Cook–Levin theorem

From Wikipedia, the free encyclopedia

In computational complexity theory, the **Cook–Levin theorem**, also known as **Cook's theorem**, states that the Boolean satisfiability problem is NP-complete. That is, any problem in NP can be reduced in polynomial time by a deterministic Turing machine to the problem of determining whether a Boolean formula is satisfiable.

The theorem is named after Stephen Cook and Leonid Levin.

An important consequence of the theorem is that if there exists a deterministic polynomial time algorithm for solving Boolean satisfiability, then there exists a deterministic polynomial time algorithm for solving *all* problems in NP. Crucially, the same follows for any NP complete problem.

The question of whether such an algorithm exists is called the P versus NP problem and it is widely considered the most important unsolved problem in theoretical computer science.

Contents

Contributions

The concept of NP-completeness was developed in the late 1960s and early 1970s in parallel by researchers in the US and the USSR. In the US in 1971, Stephen Cook published his paper "The complexity of theorem proving procedures"[1] in conference proceedings of the newly founded ACM Symposium on Theory of Computing. Richard Karp's subsequent paper, "Reducibility among combinatorial problems",[2] generated renewed interest in Cook's paper by providing a list of 21 NP-complete problems. Cook and Karp received a Turing Award for this work.

The theoretical interest in NP-completeness was also enhanced by the work of Theodore P. Baker, John Gill, and Robert Solovay who showed that solving NP-problems in Oracle machine models requires exponential time. That is, there exists an oracle A such that, for all subexponential deterministic time complexity classes T, the relativized complexity class NP^A is not a subset of T^A. In particular, for this oracle, $\mathrm{P}^A \neq \mathrm{NP}^A$.[3]

In the USSR, a result equivalent to Baker, Gill, and Solovay's was published in 1969 by M. Dekhtiar.[4] Later Levin's paper, "Universal search problems",[5] was published in 1973, although it was mentioned in talks and submitted for publication a few years earlier.

Levin's approach was slightly different from Cook's and Karp's in that he considered search problems, which require finding solutions rather than simply determining existence. He provided 6 such NP-complete search problems, or *universal problems*. Additionally he found for each of these problems an algorithm that solves it in optimal time (in particular, these algorithms run in polynomial time if and only if P = NP).

Definitions

A decision problem is *in NP* if it can be solved by a non-deterministic algorithm in polynomial time.

An *instance of the Boolean satisfiability problem* is a Boolean expression that combines Boolean variables using Boolean operators.

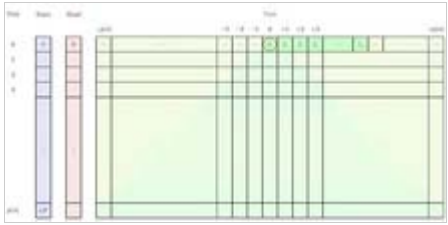
An expression is *satisfiable* if there is some assignment of truth values to the variables that makes the entire expression true.

Idea

Given any decision problem in NP, construct a non-deterministic machine that solves it in polynomial time. Then for each input to that machine, build a Boolean expression which says that the input is passed to the machine, the machine runs correctly, and the machine halts and answers "yes". Then the expression can be satisfied if and only if there is a way for the machine to run correctly and answer "yes", so the satisfiablity of the expression is equivalent to asking whether or not the machine will answer "yes".

Proof

This proof is based on the one given by Garey and Johnson.[6]



Schematized accepting computation by the machine *M*.

There are two parts to proving that the Boolean satisfiability problem (SAT) is NP-complete. One is to show that SAT is an NP problem. The other is to show that every NP problem can be reduced to an instance of a SAT problem by a polynomial-time many-one reduction.

SAT is in NP because any assignment of Boolean values to Boolean variables that is claimed to satisfy the given expression can be *verified* in polynomial time by a deterministic Turing machine. (The statements ***verifiable*** *in polynomial time by a* ***deterministic*** *Turing machine* and ***solvable*** *in polynomial time by a* ***non-deterministic*** *Turing machine* are totally equivalent, and the proof can be found in many textbooks, for example Sipser's *Introduction to the Theory of Computation*, section 7.3.).

Now suppose that a given problem in NP can be solved by the nondeterministic Turing machine $M = (Q, \Sigma, s, F, \delta)$, where Q is the set of states, Σ is the alphabet of tape symbols, $s \in Q$ is the initial state, $F \subseteq Q$ is the set of accepting states, and $\delta \subseteq ((Q \setminus F) \times \Sigma) \times (Q \times \Sigma \times \{-1, +1\})$ is the transition relation. Suppose further that M accepts or rejects an instance of the problem in time $p(n)$ where n is the size of the instance and p is a polynomial function.

For each input I, we specify a Boolean expression which is satisfiable if and only if the machine M accepts I.

The Boolean expression uses the variables set out in the following table. Here, $q \in Q$, $-p(n) \leq i \leq p(n)$, $j \in \Sigma$, and $0 \leq k \leq p(n)$.

Variables	Intended interpretation	How many?
$T_{i,j,k}$	True if tape cell i contains symbol j at step k of the computation.	$O(p(n)^2)$
$H_{i,k}$	True if the M's read/write head is at tape cell i at step k of the computation.	$O(p(n)^2)$
$Q_{q,k}$	True if M is in state q at step k of the computation.	$O(p(n))$

Define the Boolean expression B to be the conjunction of the sub-expressions in the following table, for all $-p(n) \leq i \leq p(n)$ and $0 \leq k \leq p(n)$:

Expression	Conditions	Interpretation	How many?
$T_{i,j,0}$	Tape cell i initially contains symbol j	Initial contents of the tape. For $i > n$-1 and $i < 0$, outside of the actual input I, the initial symbol is the special default/blank symbol.	$O(p(n))$
$Q_{s,0}$		Initial state of M.	1
$H_{0,0}$		Initial position of read/write head.	1
$\neg T_{i,j,k} \vee \neg T_{i,j',k}$	$j \neq j'$	At most one symbol per tape cell.	$O(p(n)^2)$
$\bigvee_{j \in \Sigma} T_{i,j,k}$		At least one symbol per tape cell.	$O(p(n)^2)$
$T_{i,j,k} \wedge T_{i,j',k+1} \rightarrow H_{i,k}$	$j \neq j'$	Tape remains unchanged unless written.	$O(p(n)^2)$
$\neg Q_{q,k} \vee \neg Q_{q',k}$	$q \neq q'$	Only one state at a time.	$O(p(n))$
$\neg H_{i,k} \vee \neg H_{i',k}$	$i \neq i'$	Only one head position at a time.	$O(p(n)^3)$
$(H_{i,k} \wedge Q_{q,k} \wedge T_{i,\sigma,k}) \rightarrow \bigvee_{(q,\ \sigma,\ q',\ \sigma',\ d) \in \delta} (H_{i+d,k+1} \wedge Q_{q',k+1} \wedge T_{i,\sigma',k+1})$	$k<p(n)$	Possible transitions at computation step k when head is at position i.	$O(p(n)^2)$
$\bigvee_{0 \leq k \leq p(n)} \bigvee_{f \in F} Q_{f,k}$		Must finish in an accepting state, not later than in step $p(n)$.	$O(p(n))$

If there is an accepting computation for M on input I, then B is satisfiable by assigning $T_{i,j,k}$, $H_{i,k}$ and $Q_{i,k}$ their intended interpretations. On the other hand, if B is satisfiable, then there is an accepting computation for M on input I that follows the steps indicated by the assignments to the variables.

There are $O(p(n)^2)$ Boolean variables, each encodeable in space $O(\log p(n))$. The number of clauses is $O(p(n)^3)$ so the size of B is $O(\log(p(n))p(n)^3)$. Thus the transformation is certainly a polynomial-time many-one reduction, as required.

Consequences

The proof shows that any problem in NP can be reduced in polynomial time (in fact, logarithmic space suffices) to an instance of the Boolean satisfiability problem. This means that if the Boolean satisfiability problem could be solved in polynomial time by a deterministic Turing machine, then all problems in NP could be solved in polynomial time, and so the complexity class NP would be equal to the complexity class P.

The significance of NP-completeness was made clear by the publication in 1972 of Richard Karp's landmark paper, "Reducibility among combinatorial problems", in which he showed that 21 diverse combinatorial and graph theoretical problems, each infamous for its intractability, are NP-complete.[2]

Karp showed each of his problems to be NP-complete by reducing another problem (already shown to be NP-complete) to that problem. For example, he showed the problem 3SAT (the Boolean satisfiability problem for expressions in conjunctive normal form with exactly three variables or negations of variables per clause) to be NP-complete by showing how to reduce (in polynomial time) any instance of SAT to an equivalent instance of 3SAT. (First you modify the proof of the Cook–Levin theorem, so that the resulting formula is in conjunctive normal form, then you introduce new variables to split clauses with more than 3 atoms. For example, the clause $(A \vee B \vee C \vee D)$ can be replaced by the conjunction of clauses $(A \vee B \vee Z) \wedge (\neg Z \vee C \vee D)$, where Z is a new variable which will not be used anywhere else in the expression. Clauses with fewer than 3 atoms can be padded; for example, A can be replaced by $(A \vee A \vee A)$, and $(A \vee B)$ can be replaced by $(A \vee B \vee B)$.

Garey and Johnson presented more than 300 NP-complete problems in their book *Computers and Intractability: A Guide to the Theory of NP-Completeness*,[6] and new problems are still being discovered to be within that complexity class.

Although many practical instances of SAT can be solved by heuristic methods, the question of whether there is a deterministic polynomial-time algorithm for SAT (and consequently all other NP-complete problems) is still a famous unsolved problem, despite decades of intense effort by complexity theorists, mathematical logicians, and others. For more details, see the article P versus NP problem.

References

1. ^ Cook, Stephen (1971). "The complexity of theorem proving procedures" (http://portal.acm.org/citation.cfm?coll=GUIDE&dl=GUIDE&id=805047). *Proceedings of the Third Annual ACM Symposium on Theory of Computing*. pp. 151–158.
2. ^ ***a*** ***b*** Karp, Richard M. (1972). "Reducibility Among Combinatorial Problems". In Raymond E. Miller and James W. Thatcher (editors). *Complexity of Computer Computations* (http://www.cs.berkeley.edu/~luca/cs172/karp.pdf). New York: Plenum. pp. 85–103. ISBN 0-306-30707-3.
3. ^ T. P. Baker; J. Gill; R. Solovay (1975). "Relativizations of the P = NP question". *SIAM Journal on Computing* **4** (4): 431–442. doi:10.1137/0204037 (https://dx.doi.org/10.1137%2F0204037).
4. ^ Dekhtiar, M. (1969). "On the impossibility of eliminating exhaustive search in computing a function relative to its graph". *Proceedings of the USSR Academy of Sciences* (in Russian) **14**: 1146–1148.
5. ^ Levin, Leonid (1973). "Universal search problems (Russian: Универсальные задачи перебора, Universal'nye perebornye zadachi)". *Problems of Information Transmission (Russian: Проблемы передачи информации, Problemy Peredachi Informatsii)* **9** (3): 115–116. (pdf) (http://www.mathnet.ru/php/getFT.phtml?jrnid=ppi&paperid=914&volume=9&year=1973&issue=3&fpage=115&what=fullt&option_lang=eng) **(Russian)**, translated into English by Trakhtenbrot, B. A. (1984). "A survey of Russian approaches to *perebor* (brute-force searches) algorithms". *Annals of the History of Computing* **6** (4): 384–400. doi:10.1109/MAHC.1984.10036 (https://dx.doi.org/10.1109%2FMAHC.1984.10036).
6. ^ ***a*** ***b*** Garey, Michael R.; David S. Johnson (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman. ISBN 0-7167-1045-5.