The NP vs P Problem

the second of two talks – why it is hard

by

Charles W. Neville, October 2000

0. Quick Review

A. Polynomial Time and Class P

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*.

Quick Review

B. Non-deterministic Polynomial Time and 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.

Quick Review

C. The NP vs P Problem

Clearly, $P \subseteq NP$. The NP vs P problem is,

Question. Is $P = NP$?

The NP vs P problem is one of the Clay Mathematics Institute's 7 Millennium Prize Problems. The generally assumed falsity of $P = NP$ provides the (somewhat shaky) theoretical foundation for all internet cryptographic security.

Quick Review

D. Boolean Satisfiability

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

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

The Boolean satisfiability problem is the problem of determining whether a given wff has an assignment of truth values making it evaluate to TRUE. It is very simple to solve the Boolean satisfiability problem for a simple wff such as the one above, but it is very difficult to solve the problem for complicated wffs, say a wff in 1,000 variables with 10,000 terms, by known techniques.

Quick Review

E. Cook's Theorem

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

$$P = NP$$

F. Other NP Complete Problems

An problem in class NP which is maximally hard, like Boolean satisfiability, is called *NP complete*. Here are some other NP complete problems:

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.

Quick Review

F. Other NP Complete Problems continued ...

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.

By now, there are thousands of known NP complete problems (c.f. Garey and Johnson).

1. Strategies for Proving $P = NP$

Pick your favorite NP complete problem, and prove it lies in class P by producing a polynomial time algorithm for it.

In the 30 years since Cook's paper, no one has succeeded. Of course, in mathematics, problems are not regarded as *really difficult* until they have been around for a century or more, so it is premature to dismiss this approach.

Interestingly enough, for several NP complete problems including Boolean satisfiability and traveling salesman, there are algorithms which run in polynomial time in the *average case.* So hard instances of NP complete problems seem to be rare!

2. Strategies for Proving $P \neq NP$

A. Prove co-$NP \neq NP$

B. Diagonalization

Let us explain each of these in turn.

3. Prove co-$NP \neq NP$

Def. A problem lies in co-NP if

a. It has a yes/no answer

b. With an inspired guess, the correctness of the *no* answer can be checked (verified) by a program running in polynomial time.

In other words, a problem lies in co-NP if its negation lies in NP.

Prove co-$NP \neq NP$ continued ...

A. Why this would show $P \neq NP$

Because, by definition, a deterministic polynomial time algorithm for solving a problem tells us whether the answer is *yes* or *no*. Thus if $P = NP$, then $NP =$ co-NP, which would be a CONTRADICTION.

PROVE CO-$NP \neq NP$ CONTINUED ...

B. Why is co-NP unlike NP?

The co-NP problem corresponding to Boolean satisfiability is: Given a wff, is it never satisfiable, that is, is it a contradiction?

Guessing a truth assignment making the given wff evaluate to TRUE shows it is not a contradiction, but guessing a truth assignment making the wff evaluate to FALSE tells us practically nothing, because to show the wff is a contradiction, we have to show it evaluates to FALSE for ALL possible truth assignments.

PROVE CO-$NP \neq NP$ CONTINUED ...

B. Why is co-NP unlike NP? continued ...

What would an NP algorithm for showing a given wff is a contradiction amount to?

Answer. It would amount to a *proof system* with polynomial length proofs for proving wffs were contradictions. The NP algorithm would consist of guessing a proof in the proof system, and then checking it.

Prove co-$NP \neq NP$ continued ...

C. What is known about the lengths of proofs of wffs?

i. Robinson and Putnam introduced an elegant proof system called resolution (technicality – works for wffs in conjunctive normal form like our sample wff above, but this is enough).

Only rule: $\neg p \vee q$ and $p \vee r$ implies $q \vee r$

(together with the special cases:

$\neg p \vee q$ and $p \vee q$ implies q

$\neg p$ and $p \vee q$ implies q (modus ponens),

and the fact that $\neg p \wedge p$ is a CONTRADICTION)

C. The lengths of proofs of wffs continued ...

ii. In 1982, Goldberg, Purdom and Brown proved

Theorem. The expected runtime of resolution theorem proving (correctly done) is $O(n)$. Thus the expected length of resolution proofs of wffs of length n is $O(n)$.

C. The lengths of proofs of wffs continued ...

Thus in the average case, the lengths of resolution proofs grow as $O(n)$. But in 1979, Cook and Reckhow introduced a family of wffs which expresses the Dirichlet pigeon hole principle, if you put n objects in $n + 1$ boxes, then at least one box is empty. In 1985, Haken (the son of four color Haken) proved,

Theorem. The shortest resolution proofs of the wffs in Cook's family grow exponentially fast in length.

PROVE CO-$NP \neq NP$ CONTINUED ...

C. The lengths of proofs of wffs continued ...

Hence, in the worst case, the lengths of resolution proofs grow exponentially fast.

iii. However, there is a more complex proof system called *extended resolution*. The wffs in Cook and Reckhow's family have polynomial length extended resolution proofs. No one knows if the worst case behavior of extended resolution is super-polynomial.

4. Diagonalization

Recall the familiar Cantor second diagonal method:

Theorem. The set of all real numbers is uncountable.

Proof. Suppose not. Then it is possible to list those in the interval $(0,1)$ in the familiar way,

$$\begin{array}{l} 0.a_{1,1}a_{1,2}a_{1,3}\cdots \\ 0.a_{2,1}a_{2,2}a_{2,3}\cdots \\ 0.a_{3,1}a_{3,2}a_{3,3}\cdots \\ \quad\vdots \qquad\qquad\qquad \ddots \end{array}$$

where the $a_{i,j}$'s are the digits of the decimal expansion of the i^{th} real a_i, with no infinite sequence of 9's.

Diagonalization

Now construct a real $r = 0.r_1 r_2 r_3$ so that $r_i \neq a_{i,i}$, again with no infinite sequence of 9's to preserve uniqueness. (For instance, choose $r_i = 7$ if $a_{i,i} \neq 7$, and choose $a_{i,i} = 3$ otherwise.) Clearly, $r \neq$ any of $a_1, a_2, a_3, \ldots$ because r differs in at least one decimal place from each of the a_i's. CONTRADICTION

DIAGONALIZATION

The same sort of argument shows,

Theorem. Let E be the time complexity class of all problems with yes/no answers solvable in time $O(e^n)$ (exponential time). Then there is a problem in E which does not lie in P.

Proof sketch. Let $\mathcal{P}$ be the set of all programs in your favorite programming language (Pascal, Java, C, etc.) which run in polynomial time and output a yes/no answer. Let t (for "twisted") be a program which behaves as follows,

Diagonalization

If $p \in \mathcal{P}$ gives a *yes* result when run with itself as input, have our twisted program t output a *no* result when run on input p. But if p when run in the above way gives a *no* result, have t output a *yes* result.

Finally, arrange to have t run in exponential time and output a yes/no answer for any input string, whether a legal (Pascal, Java, C, etc.) program or not.

Note that our twisted program t behaves in exactly the opposite way from each $p \in \mathcal{P}$ on at least one input, so t can't be in $\mathcal{P}$.

The problem solvable in exponential time which is not solvable in polynomial time is the following,

Problem T. On a given input string, will our twisted program t give a yes or no answer?

For any given input string, we can simply run t on it to solve this problem in exponential time. On the other hand, t behaves differently on at least one input from every program in $\mathcal{P}$, so it does not run in polynomial time. More strongly, no program which runs in polynomial time can correctly predict the behavior of t on all inputs, because the behavior of t differs from the behavior of the given polynomial time program on at least one input. Therefore, Problem T is *not* solvable in polynomial time.

Diagonalization

This same sort of argument, by the way, allows us to prove Turing's famous undecideability theorem,

Theorem. The halting problem is undecideable. More precisely, there is no program which can correctly predict whether each given program with given input halts, or runs forever. (Turning off the switch is not allowed.)

Proof sketch. Suppose not. Let s (for "super") be the program which does the predicting. Let t (again for "twisted") have s as a subroutine, and behave as follows,

Diagonalization

If our super program s predicts program p with itself as input halts, then our twisted program t runs forever. But if s predicts p with itself as input runs forever, then t halts.

The twisted program t disagrees with the prediction of s for every program including itself. So s CANNOT correctly predict the behavior of t. CONTRADICTION.

Diagonalization

The successes of diagonalization listed above,

- There are problems solvable in exponential time which are not solvable in polynomial time
- The halting problem is undecideable

and many, many more, have led to the hope that diagonalization can successfully be applied to prove $P \neq NP$. Let us examine the situation more closely.

5. Diagonalization and the NP vs P Problem

In most cases, diagonalization arguments show MOST objects are unusual.

- Most numbers are transcendental.
- Most sequences of 0's and 1's are not computable.

But the NP vs P problem is different.

- Satisfiability for most wffs can be decided in linear time. Only a very thin set of wffs require exponential time, even with existing algorithms.

This leads us to believe that it is very unlikely that diagonalization can be successfully applied to prove $P \neq NP$.

Even more devastating to our hopes is a result involving *oracle* computation.

Def. An oracle is a problem which a RAM machine is allowed to question, just like the ancient Greek oracle at Delphi. But unlike the oracle at Delphi, a RAM machine's oracle always gives unambiguous yes/no answers to problems. The process of questioning the oracle and getting a response counts as only a single step of the oracle computation.

Obviously, oracle computation is highly unrealistic. But the idea is that oracle computation allows one to understand the relative difficulty of problems.

For example, we might choose as an oracle an NP complete problem like Boolean satisfiability, and we might use this to investigate the comparative difficulty of solving problems involving *quantified Boolean formulas.* For example, we might ask whether for ALL values of some variables, a wff is satisfiable (has a truth assignment for the remaining variables making it evaluate to TRUE). Such problems are probably harder than NP problems because they are not solvable in polynomial time, even with the aid of an NP complete oracle, unless $NP =$ co-NP.

Diagonalization and the NP vs P Problem

In 1975, Baker, Gill and Solovay proved,

Theorem. There are oracles for which $NP = P$, and there are oracles for which $NP \neq P$.

The oracle for which $NP = P$ is relatively easy to understand: It can be any polynomial *space* (PSPACE) complete problem. Polynomial space problems are problems that take a polynomial amount of memory to solve. Because non-deterministic polynomial space computations can be done deterministically while still using only a polynomial amount of space ($NP = P$ for space as opposed to time complexity), $NP = P$ for this oracle.

The construction of an oracle for which $NP \neq P$ is considerably harder.

DIAGONALIZATION AND THE NP VS P PROBLEM

Here's why the Baker, Gill and Solovay theorem makes the outlook so dim for applying diagonalization to prove that $NP \neq P$.

MOST DIAGONALIZATION ARGUMENTS CARRY OVER VIRTUALLY WITHOUT CHANGE TO ORACLE COMPUTATION.

Thus, a purported diagonalization argument proving $NP \neq P$ would probably carry over to an oracle computation where $NP = P$.

Diagonalization and the NP vs P Problem

Of course, it is possible that a diagonalization argument might work if it relied very strongly on the *form* and *details* of an NP complete problem. But if you find such an argument and try to publish it, you had better explain exactly why the Baker, Gill and Solovay theorem does not apply to your argument, or no one will believe you.

6. A Strong Result

In 1987, Blum and Impagliazzo proved one of the few really strong results about the NP vs P problem,

Theorem. For *most* oracles, $NP \neq P$.

Thus the situation in regard to the $NP \neq P$ question resembles the situation circa 1930 for the almost everywhere convergence of Fourier series, or the situation circa 1900 for the Riemann hypothesis. In both cases, it was possible to give an extension of the problem and prove that the conjectures were true for the extended problems most of the time, but not for the original problems all of the time.

7. Conclusions

The NP vs P problem is VERY hard. Perhaps not as hard as many of the great unsolved problems of mathematics, like the Riemann hypothesis, which have been around for 150 years or more, but still very hard.

For nearly up-to-the-minute information about other possible techniques for solving the NP vs P question, consult Steven Cook's Millenium Prize Problem description listed in the references below. Also, pay particular attention to the formulation of the problem in terms of the language recognition problem, which I have skipped for the sake of simplicity.

8. References

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

T. Baker, J. Gill and R. Solovay, *Relativization of the* $P = ?\, NP$ *question.* SICOMP: SIAM Journal on Computing, 4, pages 431 – 442, 1975.

M. Blum and R. Impagliazzo. *Generic oracles and oracle classes.* In Ashok K. Chandra, editor, Proceedings of the 28th Annual Symposium on Foundations of Computer Science, pages 118 – 126, Los Angeles, CA, October, 1987. IEEE Computer Society Press.

References

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.

Stephen A. Cook and R. A. Reckhow, *The relative efficiency of propositional proof systems*. Journal of Symbolic Logic, 44, pages 35 – 50, 1979.

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.

M. Davis and H. Putnam, *A computing procedure for quantification theory*. Journal of the ACM, 1, pages 201 – 215, 1960.

M. Davis, G. Logemann and D. Loveland, *A machine program for theorem proving*. Communications of the ACM, 5, pages 394 – 397, 1962.

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

References

Allen Goldberg, Paul Purdom and Cynthia Brown, *Average time analysis of simplified Davis-Putnam procedures.* Information Processing Letters, 15, pages 72 – 75, 1982. *Corrigendum*, 16, page 213, 1983.

Armin Haken, *The intractability of resolution.* Theoretical Computer Science, 39, pages 297 – 308, 1985.

John E. Hopcroft and Jeffrey D. Ullman, *Introduction to Automata Theory, Languages and Computation.* Addison Wesley, Reading Mass., 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.

References

J. A. Robinson, *A machine oriented logic based on the resolution principle*. Journal of the ACM, 12, pages 23 – 41, 1965.

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