

Puzzling Computers

A gentle introduction to the theory of NP-Completeness.

Sudoku

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

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

Above is an *instance* of the sudoku (<http://en.wikipedia.org/wiki/Sudoku>) puzzle. The goal is to fill in numbers from 1 to 9 in the blanks such that, every row, column and 3×3 block has all the 9 different numbers. Beside is a solution for the puzzle instance and it is easy to verify that the *constraints* on the rows, columns and sub blocks are *satisfied*.

Since very few numbers are given, it is difficult to solve. Nevertheless a person might solve it in a few hours. But what if the puzzle instance was "larger". That is instead of a 9×9 , it is a 16×16 version, where numbers from 1 to 16 could be filled in. It has similar *constraints* on having all numbers in rows and columns and 4×4 sub blocks.

			4	7						8		5		12
13		9		14			1	11		12	8		6	
	8		1		6					5	10		16	
						11	10		16			7	4	13
4							6			15		11	14	
	5	16	11	1			4		6	3		8	9	
			9		8	14		4	10	2		5	16	
			8		9				1	12	11		6	10
8		15		5	14	7			6		3			
		1	13		3	6	15		12	8		2		
	10	11		13		2		15			4	6	9	5
	3	14		4				10						8
9	16	8			1			7	13					
	7		3	15						4		14		13
	13		12	9			7	2			14		10	16
1		10		12							6	7		

Such a puzzle would be mind boggling for humans to handle. But, may be there is a clever computer program, to do the job for us. There is some hope, since a solution given below for the above sudoku instance is very easy to verify.

10	2	6	4	7	16	9	3	13	14	15	8	1	5	11	12
13	15	9	16	14	4	5	1	11	7	10	12	8	3	6	2
11	8	7	1	2	6	13	12	3	4	9	5	10	15	16	14
14	12	3	5	8	15	11	10	6	2	16	1	9	7	4	13
4	1	2	10	16	7	12	6	8	9	5	15	13	11	14	3
15	5	16	11	1	2	10	4	14	6	13	3	12	8	9	7
3	6	12	9	11	8	14	13	4	10	2	7	5	16	1	15
7	14	13	8	3	9	15	5	16	1	12	11	4	6	2	10
8	9	15	2	5	14	7	16	1	11	6	13	3	12	10	4
5	4	1	13	10	3	6	15	9	12	8	16	2	14	7	11
16	10	11	7	13	12	2	8	15	3	14	4	6	9	5	1
12	3	14	6	4	11	1	9	10	5	7	2	16	13	15	8
9	16	8	14	6	1	3	2	7	13	11	10	15	4	12	5
2	7	5	3	15	10	16	11	12	8	4	9	14	1	13	6
6	13	4	12	9	5	8	7	2	15	1	14	11	10	3	16
1	11	10	15	12	13	4	14	5	16	3	6	7	2	8	9

One just needs to check if every row, column and sub block has all the numbers from 1 to 16. So an *algorithm* (a computer program) would be to go over all *assignments* of numbers from 1 to 16, to the blanks and check if the *constraints* on rows, columns and sub blocks are *satisfied*. This might look like a simple thing to do, except when one tries to run the program.

Polynomial Time Algorithms

Suppose half the blanks in the above 16×16 puzzle instance were already filled, the number of possibilities the program has to try is $16^{16 \times 16/2}$ (try all the 16 possibilities for each of the $16 \times 16/2$ blanks). This number is more than the number (http://en.wikipedia.org/wiki/Chronology_of_the_universe) of seconds since the Big Bang and more than the number (<http://www.physicsoftheuniverse.com/numbers.html>) of particles in the universe. No matter how fast or large a computer we have, it will keep running (if we can maintain it that way) till (<http://image.gsfc.nasa.gov/poetry/ask/a10395.html>) our sun goes dead.

Hence the “try all possibilities” algorithm is definitely not acceptable. We want an algorithm which, lets say finds the solution in a day. As we saw, we would also like to solve “larger” $n \times n$ versions of the puzzle. 9×9 was the case for $n = 9$ and 16×16 for $n = 16$.

The try all possibilities algorithm that we saw, needs to try (at least) n^n possibilities. Algorithms with such a bad running time are called **exponential time** algorithms. Even for $n = 16$, n^n is a humongous number, because n is in the *exponent* (<http://en.wikipedia.org/wiki/Exponentiation>).

Suppose the number of steps the algorithm runs is n^c for some constant c , it would not be that bad. Such an algorithm is called a *polynomial time* algorithm. Finding a polynomial time algorithm for sudoku is a long standing challenge for programmers, which remains unsolved till this date. All the algorithms known so far can take n^n time on some sudoku instances.

NP-Completeness

In the 1970's, Stephen Cook, Leonid Levin and Richard Karp found that sudoku is not just a one off case, for which polynomial time algorithms cannot be found. They showed that the same mystery exists for a whole class of puzzles, called NP-Complete (NPC).

They showed this, by devising a way of solving any one puzzle in NPC, by using solutions to any other puzzle in NPC. That is for any pair of puzzles, say sudoku and kakuro (<http://en.wikipedia.org/wiki/Kakuro>) in NPC, they showed how to convert an instance of sudoku to an instance of kakuro such that a solution to the kakuro instance can be converted back to a solution to the sudoku instance. If that was confusing, what it means is that,

if there is a polynomial time algorithm for any one puzzle in NPC, that would mean there is a polynomial time algorithm for all puzzles in NPC.

Till today, thousands of problems has been found to be NP-Complete and we dont have algorithms that runs in time much better than n^n for all of them. By the result mentioned above, if there was a better algorithm for any one of them, then there is an algorithm with very similar running time for each NP-Complete problem.

Why Care?

Who cares about solving puzzles when there are more pressing problems in the world. These might appear to be questions that a jobless unrealistic philosopher might ponder about. Though it is not so. A lot depends on finding answers to these questions.

Nobody would disagree that internet is one of the greatest things to happen in end of the previous century. An underlying technology that makes it possible is encryption. That is, a way of sending messages so that only the intended person is able to read it.

All the existing methods of encryption are based on the assumption that NPC problems does not have polynomial time algorithms. That is

if somebody comes up with a fast polynomial time algorithm for solving sudoku puzzles, he can break all the codes in the internet.

Dont panic!. Your internet banking accounts are safe. We dont expect this to happen. However, we need to know what other problems are in NPC. This will help us design better encryption systems and many other things (http://en.wikipedia.org/wiki/Zero-knowledge_proof).

A Philosophical Detour

Though most people are interested in solving those pressing problems of the world, there are still some stupid harmless philosophers who find it interesting to think about these for the sake of curiosity. They find a "deeper" meaning to this silly question about solving puzzles.

Mathematical proofs, sudoku solutions are all easy to verify. Does that mean they are easy to come up with? Can human creativity be truly automated?

If NPC puzzles had fast polynomial time algorithms, then in many cases humans can be replaced by computers in solving problems. The concepts of puzzles and NPC described in this article could be defined unambiguously in the language of maths. Scientists have been hard at work for over 40 years either trying show NPC does not have polynomial time algorithms or finding fast algorithms for these problems. Some day, hopefully soon, we will have an answer.


Conclusion

The results mentioned above increases our faith in the conjecture that NP-Complete problems does not have polynomial time algorithms.

This conjecture could be disproved if any one of the thousands of NP-Complete problems has a polynomial time algorithm, but yet it has stood the test of time for over 40 years. Assuming this conjecture gives a singular explanation for our inability to design good algorithms for many problems.

Further Reading

- What to do, when a puzzle is NP-Complete? ([/2014/approximation-limits.html](http://2014/approximation-limits.html))

 (<http://creativecommons.org/licenses/by/4.0/>)

This work is licensed under a Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>).