washington, D.C.) pp. 1-10.
ZACHOS, S. and M. FÜRER
[1985] Probabilistic quantifiers vs. distrustful adversaries, unpublished manuscript.