Computational Complexity Theory

MICHAEL C. LOUI

Department of Electrical and Computer Engineering, and Coordinated Science Laboratory, University of Illinois at Urbana-Champaign ⟨loui@geisel.csl.uiuc.edu⟩

INTRODUCTION

Computational complexity is the study of the resources, such as time and space (memory), required to solve computational problems. By quantifying these resources, complexity theory has profoundly affected our thinking about computation. Computability theory establishes the existence of undecidable problems that cannot be solved in principle, regardless of the amount of time invested. In contrast, complexity theory establishes the existence of decidable problems that, although solvable in principle, cannot be solved in practice, because the time and space required would be larger than the age and size of the known universe [Stockmeyer and Chandra 1979].

The quest for the boundaries of the set of feasible problems, those solvable in practice, has led to one of the most important unresolved questions in computer science: Is **P** different from **NP**? Here **P** comprises the problems that can be solved feasibly in polynomial time and **NP** comprises the problems whose solutions can be verified in polynomial time.

Hundreds of fundamental problems, including many ubiquitous optimization problems of operations research, are NP-complete—they are the hardest problems in **NP**. If there is a polynomial-time algorithm for any one NP-complete problem, then there would be polynomial-time algorithms for all of them. Despite the efforts of many scientists over several decades, no polynomial-time algorithm has been found for any NP-complete problem. Although no one knows whether **P** is different from **NP**, showing that a problem is NP-complete provides strong evidence that the problem is computationally infeasible and justifies the use of heuristics for solving the problem.

MODELS OF COMPUTATION

To develop a theory of the difficulty of computational problems, we need to specify precisely what a problem is, what an algorithm is, and what a measure of difficulty is. For simplicity, complexity theorists represent problems as languages, model algorithms by Turing machines, and measure computational difficulty by the time and space required by Turing machines. To justify these choices, some theorems of complexity theory show how to translate statements about, say, the time complexity of language recognition by Turing machines into statements about problems on more realistic computational models. Thus the principles of complexity theory are not artifacts of Turing machines but intrinsic properties of computation.

A Turing machine is *deterministic* if at every point during every computation, there is at most one possible next step. A Turing machine is *nondeterministic* if the machine can have two or more possible next steps.

A *decision problem* is a computational problem whose answer is simply yes or no. A decision problem can be expressed as a membership problem for a language L that represents the problem:

This work is supported by the National Science Foundation under grant CCR-9315696.
Copyright © 1996, CRC Press.

for an input word x, does x belong to L? A Turing machine *decides* a language L if for every input word x, the machine always halts and outputs either yes or no. For a nondeterministic machine, $x \in L$ if and only if some computation leads to yes.

COMPLEXITY CLASSES

We measure the complexity of a problem in terms of the growth of the maximum required time or space, as a function of the length n of the input word that encodes a problem instance. The complexity class **P** (respectively, **NP**) consists of all languages decided by deterministic (nondeterministic) Turing machines whose running times are bounded above by a polynomial in n, such as $n^3 + 2n$.

The class **P** characterizes the computationally feasible problems robustly: **P** remains the same when defined by other models of sequential computation.

The class **NP** comprises the languages whose membership proofs can be checked in polynomial time. For example, one language in **NP** is the set of composite numbers, written in binary. On input z, a nondeterministic Turing machine for composite numbers guesses $z_1 \geq 2$ and $z_2 \geq 2$ and then deterministically computes their product to check whether $z_1 z_2 = z$. Another language in **NP** is the set of satisfiable Boolean formulas, SAT. A Boolean formula ϕ is satisfiable if there exists a truth assignment of true or false to each variable such that the resulting value of ϕ is true. For example, $x \wedge (\bar{x} \vee y)$ is satisfiable, but $x \wedge \bar{y} \wedge (\bar{x} \vee y)$ is not satisfiable. A nondeterministic Turing machine for SAT guesses a truth assignment and then deterministically evaluates ϕ for this truth assignment.

NP-COMPLETENESS

A common way to solve a new problem is to reduce it to a previously solved problem. Frequently, an instance of the new problem is formulated completely in terms of an instance of the previous problem.

A language L_0 is *NP-complete* if $L_0 \in$ **NP** and every language in **NP** can be reduced to L_0 in the following sense: for each L in **NP**, there exists a function f computable in polynomial time such that for every word x, $x \in L$ if and only if $f(x) \in L_0$. If there is a polynomial-time algorithm A_0 to decide membership in L_0, then there is a polynomial-time algorithm to decide membership in L: for an input word x, first compute $f(x)$, and then run A_0 on $f(x)$. For example, SAT is NP-complete. Consequently, if deciding SAT is easy (in polynomial time), then factoring integers is easy—a surprising connection between ostensibly unrelated problems.

The NP-completeness concept has become pervasive throughout the research literatures of numerous science and engineering disciplines, because many of their computational problems are NP-complete.

RESEARCH ISSUES AND SUMMARY

Research in complexity theory has had direct applications to many areas of computer science and mathematics. Recent results on the existence of probabilistically checkable proofs imply that obtaining approximate solutions to NP-complete problems can be as difficult as solving them exactly. Complexity classes provide new tools for studying questions in finite model theory, a branch of mathematical logic: some questions about logical expressibility are equivalent to open questions about relationships among complexity classes. Fundamental questions in complexity theory are intimately linked to practical questions about the use of cryptography for computer security, such as the existence of one-way functions and the strength of public-key cryptosystems.

Current research in complexity theory goes well beyond the ideas presented here. Researchers are characterizing the power of randomization and interaction; developing a theory of average-case complexity; studying classes of

enumeration and optimization problems; and proving lower bounds on complexity for restricted computational models.

With precisely defined models and mathematically rigorous proofs, research in complexity theory will continue to provide insights into the difficulty of solving real computational problems.

FURTHER INFORMATION

Recent textbooks on complexity theory are those by Balcázar et al. [1995, 1990], by Bovet and Crescenzi [1994], and by Papadimitriou [1994]. Garey and Johnson [1979] explain NP-completeness thoroughly, with examples of NP-completeness proofs and a collection of hundreds of NP-complete problems.

ACKNOWLEDGMENT

Eric Allender, Donna Brown, Bevan Das, Lane Hemaspaandra, Leonard Pitt, Kenneth Regan, and Martin Tompa kindly read earlier versions of my CRC Handbook chapter and suggested numerous helpful improvements.

REFERENCES

BALCÁZAR, J. L., DÍAZ, J., AND GABARRÓ, J. 1995. *Structural Complexity I.* 2nd ed. Springer-Verlag, Berlin.

BALCÁZAR, J. L., DÍAZ, J., AND GABARRÓ, J. 1990. *Structural Complexity II.* Springer-Verlag, Berlin.

BOVET, D. P. AND CRESCENZI, P. 1994. *Introduction to the Theory of Complexity.* Prentice Hall International (UK) Limited, Hertfordshire, U.K.

GAREY, M. R. AND JOHNSON, D. S. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, San Francisco.

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

STOCKMEYER, L. J. AND CHANDRA, A. K. 1979. Intrinsically difficult problems. *Sci. Am. 240,* 5, 140–159.