Proof $P \neq NP$ Using a New Method of Circuit Analysis on 3SAT

Michael Marquard

Independent Researcher, Aston, Pennsylvania, USA.

Contributing authors: MikeMAcademic@outlook.com;

Abstract

The following paper shows that some 3SAT (3-Satisfiability) problems containing only one unique solution cannot be solved in polynomial time. To do so a new technique I call "boolean isolation" is used to create an equation defining each variable in the 3SAT problem thus creating a system of equations. From there circuit analysis and other methods are used to show that any possible equation to find the values in the 3SAT problem would have to be equivalent to the equation created by boolean isolation allowing minimum bounds to be established. This paper will also discuss the possible impacts of these findings on other fields including cryptography.

Keywords: P Versus NP, Computational Complexity, SAT, Polynomial Time, Exponential Time, Cryptography

1 Introduction

It's been demonstrated through the Cook-Levin Theorem that all NP problems can be converted to 3SAT [1-2]. Since a complexity class is defined by the worst case time complexity of its members it is only necessary to prove some 3SAT problems require superpolynomial time to solve. What this paper will prove is that some 3SAT problems are impossible to solve in polynomial time thus proving that NP is a separate complexity class from P. However, as with other NP problems there are some subclasses of problems that can be solved in polynomial time and the method used in this proof may be of use in solving those 3SAT and other NP problems including those in cryptography. In addition the method used in this proof demonstrates clearly why 2SAT can always be solved in polynomial time but 3SAT can't.

The fundamental difficulty of proving that 3SAT cannot be solved in polynomial time is that it requires proving a negative. Specifically how do you prove that a polynomial algorithm cannot exist. The approach used here is to first create an equation for each variable in the 3SAT problem using a new method I call "boolean isolation". After this creates a system of Boolean equations and these equations are simplified minimum circuit complexity can be demonstrated. In some cases this will be solvable in polynomial time but in other cases it will be shown to be impossible. A key constraint to this method is that it can only be used on 3SAT problems that only have 1 satisfiable answer. These 3SAT problems with only 1 solution we will label as "U3SAT" (U being short for unique).

Throughout this paper we will use 1 to represent True and 0 to represent False. Boolean equations and circuits will be spoken of interchangeably since boolean equations are just a mathematical representation of digital circuits. Finally if the terms "boolean isolation" and "simple or complex equations" are used in any other branch of mathematics these definitions should be ignored as these terms have a specific meaning in the context of this paper which will be explained in depth.

2 The Algorithm

Before starting any known methods of simplifications can be done first. For example the pure literal rule [3] can be used if any variable appears in only one polarity. The examples used in this paper do not allow for this simplification so they will not be discussed any further.

2.1 Boolean Isolation

The first step of this algorithm is to create an equation that defines each variable. Unlike in regular algebra isolating a variable to one side of the equation is not as simple in Boolean algebra. It is however possible in U3SAT problems with the following method. I will first show why this works with small examples (that will have more than 1 satisfiable answer to minimize the size of the equations and make the process easier to understand) before demonstrating real examples in the Examples Section 3.

$$(A \vee B \vee C) = 1 \tag{1}$$

In this example the last 2 variables $(B \vee C)$ will be called the "remainder phrase". Since every phrase in a SAT problem must come out to True/1 to be satisfiable we can say that if $B = 0$ and $C = 0$ then $A = 1$. We can write the following equation:

$$A = \neg(B \vee C) \tag{2}$$

This equation will work when the remainder phrase equals zero and it will also work if the remainder phrase equals 1 assuming there is only one satisfiable answer to the 3SAT problem. I'll demonstrate the reason why at the end of this subsection.

For equations where the variable we want to isolate is in multiple phrases we simply AND the remainder phrases and then NOT the whole thing. This works because if any remainder phrase equals zero then A has to equal 1 for the equation to be satisfied. See below:

$$\begin{aligned}(A \vee B \vee C) \wedge (A \vee D \vee E) &= 1 \\ A &= \neg((B \vee C) \wedge (D \vee E))\end{aligned} \tag{3}$$

If A has a NOT in front of it then we simply do the same thing but cancel out the NOTs. See below:

$$\begin{aligned}(\neg A \vee B \vee C) \wedge (\neg A \vee D \vee E) &= 1 \\ \neg A &= \neg((B \vee C) \wedge (D \vee E)) \\ A &= (B \vee C) \wedge (D \vee E)\end{aligned} \tag{4}$$

Combining this together we first make an equation for the positive A and then AND it with the ¬A to get the following equation:

$$\begin{aligned}(A \vee B \vee C) \wedge (\neg A \vee D \vee E) \wedge (A \vee \neg F \vee G) \wedge (\neg A \vee H \vee \neg J) &= 1 \\ (Pos)\ A &= \neg((B \vee C) \wedge (\neg F \vee G)) \\ (Neg)\neg A &= \neg((D \vee E) \wedge (H \vee \neg J)) \\ (Combined)A &= \neg((B \vee C) \wedge (\neg F \vee G)) \wedge ((D \vee E) \wedge (H \vee \neg J))\end{aligned} \tag{5}$$

Now we'll examine why this works even when the remainder phrase equals 1 assuming the 3SAT equation has only one satisfiable answer. The following example actually has multiple satisfiable answers but I am using it as a simplified example since a real example requires a large number of phrases. Pretend there is only one solution to this problem. I will give a real example in the Examples Section.

$$\begin{aligned}&(A \vee B \vee C) \wedge (\neg A \vee D \vee E) = 1 \\ &Let\ (B \vee C) = 1 \\ &If A = 1\ then\ \neg A = 0 \\ &If \neg A = 0\ then\ (D \vee E) = 1\end{aligned} \tag{6}$$

If this is the case A = 1 and A = 0 both work. If this 3SAT equation had only one satisfiable solution we would have a contradiction since this would mean we have 2 possible solutions for an equation that only has 1 solution. The only way not to have a contradiction is for the values to be $A = 0\ then\ \neg A = 1\ and\ (D \vee E) = 0$.

2.2 Boolean System of Equations

This process of Boolean isolation is done for each variable in the 3SAT equation. The result is a Boolean system of equations. We now have an equation providing a definition for each variable. Since we are using this algorithm on 3SAT problems with only one set of correct values there is only one value each variable can have. This means that

if any other algorithm for solving these 3SAT problems was ever developed it would have to be an equivalent circuit since the output has to match and the inputs would have to be the same or correlated with the input variables in the equation we develop.

Furthermore it is possible to find the simplest equivalent circuit to the initial equation we create. One method is to exhaustively try every possible circuit that is less complex than the one we developed and see if the truth tables match. Other methods are guaranteed to find the simplest circuit for 2 level circuits. This includes Karnaugh maps which are practical when 6 or fewer variables exist. Another method would be to convert the equation to a 2 level circuit and use the Quine-McCluskey algorithm [4-5]. The examples that will be shown later in this paper were developed using a python script which calls a math library using the Quine-McCluskey algorithm. Once simplified we now know the minimum complexity of the circuit and can determine whether it is a simple or complex equation (the definition of simple and complex equations are explained in the next subsection). Additionally we can determine whether an equation is simple or complex purely with truth tables which will be discussed in the next section where we define those terms.

It may at first glance appear that there may be some way to develop an algorithm to find the value of the variables but remember that to solve for any variable in the equation the algorithm must be an equivalent circuit to the definition for the that variable in the system of equations. Furthermore of the 3 typical methods for solving systems of equations (graphing, elimination and substitution) only the last one is applicable. Graphing and elimination don't work because the equations are not numerical in nature. Substitution can be done but if used on complex equations it results in an initial equation that is more complicated than the original circuit. Even if we attempt to simplify the circuit after substitution we are adding more steps to finding a solution. All methods which are guaranteed to simplify the circuit to its minimum size require superpolynomial time. For instance the easiest way would be to develop a truth table for each equation and then find a line where all values match for all truth tables. Although comparing the tables can be done in polynomial time the size of each truth table doubles with each additional variable taking 2^n entries. Karnaugh maps and Quine-McCluskey have similar problems. While faster heuristic methods exist they are not guaranteed to fully minimize the circuit and this is assuming any method would reduce the circuit to a smaller size than the original non-substituted equation. Furthermore after one or more substitutions you end up with a circuit where the variable you are trying to solve for appears on both sides of the equation. You cannot use an unknown variable to solve for the same unknown variable as this is a bit like a snake trying to eat its own tail.

2.3 Simple and Complex Equations

First we need to define what simple and complex equations are. A simple equation is defined as an equation where defining the value of one of the variables allows you to determine the value of one or more other variables in the same equation. A complex equation means the opposite. Multiple variables have to be defined to derive the value of another variable.

If enough equation(s) are simple then it is possible to easily solve for all variables in polynomial time by attempting to set one of the variables to true/1 and false/0 and deriving the rest of the values. If both work then the correct set of variables will be the one that works in the original 3SAT equation. If all equations are complex then it is impossible to derive the value of any variable until multiple other variables are determined. Therefore if all equations are complex the only way to determine all variables is with a large amount of random guessing which requires superpolynomial time.

There are 2 methods to determine whether an equation is simple or complex. The first is to use a truth table and look for any situations where setting one variable to a particular value always results in another value being the same or opposite. This method will always work even on the direct output from Boolean isolation that has not yet been simplified. This is useful because the output from Boolean isolation is a 3 level circuit and methods like Quine-McCluskey can only guarantee the absolute minimal circuit for 2 level circuits so I have used truth tables to confirm which equations are simple or complex.

$$A = B \wedge \neg C \tag{7}$$

A	B	C
0	0	0
0	0	1
1	1	0
0	1	1

In the example above we can see that every time $A = 1$ *then* $B = 1$ *and* $C = 0$. Likewise any time $B = 0$ *then* $A = 0$. Also any time $C = 1$ *then* $A = 0$. These consistent relationships define this as a simple equation. If this was a complex equation no consistent relationships would occur.

The second method is to simplify the equation and determine whether it is a 2 level circuit (or more) where every variable is at least 2 levels deep. If it meets both criteria it is a complex equation. The prior equation does not meet these qualifications and therefore is a simple equation. Other examples include:

$$\begin{aligned} A &= B \wedge (\neg C \wedge D) \\ A &= B \vee C \end{aligned} \tag{8}$$

A complex equation is the opposite. Setting the value of any one variable will not allow you to solve for any other variable in the equation. The following are examples:

$$\begin{aligned} A &= (B \wedge C) \vee (D \wedge E) \\ A &= (B \vee C) \wedge (D \vee E) \end{aligned} \tag{9}$$

In cases where a direct answer is provided such as $A = 0$ the equation is in a separate category neither simple nor complex since there aren't 2 variables to compare.

2.4 2SAT

While it is technically not possible to use this method on 2SAT since all 2SAT equations have 2 or more possible solution sets we can see why 2SAT and 3SAT are in separate complexity classes. If boolean isolation was applied to 2SAT the remainder phrase would be only one literal by itself. When these literals get ANDed together it would always result in a simple equation. By contrast 3SAT remainder phrases result in 2 literals ORed together and then these phrases are ANDed together to form an equation. In some cases this will result in a complex equation.

3 Examples

3.1 3SAT Example With Complex Equations (Superpolynomial Time)

This example demonstrates that this particular 3SAT problem cannot be solved in polynomial time. The equations were created using boolean isolation then simplified using the simplify_logic() function in the python library sympy with the argument force = True passed to the function. The argument force = True tells the function to use Quine-McCluskey for simplification. However, since this is a 3 level circuit it's not guaranteed to produce the absolute smallest equation even though I do not see any obvious way to reduce the equations more than it already has. To further prove that none of the equations are simple I have provided the truth tables for each variable. As you can see no single variable value can be used to predict the value of another variable by itself thus proving all equations are complex. Since the equations are complex this means that the value of any variable requires guessing the value of multiple other variable first requiring superpolynomial time.

Please note all truth tables are shown in the same alphabetical order instead of the standard way of placing the variable being solved for to the far right or left of the table. This is done in case the reader wants to compare tables. The variable being solved for is separated by vertical lines to make it obvious what is the solution variable.

Correct Solution $A = 1;\ B = 0;\ C = 1;\ D = 1;\ E = 1;\ F = 1$

$$(\neg B \vee \neg D \vee \neg E) \wedge (\neg A \vee E \vee \neg F) \wedge (\neg B \vee \neg D \vee F) \wedge (\neg A \vee \neg B \vee D) \wedge (B \vee \neg E \vee F) \\ \wedge (B \vee \neg C \vee E) \wedge (B \vee D \vee \neg E) \wedge (B \vee C \vee \neg D) \wedge (B \vee C \vee F) \wedge (\neg C \vee \neg D \vee E) \\ \wedge (A \vee \neg E \vee \neg F) \wedge (\neg C \vee D \vee \neg E) \wedge (\neg C \vee E \vee \neg F) \wedge (A \vee \neg B \vee E) \wedge (A \vee C \vee D) = 1$$

$$A = \neg((\neg E \vee \neg F) \wedge (\neg B \vee E) \wedge (C \vee D)) \wedge ((E \vee \neg F) \wedge (\neg B \vee D))$$
$$A = (D \vee \neg B) \wedge (E \vee \neg F) \wedge (B \vee F \vee \neg C) \wedge (B \vee F \vee \neg D) \wedge (F \vee \neg D \vee \neg E)$$

$$B = \neg((\neg E \vee F) \wedge (\neg C \vee E) \wedge (D \vee \neg E) \wedge (C \vee \neg D) \wedge (C \vee F)) \wedge ((\neg D \vee \neg E) \\ \wedge (\neg D \vee F) \wedge (\neg A \vee D) \wedge (A \vee E))$$
$$B = (A \vee E) \wedge (E \vee F) \wedge (D \vee \neg A) \wedge (\neg D \vee \neg E)$$

$$C = \neg((B \vee \neg D) \wedge (B \vee F) \wedge (A \vee D)) \wedge ((B \vee E) \wedge (\neg D \vee E) \wedge (D \vee \neg E) \wedge (E \vee \neg F))$$
$$C = (B \vee D) \wedge (D \vee \neg A) \wedge (D \vee \neg F) \wedge (E \vee \neg D) \wedge (\neg B \vee \neg E)$$

$$D = \neg((\neg A \vee \neg B) \wedge (B \vee \neg E) \wedge (\neg C \vee \neg E) \wedge (A \vee C)) \wedge ((\neg B \vee \neg E) \wedge (\neg B \vee F) \\ \wedge (B \vee C) \wedge (\neg C \vee E))$$
$$D = (B \vee C) \wedge (C \vee F) \wedge (E \vee \neg C) \wedge (\neg B \vee \neg E)$$

$$E = \neg((\neg A \vee \neg F) \wedge (B \vee \neg C) \wedge (\neg C \vee \neg D) \wedge (\neg C \vee \neg F) \wedge (A \vee \neg B)) \\ \wedge ((\neg B \vee \neg D) \wedge (B \vee F) \wedge (B \vee D) \wedge (A \vee \neg F) \wedge (\neg C \vee D))$$
$$E = (A \vee B) \wedge (B \vee D) \wedge (A \vee \neg F) \wedge (D \vee \neg C) \wedge (F \vee \neg A) \wedge (\neg B \vee \neg D)$$

$$F = \neg((\neg B \vee \neg D) \wedge (B \vee \neg E) \wedge (B \vee C)) \wedge ((\neg A \vee E) \wedge (A \vee \neg E) \wedge (\neg C \vee E))$$
$$F = (A \vee \neg C) \wedge (A \vee \neg E) \wedge (D \vee \neg B) \wedge (E \vee \neg A) \tag{10}$$

Table 1 Solving For A

A	B	C	D	E	F
1	0	0	0	0	0
0	0	0	0	0	1
1	0	0	0	1	0
1	0	0	0	1	1
0	0	0	1	0	0
0	0	0	1	0	1
0	0	0	1	1	0
1	0	0	1	1	1
0	0	1	0	0	0
0	0	1	0	0	1
0	0	1	0	1	0
1	0	1	0	1	1
0	0	1	1	0	0
0	0	1	1	0	1
0	0	1	1	1	0
1	0	1	1	1	1
0	1	0	0	0	0
0	1	0	0	0	1
0	1	0	0	1	0
0	1	0	0	1	1
1	1	0	1	0	0
0	1	0	1	0	1
0	1	0	1	1	0
1	1	0	1	1	1
0	1	1	0	0	0
0	1	1	0	0	1
0	1	1	0	1	0
0	1	1	0	1	1
1	1	1	1	0	0
0	1	1	1	0	1
0	1	1	1	1	0
1	1	1	1	1	1

Table 2 Solving For B

A	B	C	D	E	F
0	0	0	0	0	0
0	0	0	0	0	1
0	1	0	0	1	0
0	1	0	0	1	1
0	0	0	1	0	0
0	0	0	1	0	1
0	0	0	1	1	0
0	0	0	1	1	1
0	0	1	0	0	0
0	0	1	0	0	1
0	1	1	0	1	0
0	1	1	0	1	1
0	0	1	1	0	0
0	0	1	1	0	1
0	0	1	1	1	0
0	0	1	1	1	1
1	0	0	0	0	0
1	0	0	0	0	1
1	0	0	0	1	0
1	0	0	0	1	1
1	0	0	1	0	0
1	1	0	1	0	1
1	0	0	1	1	0
1	0	0	1	1	1
1	0	1	0	0	0
1	0	1	0	0	1
1	0	1	0	1	0
1	0	1	0	1	1
1	0	1	1	0	0
1	1	1	1	0	1
1	0	1	1	1	0
1	0	1	1	1	1

Table 3 Solving For C

A	B	C	D	E	F
0	0	0	0	0	0
0	0	0	0	0	1
0	0	0	0	1	0
0	0	0	0	1	1
0	0	0	1	0	0
0	0	0	1	0	1
0	0	1	1	1	0
0	0	1	1	1	1
0	1	1	0	0	0
0	1	0	0	0	1
0	1	0	0	1	0
0	1	0	0	1	1
0	1	0	1	0	0
0	1	0	1	0	1
0	1	0	1	1	0
0	1	0	1	1	1
1	0	0	0	0	0
1	0	0	0	0	1
1	0	0	0	1	0
1	0	0	0	1	1
1	0	0	1	0	0
1	0	0	1	0	1
1	0	1	1	1	0
1	0	1	1	1	1
1	1	0	0	0	0
1	1	0	0	0	1
1	1	0	0	1	0
1	1	0	0	1	1
1	1	0	1	0	0
1	1	0	1	0	1
1	1	0	1	1	0
1	1	0	1	1	1

Table 4 Solving For D

A	B	C	D	E	F
0	0	0	0	0	0
0	0	0	0	0	1
0	0	0	0	1	0
0	0	0	0	1	1
0	0	1	0	0	0
0	0	1	0	0	1
0	0	1	1	1	0
0	0	1	1	1	1
0	1	0	0	0	0
0	1	0	1	0	1
0	1	0	0	1	0
0	1	0	0	1	1
0	1	1	0	0	0
0	1	1	0	0	1
0	1	1	0	1	0
0	1	1	0	1	1
1	0	0	0	0	0
1	0	0	0	0	1
1	0	0	0	1	0
1	0	0	0	1	1
1	0	1	0	0	0
1	0	1	0	0	1
1	0	1	1	1	0
1	0	1	1	1	1
1	1	0	0	0	0
1	1	0	1	0	1
1	1	0	0	1	0
1	1	0	0	1	1
1	1	1	0	0	0
1	1	1	0	0	1
1	1	1	0	1	0
1	1	1	0	1	1

Table 5 Solving For E

A	B	C	D	E	F
0	0	0	0	0	0
0	0	0	0	0	1
0	0	0	1	0	0
0	0	0	1	0	1
0	0	1	0	0	0
0	0	1	0	0	1
0	0	1	1	0	0
0	0	1	1	0	1
0	1	0	0	1	0
0	1	0	0	0	1
0	1	0	1	0	0
0	1	0	1	0	1
0	1	1	0	0	0
0	1	1	0	0	1
0	1	1	1	0	0
0	1	1	1	0	1
1	0	0	0	0	0
1	0	0	0	0	1
1	0	0	1	0	0
1	0	0	1	1	1
1	0	1	0	0	0
1	0	1	0	0	1
1	0	1	1	0	0
1	0	1	1	1	1
1	1	0	0	0	0
1	1	0	0	1	1
1	1	0	1	0	0
1	1	0	1	0	1
1	1	1	0	0	0
1	1	1	0	0	1
1	1	1	1	0	0
1	1	1	1	0	1

Table 6 Solving For F

A	B	C	D	E	F
0	0	0	0	0	1
0	0	0	0	1	0
0	0	0	1	0	1
0	0	0	1	1	0
0	0	1	0	0	0
0	0	1	0	1	0
0	0	1	1	0	0
0	0	1	1	1	0
0	1	0	0	0	0
0	1	0	0	1	0
0	1	0	1	0	1
0	1	0	1	1	0
0	1	1	0	0	0
0	1	1	0	1	0
0	1	1	1	0	0
0	1	1	1	1	0
1	0	0	0	0	0
1	0	0	0	1	1
1	0	0	1	0	0
1	0	0	1	1	1
1	0	1	0	0	0
1	0	1	0	1	1
1	0	1	1	0	0
1	0	1	1	1	1
1	1	0	0	0	0
1	1	0	0	1	0
1	1	0	1	0	0
1	1	0	1	1	1
1	1	1	0	0	0
1	1	1	0	1	0
1	1	1	1	0	0
1	1	1	1	1	1

3.2 3SAT Example With Simple Equations (Polynomial Time)

This example can be solved in polynomial time and for that reason is not part of the proof . It is provided only to demonstrate that this algorithm can be used as a heuristic to solve some 3SAT problems. In fact in the process of developing examples for this paper I found it was much easier to create problems that can be solved in polynomial time than to create ones that need superpolynomial time. After we plug in 0 every place the C variable is at we only need to try one variable to derive all the other values. This gives us 2 possible solutions (1,1,0,0) and (0,0,0,1). The second solution (0,0,0,1) can be ruled out because it fails the original 3SAT problem. Because it is obvious these are simple equations I skipped providing the truth tables for each equation.

Correct Solution $A = 1;\ B = 1;\ C = 0;\ D = 0$

$$(\neg A \vee \neg B \vee \neg C) \wedge (\neg A \vee C \vee \neg D) \wedge (B \vee \neg C \vee \neg D) \wedge (\neg A \vee B \vee \neg C) \wedge (B \vee C \vee D) \wedge (\neg B \vee \neg C \vee \neg D) \wedge (\neg B \vee \neg C \vee D) \wedge (A \vee \neg B \vee \neg D) \wedge (A \vee C \vee D) \wedge (A \vee \neg C \vee D) \wedge (A \vee B \vee C) = 1$$

$$A = \neg((\neg B \vee \neg D) \wedge (C \vee D) \wedge (\neg C \vee D) \wedge (B \vee C)) \wedge ((\neg B \vee \neg C) \wedge (C \vee \neg D) \wedge (B \vee \neg C))$$
$$A = \neg C \wedge \neg D$$

$$B = \neg((\neg C \vee \neg D) \wedge (\neg A \vee \neg C) \wedge (C \vee D) \wedge (A \vee C)) \wedge ((\neg A \vee \neg C) \wedge (\neg C \vee \neg D) \wedge (\neg C \vee D) \wedge (A \vee \neg D))$$
$$B = \neg C \wedge \neg D$$

$$C = \neg((\neg A \vee \neg D) \wedge (B \vee D) \wedge (A \vee D) \wedge (A \vee B)) \wedge ((\neg A \vee \neg B) \wedge (B \vee \neg D) \wedge (\neg A \vee B) \wedge (\neg B \vee \neg D) \wedge (\neg B \vee D) \wedge (A \vee D))$$
$$C = 0$$

$$D = \neg((B \vee C) \wedge (\neg B \vee \neg C) \wedge (A \vee C) \wedge (A \vee \neg C)) \wedge ((\neg A \vee C) \wedge (B \vee \neg C) \wedge (\neg B \vee \neg C) \wedge (A \vee \neg B))$$
$$D = \neg A \wedge \neg B \wedge \neg C$$

(11)

4 Infinite U3SAT

We have already demonstrated an example of a 3SAT problem that cannot be solved by a polynomial algorithm. Next we are going to show that there are an infinite number of such U3SAT problems that cannot be solved in polynomial time and that U3SAT problems continue occurring as the number of variables and phrases get larger. Essentially what we are going to do is the copy the original equation from 3.1 but with 6 new variables then AND that equation with the original equation. This can be repeated an infinite number of times and will still produce only 1 correct solution and will never create a simple equation when boolean isolation is done. Please note that this is not the only way to create U3SAT problems however I chose it because it is relatively easy to understand and prove. This pattern only requires that the number of variables be a multiple of 6 (6, 12 18, 24, etc. . .). Other than that there is no limit on how many variables can be used.

To explain this let's start with a thought experiment. Let's take the U3SAT equation from 3.1. Suppose we renamed every variable and created a new equation

we'll call a "family of phrases" or just "family" for short. We'll simply shift the variable names 6 spaces alphabetically.

A becomes G
B becomes H
C becomes I
D becomes J
E becomes K
F becomes L

$$Family1 = (\neg B \vee \neg D \vee \neg E) \wedge (\neg A \vee E \vee \neg F) \wedge (\neg B \vee \neg D \vee F) \wedge (\neg A \vee \neg B \vee D)$$
$$\wedge (B \vee \neg E \vee F) \wedge (B \vee \neg C \vee E) \wedge (B \vee D \vee \neg E) \wedge (B \vee C \vee \neg D) \wedge (B \vee C \vee F)$$
$$\wedge (\neg C \vee \neg D \vee E) \wedge (A \vee \neg E \vee \neg F) \wedge (\neg C \vee D \vee \neg E) \wedge (\neg C \vee E \vee \neg F) \wedge (A \vee \neg B \vee E)$$
$$\wedge (A \vee C \vee D)$$

$$Family2 = (\neg H \vee \neg J \vee \neg K) \wedge (\neg G \vee K \vee \neg L) \wedge (\neg H \vee \neg J \vee L) \wedge (\neg G \vee \neg H \vee J)$$
$$\wedge (H \vee \neg K \vee L) \wedge (H \vee \neg I \vee K) \wedge (H \vee J \vee \neg K) \wedge (H \vee I \vee \neg J) \wedge (H \vee I \vee L)$$
$$\wedge (\neg I \vee \neg J \vee K) \wedge (G \vee \neg K \vee \neg L) \wedge (\neg I \vee J \vee \neg K) \wedge (\neg I \vee K \vee \neg L) \wedge (G \vee \neg H \vee K)$$
$$\wedge (G \vee I \vee J) \tag{12}$$

Everything would behave identically with the original equation because all the relationships between variables would be identical. There would still only be 1 possible solution to the problem and that solution would follow the same pattern. Also when boolean isolation is performed we would get the same equations. The only difference would be that all the variables are renamed to a letter 6 spaces later in the alphabet.

Now let's also consider what would happen if we took the original equation (Family1) and tested it on every combination of 12 variables (A through L). We would end up with 64 valid combinations. The first 6 letters (A through F) on every combination would be (1,0,1,1,1,1,?,?,?,?,?,?) but the remaining variables would be any possible combination of G through L. This is because each 3SAT phrase only cares about the 3 variables in the phrase. If the 3 variable combination is wrong then that phrase will return 0/false regardless of the other variables and the whole 3SAT equation will be unsatisfied. Family1 eliminates 63 of the 64 combinations of A through F. It has at least 1 phrase which will be unsatisfied if any of the values of A through F are incorrect. Since none of the phrases contain any letter above F it doesn't care what those values are.

If we tried the same procedure on Family2 we would get a similar result but in reverse (?,?,?,?,?,?,1,0,1,1,1,1). In both cases for 6 of the variables there's only 1 combination that works. If we now AND the 2 families we would end up with only one possible combination of A through L that works (1,0,1,1,1,1,1,0,1,1,1,1). This is because 1 combination * 1 combination = 1. Furthermore since 1 multiplied to any

power equals 1 this means if we repeated this procedure any number of times we would still get only 1 correct solution. The solution would be the solution to the original equation/family repeated for as many times as this copying pattern is repeated.

$$NewEquation = Family1 \wedge Family2 \tag{13}$$

Finally how do we know that ANDing 2 families will not cause a simple equation to occur once boolean isolation is done on the combined 3SAT equation? If the original equation produces no simple equations with boolean isolation then it is impossible for this pattern to produce a simple equation. This is because each family has it's own set of unique variables which are not shared with any other family. When boolean isolation is done the remainder phrases become part of the equation solving for the variable that was left out of the remainder phrase.

4.1 Full Example

Notice in the full example that each family is completely isolated from each other in the boolean isolation output. For example the equations solving for A through F contain only the variables A through F from Family1 and the equations solving for G through L contain only the variables G through L from Family2. This is because the equation that a phrase is assigned to has to contain that variable. This means that no new simplifications of any equation are possible. In addition the equations for G through L are complete copies of the equations for A through F just with the letters shifted. This means that we could repeat this process again and again infinitely and know that no new simplifications will ever be possible.

$Correct\ Solution\ A = 1;\ B = 0;\ C = 1;\ D = 1;\ E = 1;\ F = 1; G = 1; H = 0; I = 1; J = 1; K = 1; L = 1;$

$Family1 \wedge Family2 = 1$

$$(\neg B \vee \neg D \vee \neg E) \wedge (\neg A \vee E \vee \neg F) \wedge (\neg B \vee \neg D \vee F) \wedge (\neg A \vee \neg B \vee D) \wedge (B \vee \neg E \vee F)$$
$$\wedge (B \vee \neg C \vee E) \wedge (B \vee D \vee \neg E) \wedge (B \vee C \vee \neg D) \wedge (B \vee C \vee F) \wedge (\neg C \vee \neg D \vee E)$$
$$\wedge (A \vee \neg E \vee \neg F) \wedge (\neg C \vee D \vee \neg E) \wedge (\neg C \vee E \vee \neg F) \wedge (A \vee \neg B \vee E) \wedge (A \vee C \vee D) \wedge$$
$$(\neg H \vee \neg J \vee \neg K) \wedge (\neg G \vee K \vee \neg L) \wedge (\neg H \vee \neg J \vee L) \wedge (\neg G \vee \neg H \vee J) \wedge (H \vee \neg K \vee L)$$
$$\wedge (H \vee \neg I \vee K) \wedge (H \vee J \vee \neg K) \wedge (H \vee I \vee \neg J) \wedge (H \vee I \vee L) \wedge (\neg I \vee \neg J \vee K)$$
$$\wedge (G \vee \neg K \vee \neg L) \wedge (\neg I \vee J \vee \neg K) \wedge (\neg I \vee K \vee \neg L) \wedge (G \vee \neg H \vee K) \wedge (G \vee I \vee J) = 1$$

$Family1$

$$A = \neg((\neg E \vee \neg F) \wedge (\neg B \vee E) \wedge (C \vee D)) \wedge ((E \vee \neg F) \wedge (\neg B \vee D))$$
$$A = (D \vee \neg B) \wedge (E \vee \neg F) \wedge (B \vee F \vee \neg C) \wedge (B \vee F \vee \neg D) \wedge (F \vee \neg D \vee \neg E)$$

$$B = \neg((\neg E \vee F) \wedge (\neg C \vee E) \wedge (D \vee \neg E) \wedge (C \vee \neg D) \wedge (C \vee F)) \wedge ((\neg D \vee \neg E)$$
$$\wedge (\neg D \vee F) \wedge (\neg A \vee D) \wedge (A \vee E))$$
$$B = (A \vee E) \wedge (E \vee F) \wedge (D \vee \neg A) \wedge (\neg D \vee \neg E)$$

$$C = \neg((B \vee \neg D) \wedge (B \vee F) \wedge (A \vee D)) \wedge ((B \vee E) \wedge (\neg D \vee E) \wedge (D \vee \neg E) \wedge (E \vee \neg F))$$
$$C = (B \vee D) \wedge (D \vee \neg A) \wedge (D \vee \neg F) \wedge (E \vee \neg D) \wedge (\neg B \vee \neg E)$$

$$D = \neg((\neg A \vee \neg B) \wedge (B \vee \neg E) \wedge (\neg C \vee \neg E) \wedge (A \vee C)) \wedge ((\neg B \vee \neg E) \wedge (\neg B \vee F)$$
$$\wedge (B \vee C) \wedge (\neg C \vee E))$$
$$D = (B \vee C) \wedge (C \vee F) \wedge (E \vee \neg C) \wedge (\neg B \vee \neg E)$$

$$E = \neg((\neg A \vee \neg F) \wedge (B \vee \neg C) \wedge (\neg C \vee \neg D) \wedge (\neg C \vee \neg F) \wedge (A \vee \neg B))$$
$$\wedge ((\neg B \vee \neg D) \wedge (B \vee F) \wedge (B \vee D) \wedge (A \vee \neg F) \wedge (\neg C \vee D))$$
$$E = (A \vee B) \wedge (B \vee D) \wedge (A \vee \neg F) \wedge (D \vee \neg C) \wedge (F \vee \neg A) \wedge (\neg B \vee \neg D)$$

$$F = \neg((\neg B \vee \neg D) \wedge (B \vee \neg E) \wedge (B \vee C)) \wedge ((\neg A \vee E) \wedge (A \vee \neg E) \wedge (\neg C \vee E))$$
$$F = (A \vee \neg C) \wedge (A \vee \neg E) \wedge (D \vee \neg B) \wedge (E \vee \neg A)$$

(14)

$Family2$

$$
\begin{aligned}
G &= \neg((\neg K \vee \neg L) \wedge (\neg H \vee K) \wedge (I \vee J)) \wedge ((K \vee \neg L) \wedge (\neg H \vee J)) \\
G &= (J \vee \neg H) \wedge (K \vee \neg L) \wedge (H \vee L \vee \neg I) \wedge (H \vee L \vee \neg J) \wedge (L \vee \neg J \vee \neg K)
\end{aligned}
$$

$$
\begin{aligned}
H &= \neg((\neg K \vee L) \wedge (\neg I \vee K) \wedge (J \vee \neg K) \wedge (I \vee \neg J) \wedge (I \vee L)) \wedge ((\neg J \vee \neg K) \\
&\quad \wedge (\neg J \vee L) \wedge (\neg G \vee J) \wedge (G \vee K)) \\
H &= (G \vee K) \wedge (K \vee L) \wedge (J \vee \neg G) \wedge (\neg J \vee \neg K)
\end{aligned}
$$

$$
\begin{aligned}
I &= \neg((H \vee \neg J) \wedge (H \vee L) \wedge (G \vee J)) \wedge ((H \vee K) \wedge (\neg J \vee K) \wedge (J \vee \neg K) \wedge (K \vee \neg L)) \\
I &= (H \vee J) \wedge (J \vee \neg G) \wedge (J \vee \neg L) \wedge (K \vee \neg J) \wedge (\neg H \vee \neg K)
\end{aligned}
$$

$$
\begin{aligned}
J &= \neg((\neg G \vee \neg H) \wedge (H \vee \neg K) \wedge (\neg I \vee \neg K) \wedge (G \vee I)) \wedge ((\neg H \vee \neg K) \wedge (\neg H \vee L) \\
&\quad \wedge (H \vee I) \wedge (\neg I \vee K)) \\
J &= (H \vee I) \wedge (I \vee L) \wedge (K \vee \neg I) \wedge (\neg H \vee \neg K)
\end{aligned}
$$

$$
\begin{aligned}
K &= \neg((\neg G \vee \neg L) \wedge (H \vee \neg I) \wedge (\neg I \vee \neg J) \wedge (\neg I \vee \neg L) \wedge (G \vee \neg H)) \\
&\quad \wedge ((\neg H \vee \neg J) \wedge (H \vee L) \wedge (H \vee J) \wedge (G \vee \neg L) \wedge (\neg I \vee J)) \\
K &= (G \vee H) \wedge (H \vee J) \wedge (G \vee \neg L) \wedge (J \vee \neg I) \wedge (L \vee \neg G) \wedge (\neg H \vee \neg J)
\end{aligned}
$$

$$
\begin{aligned}
L &= \neg((\neg H \vee \neg J) \wedge (H \vee \neg K) \wedge (H \vee I)) \wedge ((\neg G \vee K) \wedge (G \vee \neg K) \wedge (\neg I \vee K)) \\
L &= (G \vee \neg I) \wedge (G \vee \neg K) \wedge (J \vee \neg H) \wedge (K \vee \neg G)
\end{aligned}
$$

(15)

4.2 Further Considerations

For the sake of further exploration and to answer any possible objections to the method just described I will now show how slight variations of this pattern can also produce U3SAT problems that do not offer solutions that can be sped up with any simple heuristic.

An argument could be made that solving a multifamily U3SAT problem is not significantly harder than solving one family since in the previous section we described a pattern where each family is essentially a copy of the previous family and that if we found the values of the variables in any family we could just copy them to the corresponding variables in the other families assuming it would be possible for an algorithm to spot this pattern in polynomial time. However, even this can be broken if we introduce something I call multifamily phrases. A multifamily phrase is one where

each variable comes from a different family. This can only be done on U3SATs with at least 3 families. Here is an example:

$$\begin{aligned} &Family1\,(A-F)\\ &Family2\,(G-L)\\ &Family3\,(M-R)\\ \\ &Multifamily\,Phrase\,(A\vee G\vee M) \end{aligned} \tag{16}$$

Adding this phrase to the new equation now means that the variables in one family cannot be calculated independently of the other families since for example to determine the value of A we now need to know the values of G and M. However, the values of G and M are decided by the equations in family 2 and 3.

To explain why this would not risk any simplifications in the existing boolean equations we first need to define a term and establish a lemma. First we will define two remainder phrases as being in "conflict" if it is possible for the 2 phrases to be simplified. This first requires the two remainder phrases to be in the same equation. Next simplification is not possible if the two remainder phrases don't share any variables in common. If one or more variables are shared then they might be in conflict but this depends on the specifics of the two phrases. We will define two 3SAT phrases as being in conflict if the two phrases produce remainder phrases that come into conflict. This leads to a natural consequence.

Lemma: If two 3SAT phrases have 1 or fewer variables in common they will never be in conflict.

The reason for this is that 2 remainder phrases can only occur in the same equation if the 3SAT phrases they come from share at least 1 variable. However, if there is only 1 shared variable then the remainder phrases cannot contain a shared variable since the solution variable is not used in the remainder phrases.

An important consequence of this is that multifamily phrases will never come into conflict with the phrases for each family so long as each variable in a multifamily phrase comes from a different family. Similarly the U3SAT equation can have multiple multifamily phrases so long as the multifamily phrases share at most 1 variable. How many multifamily phrases can be added is essentially a combination problem and the number increases factorially. For example the following multifamily phrases are not in conflict with each other or the 3 families:

$$
\begin{aligned}
&Family1\,(A-F)\\
&Family2\,(G-L)\\
&Family3\,(M-R)\\
&\\
&(A \vee G \vee M)\\
&(A \vee H \vee N)\\
&(A \vee K \vee P)
\end{aligned}
\tag{17}
$$

Notice in the 2nd and 3rd phrases that each variable in a multifamily phrase does not need to correspond with its equivalent in another family. This allows for a large number of combinations of multifamily phrases. It also means the system of boolean equations do not need to be symmetrical for each family. This makes it difficult for any simple heuristic to tell that the families are similar.

Next we could also use multifamily phrases to change the solution for 1 or more families. To do this we could remove a phrase from one of the families. This would give us 2 or more solutions to that family. We could then use multifamily phrases to invalidate the solution(s) we don't want including the original solution. This could be used to change the solution of 1 or more families and thus break up any pattern in the overall solution.

Finally we have used one family as the template for all other families but there is nothing preventing us from for example using 2 randomly generated families as templates and alternating which one is copied. The example in 3.1 that is used was randomly generated by a python script I wrote which uses the random number generator to produce U3SAT problems. I know from testing that more than 1 non polynomial U3SAT problem can be generated.

In conclusion we have now demonstrated that there are an infinite number of U3SAT problems with superpolynomial time complexity. These problems keep occurring as the number of variables increases and the time required to solve these problems increases superpolynomially as the number of variables increases.

5 Conclusions and Observations

In this paper we discussed how all NP problems can be converted to 3SAT problems. We discussed how the complexity class it belongs to is determined by the worst case outcome. This means that showing examples of a problem needing superpolynomial time that continue to occur as the number of variables increase proves that it is in a separate complexity class from P. In the Boolean Isolation section 2.1 we showed that if a 3SAT equation has only 1 solution that the method I called boolean isolation will create an equation for each variable. In Boolean Systems of Equations 2.2 we discussed how any algorithm produced by any other method would have to be equivalent to the one we created by boolean isolation. In Simple and Complex Equations 2.3 we discussed how some equations do not allow variables to be derived until several other

variables are known first. Finally in the Examples we showed a real equation that produces a complex system of equations 3.1. Finally in Infinite U3SAT 4 we showed that there are an infinite number of U3SAT problems that continue to occur as the number of variables increase and that these larger U3SAT problems take longer and longer to solve and cannot be solved in polynomial time. I therefore conclude that this paper proves that $P \neq NP$.

Beyond resolving the P versus NP question this algorithm has other potential uses. The algorithm represents a new heuristic method for solving NP problems that have only 1 solution.

In the process of solving this problem I developed a couple of python scripts to speed up the process of creating example U3SAT problems and then automating the boolean isolation and the simplification of the system of equations that it creates. The goal of creating this script was to create the simplest possible example that would still successfully prove that 3SAT cannot be solved in polynomial time. The script would randomly create phrases to eliminate potential wrong solutions until only one answer could work. If a phrase already existed that eliminated that particular wrong answer then the script would discard the new phrase to minimize the size of the equation. Since I was trying to make the smallest example possible and Quine-McCluskey becomes impossibly slow with large numbers of variables I limited my tests to 8 or fewer variables and usually 6 or fewer variables. I noticed a few patterns and observations.

First the overwhelming majority of randomly created 3SAT problems could be solved in polynomial time. Typically if even 1 of the equations is simple then the values of all the variables are typically trivial to determine. It took quite a few attempts to come up with even 1 example of a 3SAT problem that needs exponential time. When Boolean isolation created a simple equation it would always be either a variable(s) ANDed together sometimes with a 2 variable phrase or it would simply spit out a false/0 as the value for the variable. If it directly spits out the answer it would always be false/0 and never true/1. Similarly I never saw a simple equation which was a series of just ORs.

This algorithm could be particularly relevant to cryptography because cryptographic functions are typically restricted to having one valid answer and typically have a one to one relationships. In theory this algorithm could open up new methods of cryptanalysis as well as potentially open the door to creating provably unbreakable encryptions. One major difficulty in practice though is that determining whether an equation is simple or complex requires techniques that are themselves NP. If truth tables are used the size of the table doubles with each additional variable. With boolean simplification methods like Quine-McCluskey superpolynomial time is required and quickly becomes intractable with a significant number of variables. This seems like an area deserving further investigation.

Declarations

The author has no competing interests to declare. This work was developed and written by Michael Marquard with no outside funding or obligations.

6 References

[1] Cook, Stephen (1971). "The complexity of theorem proving procedures". Proceedings of the Third Annual ACM Symposium on Theory of Computing. pp. 151–158. doi:10.1145/800157.805047.

[2] Karp, Richard M. (1972). "Reducibility Among Combinatorial Problems". In Miller, Raymond E.; Thatcher, James W. (eds.). Complexity of Computer Computations. New York: Plenum. pp. 85–103.

[3] Davis, Martin; Putnam, Hilary (1960). "A Computing Procedure for Quantification Theory". Journal of the ACM. 7 (3): 201–215. doi:10.1145/321033.321034.

[4] Quine, Willard Van Orman (October 1952). "The Problem of Simplifying Truth Functions". The American Mathematical Monthly. 59 (8): 521–531. doi:10.2307/2308219.

[5] McCluskey, Edward Joseph Jr. (November 1956). "Minimization of Boolean Functions". Bell System Technical Journal. 35 (6): 1417–1444. doi:10.1002/j.1538-7305.1956.tb03835.x.