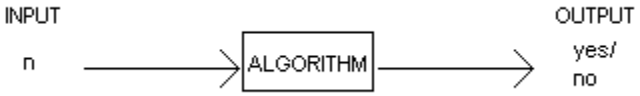
The P Vs NP Problem, NP Completeness, and Minesweeper

Justin Palumbo
Newer Math
12-11-03

The P vs. NP Problem

The P vs. NP problem is a long-standing widely known dilemma in the fields of mathematics and computer science and it has become very important to a great number of people working in those fields. In fact, it has become so important to some people that the Clay Mathematics Institute is willing to pay one million dollars to anyone who can solve it and finally put it to rest (#3). The applications of a solution to the P vs. NP problem are numerous and a solution would either dramatically increase the efficiency of scheduling, organization, traveling, and hundreds of other human endeavors or show that there is no way to do so.

P and NP are two sets of decision problems that can be solved by computers. A decision problem is a problem with a yes or no answer. A successful algorithm to find the solution to a decision problem takes some input n, performs a series of steps to answer the question, and outputs a yes or no answer (#4). The figure below gives a graphical representation of a decision problem's solution algorithm.



One example of a decision problem is, "Is n composite?" A successful solving algorithm would take the input n, perform a number of steps, and output yes if n is composite or output no if it is instead prime. Another example is the "subset sum problem": given a set of positive integers $Q=\{a_1, a_2...a_n\}$ and a positive integer S, is there a subset of the given set that sums to S? Here the input is the given set and the integer S. Yes is the output if an appropriate subset exists and no is the output otherwise.

It is very useful to evaluate how efficient a solution to a problem is. Because of the nature of computers, an algorithm's efficiency (also called its computational complexity) is usually given as the number of steps it takes to complete an algorithm as a function of the length (or number of characters) of the input. For example, if the input to an algorithm were 16732 its length would be considered to be 5. This way of looking at the input is useful because computers must parse input into its separate characters before they can do anything with it (#4).

A problem is in P if it can be solved on a standard computer in a number of steps that varies with the length of the input as a polynomial. Let #n be the length of the input. Then #n+2, #n^3+#n, and #n^99 are all examples of polynomial running times. A standard computer is defined as essentially what its name suggests. It can add, multiply, subtract, divide and do a host of other things normally associated with computers with the added stipulation that it is given an unlimited supply of memory (#1).

A problem is in NP if it can be solved in polynomial running time on a non-deterministic computer. A non-deterministic computer is a hypothetical machine with all the capabilities of a standard computer plus one added ability: if asked to guess a number, it will always make the right guess if such a guess is possible. It cannot, however, be sure whether or not it has made the right guess - the guess must be verified. This prevents such a computer from simply guessing the answer (#1).

For example, a non-deterministic solution for "Is n composite?" as mentioned above is as follows.

1. Guess numbers x and y such that x*y=n
2. If x*y=n, then output yes. Otherwise, output no.

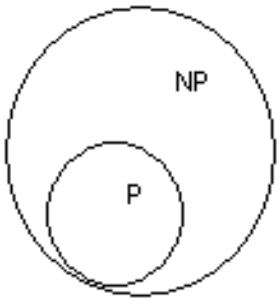
As a further example, a non-deterministic solution to the "subset sum problem" defined above is as follows.

1. Guess a set of numbers R such that R is a subset of Q and the sum of all the numbers in R is equal to S.
2. If the sum of all the numbers in R is equal to S, output yes. Otherwise, output no.

Trying to solve the "subset sum problem" deterministically is very difficult and so far no one has found such a solution that has a polynomial running time (one that is in P) but no one is sure that one doesn't exist. Deterministic solutions require looking through huge lists of possible subsets with the correct sum and doing so is very inefficient. The guess ability of non-deterministic machines makes such searching unnecessary. If available, the object of the search is automatically found.

What the P versus NP problem boggles down to essentially is whether or not there is ever a situation where the "guess" ability of a non-deterministic computer is ever necessary to solve a problem efficiently. It may be that there is a way to circumvent the need for "guessing" in situations where looking through huge groups of items is necessary but no one is sure.

Because non-deterministic computers have all the abilities of standard computers, it follows that anything in P is in NP and the two sets of problems can be represented as follows.



Thought of this way the P vs. NP question becomes the question of whether or not the area in NP and outside of P exists. Does P=NP or do there exist problems that can't be solved in polynomial time without guessing? Although there are a great number of known NP problems where a deterministic polynomial solution has not been found, there are no NP problems that have been proven to not have such a solution.

The Satisfiability Problem (SAT) and NP-Completeness

The Satisfiability Problem (or SAT) is an NP problem and a very important one - both to this paper and to the P vs. NP dilemma itself. To familiarize oneself with the problem one must have at least a rudimentary understanding of Boolean circuitry.

A Boolean circuit is made up of wires and gates. Wires transmit bits of data - carrying a value of either 0 (false) or 1 (true) until they terminate or reach a gate. A gate takes one or two wires as its inputs and outputs to another wire a new value after performing some function on the values given by the inputs. The three 'main' types of gates are the AND gates, the OR gates, and the NOT gates. An AND gate takes two inputted bit values and outputs true if both are true and false otherwise. An OR gate takes two inputted bit values and outputs false if both are false and true otherwise. A NOT gate simply takes one inputted bit value and outputs the opposite value - true if given false or

false if given true.

The SAT problem is as follows: given a Boolean circuit with some number of wires serving as inputs and one wire serving as an output, is there a combination of input values that will result in an output of true? A non-deterministic polynomial running-time solution is easy; guess the input values and verify that true is outputted at the end of the circuit. Checking an answer here is very easy (#4).

Deterministically, no polynomial running-time solution has been found. The obvious deterministic solution is to check every combination of inputs - and with 2^n possible combinations (where n is the number of inputs) this algorithm is definitely not in P.

In 1971 Stephen Cook proved that it is possible to reduce every NP problem to SAT using an algorithm with a polynomial running time. What this shows is that if SAT is found to be in P then every NP problem must also be in P since composing two polynomial running times together produces a polynomial running time. The SAT problem and others with this distinguishing property are known as NP-complete problems (#4).

The Minesweeper Consistency Problem and NP-Completeness

The computer game Minesweeper was invented by Robert Donner of Microsoft and has been included with every Windows operating system since Windows 3.1 in 1992 (#6). Thanks to the simplicity of the game's premise and the complexity of an adequate solution, not to mention the overwhelming success of Bill Gates's company, Minesweeper has become a name familiar to most people who have experience with a computer. However, that doesn't necessarily imply that the actual rules of the game are well known.

In Minesweeper, the player is presented with a rectangular grid of spaces much like a chess or checkers board. A number of mines are randomly and secretly distributed throughout the board. The player is asked to choose a space; if he chooses one with a mine he has lost the game. Otherwise, the space is revealed to contain a number. This number is equal to the number of mines surrounding that space. (Note: In the Windows version of the game, spaces with the number zero are shown as blank and all surrounding spaces are automatically revealed to save time for the player). Using this information the player can now select another space, and play continues in this way until all the spaces empty of mines are revealed and the player wins or until a space with a mine is revealed and the player loses.

Sometimes the correct move for a player to make is very obvious. For example, finding a space safe from mines in the following figure (taken from #6) is not terribly difficult.

When playing the game it is easy to take for granted the fact that a definite solution exists. For example, in the next figure (taken from #5) it is not immediately clear that a combination of mines to satisfy the known numbers even exists. (There is a definite solution - see if you can find it.)

Given situations like the above, it is easily seen that Minesweeper would be much more difficult if the player were responsible for making sure that a board is consistent with the rules of the game. In order to know for sure he would have to find a possible location for the mines - standard for play anyway - but also account for the fact that maybe there is no possible location for the mines. It is this problem, the Minesweeper Consistency Problem, that will serve as the link between the computer game and NP-completeness. The problem is a decision problem; the input for a solution is the minesweeper board and the output is the yes or no answer to the question "Is this board consistent with the rules of the game?"

The next Minesweeper configuration is an important one. (Note: All of the following Minesweeper configurations and the logic surrounding them are taken from Richard Kaye's paper - reference #1 - and are the product of his cleverness.)

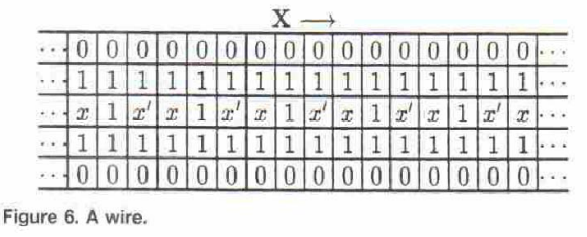


Figure 6. A wire.

Here x and x' represent unidentified spaces and without too much difficulty it can be shown that there are just two valid possibilities for mine locations. Either the spaces marked x contain mines and those marked x' do not, or those marked x' contain mines and those marked x do not. This board can be expanded to any length or terminated and

because of this the wire can be seen as “transmitting” the value of x much like a wire in a computer transmits the value of a bit, where x being a mine is the ‘true value’ or the 1 bit, and x’ being a mine is the ‘false value’ or the 0 bit. The value of x can thus be transmitted to another board, for example, the following.

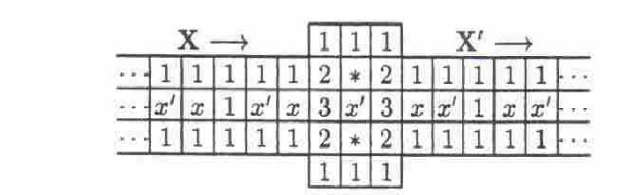


Figure 9. A NOT gate.

This board also transmits data but as the symbols show and as a small amount of tracing will prove the location of the x and the x’ spaces are switched after passing through the middle chunk. In fact, this configuration serves exactly as a NOT gate in Boolean circuits - x is inputted and NOT x is outputted.

It turns out that with enough work any Boolean circuit can be simulated with Minesweeper boards. Following9 are some more, although these are far uglier to prove correctly functioning.

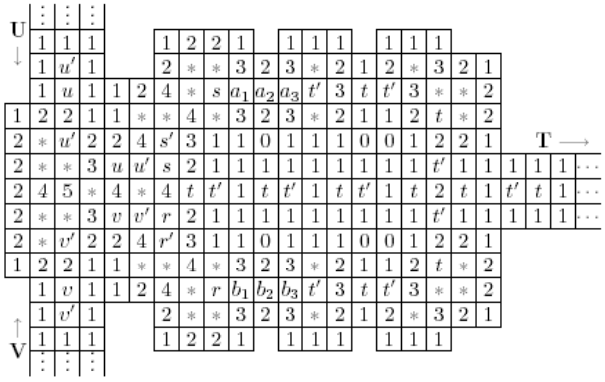


Figure 9: An AND gate.

U and V are input - U AND V is outputted.

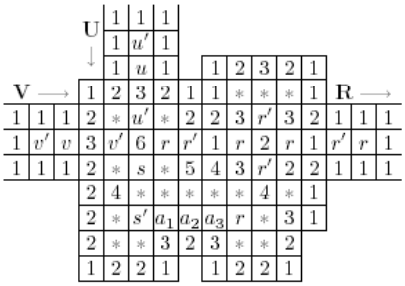


Figure 10: An OR gate

U and V are input - U OR V is outputted.

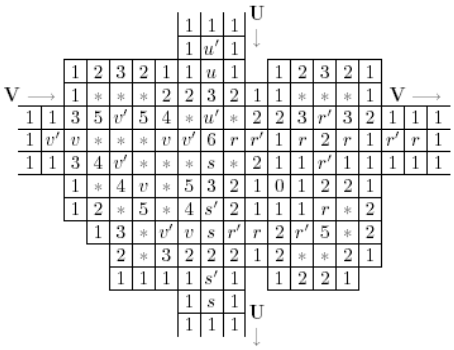


Figure 11: A wire crossing

This configuration allows two wires to cross without interfering with the data transmitted.

Other facets of Boolean circuits not talked about here - NAND gates and XOR gates and the other types of wires - can also be simulated through Minesweeper configurations allowing for any imaginable Boolean circuit to be emulated by a Minesweeper board. With every possible Boolean circuit representable by a Minesweeper configuration it can be shown that the Satisfiability problem is in fact just a special case of the Minesweeper Consistency Problem - the case where the Minesweeper board input is comprised exclusively of smaller boards that are equivalent to Boolean circuits. All this is then superimposed onto a much larger rectangular board that contains the Minesweeper Boolean circuit and spaces clear of mines everywhere else.

Since the Minesweeper Consistency Problem is equivalent to the Satisfiability problem it follows that by definition the Minesweeper Consistency Problem is NP-complete. This means that if a deterministic polynomial solution is ever found than P=NP and the P vs. NP problem can finally be put to rest. Right now, the only known deterministic solution that is effective for every conceivable board is to check the inputted board against every possible consistent configuration of the same size - a task with a horrifyingly large and non-polynomial running time.

When Robert Donner invented the game and when Bill Gates spread it to the masses, neither knew that they were sending a variant of one of the greatest problems of math and computer science with it. Perhaps their unwitting efforts will someday bring about a solution when a Minesweeper genius comes forward with a solution. What all of this really proves is that Minesweeper is a good game that will never become boring for lack of difficulty.

References

1. Richard Kaye, "Minesweeper is NP-complete", Mathematical Intelligencer, vol. 22, no. 2 (2000) pp. 9-15;

2. "Richard Kaye's Minesweeper Page." http://web.mat.bham.ac.uk/R.W.Kaye/minesw/minesw.htm

3. Ian Stewart. "Minesweeper." Hosted by the Clay Mathematics Institute. http://web.mat.bham.ac.uk/R.W.Kaye/minesw/minesw.htm

4. "Eric Weisstein's World of Mathematics." http://mathworld.wolfram.com/

5. "The Minesweeper Page." http://www.frankwester.net/winmine.html

6. Walter Schneider. "Matthews: the Archive of Recreational Mathematics - Minesweeper." http://www.wschnei.de/puzzles/minesweeper.html