prev

16 Beyond NP

The theory of NP-completess is a bedrock of computer science because there are so many NP-complete problems, and they crop of everywhere. There are NP-complete problems from mathematics, the theory of databases, the theory of compilers and even from politics.

But even though NP-completeness is central, there is more to the world than that. This section looks at two other classes of problems.

16.1 Co-NP and the validity problem

We started looking at difficult problems in Section 12.3 with the Validity Problem for Propositional Logic (VALIDPL). But we have not said anything more about it yet. We have not shown that it is NP-complete, nor have we shown that it is in P.

There is a good reason for that. If Conjecture 15.1 is true and P $\neq$ NP, then the validity problem is neither in P nor NP-complete. That is a consequence of the asymmetry of NP: if $A \in$ NP, then there are short, easily checkable proofs that things are in A, but there is no requirement that there are short, easily checkable proofs that things are not in A.

But VALIDPL has short, easily checkable proofs of nonmembership. To show that ϕ is not valid, show that $\neg\phi$ is satisfiable by finding a truth-value assignment that makes $\neg\phi$ true. The following is obvious.

Theorem 16.1. Suppose ϕ is a propositional formula. ϕ is valid if and only if $\neg\phi$ is not satisfiable.

Define $\overline{\text{SATPL}}$ to be the set of propositional formulas that are not satisfiable. Then $f(\phi) = \neg\phi$ is a polynomial-time reduction from VALIDPL to $\overline{SATPL}$. The same function is a polynomial-time reduction from $\overline{SATPL}$ to VALIDPL. So VALIDPL is equivalent in difficulty to $\overline{SATPL}$.

What can we say about languages that are complements of NP-complete languages?

Definition 16.1. Co-NP $= \{X \mid \overline{X} \in \text{NP}\}$ is the class of all languages whose complements are in NP.

Pay close attention to the definition of Co-NP. Co-NP is not the complement of NP.

Definition 16.2. Language A is *Co-NP-complete* if $A \in$ Co-NP and $X \leq_p A$ for every language $X \in$ Co-NP.

Theorem 16.2. A is NP-complete if and only if $\overline{A}$ is Co-NP-complete.

Proof. By the definition of Co-NP, $A \in \text{NP} \rightarrow \overline{A} \in \text{Co-NP}$ and $\overline{A} \in \text{NP} \rightarrow \overline{\overline{A}} \in \text{Co-NP}$. But $\overline{\overline{A}} = A$.

Suppose that A is NP-complete. Then $X \leq_p A$ for every X in NP. Suppose that f is a polynomial-time reduction from X to A. By the definition of a polynomial-time reduction,

$$x \in X \leftrightarrow f(x) \in A.$$

So

$$x \notin X \leftrightarrow f(x) \notin A.$$

or, equivalently,

$$x \in \overline{X} \leftrightarrow f(x) \in \overline{A}.$$

So f is a polynomial-time reduction from $\overline{X}$ to $\overline{A}$. So $\overline{A}$ is Co-NP-complete.

The other direction, showing that if $X \in$ Co-NP then then $\overline{X}$ is NP-complete, is true by symmetry.

♢———————————————————————————————♢

Since VALIDPL is equivalent to $\overline{SATPL}$, VALIDPL is Co-NP-complete.

16.2 NP ∩ Co-NP and factoring

We know that P $\subseteq$ NP. By symmetry, P $\subset$ Co-NP. So P $\subset$ NP $\cap$ Co-NP. An obvious question is: Is P $=$ NP $\cap$ Co-NP. That is conjectured to be false.

Conjecture 16.1 P $\neq$ (NP $\cap$ Co-NP).

Proving Conjecture 16.1 is difficult because, if P = NP then P = Co-NP and the conjecture is false.

But what would lead people to make Conjecture 16.1? There must be some decision problem that is conjectured to be in NP ∩ Co-NP but not in P. And there is: factoring integers. Quick, what are the prime factors of 109,938,432,277?

The problem of factoring a given integer cannot be in NP ∩ Co-NP because it is not a decision problem; the result is a list of factors. But there is a decision problem that is equivalent to that in computational difficulty.

Definition 16.3. FACTOR is the following decision problem.

> **Input.** Two positive integers x and k.
> **Question.** Does there exist a factor y of x where $1 < y < k$?

If you have a polynomial-time algorithm that finds the factors of an integer then it is easy to decide FACTOR. And if you have a polynomial-time algorithm for FACTOR then you can find the smallest factor of an integer using binary search. Having found the smallest factor, you divide x by that factor and continue finding factors, stopping when the number that you have is prime. (As mentioned in Section 12, there is a known algorithm that determines whether a given integer is prime.)

Conjecture 16.2 FACTOR $\in (NP \cap$ Co-NP$) -$ P.

16.3 Public key cryptograpy

Cryptographic systems are based on keys. To encipher a message you use the encipher key, and to decipher a message you use the associated decipher key. A person who has the decipher key is said to decipher a message. A person who attempts to do the same job without the benefit of the decipher key is said to *decrypt* the message.

Traditional cryptography is based on the idea that someone attempting to decrypt an enciphered message does not have enough *information* to do so. The standard traditional cryptosystem is a one-time pad, where a randomly chosen and agreed on text or number is used as a key, which must only be used once without giving away information to an adversary. With a one-time pad, the enciper key and decipher keys are the same.

Public key cryptography takes a different viewpoint. The encipher and decipher keys are different, and the strength of the system is based on the idea that someone trying to decrypt a message does not have enough *time*. And that is based on the problem of decrypting a message being a computationally difficult problem.

Definition 16.4. A public key cryptosystem is described by two functions $E(k, x)$ and $D(j, y)$ where k is a public encipher key and j is a private decipher key, such that

1. For every x in a limited range, $D(j, E(k, x)) = x$ and $E(k, D(j, x)) = x$. That is, deciphering an enciphered message gives the original message, and enciphering a deciphered message also gives the original message.

2. There is a polynomial-time algorithm to compute $E(k, x)$ and another to compute $D(j, y)$.

3. The *decryption function* $D(y)$ is defined to take y and yield a value x such that $E(k, x) = y$; $D(y)$ decrypts without the benefit of j. There is no polynomial-time algorithm that computes $D(y)$.

It is important to realize that the encipher key k is public, available to anyone. The strength of a public key cryptosystem is tied to the (at least apparent) computational difficult of computing $D(y)$.

Several public key cryptosystems are known, and it is not our concern here to describe one. Rather, let's ask whether a public-key cryptosystem exists. Assume that function pair $(E(k, x),\ D(j, y))$ is such a cryptosystem. For simplicity, assume that messages (x and y) are integers. Text can always be encoded using integers.

Consider the following decision problem DECRYPT determined by the decryption function $D(y)$.

$$\text{DECRYPT} = \{(y, i) \mid D(y) < i\}.$$

DECRYPT must be in NP. As evidence, use the decipher key j. First compute $x = D(j, y)$, then compute $z = E(k, x)$. Accept j as evidence that $(y, i) \in$ DECRYPT provided $z = y$ and $x < i$. Requirement $z = y$ tells

you that j is the correct decipher key and requirement $x < i$ tells you that $D(y) < i$.

DECRYPT must also be in Co-NP. The complement of DECRYPT is equivalent to language $\{(y, i) \mid D(y) \geq i\}$, and almost the same evidence checker works for that.

So DECRYPT is in NP $\cap$ Co-NP. But if DECRYPT is in P then there is a polynomial-time algorithm to compute $D(y)$. Simply use binary search to search for the smallest i such that $D(y) < i$. Then $D(y) = i - 1$. That leads to the following conclusion.

Theorem 16.3. A public key cryptosystem can only exist if P $\neq$ NP $\cap$ Co-NP.

It is no accident that public key cryptosystems are based on factoring or on other problems that are in NP $\cap$ Co-NP.

16.4 Polynomial Space

Definition 16.5. *PSPACE* is the class of all decision problems that can be solved using $O(n^k)$ bits of memory for some fixed k, where n is the length of the input.

It is known that NP $\subseteq$ PSPACE and Co-NP $\subseteq$ PSPACE, and it is conjectured that NP $\neq$ PSPACE (and Co-NP $\neq$ PSPACE). Polynomial space allows a lot of room for computations. Typical exponential-time algorithms only use a polynomial amount of memory.

A PSPACE-complete problem is one of the hardest problems in PSPACE.

Definition 16.6. A decision problem A if *PSPACE-complete* if

(a) A is in PSPACE and

(b) $X \leq_p A$ for every problem $X \in$ PSPACE.

PSPACE is closely related to computations where quantifiers alternate between universal and existential. For that reason, several PSPACE-complete

problems are related to two-person games. The following are some PSPACE-complete problems.

Definition 16.7. *Generalized Checkers* is the following decision problem.

> **Input.** A placement of red and black kings on an $n \times n$ checkerboard.
> **Question.** Assuming that it is red's move, does red have a winning strategy from the given configuration?

Theorem 16.4. Generalized Checkers is PSPACE-complete.

Geography is a game that children can play without any props. A child thinks of the name of a country (or other chosen category). If the first child select Sweden, then the next child must select a country name that begins with N, the last letter of Sweden. Suppose that child chooses Nepal. Now the next player must choose a country name that starts with L. Countries cannot be reused, and the first child who cannot think of a country name loses.

There is a version of Geography that is played on a directed graph. A vertex is selected as the start vertex s. The first player selects a vertex u where there is a directed edge from s to u. The second player selects a vertex v where there is a directed edge from u to v. Play alternates. No vertex that was previously selected can be selected again.

Definition 16.7. The *Generalized Geography* problem is the following decision problem.

> **Input.** A directed graph G with a selected start vertex.
> **Question.** Does the first player have a winning strategy on the game of Geography played on G?

Theorem 16.5. Generalize Geography is PSPACE-complete.

In practice, PSPACE-complete problems appear very difficult to solve. If the conjecture that NP $\neq$ PSPACE is true, then PSPACE-complete problems do not have polynomial-time evidence checkers.

Surprisingly, however, nobody knows whether P = PSPACE. Even the huge jump from polynomial time to polynomial space is not enough for us to demonstrate a separation. If you want to prove that P $\neq$ NP, you might warm up by proving that P $\neq$ PSPACE; that ought to be easier.

16.5 Exponential time

Definition 16.8. *EXPTIME* is the class of decision problems that are solvable in time $O(2^{n^k})$ for some fixed k, where n is the length of the input.

Notice that EXPTIME allows an amount of time that is two to a polynomial in n. It is known that PSPACE $\subseteq$ EXPTIME, and PSPACE is conjectured to be a proper subset of EXPTIME.

It is known that P $\neq EXPTIME$. So at least one of the subset relations P $\subseteq$ NP $\subseteq$ PSPACE $\subseteq$ EXPTIME is surely a proper subset relation. Of course, they are all conjectured to be proper.

There is a notion of an EXPTIME-complete problem, defined in the usual way as a hardest problem in EXPTIME.

Definition 16.9. A decision problem A is *EXPTIME-complete* if

(a) A is in EXPTIME and

(b) $X \leq_p A$ for every problem $X \in$ EXPTIME.

TRhere are EXPTIME-complete problems. There is a typed programming language called ML. The ML Type Checking problem is as follows.

Definition 16.10. The *ML Type Checking Problem* is the following decision problem.

Input. An ML program p.
Question. Is p well-typed (free of type errors)?

The ML Type Checking Problem is known to be EXPTIME-complete. Fortunately, ML programmers tend not to write programs that are difficult to type check. But if you want to, you can write an ML program that will bring an ML compiler to its knees; in practice, the memory requirements overwhelm the compiler, and it gives up.

prev