Complexity Classes

Simon Meyers

1. Introduction

In computational complexity theory, complexity classes are group problems based on their resource demands, like how much time or memory is needed to solve them. They are defined by the difficulty of solving problems with specific computational resources. Typically, these classes include decision problems solvable by Turing machines, and they're categorized by their time or space requirements. Decision problems are the kinds of problems that can be posed as yes–no questions, like whether or not a given number is prime. Complexity classes also consist of other problems such as function problems, counting problems, and optimization problems.

There are established hierarchies among complexity classes. For example, we know that fundamental time and space complexity classes relate to each other in this way:

$$\mathrm{L} \subseteq \mathrm{NL} \subseteq \mathrm{P} \subseteq \mathrm{NP} \subseteq \mathrm{PSPACE} \subseteq \mathrm{EXPTIME} \subseteq \mathrm{NEXPTIME} \subseteq \mathrm{EXPSPACE}.$$

These classes also exhibit various closure properties. For instance, some are closed under operations like negation, disjunction, and conjunction. The class P is

notable for being closed under all Boolean operations and for quantification over polynomially sized domains.

Complexity classes are so useful because they help computer scientists group problems based on how much time and space they require to be solved and for the solution to be verified. They are also the basis of problems in theoretical computer science such as the famous P = NP problem, which asks whether every problem whose solution can be quickly verified can also be quickly solved.

2. Types of Complexity Classes

- **P complexity/Polynomial time complexity:** P complexity is the set of all decision problems that can be solved by a deterministic Turing machine using polynomial time, meaning a polynomial amount of computation time. In other words, P is the class of computational problems that are efficiently solvable. P complexity consists of problems such as calculating the greatest common divisor, merge sort, and finding a maximum matching.

- **NP complexity/Nondeterministic polynomial time complexity:** NP complexity is the set of all decision problems for which the problem has a proof verifiable in polynomial time by a deterministic Turing machine. It is alternatively defined as the set of problems that can be solved in polynomial time by a nondeterministic Turing machine. In other words, NP complexity is the class of computational problems that are efficiently verifiable. NP complexity consists

of problems such as the Boolean Satisfiability Problem, graph coloring, and the Hamiltonian Path Problem.

- **BPP complexity/Bounded-error probabilistic polynomial time complexity:** BPP complexity is the set of all decision problems that are solvable by a probabilistic Turing machine in polynomial time with an error probability bounded by 1/3 for all instances. BPP complexity is the class of computational problems that have an algorithm which is allowed to flip coins and make random decisions, is guaranteed to run in polynomial time, and has a probability, on any given run of the algorithm, of at most 1/3 of giving the wrong answer, whether the answer is YES or NO. BPP complexity consists of problems such as polynomial identity testing.

- **PSPACE complexity/Polynomial space complexity:** PSPACE complexity is the set of all decision problems that can be solved by a Turing machine using a polynomial amount of space. PSPACE complexity consists of problems such as the quantified Boolean formula problem, and finding optimal play styles in games like solitaire and mahjong.

- **NP hard complexity**: NP hard complexity is the set of all decision problems A, such that for every problem L in NP, there exists a polynomial-time reduction from L to A. NP hard complexity consists of problems such as the halting problem.

3. The P versus NP problem

The P versus NP problem is one of the seven Millennium Prize Problems selected by the Clay Mathematics Institute, each of which carries a 1,000,000 dollar prize for the first correct solution. It is a major unsolved problem in theoretical computer science that asks whether every problem whose solution can be quickly verified can also be quickly solved. In other words, P versus NP is asking if every problem in complexity class P is also in complexity class NP, and vice versa.

An answer to the P versus NP question would determine whether problems that can be verified in polynomial time can also be solved in polynomial time, since we already know that P is contained in NP. So, if $P \neq NP$, it would mean that there are problems in NP that are harder to compute than to verify. On the other hand, if we found that $P = NP$, then we would know that all the hard problems in complexity class NP, which are easily verifiable, are also in complexity class P, meaning that they are easily solvable. These problems, similar to the ones I stated earlier when discussing the complexity class NP, consist of problems such as the subset sum problem, the vertex cover problem, and the traveling salesman problem.

In conclusion, aside from being an important problem in computational theory, a proof of the P versus NP problem either way would have profound implications for mathematics, cryptography, algorithm research, artificial intelligence, game theory, multimedia processing, philosophy, economics and many other fields.

References

[1] Rubinstein-Salzedo, Simon, *Cryptography*, Springer, 2018.

[2] Fortnow, Lance, *The Status of the P versus NP Problem*, 2009, https://wayback.archive-it.org/all/20110224135337/http://people.cs.uchicago.edu/~fortnow/papers/pnp-cacm.pdf

[3] Dawar, Anuj, *Complexity Theory*, 2013, https://www.cl.cam.ac.uk/teaching//1213/Complexity/lecture12.pdf

[4] *Complexity Theory* https://web.stanford.edu/class/archive/cs/cs103/cs103.1208/lectures/20-Complexity/Complexity%20Theory.pdf

[5] Art of Problem Solving, *P versus NP* https://artofproblemsolving.com/wiki/index.php/P_versus_NP