Jacob Barger
CMPSCI 5130
Badri Adhikari
12/15/20

P=NP?

NP-Completeness

A phenomenal visual explanation of P vs. NP can be found here!

P vs. NP is a major unsolved 'Millenium problem, and generating a correct proof carries a $1 Million prize.

Intended module reading flow: P-> NP -> NPC

P: Polynomial Time

NP: Non deterministic Polynomial Time

NPC: NP-Complete

A problem is NP complete if its solution can be verified in polynomial time, but it is unknown if it can be solved in polynomial runtime. A major thereom in the P vs. NP proof postulates that If **any** NP problem can be solved in a polynomial runtime, then all NP problems can be solved in polynomial runtime. NP Completeness

P Problems are problems / tasks which can be solved and verified in polynomial runtime, or where runtime can be expressed as O(n^k)

"Does being able to quickly recognize NP correct answers mean there is also a way to find them?" Per the video above

NP problems are problems which can be 'verified' in polynomial time, but can either definitely or definitely not be solved in polynomial runtime; where verifiable means the solution can be realistically validated in a runtime expressed as O(n^k)

Think about Multiplication! If you want to multiply 50 and 7, you don't add 7 to itself 50 times, you perform a shortcut, like the one to the right. You can carry numbers and so on to get a result faster than the slowest method. This is a great example of a Polynomial or P runtime problem.

50
x 7
350

SUDOKU

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

ANSWER:

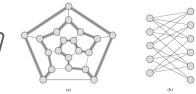
5	4	3	9	2	1	8	7	6
2	1	9	6	8	7	5	4	3
8	7	6	3	5	4	2	1	9
9	8	7	4	6	5	3	2	1
3	2	1	7	9	8	6	5	4
6	5	4	1	3	2	9	8	7
7	6	5	2	4	3	1	9	8
4	3	2	8	1	9	7	6	5
1	9	8	5	7	6	4	3	2

To clarify this thereom, we again we can turn to Suduku! If it is possible to solve a sudoku puzzle in the same time we can verify the solution, or atleast even slightly faster than the brute force method, then we can make the same improvements in solution runtime on **all** of the problems classified as NP. This is called a polynomial time reduction. So solving sudoku faster also means solving something like chess or protein folding faster.

Think about Sudoku! There are nearly no, if any, legitimate shortcuts in solving a sudoku puzzle, each square must essentially be 'guess and checked' for compability. This ensures it is not a P problem, but, what about validating a correct solution? You can do that much quicker than finding it! This is what makes it a NP problem

Hamilton Cycles have been proven to be an NP complete problem

Find a phenomenal proof of the NP-Completeness for Hamilton Cycles can be found here!



The problem of verification must be rigoursly defined in order to clearly draw the boundary between the three classes of problems

A major point of uncertainty in these ideas, and the core idea behind the P vs NP. problem, has to do with that of sets! Are all NP problems in the same set? Are P problems a subset of that same set? Or are P and easier NP problems in a set of their own?

Verification Algorithms

A two argument algorithm where one argument is an ordinary string, the other is a string called a certificate.

The algorithm verifies the initial input string: if there exists the certificate such that Algorithm(Initial, Certificate) = 1

Reducabiliy

This is the idea of reducing a problem which exists in one set, say NP hard, to component(s) which are member(s) of a set of easier problems

Think of basic algebra! Rearranging pieces of an equation, and setting one half of the equation equal to zero, etc, these are tactics used to reduce the overall complexity of the problem

In more specific terms, a reduction function ***f*** is found by a fedictopm algorithim **F**

Below we can see the primary two formulations of the famous P vs. NP problem, if P is not equal to NP, then they are in a seperate set entirely than NP-hard problems. It is currently unknown if NP hard problems can be solved with the same speed as regular NP or P problems. Part of the proof would handle if NP complete are verifiable and solvable within polynomial running time constraints. If P is equal to NP, then all problems can be verified and solved in polynomial running time, we just haven't found out how to do it yet. This proof could change how we see a majority of unsolved computational problems accross many different disciplines!