DOES NP = P? A SHORT BUT COMPREHENSIVE PROOF (Original Copy)

Preprint · April 2020

CITATIONS
0

READS
25

1 author:

Jamell Ivan Samuels
N/A
21 PUBLICATIONS **0** CITATIONS

SEE PROFILE

DOES NP = P?

A SHORT BUT COMPREHENSIVE PROOF

By

JAMELL IVAN SAMUELS

IMPERIAL COLLEGE LONDON
Department of Aeronautical Engineering

APRIL 2020

ACKNOWLEDGMENT

All glory goes to God first and foremost, but this couldn't have been done without the love and dedication of my Father. Thank you to both my Mothers for the hand they gave in raising me, love to my brothers and sister. A special mention to my cousin Jamil for whom without I wouldn't even be sitting here writing this thesis and the list goes on Mayur, Wale, Aaron, Andy, Issac, Jaron and Vera have all been instrumental in my life and have been with me through some of my darkest hours, I do this for them.

Abstract

The question does NP = P has confounded mathematicians and computer scientists alike for over 50 years and although there is an almost unanimous agreement that it in fact does not, there still is no definitive proof. In this paper, I will attempt to definitively prove to the point of absurdity that NP does not in fact equal P, by establishing the simplest exponential problem and proving it impossible to be done in less then exponential time.

TABLE OF CONTENTS

LIST OF FIGURES

Chapter One

Introduction

1.1 Historical Background

The question of whether NP = P was first established in 1971 by Stephen Cook, although the problem had been discussed as early as the 1950's by John Forbes Nash Jr in a series of letters to the National Security Council and independently by Kurt Gödel to John Von Neumann. The potential ramifications of a solution were understood even then, with proof of it being true leading to the possibility of the automation of mathematical proofs and a solution disproving the statement leaving us to all going home to focus on quantum computers.

1.2 Description of the Problem

The P =? NP problem can best be described as a mathematical statement concerning the very nature of the algorithm. Can every problem, that can be verified in polynomial time, be solved in polynomial time? Now the question is posed this way so that it can be properly understood, as a large part of whether something is true or not is based upon how the question or statement is posed other than the universality of the statement. So the question is posed as stupidly as possible to ensure that we get to the root of the problem, because really it's just a funny question.

Chapter Two
Counting and Totalling

To establish the problem we have to first introduce the two most basic operations in mathematics, counting and totalling. Counting and Totalling inhabit a region I call the Method Space, which is an area used to categorise all the operations known to man, however they can all be reduced to counting and totalling.

- Counting is the act of taking account of a variable or value in a series or group. In the Method Space counting can be denoted as $M(n, i)$
- Totalling is the summation of the variables or values you have counted. In the Method Space totaling can be denoted as $M(n + i, i)$
- M represents Method - which is any process or operation which is used to solve an algorithm or problem. The Method Space is denoted as M(current operation, next variable)
- n is the current number
- i is the next number in the group or series

We must now establish the limits of counting and totaling from 1 to ∞. In this paper we will not consider any number that is not a real integer as computational steps are measured from i = 1 onwards. However the principals of this derivation still hold for negative and non-integer numbers.

2.1 Limits of Counting and Totalling

The Limit of Counting as i tends to 0 is

$$M(n,i)\lim \xrightarrow{i=0} M(n,0) \tag{2.1}$$

The Limit of Totalling as i tends to 0 is

$$M(n+i_i,i_{i+1})\lim \xrightarrow{i=0} M(n,0) \tag{2.2}$$

Which in laymans' terms states that you can't have counted any less then n and you must have totalled at least n.
The limit of Counting as i tends to infinity is

$$M(n,i)\lim \xrightarrow{i=\infty} M(n,\infty) \tag{2.3}$$

The Limit of Totalling as i tends to infinity is

$$M(n+i_i,i_{i+1})\lim \xrightarrow{i=0} M(\infty,\infty) \tag{2.4}$$

Which in layman's terms states that you must have counted at least infinity and you must have totalled at least infinity.

We are now going to establish the difference between counting and totaling, using Newtonian differentiation.

$$\frac{d\Delta T}{d\Delta C} = \frac{M(\infty,\infty)-M(n,0)}{M(n,\infty)-M(n,0)} = \frac{M(\infty,1)}{M(0,1)} \equiv \frac{(\infty,1)}{(0,1)} \tag{2.5}$$

Which is clearly not a calculable number, but shows that the change in number between counting and totalling is so great that the two are clearly distinct operations. Therefore counting $\neq$ totalling, but like how 0 is contained within infinity, counting is contained in totalling, however totalling is not contained in counting. We can therefore state that there is no function method or operation that will link counting to totalling. By taking the modulus of the top and bottom you get,

$$\frac{|(\infty,1)|}{|(0,1)|} = \frac{\infty}{1} \tag{2.6}$$

Which proves that 0 is not a countable number.

2.2 Steps & Time

In this paper we are going to use the simplest computer available, a Deterministic Turing Machine, where only one action is permitted for a given situation and a function is just a sequence of actions for a given scenario. In the scenarios described our actions will be to count and total with each count and each new addition in a total taking a step (i).

In computational complexity the time taken to perform a given function is proportional to the number of steps required to complete that function. For simplicity's sake we calculate the time taken to perform each step as a function of the operation. i.e if we had to square x on step i we would say the time taken to perform step i was t^2. The steps required to perform any method can be considered a permutation of a Brute Force Step algorithm, that is if you could randomise the steps taken by Brute Force you would eventually create all the algorithms currently known and more. For example in the case of Sudoku regardless of what the machine is doing there are only 9 steps it can take.

Chapter Three
Checking and Solving

The question does NP = P, asks us if a problem that can be checked in polynomial time, can also be solved in polynomial time. For this section we have to properly define what checking and solving are.

- Checking is the process of where you assure that the solution you have is valid.
- Solving is the method used to gain that solution.

Your best solving method can not run faster than your best checking method. Therefore the limit of solving is checking.

$$Solving \lim \xrightarrow{Method} Checking \tag{3.1}$$

Whether a problem can be reduced depends on the relationship between its checking method C and its solving method S. All methods in some form are composed of their check. Problems composed purely of their check, or which can be factorised for their check are capable of being run at faster computational times.

You can define the relationship between checking and solving in the Method Space as:

$$M[C, S] \tag{3.2}$$

If a problem can be reduced it can be described as

$$M[C, S[K, C]] \tag{3.3}$$

Where K is an arbitrary part of the solution algorithm. It can then be reduced to

$$M[C, C * S[K, C]] \tag{3.4}$$

Where the check multiplier of C can be eliminated leaving a new algorithm

$$M[C, S[K, C]]$$

Chapter Four
Polynomial Time Problems

For a polynomial time problem such as solving for the highest common factor of a and b, all steps i required to calculate the result R are dependent on one scale, which is the value of the numbers v and if we were to imagine a random brute force algorithm made up of all possible actions we could write.

$$v = v(a, b) \tag{4.1}$$

All the numbers contained from 1 to a or b are.

$$a = [a_0, a_1, ..a_{a-1}]$$

$$b = [b_0, b_1, ..b_{b-1}]$$

The probability that you get the number right is.

$$P \sim (\frac{1}{a+b})$$

$$R = (\frac{1}{a+b})^{a+b}$$

The length n of the calculation is

$$a + b = n$$

Where $\frac{1}{a+b}$ would have the same length n.

$$R = n^n \tag{4.2}$$

$$C = f(n)$$

$$C = n^2$$

Where C is of the order of your typical polynomial algorithm (gcd algorithm/non-binary variant)

$$R = n^2(n)^{n-2} \tag{4.3}$$

$$R = C(n)^{n-2}$$

This algorithm like all polynomial algorithms can be reduced in computational time, in this case from $O(n)^n$ to $O(n)^2$ as long as $n \geq 3$. And we can therefore conclude that out of all the random possibilities available it is possible to solve the highest common factor problem in $O(n^2)$ time even if we assumed we could only check with the gcd method.

Chapter Five

Non-Polynomial Time Problems

Non-polynomial problems are a lot more complex and therefore require a bit more deriving. Taking Sudoku as an example, Sudoku has two scales to consider, value and order $[v, o]$. In a random brute force search the probability that you get the value correct is $\frac{1}{9}$. The value and the order despite being on distinct scales are linked by probability so that the probability of getting the order correct is 1 given that the value is also correct.

We are going to begin by imagining a problem whereby the probability of getting the value and order $[v, o]$ correct are $[1, 1]$. This problem has a length n and we are going to derive what a random Brute Force algorithm would do in this given scenario.

$$R = (1+1)^n \tag{5.1}$$

$$R = 2^n$$

$$R = e^{n(ln2)}$$

$$nln(2) = u$$

$$R = e^u$$

The Taylor Expansion of e^u is.

$$R = 1 + u + \frac{u^2}{2} + \frac{u^3}{6} + ...$$

$$C = f(n) \tag{5.2}$$

$$C = n^2$$

Which as a power series contains the Check, but can not be factored for the Check.
The standard exponential is given to the order of $\mathrm{O}(2^{n_2})ln(R) = n^2 ln(2)$

$$\frac{1}{2}ln(R) = \frac{1}{2}nln(2)$$

$$R^{\frac{1}{2}} = e^{\frac{1}{2}}nln(2)$$

$$R = e^{nln(2)}$$

Like the previous example this expression can not be factored for the Check
This trend continues for all powers of 2^{nn}.

$$R = 2^{n^n} \tag{5.3}$$

$$ln(R) = n^n ln(2)$$
$$\frac{1}{n} ln(R) = \frac{1}{n} n ln(2)$$
$$\frac{1}{n} ln(R) = \frac{l}{n}(2)$$
$$ln(R) = n ln(2)$$
$$R = e^{nln(2)}$$

Therefore we can say $2^n = 2^{nn}$, as any value counted by n^n will eventually be counted by n. What is to note is that if we are to consider integers only than you can not count n^n and say you have counted n. Which is why I pose a new question and ask if you could solve linear equations using a polynomial.

5.1 Sudoku

3		6	5		8	4		
5	2							
	8	7					3	1
		3		1			8	
9			8	6	3			5
	5			9		6		
1	3					2	5	
							7	4
		5	2		6	3		

Figure 5.1 Sudoku Grid

Taking the more specific example of Sudoku we have.

$$P \sim [v, O] \tag{5.4}$$

And in our particular case.

$$P \sim [\frac{1}{9}, 1]$$

As a quick aside whichever way around you do this the result is still the same i.e if you were to say $o = \frac{1}{9}$, then $v = 1$)
Which leads on as before to.

$$C = f(n) \tag{5.5}$$

A standard Sudoku Check is the time take to count and total all the variables in the grid, so for a grid of size n^2 the time take to check would be $2n^2$.

$$C = 2n^2$$

$$R = f(v, O)$$

The probability of finding a solution for n squares is therefore.

$$R = (1 + v)^n$$

Which if we were to substitute our values in would be.

$$R = (1 + \frac{1}{9})^{81}$$

A Binomial expansion of the general expression would be would be.

$$R = \binom{n}{0}v^0 + \binom{n-1}{1}v^1 + \binom{n-2}{2}v^2 ... + \binom{0}{n}v^n \tag{5.6}$$

Which as a power series can not be factored for C but may contain C.

$$R\ isnot \sim C$$

We state that it can not be factored for C as it can not be done without introducing negative exponents. Negative exponents in computational terms are nonsense as you can not have a negative step, as we can only count from 1 to ∞. Therefore we can now state that out of all the random possibilities that are available to us. None of them can solve Sudoku in polynomial time.
We are now going to cover the only time a power series and therefore an exponential problem can be factored to remove the Check.

The difference between non-polynomial problems and polynomial problems can best be visualised as in the image below.

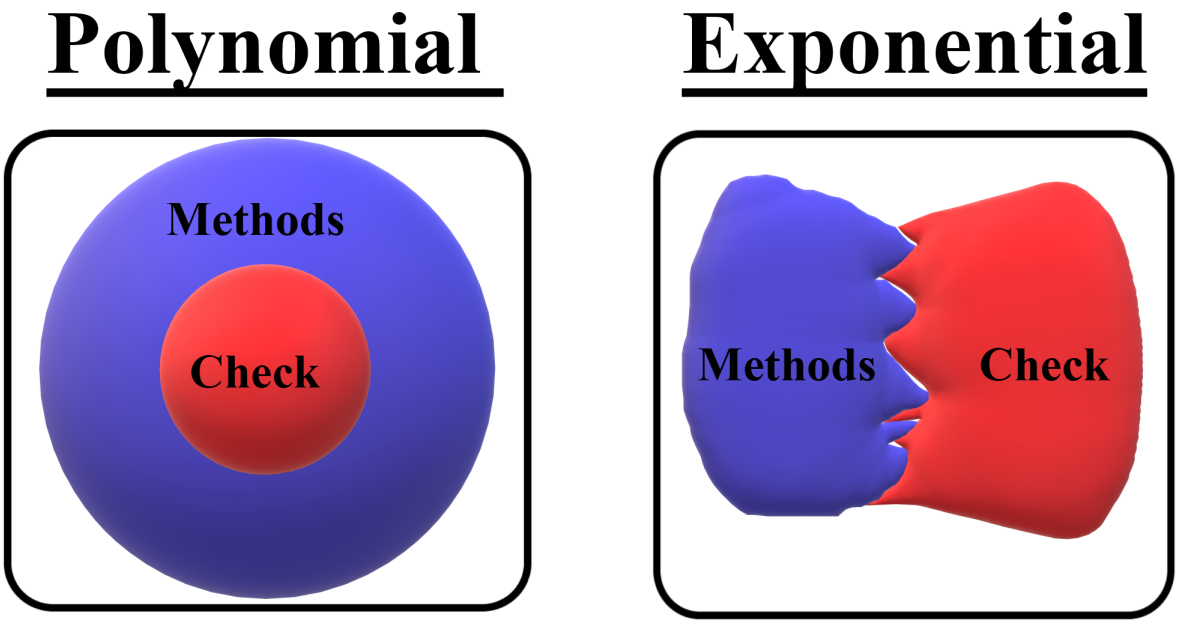

Figure 5.2 Method Space for polynomial and non-polynomial problems

- When reducing an exponential problem, the space that includes the Check is not contained within the solution.
- When reducing a polynomial problem the space that includes the Check is always contained in the solution.

Although Binomial Expansions are technically polynomials you can not create an equivalent polynomial that is not a Binomial Expansion. And as all exponentials can be expressed as an expansion of some kind, there is no way to solve exponential problems as polynomials as fundamentally the probability of something happening can not exceed 1.

Chapter Six

A Simple Exponential Problem

Although we have essentially proved why NP can not be P in the previous chapter, in this chapter we are going to go into a little further detail by establishing a simple exponential problem. Imagine you were asked to calculate the $i^{th} value of e^x$ in a Taylor Series that produced a given sum.

We have already established counting and totalling in the first chapter and it's good to remember that you must keep count whilst also doing your total.

Your count would look like this:

$$e^{x_i} = [1, x, \frac{x^2}{2}, ...] \tag{6.1}$$

Your total would look like this

$$e^{x_T} = [1, 1 + x, 1 + x + \frac{x^2}{2}, ...] \tag{6.2}$$

Your check would be the given value and you would count and total until you reached your check which could be expressed as e^x_{Ts}. From what we have already established about counting and totalling, although it is possible to reach the value of e^x_{Ts} in fewer steps then the summation of e^x_i. It is impossible to do so whilst keeping a full count of e^x_i. Therefore it is impossible for NP = P off of one algorithm alone.

Furthermore, any problem that can be expressed as $(1 + x)^n$ can not be solved in P time and this is evident even in Pascals Triangle. For example a problem composed purely of it's check would still would not be able to be solved in P time.

$$(1 + C)^n = 1 + C + \frac{C^2}{2} + \frac{C^3}{6} + ... \tag{6.3}$$

$$C^0 + C^1 + \frac{C^2}{2} + \frac{C^3}{6} + ...$$

As you are still unable to factorise for the check without introducing negative exponents.

6.1 Conclusion

The question of whether P = NP is actually a question of the universe, as you do not say that theorem X offers better results than the observed phenomena and although intuitively it is untrue our mind always holds on to the hope that perhaps it maybe. It has been and always will be a bit of a prank and I would not be surprised if the fellows at the Clay Mathematics Institute realised this. Although it is one of the defining questions of our generation a lot of work must be done to ensure that these types of fallacies are not so easily fallen for, especially by a large segment of the mathematics community, in short NP does not equal P and it is a bit of a foolish question to ask, although it is really funny.

Chapter Seven

Some Ideas that I liked

Whilst searching for a method to prove that NP does = P I came across a few interesting ideas, although most of them have been spoken about before.
The first of these is to count and total at the same time which is essentially what quantum computers do. By being able to store two variables as one you can effectively count and total at the same time.

count value	count binary	total binary	total value
1	1	0	0
1	0	1	1
2	1	1	2

From this example a few things are clear, given that binary is our method of choice and we only have one q bit we can't count and total past 2 without introducing a second bit. The second is that if a q bit has a rest state of $[1, 0]$ we can't solve as fast as our check so the limit of solving is still checking. Therefore, for a problem such as Sudoku the limit for how fast you would be able to solve it would be $2n^2$ time.

Another method for solving an exponential in polynomial time would be to use two separate algorithms to solve parts of the problem. They would both have to themselves run in polynomial time and we could either have them run in succession of each other or in tandem, although running them in tandem would be a bit tricky as it couldn't total what the other one hasn't counted so it would have to be designed so that one algorithm ran slightly ahead of the other.

An experimental design based off this idea and what we had derived about counting and totaling was the multi-voltage processor. This would differ from the standard multi-voltage processor in that we would be using the voltage difference to calculate. This would be done by using a lower voltage to count and a higher voltage to total. They would be running different algorithms as mentioned above and the machine would pass the instructions to the higher voltage which would then initiate the count in the lower voltage. The algorithm running on the higher voltage would measure the voltage change in the lower processor to total what the lower voltage counted through subtle voltage changes.

The last of these is to use a Chaotic System to solve problems like Sudoku. The system I intend to use is the Rössler System as it can be scaled to both 2D and 3D due to two of

it's equations being linear. A typical Rössler System looks like this.

$$X = [x, y, z] \tag{7.1}$$

$$\dot{X} = [\dot{x}, \dot{y}, \dot{z}]$$

$$\frac{x}{t} = y - z$$

$$\frac{y}{t} = x + ay$$

$$\frac{z}{t} = b + z(x - c)$$

With parameters a,b,c being the strange attractors. Which typically have values of 0.1, 0.1 and 14.

The system I imagined would look like this.

$$X = [v, o, \frac{v}{o}] \tag{7.2}$$

$$\dot{X} = [\dot{v}, \dot{o}, \frac{\dot{v}}{o}]$$

$$\frac{v}{t} = o - \frac{v}{o}$$

$$\frac{o}{t} = v + ao$$

$$\frac{\frac{v}{o}}{t} = b + \frac{v}{o}(v - c)$$

The strange attractors would be calculated from the known values in the grid.

- (a) would be based upon the average values in the row
- (b) being based on the order you can't do
- (c) would based on the order you can do

Calculations would be run either a row at a time or for a single value then a,b and c would be recalculated. A solution is found when $\frac{v}{o} = 1$. This system is yet to be tested, although I'm getting sick of looking at NP problems and might try the Riemann Hypothesis.