COMPLEXITY CLASSES

SADHIKA MALLADI

1. Introduction to Complexity Classes

Complexity classes provide a way for us to understand problems in the context of the amount of resources required. Another way to understand this notion is by observing how that quantity of resources varies with the size of the input. Resources can be considered in the context of either memory or time. The O function is thus closely related to complexity classes as it allows us to generalize the run time of an algorithm in terms of the size of the input.

2. Common Complexity Classes

Common complexity classes include P, NP, and PSPACE.

Definition 2.1. A problem belongs to the class P, which stands for polynomial-time, if a solution can be found by an algorithm that runs in $O(n^a)$, where n is the size of the input and a is some constant.

An example of an algorithm we're familiar with that belongs to P is the Euclidean algorithm for calculating the greatest common divisor. To briefly summarize, $\gcd(a, b) = \gcd(b, a \bmod b)$. A long division algorithm can run in polynomial time, as can subtraction and multiplication, so the Euclidean algorithm runs in polynomial time.

Definition 2.2. A problem belongs to the class NP if a solution can be verified for some specific instance of that problem in polynomial time.

Note that there is another formal definition for NP that involves a non-deterministic Turing machine, but we will only consider deterministic Turing machines in this presentation.

A commonly cited NP problem is the traveling salesman problem: given an input matrix detailing the distance between n cities, determine if there is a route that traverses a total distance less than k visiting all cities. A solution that we may want to verify would be a list of cities in the order that we could visit them. Verifying that solution would require simply adding the entries of the matrix and verifying that the sum is less than k.

While both of these classes involve run time of an algorithm, problems belonging to PSPACE require a polynomial amount of space to be solved.

3. P versus NP

Intuitively, the complexity class P is contained in NP, because finding a solution in polynomial time to a problem implies that any given solution can be verified simply by solving the problem. However, an important open question in the field is if P=NP. Informally, the question asks if every problem that allows an algorithm to verify solutions in polynomial time can also be solved in polynomial time.

Date: March 14, 2016.

As an example, consider what is known as the subset sum problem: given a set of integers, determine if some nonempty subset sum to 0. Verifying a solution can be done in polynomial time: given a subset, sum the numbers (which runs in polynomial time), and compare the output to 0. Thus, the problem is in NP. However, no known algorithm can identify such a subset in polynomial time, so the problem's membership in P is unknown.

4. NP-Complete

The question of P=NP is often approached from the standpoint of NP-complete problems.

Definition 4.1. NP-complete problems are a set of problems, contained in NP, to which any other NP problem can be reduced in polynomial time.

Informally, NP-complete problems are the "hardest" of the NP complexity class. Proving that an NP-complete problem has a solution that runs in polynomial time (thus making that problem a member of the class P) is equivalent to proving that P=NP, since all NP problems can be reduced to any one of the NP-complete problems.

Proving membership of a problem to NP-complete is now largely facilitated through the proof of the Boolean Satisfiability Problem as an NP-complete problem. The Boolean Satisfiability Problem asks if there exists an interpretation of a given Boolean formula that evaluates to TRUE. In other words, can the variables of a Boolean formula be replaced consistently to ensure that the formula evaluates to TRUE.

Theorem 4.2 (Cook-Levin Theorem)**.** *The Boolean Satisfiability Problem is NP-complete.*

The Cook-Levin Theorem is particularly important because many problems have been shown to be NP-complete through reduction, including the subset sum problem discussed above. A vast set of problems can thus be reduced to a simply stated equivalent problem.

5. Speculation about P versus NP

Due to the Cook-Levin Theorem, thousands of important problems have been identified as NP-complete. A polynomial time solution to any one of those problems would establish that P=NP, but no polynomial time algorithms have been found for any of these problems. Intuitively, if P=NP, then solving a problem is as simple as verifying a given solution, which seems fundamentally untrue in the world. If it was shown that P≠NP, then people would be able to show formally that many problems cannot be solved in polynomial time.

However, some people argue that P=NP. The argument that exhaustive searches have uncovered no polynomial time solutions is countered by the idea that deep theoretical arguments are sometimes required for fundamentally simple problems, such as Fermat's Last Theorem. A proof of P=NP alone has major theoretical consequences. If such a proof can lead to the discovery of methods to solve important NP problems, then many of the cryptographic methods we learned may be easily broken. For example, the problem of reverse hashing, or finding a value that hashes to another given value, is difficult. Current algorithms require exponential time, but if P=NP, then the hash can be compromised in polynomial time.

Regardless of speculation, a proof in either direction would greatly advance theoretical and practical computer science, as well as many fields that rely on them, such as cryptography.