P VERSUS NP: A SURVEY

Ariz Ahmad

December 1, 2019

1 Introduction

The **P** vs **NP** problem is one of the central questions plaguing the field of computer science for decades. It all began with the 1900 challenge by Hilbert and has today morphed into one of the most sought after puzzles of the present age. The Clay institute has placed a prize of 1 million dollars on solving this problem; five of the other seven questions would also see a major breakthrough - making it all the more lucrative for someone to come up with a solution. Though it has been difficult to prove that $\mathbf{P} \neq \mathbf{NP}$, it is widely believed so by the scientific community, that these are two separate classes. Despite the hype surrounding new technologies like quantum computing, this is likely to remain so.

---

2 History

While working on David Hilbert's entschiedungsproblem of finding a "purely mathematical procedure" for giving an answer to a logical question, Turing, Church and Gödel proved that testing whether a problem has a proof is unsolvable. The **P** vs **NP** problem can be thought of a finite version of the entschiedungsproblem. Neither Church nor Turing provided any special cases where this problem may be solvable. Also, Turing machines did not provide any models for the amount of time or memory that would be required by any program running on it. Hartmanis and Stearns proposed a way to truly learn that, and hence the idea of complexity was born.

In a letter to von Neumann in 1956, Gödel wrote about the issue of brute force algorithms on problems of exponential size. Edmonds was the first one to give a lucid account of the problem in western literature. In the Soviet Union, researchers knew about the problem associated with exponential

search in the 1950s. Cobham, Edmunds and Rabin were the first to propose the class P, as a set of reasonably solvable problems.

Cook and Levin working independently proved the existence of complete problems in the NP class. Karp demonstrated the wide applicability of NP-complete problems by showing the extent to which they occur in everyday problems. So we see that rough references to this were made by several mathematicians. However, Cook was the first to explicitly conjecture the **P vs NP** problem in his 1971 paper.

---

3 Statement of the Problem

Despite being conceived in the 1930s before the existence of physical computers, the Turing machine can be used as a proper computer model to define this problem. The class **P** can be defined in terms of various computer models, but here it is defined as a set of languages accepted by the Turing machine, solvable in finite time.

Let Σ be an alphabet and consequently Σ^* is the set of all possible strings over Σ. We assume M to be a Turing machine which accepts the language L defined in terms of Σ. Every string $w \in \Sigma^*$ has a computation associated with it in M. The language L accepted by M can therefore be defined by the following:

$$L(M) = \{ w \in \Sigma^* \mid M \text{ accepts } w \}$$

Let's define $t_M(w)$ to be the steps required by M on the computation of input w. If the computation goes on indefinitely, $t_M(w) = \infty$. $T_M(n)$ can be defined to be the worst case time for any input n

$$T_M(n) = max\{t_M(w) \mid w \in \Sigma^n\}$$

Therefore, we can formulate the class **P** of languages as the following:

$$P = \{L \mid L = L(M) \text{ for some Turing machine M that is polynomial}\}$$

We use the formulation **NP** for historical reasons because these problems were defined in terms of non-deterministic Turing machines. These machines could be in one of several possible states following any particular instance. These days it is commonly defines in terms of a relation $R \subseteq \Sigma^* \times \Sigma_1^*$. For each relation R we define L_R over $\Sigma \cup \Sigma_1 \cup \{\#\}$ as

$$L_R = \{w\#y \mid R(w, y)\}$$

Finally, the language L is defined over Σ to be in **NP** iff there exists a $k \in \mathbb{N}$ a relation R which can be checked in polynomial time, for all $w \in \Sigma^*$

$$w \in L \Leftrightarrow \exists y(|y| \leq |w|^k \text{ and } R(w, y))$$

Does P = NP?
The problem is not dependent of the size of Σ. It is also intuitive, because a string in any alphabet can be coded to a string in our binary alphabet. Therefore, it is easy to show that $\mathbf{P} \subseteq \mathbf{NP}$, because for each language L we can define $R \subseteq \Sigma^* \cup \Sigma^*$

$$R(w, y) \Leftrightarrow w \in L$$

a polynomial time checking algorithm $\forall w, y \in \Sigma^*$

---

4 Status

Several failed attempts have been made to prove $\mathbf{P} \neq \mathbf{NP}$. Some of them are as follows:

Diagonalization
It was shown by Cantor in 1874, that real numbers are uncountable using this trick. The same technique was used by Turing to show the incomputability of the Halting problem. It has also been suggested that if we are given more time and memory, we could easily solve the **NP** problems. The problem with this approach however, is how to simulate a nondeterministic **NP** machine with an determinisitic **P** machine. Another problem is that of relativization - all machines would work despite having access to the same extra information. Diagonalization techniques have been used to show that some specific problems can not have both small memory and small runtime at the same time. But this is not sufficient to obtain any useful result.

Circuit Complexity
If it can be shown successfully that a problem can not be solved using a small number of gates AND, OR and NOT, then it can be proved that $\mathbf{P} \neq \mathbf{NP}$. Furst, Saxe an Sipser in 1984 showed that parity function cannot be solved properly by limiting the number of layers; then in 1985, Razborov et al showed that clique problems can not be solved using only AND and OR gates. These results however, fail abruptly if NOT gates are allowed, and

hence there has not been any major breakthrough in this direction in the past two decades.

Proof Complexity
If it could be proved that there do not exist any short proofs of a tautology, then this would mean that $\mathbf{P} \neq \mathbf{NP}$. If a formula ϕ is not a tautology, we can easily disprove it by assigning values to variables that make ϕ false. If however, ϕ is indeed a tautology, we would not expect a short proof of this. If it can be shown that a tautology hasn't got a short proof, then we would have effectively proved $\mathbf{P} \neq \mathbf{NP}$.

Restricted Systems
We take the approach of pruning the ways we approach the problem. These methods fall into two categories. In the *natural methods* we only allow operations which are relevant to the problem we are concerned with. In the *handicapped models*, we weaken the problem sufficiently so that it is easy to obtain a lower bound.

---

5 Current Approaches

These are some of the approaches we can use, once we have established that a problem is **NP**-complete.

Brute Force
Solving several **NP**-complete problems for practical purposes can be done using several tweaks. Best practical algorithms in practice can be used to solve the standard SAT problems for upto one hundred variables. Similarly, cutting plane method can be used to solve traveling salesman problem for up to ten thousand cities. But most **NP**-complete problem remain outside the purview of such hacks.

Parameterized Complexity
Downey and Fellows used parameter sizes to develop a approach using parameterized complexity approach to several problems. For example, vertex cover problem can be slightly morphed to find the central k elements with the condition that for every element pair, there is one central element. For small values of k, we can solve the vertex cover problem efficiently, without any effect of the total number of vertices present in the graph. For most problems though, like clique, the problem is difficult even for small k.

Approximation
Traveling salesman problem was solved by Arora such that it gives a very close approximation to the best possible route. Goemans and Williamson solved the MAX-CUT problem such that his algorithm provides a 0.878567 factor of the best possible answer. The approximation methods also have provable limitations which can be found using the theory of probabilistic proof systems.

Heuristic and Average-case Complexities
Average-case complexity is very useful in most cases of dealing with real-world problems. Instances which are generated through some specific distribution make it easier to deal with problems like SAT. Levin studied these scenarios in detail and proposed a version of the **P** vs **NP** problem which takes into account these distributional features.

---

6 Use of NP-Hardness

Cryptography
Cryptography would be impossible if **P** were equal to **NP** . The concept of zero-knowledge proofs can be used for a lot of applications besides public key transmissions and verification. This concept allows the verifier to know nothing other than some property, like the presence of a solution for a Sudoku puzzle. All NP search problems satisfy this properly. This concept if extended further, obviates the need of a trusted third-party between any two entities.

Eliminating Randomness
Since the 70s, several algorithms have been proposed which use random bits to solve a problem. This approach is often thought to also work for **NP**-complete problems. A long list of research has shown that a problem which has an efficient algorithm in a probabilistic approach, also has a corresponding algorithm that can be solved using a pseudo-random number generation procedure. Thus, it is widely believed by the experts on complexity theory that **NP** search problems can not be solved using random bits.

---

7 Quantum Computers

In 1994, Peter Shor in his ground-breaking paper showed that a hypothetical quantum computer could factor a number in exponentially less time. This could be used to break the RSA public-key encryption protocols, and was hailed as a major breakthrough in proving that **P** = **NP** by many. The problem, however, was that the algorithm relied on the algebraic properties of numbers - an attribute not shared by other **NP**-complete problems. Lov Grover was also successful in suggesting general algorithms for NP problems using quantum computing; but was only successful in achieving quadratic speed increase, with evidence that we can not do much better. So it is widely believed that quantum computers would not be as effective in solving the **P** vs **NP** puzzle as people would like to believe.

---

8 New Hope?

Ketan Mulmuley and Milind Sohoni used algebraic geometry, or GCT to suggest a solution. They avoided the generic approaches to showing **P** $\neq$ NP, but this approach however, was much more involved and is expected to take decades to see fruition.

They broke down this problem by defining a family of polygons of high-dimension P_n corresponding to some properties. For each integer n, if P_n has an integrer, then a group of Hamiltonian circuits must have a size greater than or equal to n^{logn} on an input of size n. This would prove that **P** $\neq$ **NP** . There are three sub-problems they think need to be proved for this result to hold:

1. Integer relaxation problem can be solved by LP relaxation in polynomial time for P_n
2. find a simple algorithm for the integer relaxation problem
3. prove that this algorithm always has the answer in the affirmative

The authors have given a set of approaches to solve (1), but the approach required to solve (2) and (3) seems much less obvious. In essence, the authors have reduced the problem by proposing to work on the proof of fast algorithms for **NP**-complete problems; albeit for specific problems, and with some specifications. But Mulmuley himself believes that proving the above three results will take any less than a hundred years, even if they are proved.

---

9 Conclusion

The **P** vs **NP** problem, during the course of its existence has transformed from a problem of logic to the most well-known mathematical puzzle of our times. The question has featured in popular shows, capturing the interest of experts and the general masses alike. The proof of $\mathbf{P} \neq \mathbf{NP}$ would not be of the final consequence, because several more challenges will remain, waiting to be proved.

---

10 References

[1] Fortnow, Lance. "The status of the P versus NP problem." Communications of the ACM 52.9 (2009): 78-86.

[2] Cook, Stephen. "The P versus NP problem." The millennium prize problems (2006): 87-104.

[3] Sipser, Michael. "The history and status of the P versus NP question." Proceedings of the twenty-fourth annual ACM symposium on Theory of computing. ACM, 1992.

[4] Aaronson, Scott. "

$$P \stackrel{?}{=} NP$$

." Open problems in mathematics. Springer, Cham, 2016. 1-122.

[5] The P-versus-NP page https://www.win.tue.nl/ gwoegi/P-versus-NP.htm

[6] Garey, Michael R., and David S. Johnson. Computers and intractability. Vol. 29. New York: wh freeman, 2002.

[7] Leiserson, Charles Eric, et al. Introduction to algorithms. Vol. 6. Cambridge, MA: MIT press, 2001.

[8] Dasgupta, Sanjoy, Christos H. Papadimitriou, and Umesh Virkumar Vazirani. Algorithms. McGraw-Hill Higher Education, 2008