Computational Complexity: Extremely Short Guide

Vitaly Parnas

March 13, 2018

I summarize the material I consider crucial in the computational complexity branch of Theoretical Computer Science, omitting detailed proofs in favor of higher level concepts.

Helpful prerequisites: Asymptotic notation, set-builder notation, Turing Machines, mapping reductions.

Introduction

Computational complexity concerns itself with the algorithm runtime analysis. In Theoretical Computer Science, the runtime is measured not in absolute units bound to an arbitrary CPU, but asymptotically. Asymptotic runtime can encompass constant, sublinear, linear, polynomial, or exponential runtime with respect to the increasing input.

In this guide, I focus not on specific asymptotic runtime, but on the two major classes of problems: P and NP, the first corresponding to problems with a known polynomial or better solution, and the second corresponding to problems for which no solution better than exponential is known to exist. I then provide a high level explanation in demonstrating the problem class membership. Lastly, I survey a few among the *hardest* problems in the NP class, known as NP Complete problems.

Classes P and NP

The class P contains problems solvable in polynomial asymptotic runtime (polytime) on a deterministic Turing Machine (TM). Now, since a TM is equivalent to any other reasonable computational model within a polynomial factor[1], we can simply disregard the TM part of the definition for the sake of simplicity. For example, the Shortest Path problem between any pair of vertices in a graph is a class P problem, and has a number of different polytime solutions[2] depending on the type of graph, input representation, and data structure used.

The class NP is formally defined as problems solvable in polynomial time (still) but on a Nondeterministic Turing Machine.

As a refresher, a Nondeterministic Turing Machine (NTM) can can have multiple transitions from the same state and input, in contrast to a traditional (deterministic) TM or non-quantum computational unit in general that defines only one transition per state/input pair. To phrase it another way, a NTM can be viewed as a parallel computational unit with infinite nodes and zero inter-node communication cost. If any one branch accepts the input, the entire NTM accepts. If all the branches reject, the NTM rejects. The notion, although entirely abstract, helps in formally demonstrating not only the NP class but also specific problem class membership.

The NP class can alternatively be defined to contain problems that can be verified on a standard TM in polynomial time. That is, one can construct a *verifier* TM that, given the encoded problem and a *certificate*, the verifier can efficiently (in polynomial time) determine whether the certificate is a solution to the problem. This makes sense as verifying a potential solution is

often easier than solving the problem.

An example of a straightforward NP problem is that of finding a correct subset among a sequence of n numbers. Provided that we can efficiently determine whether any particular subset is the correct one, it remains to simply test all the 2^n subsets (an exponential amount with respect to n), which we can do by spawning a parallel NTM branch for every subset. Since the verification takes place in polytime, one of the branches will accept and the entire NTM will thus halt in polytime. (If no subset is 'correct' for this problem, then all branches will fail verification in polytime, and the entire NTM will still efficiently halt.)

P vs NP is an open problem, and although we suspect $P \neq NP$, no affirmative or negative proof exists. If $P = NP$, then any hard problem would suddenly acquire an efficient solution, resulting in radical changes to our world.

Polynomial Reductions

Similar to mapping reductions[3] that map one problem to another problem at least as hard in order to determine the problem computability, a polynomial reduction is similar, but useful in proving the problem class membership.

Formally, problem A polynomially reduces to problem B if we can construct a function f on a TM such that 1) f can be computed in polytime, and 2) for any input w, A accepts input w if and only if B accepts $f(w)$. Once we devise such a reduction, the immediate implication is that if A is not in the P class, then neither is B (being at least as hard). As a contrapositive, $B \in P$ implies $A \in P$.

For example, the Independent Set problem is defined as finding a certain (typically maximal) number of vertices in a graph that don't share any common edges. This is an NP problem. It also polynomially reduces to the Vertex Cover problem, defined by finding a certain (typically minimal) number of vertices incident (connected) to all the edges. A quick observation reveals that any vertex subset S in the graph is independent if and only if the complement subset, $V - S$, is a vertex cover. Mapping S to $V - S$ is also a trivially simple, polytime operation. The resulting implication is that Vertex Cover is also not in P, or rather, at least as hard as the NP class.

NP Complete Problems

NP Complete (NPC) problems are those considered the hardest in the NP class. Formally, problem A is NPC if $A \in NP$ AND either 1) ALL other problems $A' \in NP$ can be polynomially reduced to A or 2) some other NPC problem polynomially reduces to A.

NPC problems are special because the corollary to the above is that finding an efficient solution to any one NPC problem carries the immediate implication that $P = NP$, since an NPC problem is among the hardest in NP and all others can polynomially reduce to it.

I want to highlight a handful of NPC problems for their interesting hardness proofs.

SAT

The Boolean satisfiability problem [4], also known as SAT, is defined as follows: given a number of variables, literals, and clauses, the problem has a solution if there exists a satisfying truth assignment to the variables, or rather, if the boolean formula can be satisfied. SAT was the first problem to be proven NPC by the Cook-Levin theorem [5], demonstrating the mapping of all NP problems to SAT.

A number of other problems have consequently been proven NPC by virtue of transitive polynomial reductions initially SAT, including the aforementioned Independent set, Subset Sum, Integral Knapsack, Hamiltonian Cycle, Travelling Salesman, and a myriad of others [5].

3-colourable

The 3-Colourable problem carries the following definition: given an undirected graph G, the vertices of G can be colored in three colors (or less) such that no two adjacent vertices share a color.

3-Colourable is NPC, and the proof uses gadgets[4, 6] in polynomially reducing SAT to 3-Colourable. Specifically, a SAT formula is converted to a graph by creating vertices and edges that correspond to the formula clauses, variables, and literals. The resulting graph is shown to be 3-colourable if and only if the SAT formula has a solution.

Gadgets have since been used to demonstrate NP completeness (or at least NP hardness) in problems that don't strictly lend themselves to this approach in an intuitive way.

Super Mario

Video games also lend themselves to computational complexity analysis and hardness proofs. A handful of Super Mario games, Donkey Kong, Legend of Zelda, Metroid, and Pokémon have been demonstrated as NP Hard [7] by means of gadgets in reducing from SAT to each respective game. Proving NP hardness is not as strong as NP Completeness, since a NP hard problem has only been reduced from all other NP problems, but has not itself been restricted to NP, and could belong to an even harder, superset class of problems.

A later publication, however, demonstrates an upper bound on the hardness of Super Mario, that of not only belonging to the class PSPACE, but the set of PSPACE-complete problems [8]. (PSPACE, a superset of NP, is a space complexity class not covered in this publication.) The proof reduces from the Knapsack problem by means of the appropriate gadgets.

Conclusion

Computational complexity together with the broader theory of computation provide a framework to identify problem tractability, complexity, as well as the problem interrelationships. This publication aims to only introduce some of the fundamental components in cultivating a stronger appreciation for the area. For further in-depth explanations and broader insight, I encourage you to explore the references.

References

[1] Wikipedia contributors. Church–turing thesis — wikipedia, the free encyclopedia, 2017. [Online; accessed 13-March-2018].

[2] Wikipedia contributors. Shortest path problem — wikipedia, the free encyclopedia, 2018. [Online; accessed 13-March-2018].

[3] Wikipedia contributors. Reduction (complexity) — wikipedia, the free encyclopedia, 2018. [Online; accessed 13-March-2018].

[4] Michael Sipser. *Introduction to the Theory of Computation*. Course Technology, Boston, MA, third edition, 2013.

[5] Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990.

[6] M.R. Garey, D.S. Johnson, and L. Stockmeyer. Some simplified np-complete graph problems. *Theoretical Computer Science*, 1(3):237 – 267, 1976.

[7] Greg Aloupis, Erik D. Demaine, and Alan Guo. Classic nintendo games are (np-)hard. *CoRR*, abs/1203.1895, 2012.

[8] Erik D. Demaine, Giovanni Viglietta, and Aaron Williams. Super mario bros. is harder/easier than we thought. In Erik D. Demaine and Fabrizio Grandoni, editors, *FUN*, volume 49 of *LIPIcs*, pages 13:1–13:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016.