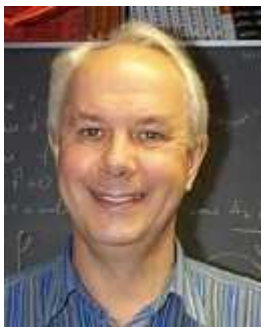
Million-Dollar Minesweeper

Lecture: November 1, 2000 (Video Online)

Ian Stewart, Department of Mathematics, University of Warwick, UK



It's not often you can win a million dollars by analysing a computer game, but by a curious conjunction of fate, there's a chance that you might. However, you'll only pick up the loot if all the experts are wrong and a problem that they think is extraordinarily hard turns out to be easy. So don't order the Corvette yet.

The prize is one of seven now on offer from the newly founded Clay Mathematics Institute in Cambridge MA, set up by businessman Landon T. Clay to promote the growth and spread of mathematical knowledge, each bearing a million-buck price-tag. The computer game is Minesweeper, which is included in Microsoft's Windows© operating system, and involves locating hidden mines on a grid by making guesses about where they are located and using clues provided by the computer. And the problem is one of the most notorious open questions in mathematics, which rejoices in the name 'P=NP?'.

The connection between the game and the prize problem was explained by Richard Kaye of the University of Birmingham, England ('Minesweeper is NP-complete', Mathematical Intelligencer volume 22 number 4, 2000, pages 9-15). And before anyone gets too excited, you won't win the prize by winning the game. To win the prize, you will have to find a really slick method to answer questions about Minesweeper when it's played on gigantic grids and all the evidence suggests that there isn't a slick method. In fact, if you can prove that there isn't one, you can win the prize that way too.

Let's start with Minesweeper. The computer starts the game by showing you a blank grid of squares. Some squares conceal mines; the rest are safe. Your task is to work out where the mines are without detonating any of them. You do this by choosing a square. If there's a mine underneath it, the mine is detonated and the game ends---with a loss for you, of course. If there is no mine, however, the computer writes a number in that square, telling you how many mines there are in the eight immediately adjacent squares (horizontally, vertically, and diagonally).

If your first guess hits a mine, you're unlucky: you get no information except that you've lost. If it doesn't, though, then you get partial information about the location of nearby mines. You use this information to influence your next choice of square, and again either you detonate a mine and lose, or you gain information about the positions of nearby mines. If you wish, you can choose to mark a square as containing a mine: if you're wrong, you lose. Proceeding in this way, you can win the game by locating and marking all the mines.

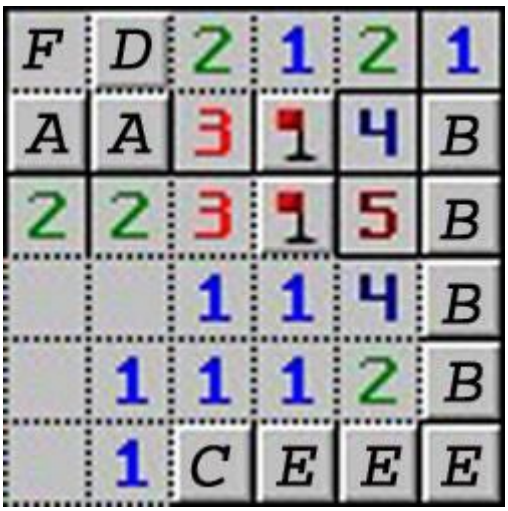


Fig.1 A Typical Minesweeper Position

For instance, after a few moves you might reach the position shown in Fig.1. Here a flag shows a known mine (position already deduced), the numbers are the information you've gotten from the computer, and the letters mark squares whose status is as yet untested. With a little thought, you can deduce that the squares marked A must contain mines, because of the 2's just below them. The squares marked B must also contain mines, because of the 4's and 5's nearby. In the same way, C must contain a mine; and it then follows that D and E do not. The status of F can then be deduced, after a few moves, by uncovering D and seeing what number appears.

Now, the P=NP? problem. Recall that an algorithm is a procedure for solving some problem that can be run by a computer: every step is specified by some program. A central question in the mathematics of computation is: how efficiently can an algorithm solve a given problem? How does the running time--- the number of computations needed to get the answer--- depend on the initial data? For theoretical purposes the main distinction is between problems that are of type P--- polynomial time --- and those that are not. A problem is of type P if it can be solved using an algorithm whose running time grows no faster than some fixed power of the number of symbols required to specify the initial data. Otherwise the problem is non-P. Intuitively, problems in P can be solved efficiently, whereas non-P problems cannot be solved algorithmically in any practical manner because any algorithm will take a ridiculously long time to get an answer. Problems of type P are easy, non-P problems are hard. Of course it's not quite as simple as that, but it's a good rule of thumb.

You can prove that a problem is of type P by exhibiting an algorithm that solves it in polynomial time. For example, sorting a list of numbers into numerical order is a type P problem, which is why commercial databases can sort data; and searching a string for some sequence of symbols is also a type P problem, which is why commercial wordprocessors can carry out search-and-replace operations. In contrast, the Travelling Salesman Problem--- find the shortest route whereby a salesman can visit every city on some itinerary--- is widely believed to be non-P, but this has never been proved. Finding the prime factors of a given integer is also widely believed to be non-P, too, but this has never been proved either. The security of certain cryptosystems, some of which are used to send personal data such as credit card numbers over the Internet, depends upon this belief being correct.

Why is it so hard to prove that a problem is non-P? Because you can't do that by analysing any particular algorithm. You have to contemplate all possible algorithms and show that none of them can solve the problem in polynomial time. This is a mindboggling task. The best that has been done to date is to prove that a broad class of candidate non-P problems are all on the same footing--- if any one of them can be

solved in polynomial time, then they all can. The problems involved here are said to have 'nondeterministic polynomial' running time: type NP.

NP is not the same as non-P. A problem is NP if you can check whether a proposed solution actually is a solution in polynomial time. This is --- or at least, seems to be --- a much less stringent condition than being able to find that solution in polynomial time. My favourite example here is a jigsaw puzzle. Solving the puzzle can be very hard, but if someone claims they've solved it, it usually takes no more than a quick glance to check whether they're right. To get a quantitative estimate of the running time, just look at each piece in turn and make sure that it fits the limited number of neighbours that adjoin it. The number of calculations required to do this is roughly proportional to the number of pieces, so the check runs in polynomial time. But you can't solve the puzzle that way. Neither can you try every potential solution in turn and check each as you go along, because the number of potential solutions grows much faster than any fixed power of the number of pieces.

It turns out that a lot of NP problems have 'equivalent' running times. Specifically, an NP problem is said to be NP-complete if the existence of a polynomial time solution for that problem implies that all NP problems have a polynomial time solution. Solve one in polynomial time, and you've solved them all in polynomial time. A vast range of problems are known to be NP-complete. The P=NP? problem asks whether types P and NP are (despite all appearances to the contrary) the same. The expected answer is 'no'. However, if any NP-complete problem turns out to be of type P--- to have a polynomial time solution--- than NP must equal P. We therefore expect all NP-complete problems to be non-P, but no one can yet prove this.

One of the simplest known NP-complete problems is SAT, the logical satisfiability of a Boolean condition. Boolean circuits are built from logic gates with names like AND, OR and NOT. The inputs to these circuits are either T (true) or F (false). Each gate accepts a number of inputs, and outputs the logical value of that combination. For instance an AND gate takes inputs p, q and outputs p AND q, which is T provided p and q are both T, and F otherwise. A NOT gate turns input T into output F and input F into output T. The SAT problem asks, for a given Boolean circuit, whether there exist choices of inputs that produce the output T. If this sounds easy, don't forget that a circuit may contain huge numbers of gates and have huge numbers of inputs.

Fig.2 Impossible Minesweeper position.

The link to the computer game comes when we introduce the Minesweeper Consistency Problem. This is not to find the mines, but to determine whether a given state of what purports to be a Minesweeper game is or is not logically consistent. For example, if during the state of play you encountered Fig.2, you would know that the programmer had made a mistake: there is no allocation of mines consistent with the information shown. Kaye proves that Minesweeper is equivalent to SAT, in the following sense. The SAT problem for a given Boolean circuit can be 'encoded' as a Minesweeper Consistency Problem for some position in the game, using a code procedure that runs in polynomial time. Therefore, if you could solve the Minesweeper

Consistency Problem in polynomial time, you would have solved the SAT problem for that circuit in polynomial time. In other words, Minesweeper is NP-complete. So, if some bright spark finds a polynomial-time solution to Minesweeper, or alternately proves that no such solution exists, then the P=NP? problem is solved (one way or the other).

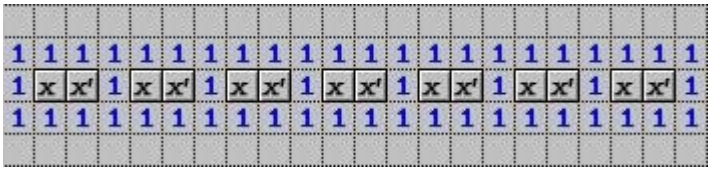


Fig.3 A Minesweeper wire.

Kaye's proof involves a systematic procedure for converting Boolean circuits into Minesweeper positions. Here a grid square has state T if it contains a mine, and F if not. The first step involves not gates, but the wires that connect them. Fig.3 shows a Minesweeper wire. All squares marked x either contain a mine (T) or do not contain a mine (F), but we don't know which. All squares marked x' do the opposite of x. You should check that all the numbers shown are correct whether x is T or F. The effect of the wire is to 'propagate' the signal T or F along its length, ready to be input into a gate.

Fig.4 shows a NOT gate. The numbers marked on the block in the middle force an interchange of x and x' on the exit wire, compared to the input wire.

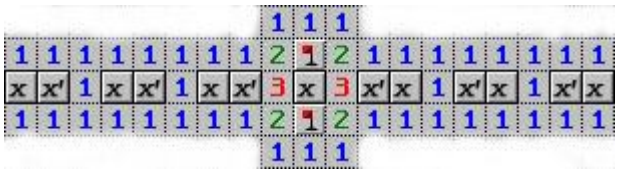


Fig.4 The NOT gate.

The AND gate (Fig. 5) is more complicated.

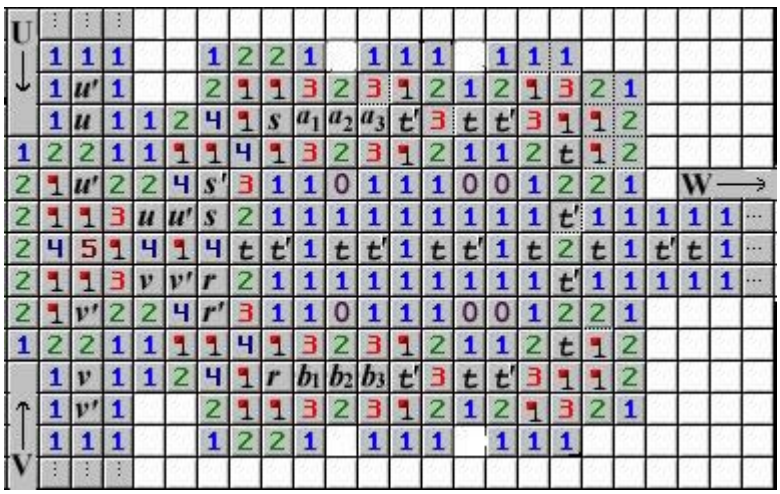


Fig. 5 The AND gate.

It has two input wires U, V, and one output W. To establish that this is an AND gate, we assume that the output is T and show that both inputs have to be T as well. Since the output is T, every symbol t must indicate a mine and every t' a non-mine. Now the 3 above and below a3 implies that a2 and a3 are mines, so a1 is not a mine, so s is a mine. Similarly, r is a mine. Then the central 4 already has four mines as neighbours, which implies that u' and v' are non-mines, so u and v are mines--- and this means that

U and V have truth-value T. Conversely, if U and V have value T then so does W. In short, we have an AND gate as claimed.

There's more to Minesweeper electronics than this--- for example, we need to be able to bend wires, split them, join them, or make them cross without connecting. Kaye solves all these problems, and other more subtle ones, in his article. The upshot is that solving the Minesweeper Consistency Problem is algorithmically equivalent to the SAT problem, and is thus NP-complete. To virtually every mathematician and computer scientist, this means that the Minesweeper Consistency Problem must be inherently hard. It is astonishing that such a simple game should have such intractable consequences, but mathematical games are like that.

If you're interested in those million-dollar prizes, a word of warning. The Clay Institute imposes strict rules before it will accept a solution as being valid. In particular, it must be published by a major refereed journal, and it must have been 'generally accepted' by the mathematical community within two years of publication. But even if you're not going to tackle anything as daunting as that, you can have a lot of fun playing Minesweeper, secure in the knowledge that it encompasses one of the great unsolved problems of our age.

Authoritative Minesweeper thanks Ian Stewart for permission to host this article. The original article appeared 1 Nov 2000 as one of seven popular lectures sponsored by the Clay Mathematics Institute. Richard Kaye at the University of Birmingham discovered Minesweeper to be NP-complete, his work being published as "Minesweeper is NP-complete" in The Mathematical Intelligencer, 22(2):9--15, Spring 2000.