Computational Complexity Theory:

The P versus NP Problem

Melchior M. Philips
Under supervision of R. Nijboer

TTO HAVO, 2011-2012
Mathematics A and Physics
Lorentz Lyceum Arnhem, 15-02-2012

Table of Contents

Preface

Having heard of a group of 5 Dutch teenagers that solved a mathematical problem with their profile research, I was tempted to try and achieve a similar feat. Being fond of Mathematics, I used Google to search for an unsolved mathematical problem that I could analyze, and possibly solve. I stumbled upon a problem concerning computer science, of which I am also incredibly fond. It was called the P versus NP problem, and the Clay Institute of Mathematics would grant a sum of $1,000,000 to the first person to give viable proof to solve the problem. I read on, and was intrigued by the problem (and the enormous sum of money), so I decided to submit it as the subject of my profile research.

I have tried to comprehend the P versus NP problem to the best of my abilities, and I aimed to explain it as clearly as possible, so that people with an average understanding of math can understand the essence of the problem. This proved a pretty difficult task, since the problem is based upon levels of mathematics and certain very complex elements that I haven't seen before. I have had to educate myself on the matter, which took a substantial amount of my time. You try not to lose track of the problem along the way, but it is inevitable when you go into some bits in-depth. Nonetheless I believe that this profile research should be sufficiently comprehensible for the averagely-educated person, and coherent enough to be scientifically representable and, more importantly, correct.

I also wish to thank a few people:

- Mr. Nijboer, for supervising my profile research as I went along.
- My previous math teachers, Mr. Naughton & Mr. Guezen, for increasing my knowledge on the fields of mathematics enough to solve this problem.
- My good friend Joey van Hummel, for stimulating my interest in science and scientific research.
- Gerardo Adesso, Thomas Woolley and others, for taking their time to answer my survey about the P versus NP Problem.
- Stephen Cook, Leonid Levin, Brian Hayes, Vinay Deolalikar, Koji Kobayashi, William Gasarch, Walter Savitch, Neil Immerman and everyone who has tried to solve P versus NP for their research on the topic.

Introduction to P versus NP

The P versus NP Problem is a fundamental, unsolved problem in computer science. Many mathematicians believe it to be THE most fundamental problem in computer science. It is a Millennium Prize Problem, and the Clay Institute of Mathematics has promised $1.000.000 to the first person to provide solid proof of an answer. The problem was first introduced by Stephen Cook in his seminal paper "The complexity of theorem proving procedures", which was published in 1971. Computer Scientists and mathematicians alike have tried to solve to problem since, but unfortunately, to no avail. So what is the P versus NP Problem?

Informally, the problem asks whether problems that can be easily verified by a computer (NP) can also be quickly solved by a computer (P). P and NP are the names of so-called complexity classes. Complexity classes are used to index the difficulty of a certain problem.

P

Problems in complexity class P (P standing for Polynomial) can be assessed and solved quickly by a computer, using a relatively simple algorithm. An example of a problem in complexity class P would be a mathematical sum. i.e.: what is 5+7? Or what is the square root of 9? The answers are 12 and 3 respectively, and can easily be solved by any household calculator. Problems in complexity class P are "easy" and can be solved in a "short" timespan.

The formal definition for a problem in complexity class P is: *"A problem that can be solved by using a deterministic algorithm by a deterministic Turing machine in polynomial time."*
That's a lot of difficult terminology to be understood at once, so let's tackle everything one by one:

A **Deterministic Algorithm** is a predictable, step-by-step calculation. A problem solved by a deterministic algorithm is solved step-by-step, using only the presented input, in a deterministic timespan. Deterministic, as the name suggests, means that something can be determined. In the case of a time span, a deterministic time span means a relatively short time span, a time span that can be measured by for example a stopwatch. Simply put: a computation that isn't "slow". The best definition of a slow computation is probably Brian Hayes' 'Coffee-Break Criterion': "A computation is slow if it's not finished when you come back from a coffee break."[HAYB08]

A **Turing Machine** is a hypothetical machine that preforms computations or manipulations according to a set of rules, used to describe the workings of an algorithm. A Turing machine is deterministic if it can only compute using deterministic algorithms, as described above. The time it takes for a deterministic Turing machine to complete a calculation is called PTIME. An example of a deterministic Turing machine would be a computer. In some ways, the human brain is also a deterministic Turing machine.

Polynomial Time is the formal term for a "short" timespan. Polynomial time is deterministic, and it exists. Any timespan that can be measured or indicated is Polynomial.

So to recap, a problem falls under complexity class P if it can be solved easily (deterministic algorithm) by a computer (deterministic Turing machine) in a short timespan (polynomial time).

NP

Problems in complexity class NP (NP standing for Non-Polynomial) cannot be assessed and solved quickly by a computer. In a way, NP is the opposite of P, but there's some fundamental differences. NP problems are "hard". A deterministic Turning machine would take an incredibly long time to compute an NP Problem, and the time taken might even be unimaginable, it might literally take forever. That's why we have a different method of computing NP Problems: The Non-Deterministic Turing Machine. As the name suggests, the non-deterministic Turing machine is the opposite of a deterministic Turing machine. It uses nondeterministic algorithms to solve problems in polynomial time.

Nondeterministic Algorithms are the opposite of deterministic algorithms. They are unpredictable, and may use data that was not provided. Nondeterministic Algorithms may run several calculations at the same time, and assess an incredible amount of data at once. In addition, there is a theoretically infinite number of ways to run this algorithm, and at least one of them is able to run in polynomial time.

A **Nondeterministic Turing Machine** is a Turing machine that is able to compute Nondeterministic Algorithms, and can solve NP Problems in Polynomial time, whereas a deterministic Turing machine could only solve an NP problem in non-Polynomial time.

Non-Polynomial Time is a "long" timespan. It is a time span that would be illogical to measure for an experiment or calculation. It might take several days, years, lightyears, or might even refer to *forever.* Non-Polynomial time, either real or fictional, is incredibly, *incredibly* long when put alongside polynomial time, to such an extent that it would be impractical to calculate whatever it is you are trying to calculate

So you might say that NP is the exact opposite of P, however, there's a good reason we do not. This is because there is one important characteristic of NP that I didn't mention yet. The answer to every problem in complexity class NP can be verified by a deterministic Turing machine in polynomial time.

Imagine two identical 1000 piece jigsaw puzzles, being solved by two different computers. The first computer is deterministic, it takes a puzzle piece and analyzes it, checks if it fits with any other piece, and puts it down in the correct position. The machine takes about 2 hours to complete the puzzle. The second machine is nondeterministic. It looks at every single puzzle piece at the exact same time, and places all the pieces of the puzzle in the correct position at the same time. The machine takes about 5 seconds to solve the puzzle. However, the first computer, being presented the correct solution, is perfectly able to comprehend the solution, and can verify it. Every puzzle piece fits, and the image displayed is comprehensible, so the first computer can see it has been solved correctly at a glance.

Another example of an NP problem is the Subset Sum problem. An example of a subset sum problem is: “does a subset of the numbers {-6, 1, 5, 17, 26, 55} add up to 32?” A deterministic Turing machine would have to try every possible subset, and check if it adds up to 32. The number of possible subsets to check is quite large. There is a total number of 63 subsets that can be created. ($\binom{6}{1} + \binom{6}{2} + \binom{6}{3} + \binom{6}{4} + \binom{6}{5} + \binom{6}{6}$) The time it takes for a deterministic Turing machine would be n^t where *n* is the number of possible combinations and *t* is the amount of time needed to check one combination. If the computer would take 1 second to check one combination, it would take 32^1 = 32 seconds to complete the question. A nondeterministic Turing machine could solve the question in the same time that the deterministic Turing machine checks one combination; 1 second. The correct answer is yes, 1, 5 & 17 add up to 32. A deterministic Turing machine could easily solve this problem, because it can calculate 1 + 5 + 17 in the same amount of time that it would take to check one combination; 1 second.

P = NP?

The P versus NP problem is all about whether P is equal to NP. If P is equal to NP that would mean that a problem that can be verified quickly by a deterministic Turning machine, can also be solved quickly by a deterministic Turing machine.

So what is the significance of the problem? The answer to the problem could greatly affect how mathematicians and computer scientists think about computational technology. A lot of fundamental computational rules and workings are based on the assumption that P is not equal to NP. If P were to be proven equal to NP, polynomial hierarchy would be proven infinite.

Nobody knows exactly how to solve the P versus NP problem. There are several suggested methods, and problems spawning from the P versus NP problems are most likely solved first. Over the past few years, a lot of research has been done, and several of those P versus NP-related sub questions have been solved. Since these sub questions form the basis of the P versus NP problem, let’s have a look at those first, before moving on to solving the actual P versus NP problem.

$$\mathrm{P} = \mathrm{NP} \ \vee \mathrm{P} \neq \mathrm{NP}\ ?$$

Beyond P and NP

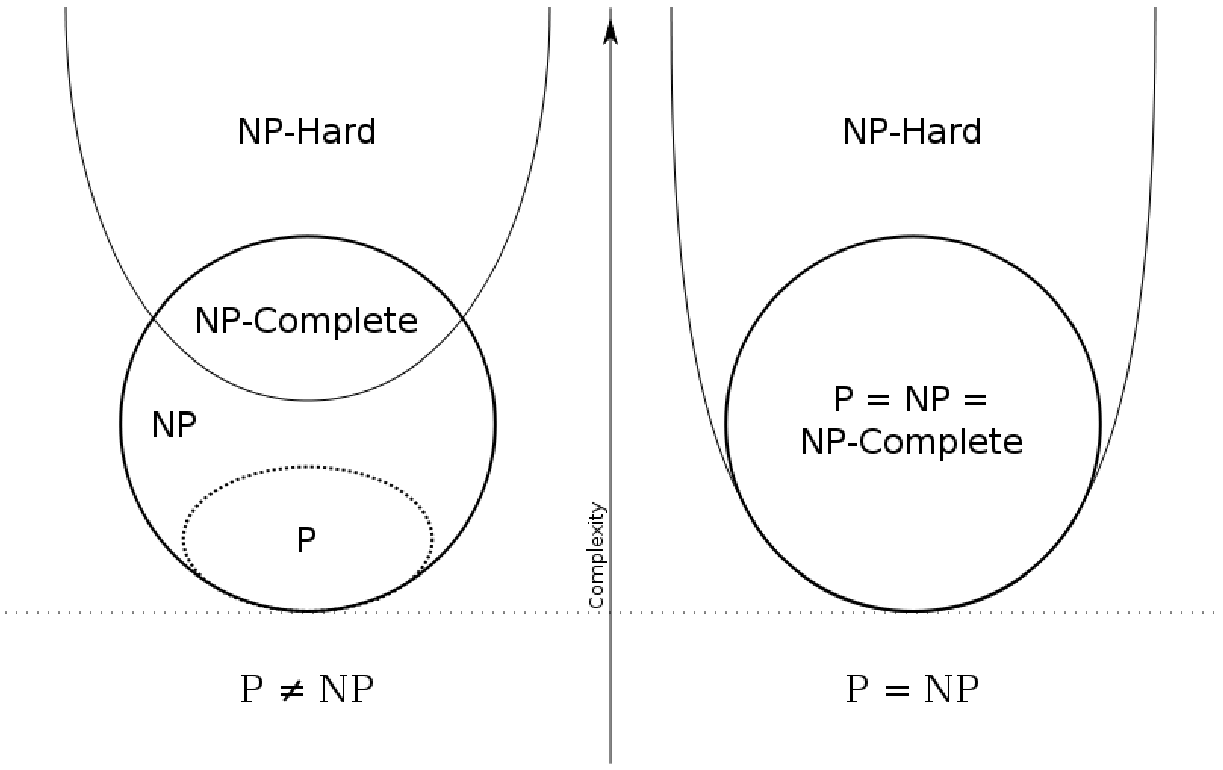
Since the P versus NP Problem's introduction in 1971, complexity theories have changed and have spawned new questions and categories. Problems aren't simply P or NP anymore; they have subcategories with each their own classification, name and specifics. So let's have a look at those first.

NPH & NPC

An example of such a subcategory is the group of **NP-Hard** problems, often abbreviated as **NPH**. All NP-Hard problems are *Decision Problems*. Decision problems are problems that have a simple yes or no decision as an answer. NP-Hard problems, despite their name, do not necessarily belong to the group op NP problems (and certainly not in P), meaning that some NP-Hard problems aren't known to be able to be solved in polynomial time. It is assumed that for these NP-Hard problems, which are not in NP, not a single path of the nondeterministic algorithm used to solve it is in polynomial time. Additionally, a problem is only NP-Hard if and only if every problem in NP is polynomial-time reducible to the decision problem.

Reduction is changing one problem into another known problem. When we encounter a new problem, and it is similar to a problem that has already been solved, we can use the answer to the already solved problem to solve the new one. In case of a **Turing Reduction**, two similar, unsolved problems are compared, and the new is solved by assuming an answer for the first problem. **Polynomial-time Reduction**, also known as a **Cook Reduction**, is a Turing reduction that can be operated by a deterministic Turing machine. In other words: a deterministic Turing machine is able to compare the new problem to an already known problem, and solve it by assuming an answer for the already known problem, and transforming it into instances of the new problem.

NP-Complete problems, often abbreviated as **NPC**, are problems that fall under both NP and NP-Hard. They are NP-Hard decision problems that can be solved using a nondeterministic algorithm in polynomial time.



The classifications of NPH and NPC are based on the assumption that P ≠ NP. As you can see in the image above, NPC is a subset of both the NP and the NPH groups. If P is equal to NP, then classes P, NP and NPC are the same, since all problems in NPC must be part of NP. NPH would be a class existing partly in the combined P/NP/NPC class, and partly of decision problems outside of the combined P/NP/NPC class.

PSPACE & NPSPACE

Problems in complexity class P also have their own subcategories, and one of them is called **PSPACE**. In the introduction I called the amount of time taken by a deterministic Turing machine to solve a deterministic algorithm PTIME. Likewise, PSPACE is a subset of problems in P where the amount of space required by a deterministic Turing machine to solve a deterministic algorithm is polynomial. Just like polynomial time, polynomial space is relatively small, and can be measured (i.e. by a tape measure).

Walter Savitch proved that PSPACE = NPSPACE in 1970. He proved that if a nondeterministic Turing machine can solve a problem using a function of a certain number of space, a deterministic Turing machine can solve the same problem in a square of that number of space.[SAVW70]

The hardest problems are known was PSPACE-Complete. These are problems which are believed to be the hardest in PSPACE, but do not belong to NP.

Other Classes

There is quite a large number of these complexity classes. They include L, NL PH, FL, FP, NFP, EXPTIME and EXPSPACE. They are all fascinating in their own way, but if I were to talk about them all this research would be enormous. I have chosen the complexity classes that are most closely related to the P versus NP problem, or have a similar unsolved problem with some significance towards the P versus NP Problem.

Opinions on P versus NP

Poll

William I. Gasarch of the University of Maryland conducted a poll about complexity theory in 2002, in which he questioned 100 mathematicians about the P versus NP problem.[GASW02] He asked when they thought it would be solved, what the answer would be, how it would be solved and whether they believed the answer to the problem would be easy to understand.

Inspired by Gasarch's poll, I set up a small survey of my own. I've contacted a few mathematicians and asked them about P versus NP. I asked them:

- Whether they knew what P versus NP was
- Whether they believed P is equal to NP
- When they believed the problem would be solved
- How the problem would be solved
- Whether the answer would be hard to follow
- What they believed is the significance of the problem

The people that I have contacted and that responded are:

- Gerardo Adesso MSc, Researcher at the School of Mathematical Sciences of the University of Nottingham
- Thomas E. Woolley MMATH, AHEA, DPhil Mathematician at the Centre of Mathematical Biology of the University of Oxford
- A professor in Computational and Analytical Mathematics who wishes to remain anonymous

That's not a lot of people to back up any claims, but the opinions presented were verily in line with the results of Gasarch's poll in 2002. All three interviewees believed that P and NP are not equal. They agreed that the P versus NP problem is significant, perhaps THE most significant problem in computer science, backed up by the fact that it is a Millennium Prize Problem.

Furthermore, Scott Aaronson, Associate Professor of Electrical Engineering and Computer Science provided his opinion regarding the poll held by Gasarch on his personal blog on June 25th 2011.[AARS11]

Solving P Problems

Problems belonging to complexity class P are informally the easiest, so it would make sense to start with those. I have highlighted a few problems that are known to belong into P and I have added an in-depth explanation of how to solve it, why it is in the complexity class it is and proof of the solution of the problem, both theoretical and mathematical. For solving these P problems, and the NP problems in the next chapter, we will be using two deterministic Turing machines:

- The Human Brain
- A Computer

Arithmetic

Simple mathematical sums, such as additions, subtractions, divisions or multiplications, are the most basic problems in complexity class P. They are easy to resolve and in most cases only require a single step. They can be solved using a deterministic algorithm by a deterministic Turing machine in polynomial time. Small additions and subtractions such as 16-1 take about a second to be processed into a human brain. Slightly larger ones such as 66+15 might take about 5 seconds. More complicated divisions and multiplications take a significantly larger amount of time to process with a human brain. A randomly generated sum, 9086 x 192, took me 173.62 seconds to solve, with the aid of a piece of paper. When such sums are inserted into a computer or a calculator, it is able to produce the correct answer in a fraction of a second.

Arithmetic (including, but not limited to: additions, subtractions, divisions and multiplications) can be solved using a deterministic algorithm by a deterministic Turing machine in polynomial time, and therefore, it is in complexity class P.

Solving NP Problems

Problems belonging to complexity class NP are significantly more difficult and more interesting than the problems belonging to complexity class P.

Sudoku

Sudoku's are Japanese puzzles which have gained massive popularity in the past years. A Sudoku is a grid of 9x9 squares, which is divided into 9 3x3 grids. The point of the game is that with 17 clues or more, you have to fill in the 9x9 grid in such a way that:

- In every row, the numbers 1-9 appear only once
- In every column, the numbers 1-9 appear only once
- In every 3x3 grid, the numbers 1-9 appear only once

While meant for recreation, Sudoku puzzles have proven to be extremely difficult. They may take the average adult anything between half an hour and a day to solve one, or whatever timespan is needed to force one to quit trying to solve the puzzle as a result of rage resulting from the inability to solve one.

However, Sudoku's can be solved by humans. Recently, some computer programs have been written that can solve Sudoku's, so computers are able to solve Sudoku's in polynomial time aswell. However, it can only be done using a nondeterministic algorithm; the set of rules the Turing machine has to follow is dependent on the data it generates along the way. However, a deterministic Turing machine can easily and quickly check that the solved Sudoku is correct, because the numbers 1-9 only appear once in every 3x3 grid, row and column. Thusly, Sudoku's belong to class NP.

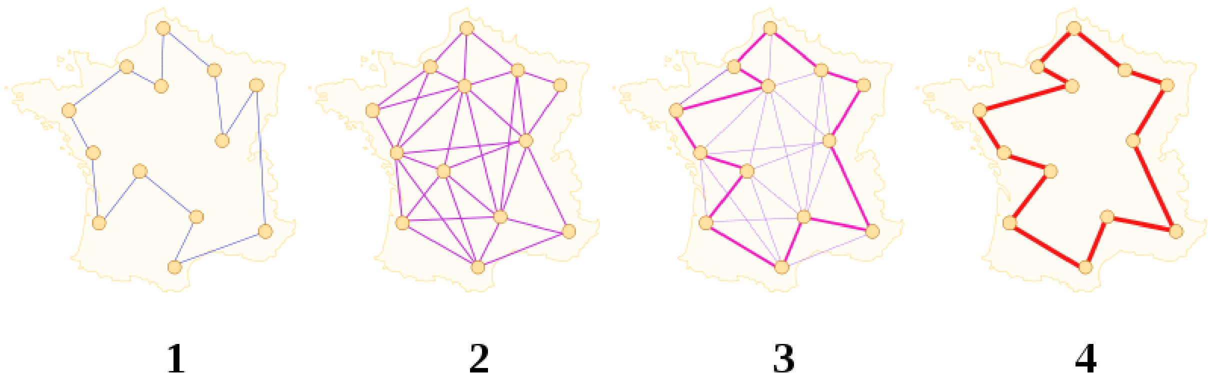
Minesweeper

Minesweeper is a computer game where the player must click random squares, which reveal a blank space, a number, or a mine. The player must then figure out where the mines are located, and click every square on the grid apart from the mines. The game is pretty popular among masses, and perhaps the most well-known iteration of the minesweeper game is the one created for OS/2 by Curt Johnson in 1990, which was included with Windows XP.

There is no real predefined algorithm to win minesweeper. You make it up as you go along. Since you haven't a clue where the mines will be placed before starting the game, a continuous flow of data is required to be inputted and generated in order to solve the computer. Because there is not one simple algorithm for minesweeper, it can only be solved in polynomial time by a nondeterministic Turing machine using a nondeterministic algorithm. However, a deterministic Turing machine can very easily see that the game of minesweeper has been successfully won, for all clicked blocks do not contain a mine, and all the remaining unclicked blocks contain mines. Because of this, Minesweeper is an NP-Complete problem.

Travelling Salesman

The Travelling Salesman problem, also known as the travelling businessman problem, is probably the sole embodiment of the NP-Complete set of decision problems. As such, it is usually the target for people trying to solve the P versus NP problem. Given a list of cities to visit, and their relative distances, the objective is to find the shortest possible route through all the cities, without visiting any city twice.



Pictured above is an example of a travelling salesman problem in Germany. 13 cities are to be visited. To calculate the shortest possible route, first, the outline has been drawn, connecting every city with each other (1). Next, a so-called Hamiltonian cycle was drawn. (2) In a Hamiltonian cycle, every dot in a planar graph is connected with each other. Next, the shortest possible route was calculated by selecting all the shortest possible connections in between cities. (3) Finally, the other routes are removed, so that only the shortest possible route is left. (4)

While seemingly easy, every single route in the complete Hamiltonian cycle has to be check to figure out which are the shortest, and in what cases the shortest distances can be utilized. The total number of possible routes is enormous, and it could take a human and even a computer an incredibly long time to compute the shortest possible route. The constant flow of selected routes has to be added to the data accumulated for the algorithm used, and thusly, it is a nondeterministic algorithm.

However, given the shortest possible route, a deterministic Turing machine can very easily verify that there is no other, shorter route for travelling through every city. Therefore, it has every characteristic of an NP-Complete problem. Therefore, it is NP-Complete.

Solving P versus NP

The question on the minds of a lot of mathematicians, physicists, engineers and computer scientists remains: is P equal to NP? Many have attempted to write comprehensible and correct proof of either outcome, and many have failed. The problem has piqued the interest of many an amateur mathematician (such as myself), and as a result, a lot of faulty, incomprehensible, incoherent or simply gibberish proof papers have been written. The Technical University of Eindhoven has made it their hobby to collect papers proving or disproving that P is equal to NP. There is a list totaling 85 papers on the subject. 41 conclude P to be equal to NP, 37 conclude P and NP to be unequal, and the remaining 5 suggest that the P versus NP problem cannot be proved without having a contradiction at some point.

a. Deduction
b. Algebra
c. Combinatorics
d. Other Techniques
e. New Technologies

Deduction

Deduction is a logical way of approaching a problem. The P versus NP problem has been approached with deduction in attempts to solve it over the past few years. Ted Swart of the University of Guelph wrote a few papers concerning the P versus NP Problem in 1986. He believed that P is equal to NP, since the Hamiltonian cycle, an NP-Hard problem, is solvable in polynomial time. With this knowledge he deduced that P must be equal to NP.[SWAT86] Seenil Gram also used deduction to solve P versus NP, but his conclusion was somewhat different. In his paper "*Redundancy, Obscurity, Self-Containment & Independence*" he first proves the so-called "Indistinguishability Lemma" and then proceeds to deduce that EXP must be in NP, concluding that P and NP are unequal.[GRAS01] Another person to rely on deduction for his evidence is Nicholas Argall. He explained in a short piece that P and NP cannot be defined consistently and completely, as supported by Goedel's Theorem. Argall concluded that since the question cannot be formulated successfully, it cannot be answered successfully.[ARGN03] In 2008, Rafee Ebrahim Kamouna proved that SAT is not NP-Complete. From this he deduces that there are no NP-Complete problems, concluding P=NP.[KAMR08]

Algebra

Algebra has also been used to try and solve the P versus NP Problem. The best example of this would be the paper '*Linear Algebra, Lie Algebra and their applications to P versus NP'* written by Ki-Bong Nam, S.H. Wang and Yang Gon Kim in 2004. In this paper, algebra is used to point out a fundamental counting error which purports to be a counterexample of P=NP, resulting in the conclusion that P is not equal to NP.[NAMK04] Richard K. Molnar has reviewed this paper in AMS Mathematical Reviews. Molnar explains that the crux of the proof is that the reader is asserted that the complex and hard-to-follow calculations are not computable in polynomial time, and that they rely on random data that wasn't predefined. Molnar is unimpressed by the paper due to the large number of unclear assumptions and faulty conclusions.

Matt Groff also used Algebra to try and solve P versus NP. However, his conclusion was that P is equal to NP. He established his answer through a time algorithm for the satisfiability problem (a problem in class NP) in which he used linear algebra. As a result he showed that P is equal to NP.[GROM11]

Combinatorics

Combinatorics is a field of mathematics concerning the study of countable discrete structures. Informally: it's the study of things you can count, and their maximal, minimal and optimal values. Being a broad field of mathematics, a lot of combinatorialists have used combinatorics to solve the P versus NP problem, yielding different results.

In response to Swart's paper, Mihalis Yannakakis wrote his paper "*Expressing combinatorial optimization problems by linear programs*" in 1988, in which he showed that solving the Travelling Salesman problem by using symmetrical lines (like Swart did) does not require exponential space (NPSPACE), concluding that P and NP are equal.[YANM88] In 2010, Sergey Gubin once again proved that the Travelling Salesman problem can be solved in polynomial space[GUBS10], but his proof was later refuted by Romeo Rizzi in 2011. Gubin had previously proved that P is equal to NP in 2006.[GUBS06]

Viktor Ivanov[IVAV05] and Lev Gordeev[GORL05] both proved P and NP to be unequal in their independent papers, both published in the summer of 2005, and both using combinatorics. Ivanov rewrote his paper in 2010 to reflect new advancements made in the P versus NP matter. Gordeev is still trying to perfect his paper to make his evidence foolproof, and thusly, his paper is still a draft version.

Other Techniques

Other, more experienced mathematicians that have attempted to solve P versus NP have relied on more advanced and infinitely more complex techniques of verification that I couldn't possibly begin to explain. Ehrenfeucht-Fraïssé Games, Boolean Circuit Lower Bounds, Finite Model Theory, Paris-Harrington Theorem; they are all examples of complex mathematical techniques that have yielded a wide variety of results concerning the P versus NP Problem.

New Technologies

A lot of mathematicians believe that the P versus NP problem cannot be solved using the mathematical techniques we possess now. As is evident in Gasarch's poll, 18 people out of the 100 surveyed suggested that an entirely new technique could have to be invented in order to solve the P versus NP Problem. Argall quite elaborately explained that we do not have the techniques to properly define the P versus NP problem, let alone to solve it. Therefore, new techniques must be constructed before new discoveries regarding the P versus NP problem can be made. However, this does not stop amateur and advanced mathematicians to try and resolve the problem anyway.

Is P equal to NP?

Whether P is actually equal to NP is still unknown. A lot of proof and evidence has surfaced, but nothing conclusive as of yet. However, most evidence found leads many to believe the P is not equal to NP. Most mathematicians interviewed by Gasarch in 2002 believed it P is not equal to NP and the mathematicians I personally surveyed also believed P and NP are unequal. Most methods in the previous chapter have yielded inconclusive proof that P in fact, does equal NP, though the acceptability of these proofs is questionable for they are mostly written by amateurs.

I have done tons of research into the matter, both trying to comprehend and solve it, and I have been led to believe that P versus NP might never be solved. I am incredibly certain that the current techniques will not solve P versus NP. I think that the computers we possess will become much and much more advanced, expanding the abilities of the deterministic Turing machine as we know it, up to a point where the P versus NP problem will become irrelevant. The way the question is asked is vague, and the definitions of its elements are hard to understand. Alleged proofs are either refuted, or infinitely complex. The question shouldn't be "is P equal to NP?" but rather "Will humans be able to solve the P versus NP problem before they are wiped out of existence by cyborgs with a superior intellect?" and if not, will the cyborgs be able to solve P versus NP?

Humans currently possess an enormous but finite amount of knowledge. Because of its finiteness, the answer whether P is equal to NP is outside of the grasps of our imagination. I am sure some advanced will be made, and I am without a doubt that people will not cease to try and solve this problem until it is resolved, but currently, with our level or knowledge and technology, the P versus NP problem is impossible to properly define or solve without a contradiction somewhere along the line.

So whether P is really equal to NP, we do not know. It is impossible to say at this point in time.

∎

Sources and Consulted Literature

- [GASW02] *The P=?NP Poll*, William I. Gasarch, 2002. Department of Computer Sciences, University of Maryland.
- [AARS11] *My responses to GASARCH's P vs. NP poll*, Scott Aaronson, 2011. www.scottaaronson.com
- [HAYB08] *Accidental Algorithms,* Brian Hayes, 2008. Computing Science issue January-February 2008, pp. 9-13.
- [SAVW70] *NPSPACE is contained in DSPACE*, Walter Savich, 1970.
- [SWAT86] *P=NP,* Ted Swart, 1986/87. Journal of Computer and System Sciences 43, 1991, pp. 441-466.
- [GRAS01] *Redundancy, Obscurity, Self-Containment & Independence*, Seenil Gram, 2001.
- [ARGN03] *P=NP – An Impossible Question*, Nicholas Argall, 2003. http://www.win.tue.nl/~gwoegi/P-versus-NP/argall.txt
- [KAMR08] *The Kleene-Rosser Paradox, The Liar's Paradox & A Fuzzy Logic Programming Paradox Imply SAT is (NOT) NP-complete*, Rafee Ebrahim Kamouna, 2008. arXiv:0806.2947v8
- [GROM11] *Towards P = NP via k-SAT: A k-SAT Algorithm Using Linear Algebra on Finite Fields*, Matt Groff, 2011. arXiv:1106.0683v2
- [YANM88] *Expressing combinatorial optimization problems by linear programs*, Mihalis Yannakakis, 1988.
- [IVAV05] *A PROOF OF NP ≠ P*, Viktor V. Ivanov, 2005.
- [GORL05] *Combinatorial sentence that infers P < NP*, Lev Gordeev, 2005.
- [GUBS10] *Complementary to Yannakakis' theorem*, Sergey Gubin, 2010.
- [GUBS06] *A Polynomial Time Algorithm for The Traveling Salesman Problem*, Sergey Gubin, 2006.