Computational Complexity Theory

Alternative course titles

- Introduction to Computational Complexity
- Basics of Computational Complexity Theory

Annotation

One of the main questions that bother modern algorithm designers is how would the algorithm scale when the amount of information to be processed grows, what order of processor time and computer memory will the algorithm require. The answer to the question depends on the type of the computer: do we have *multiple cores or just one of them*? Are we allowed to get an imprecise result with some *small probability*? What space/time tradeoffs are we expecting? Mathematical framework that is aimed at enabling us to ask these questions in a formal and quantifiable way is exactly the *computational complexity theory*. Having emerged from mathematical logic in the middle of the XX century, it later spawned connections with applied computer science and is now a standard element in the education of a computer scientist or an informed programmer. In the course we will talk about various *computational models* (i.e. formal definition of *what* is a computer) and discuss what properties of the real world devices they capture. We'll introduce basic *complexity classes* (groups of relatively hard and easy problems that we can solve using a computer). We will provide several formal definitions of *oracles* and *reduction* (mathematical equivalent of using some library function when writing code). Among others we'll learn what is the difference between *probabilistic* and *randomized* algorithms. We will also learn why the question "P=NP?" is considered to be so important for modern theoretical computer science, but why we shouldn't feel doomed when dealing with *NP-hard* problems in practice.

Prerequisites

The learners are assumed to have a solid (although basic) understanding of

- discrete mathematics (pigeonhole principle, induction, operations with sets, quantifiers, logical connectives),
- graph theory (graphs, digraphs, trees, circuits, coloring),
- probability (finite/discrete probability space, random variable, expectation).

Some familiarity with boolean functions is a plus, yet not required.

Topics

- Introduction. Easy and hard problems. Algorithms and complexity.
- Models of computation. Circuits, automata, and Turing machines. Multitape deterministic and non-deterministic Turing machines. How it all translates to real life computing machines.
- Decision problems. The Halting Problem and Undecidable Languages. Counting and diagonalisation. Reductions.
- Deterministic Complexity Classes. DTIME[t]. Linear Speed-up Theorem. PTime. Polynomial reducibility.
- NP and NP-completeness. Non-deterministic Turing machines. NTIME[t]. NP. Polynomial time verification. NP-completeness. Cook-Levin Theorem. Polynomial transformations: 3-satisfiability, clique, colourability, Hamilton cycle, partition problems. Pseudo-polynomial time. Strong NP-completeness. Knapsack. NP-hardness. 2-SAT in PTime.
- Space complexity and hierarchy theorems. DSPACE[s]. Linear Space Compression Theorem. PSPACE, NPSPACE. PSPACE = NPSPACE. PSPACE-completeness. Quantified Boolean Formula problem is PSPACE-complete. L, NL and NL-completeness. NL=coNL. Hierarchy theorems.
- Optimization and approximation. Combinatorial optimization problems. Relative error. Bin-packing problem. Polynomial and fully polynomial approximation schemes. Vertex cover, travelling salesman problem, minimum partition.
- Randomized Complexity. The classes BPP, RP, ZPP.
- Interactive proof systems. IP = PSPACE.
- Introduction to communication complexity.

References

- Sanjeev Arora, Boaz Barak. Computational Complexity: A Modern Approach. Cambridge University Press, 2009. ISBN: 978-0-5214-2426-4
- Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein. Introduction to Algorithms (3rd ed.). MIT Press and McGraw-Hill, 2009. ISBN 978-0-2620-3384-8
- Cristopher Moore, Stephan Mertens. The Nature of Computation. Oxford University Press, 2011. ISBN 978-0-1992-3321-2