The NP = P Conjecture

a general introduction

by

Charles W. Neville, October 2000

©Charles W. Neville, February 2002

Verbatim copying and redistribution of this document is permitted in any medium provided this notice and the copyright notice are preserved.

Introduction

The $NP = P$ conjecture is one of the Clay Mathematics Institute's 7 Millennium Prize Problems. The generally assumed falsity of the $NP = P$ conjecture provides the (somewhat shaky) theoretical foundation for all internet cryptographic security. So this much is known to be true:

THEORY = BIG_BUCKS

0. Introduction

In 1971, Cook proved a remarkable theorem: Among the class of problems which can be solved quickly by inspired guessing, there is a hardest problem – the problem of determining that a given well formed propositional calculus formula has an assignment of truth values making it true. The conjecture that this hardest problem can also be solved quickly without guesswork is known as the $NP = P$ conjecture.

1. Time Complexity Classes

Consider the problem of sorting a list of n elements. The time needed to sort such a list depends on the particular computer used, but with a given computer, using the provably best algorithm possible, the time required is bounded above by a constant M times $n \log n$,

$$T \leq Mn \log n$$

Using O notation, we say

$$T = O(n \log n)$$

or that the sorting problem has time complexity $O(n \log n)$, or that the sorting problem is solvable in $n \log n$ time.

Time Complexity Classes

Let's look at the sorting problem more deeply.

A. It takes more time to fetch a 1 megabit record than a 32 bit integer, so we need to take the size of the sorted data items into consideration. Therefore, measure the size of the list, not by the number of data items but by the *total number of bits* needed to represent all the items on the list. If n now represents the number of bits needed, it still is true that

$$T = O(n \log n)$$

so the sorting problem still has time complexity $O(n \log n)$.

Time Complexity Classes

For the sorting problem, even with a Turing machine,

$$T = O(n \log n)$$

(use mergesort with 3 or more tapes) but for other problems with RAM machine time complexity $O(n \log n)$, there is a quadratic slowdown, and the time complexity changes to $O((n \log n)^2)$. The reason is that we can no longer directly access data on the tapes, but instead must read through the tapes to reach a desired data item.

Time Complexity Classes

B. But suppose n is so large that we can't store the whole list in memory, but must use tape. Then we actually have changed our model of computation from

A VON NEUMANN RAM MACHINE

to

A TURING MACHINE

Time Complexity Classes

So

TIME COMPLEXITY DEPENDS ON THE MODEL OF COMPUTATION

but fortunately only by an exponent of order ≤ 2. And in general we have

Theorem. A problem with RAM machine time complexity $O(f(n)$ has Turing machine time complexity $O(f(n)^2)$ or better.

Time Complexity Classes

C. Time Complexity Class P and Polynomial Time

Alan Cobham introduced time complexity class P in 1964 to provide a robust, model independent time complexity class.

Def. A problem lies in time complexity class P if

a. It has a yes/no answer.

b. The answer can be determined by a program running in time $O(n^p)$ for some exponent p, where n is the number of bits needed to specify the input data for the problem.

If a problem lies in time complexity class P, we say it runs, or is solvable, in *polynomial time*.

Time Complexity Classes

$$W = X_1 \wedge (\neg X_2 \vee X_3) \wedge X_2 \wedge (\neg X_1 \vee X_3)$$

X_1	X_2	X_3	W
T	T	T	T
T	T	F	F
T	F	T	F
T	F	F	F
F	T	T	F
F	T	F	F
F	F	T	F
F	F	F	F

Time Complexity Classes

D. Time Complexity Class NP and Non-Deterministic Polynomial Time

Consider a well formed formula (a *wff*) in the propositional calculus, for example

$$W = X_1 \wedge (\neg X_2 \vee X_3) \wedge X_2 \wedge (\neg X_1 \vee X_3)$$

We might write down its truth table to show it is satisfiable (has an assignment of truth values making it evaluate to TRUE),

Time Complexity Classes

This takes

$2^3 = 8$ steps $\times$ the time necessary to evaluate the formula once.

Or, we might make an inspired guess – assign each of X_1, X_2, X_3 the value TRUE. It takes much less time,

1 step $\times$ the time necessary to evaluate the formula once,

to check that the guessed truth assignment makes the formula evaluate to TRUE.

Time Complexity Classes

The first algorithm, evaluate the truth table, is *deterministic* and can be implemented on a RAM computer (or Turing machine). It runs in $T = O(2^n)$ time (exponential time), where n is the number of bits needed to specify the formula.

Time Complexity Classes

In 1971, Cook introduced time complexity class NP.

Def. A problem lies in time complexity class NP if

a. It has a yes/no answer

b. With an inspired guess, the correctness of the yes answer can be checked (verified) by a program running in polynomial time.

The N in NP stands for *non-deterministic*. The P stands for *polynomial*. If a problem lies in time complexity class NP, we say it runs, or is solvable, in non-deterministic polynomial time.

Time Complexity Classes

The second algorithm is *non-deterministic*, it requires inspired guesswork.

We could implement it on a computer – have the computer generate a pseudo-random assignment of truth values and then check the resulting truth value of the formula, but we would have to be very lucky to get an answer of TRUE. An answer of FALSE would tell us next to nothing, so it would seem that the non-deterministic algorithm is not very useful. However it does have the merit that it runs quite fast, in polynomial time – in fact in linear time, *ie.* in time $T = O(n)$ – with the right guess. We shall see later that non-deterministic algorithms can be quite useful.

Time Complexity Classes

E. Other Problems in Class NP.

The *Boolean satisfiability* problem, the one we just discussed, lies in class NP. Here are some other problems in the class.

i. *Integer Programming.* Given a system of linear inequalities in many variables with integer coefficients, does the system have a solution with integer values for the variables.

ii. *Quadratic Programming.* Given a system of quadratic inequalities in many variables with integer coefficients, does the system have a solution with rational values for the variables.

Time Complexity Classes

iii. Given k cities and a $k \times k$ matrix of intercity distances, is there a traveling salesman tour with total distance less than a given constant M.

iv. And even, given a parabola $dy = ax^2 + bx + c$ with integer coefficients, does it pass through an integer point (a point with integer x and y coordinates) with x coordinate in the interval $0 \leq x \leq M$. Here M is a pre-assigned positive constant.

Time Complexity Classes

To see why factorization is hard, consider the following 200 digit integer,

$$74068877515858675692517951430592361934474770774867281974065794969172976228890022037588025244128056810366427833146859564956939017143360568437769525713167390005495312574690062280062457161088810028950595 7$$

I'll bet you didn't know that its prime factorization is

$$1509402497293445260998365996277047451139493435867383880425876691549588470411353603813444238679891122 1 \times 4907165427954027778108059597498789269411765580199490474427239837091547927832034451262331586358355121 7$$

Time Complexity Classes

All of these problems are equally hard, in fact exactly as hard as the Boolean satisfiability problem, because each can be reduced to the others.

Here is a problem in class NP which is believed to be very hard, but not as hard as the ones just mentioned.

iv. Factorization.

Given an integer m, does it have a factor less than a given constant k.

Time Complexity Classes

But now that you know what I claim to be the factorization, you can easily check my assertion by multiplication. (At least the check is easy in computer terms.) Thus factorization is both hard and in NP.

Time Complexity Classes

The factorization problem forms the basis for the RSA public key encryption algorithm used every time you buy something over the internet. So we have another experimentally verifiable fact,

NUMBER_THEORY = BIG_BUCKS

(but, alas, not to number theorists).

Cook's Theorem and the $NP = P$ Conjecture

Most experts believe the $NP = P$ conjecture is false, but think what a proof that it is true would mean.

i. All the coding schemes used on the internet would be rendered useless, because factorization would be solvable in polynomial time*.

ii. Circuit routing on integrated circuit chips would be much easier and better because circuit routing is an NP problem, and one of great practical importance.

* Well, maybe not, because an n^{57} time algorithm would be as bad as our known exponential time algorithms in a practical sense. But for theory, class P is taken to be the class of tractable problems, and the complement of class P is taken to be the class of intractable problems.

2. Cook's Theorem and the $NP = P$ Conjecture

In 1971, Cook proved the following remarkable theorem:

Theorem. There is a hardest problem in class NP, namely Boolean satisfiability. In other words, if there were a (deterministic) polynomial time algorithm for solving Boolean satisfiability, then every other problem in class NP would also be solvable in polynomial time.

We can rephrase this as,

Theorem. If Boolean satisfiability lies in class P, then so does every other NP problem,

or even as,

Theorem. If Boolean satisfiability lies in class P, then $NP = P$.

Cook's Theorem and the $NP = P$ Conjecture

iii. Military force mix calculations would be much easier and better because they are integer programming problems – it is hard to use half an aircraft carrier.

iv. As a result, pork barrel politics would be eliminated – well probably not, but it's a nice thought.

v. Whoever proved $NP = P$ would be famous, and might, like Wiles, be given a *silver* medal by the International Mathematical Union – but only if over the age of 40.

3. References

Alfred V. Aho, John E. Hopcroft and Jeffrey D. Ullman. *The design and analysis of computer algorithms*, Addison Wesley, Reading Mass., 1974.

Alan Cobham. *The intrinsic computational difficulty for functions*. In Y. Bar-Hillel, editor, Proceedings of the 1964 International Congress on Logic, Methodology and Philosophy of Science, pages 24 – 30, Elsevier/North-Holland, 1964.

Stephen A. Cook. *The complexity of theorem-proving procedures*. In Conference Record of Third Annual ACM Symposium on Theory of Computing, pages 151 – 158, Shaker Heights, Ohio, 3 – 5 1971 1971.

References

Stephen A. Cook. *The P versus NP Problem*. Clay Mathematics Institute Millennium Prize Problems, http://www.claymath.org/prize_problems/p_vs_np.pdf, accessed 10/10/2000.

Michael R. Garey and David S. Johnson. *Computers and intractability, a guide to NP-completeness*. W. H. Freeman and Co., San Francisco, 1979.

Richard M. Karp. *Reducibility among combinatorial problems*. In R. E. Miller, and J. W. Thatcher, editors, Complexity of Computer Computations, pages 85 – 103, Plenum Press, New York, 1972.

Dominic Welsh. *Codes and cryptography*, Oxford University Press, Oxford, 1988.