A reprint from
American Scientist
the magazine of Sigma Xi, The Scientific Research Society

Candy Crush's Puzzling Mathematics

This simple game has deceptively difficult computational problems behind it, which might be why it's so addictive.

Toby Walsh

It's been said that in a city, you're never more than a few feet away from a rat. But these days it seems more likely that you're never more than a few feet away from someone playing Candy Crush Saga. It is currently the most popular game on Facebook. It has been downloaded and installed on phones, tablets, and computers more than half a billion times. Largely based on this success, its developer, Global King, listed recently on the New York Stock Exchange in an initial public offering valuing the company in the billions of dollars. That's not bad for a simple game of swapping candies to form chains of three or more identical pieces.

A big part of the appeal of Candy Crush for players is that there are complex underpinnings to the seemingly simple puzzle. Surprisingly, the game holds a lot of interest for researchers as well: It offers insight into one of the most important open problems in mathematics, as well as into the security of computer systems.

In a recent proof, I demonstrated that Candy Crush is a mathematically hard puzzle to solve (the paper is available at http://arxiv.org/abs/1403.1911). To prove this point, I needed to call upon one of the most important and beautiful concepts in the whole of computer science, the idea of a *problem reduction*. This idea maps one problem onto another, or as computer scientists like to say, it reduces one problem into another. At its heart, this concept arises because computer code is versatile: You can use the same type of code to solve more than one problem, even if the variables differ. If the problem you started with was hard, then the problem you map onto must be at least as hard. The second problem can't be easier because you must be able to solve the first problem with a computer program that can solve the second problem. And if you can show the reverse, that the second problem can also be reduced to the first problem, then in some sense the two problems are equally as hard as each other, and take a similar time to solve.

Determining the difficulty of a problem is a fundamental tenet of mathematics. But it's not a semantic point. If you can classify a problem by how hard it should be to solve, you know what kind of computing power to throw at it—and even if it's worth trying to solve at all. In some ways, at least for mathematicians, looking at Candy Crush as a math problem can be as addictive as playing it.

Hard Solutions, Easy Checks

In our analysis of Candy Crush, my collaborators and I started with the most famous class of computationally hard problems, called NP for "nondeterministic polynomial time," the "time" part of the term indicating how long these problems could take to solve. NP contains all the problems for which, if you give me a solution, I can quickly check that it is a correct answer, in a time that is just a polynomial function of the size of the problem. However, finding the solution in the first place appears to be computationally challenging. Many well-known math problems—such as determinig whether a complex logical formula can be satisfied, or whether a graph can be colored so that neighboring nodes have different colors—belong to this class of computationally hard problems.

Beneath the NP class, in terms of complexity, we have the class P of computationally "easy" problems. In this case, P stands just for polynomial. P contains problems such as sorting a list or finding a record in a database. The time it takes for an efficient computer program to solve such problems is short, even in the worst case. Mathematically, the runtime of a problem in P is a polynomial that scales to the size of the problem. For example, one well-known sorting algorithm, BubbleSort, repeatedly "bubbles" the next largest item to the top of the list like a competitor in a potato race. This process takes a time that grows as the square of the size of the list to be sorted. Even if we doubled the size of the list, the algorithm would take four times as long in the worst case. This worst case is when the list is in reverse order and every item must bubble past every smaller item. If the list is not in reverse order, the algorithm will stop even more quickly.

Above NP in complexity, we have problems that are extremely hard computationally. There are even problems above NP for which our standard model of computation, the one that all our computers implement, is inadequate. For such problems, there is no computer program that is guaranteed to stop and return an answer. These examples fall in the so-called undecidable class of problems. This class includes such questions as deciding whether a computer program will stop rather than run forever

Toby Walsh is reseach group leader at the Neville Roach Lab of NICTA (National Information Communications Technology Australia). He is also a conjoint professor in the department of computer science and engineering at the University of New South Wales, and an external professor of the department of information science at Uppsala University. E-mail: Toby.Walsh@nicta.com.au

in some loop, which computer scientists call the *halting problem.* Alan Turing, one of the fathers of computation, proved that the halting problem is undecidable. No computer program exists that can both decide whether another computer program halts and is itself guaranteed to halt, therefore making it a really, really hard computational problem.

NP lies right at the boundary between easy and hard. Within NP, we have many challenging problems such as how to route trucks to deliver parcels, roster staff in a hospital, or schedule classes in a school. It turns out that winning Candy Crush falls into this category as well. Any one of these problems can be reduced to any of the others. In this sense, they're all equally as hard.

Unfortunately, the best computer programs we have for problems in NP have a runtime that grows dramatically as we increase the size of the problem. On my desktop computer, I have a program that takes a few hours to find the optimal routing for 10 trucks and to demonstrate that this solution was the best possible. But for 100 trucks, the same program would take more than the lifetime of the universe. Mathematically, the runtime of my program is an exponential of the size of the problem.

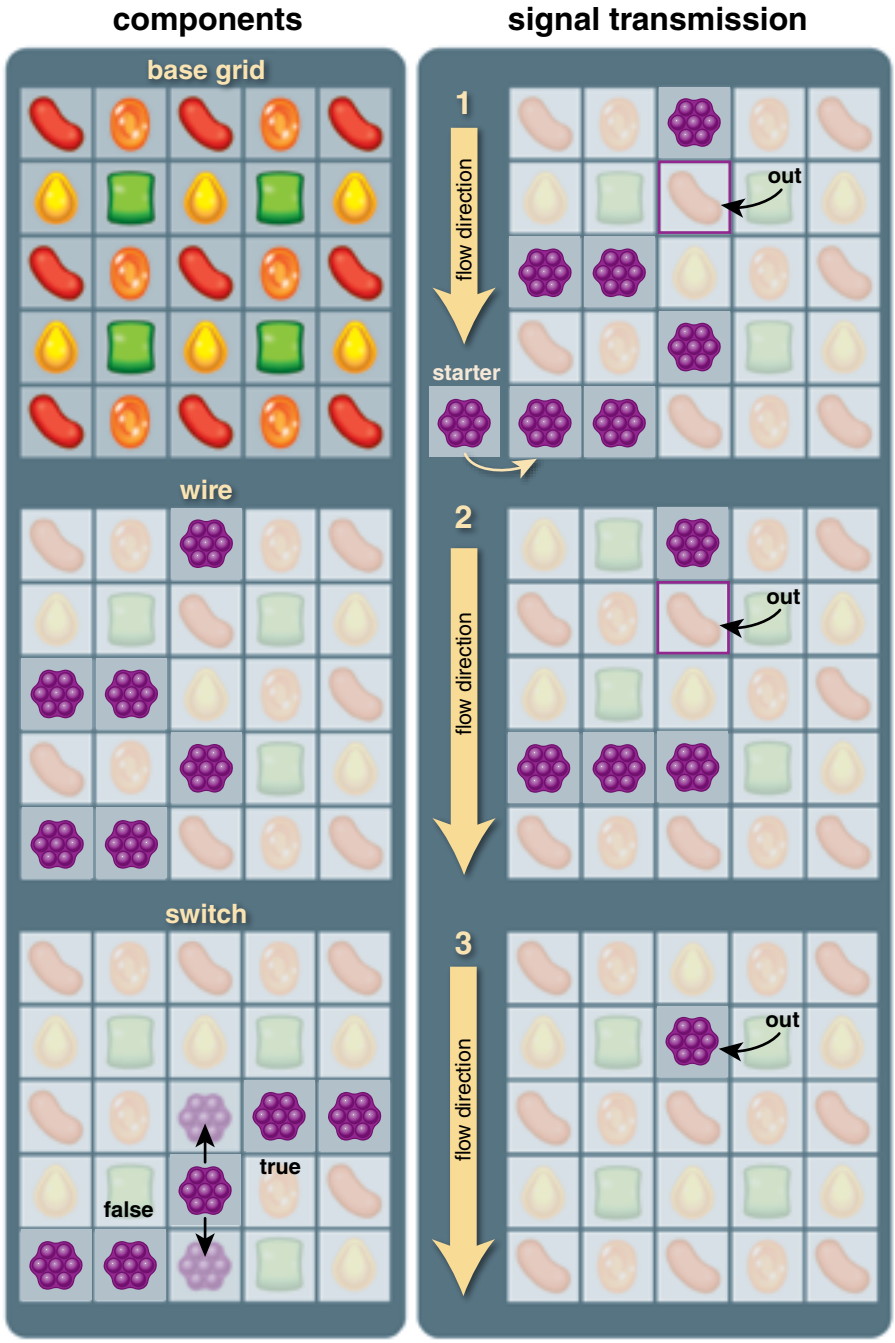
And exponentials quickly grow very large, as exemplified in the classic fable where a vizier wins any prize he wants from a sultan, and asks for one grain of wheat on the first square of a chessboard, then to have it doubled for each subsequent square. So there's one grain of wheat on the first square, two on the second, four on the third, and so on. On the 64th and final square of the board, you would need 18,446,744,073,709,551,615, or more than 18 quintillion, grains of wheat. That's approximately the amount of wheat produced worldwide in hundreds of years. Exponentials quickly sneak up on you.

Although computer scientists widely agree with my statement that NP problems are on the boundary between easy and hard, for any specific problem there is no way to know for sure which side it lies on. The best computer programs we currently have take exponential time to solve problems in NP. But we don't know if there's some exotic algorithm out there waiting to be discovered that will solve problems in NP efficiently, in polynomial time. (Mathematicians abbreviate this question as "Does P=NP?") In fact, this is one of the most important, famous open problems in mathematics today. The Clay Mathematics Institute has even offered a $1 million prize for the answer to this question. The prize remains unclaimed since it was first offered in 2000.

In this screen shot from Candy Crush Saga, a Color Bomb booster is obliterating all of the purple candies on the board. Part of the reason the game is so engaging is that it's actually quite hard, mathematically speaking. (Image courtesy of Global King.)

In the most recent poll on whether P=NP is true, 83 percent of computer scientists thought that P was not equal to NP. That is, they think there are no efficient algorithms for solving problems in NP and there never will be. Another poll of computer scientists was used to decide what to call problems that are as hard to solve as those in NP, whether or not they are in this class. The final name chosen was the rather prosaic NP-hard. But the poll did demonstrate a refreshing and geeky sense of humor: Some alternative write-ins were NP-impractical, NP-tricky, and NP-hard-ass.

The idea of problem reduction is central to the P=NP question. If we did find an algorithm that could solve any one of these problems in NP efficiently, then we also could solve all of the problems in NP efficiently. The world would be a very different place if this outcome ever happened. On the plus side, we'd be able to go about our lives with better time management, optimally routing trucks, timetabling flights, and scheduling staff to save money (and routinely winning at Candy Crush). However, we depend on other tasks, such as crack-



To prove that Candy Crush is in a class of problems called NP, it can be turned into the equivalent of a logic puzzle that is also in this class, by devising a model electrical circuit made of candies. The first component needed is a base board that has a neutral pattern of candies (*top left*). A wire is made from purple clusters (*middle left*). A switch lets a user decide which wires to use (*bottom left*). The signal travels through the wire in a process kicked off by a starter input that creates three candies in a row, which in the game are always deleted (*top right*). The next cluster moves down (*indicated by the arrow labeled flow direction*), also creating a row of three (*middle right*). When this row deletes, a candy falls into the output slot, completing the transmission (*bottom right*).

ing codes, to be computationally challenging so that our passwords and bank accounts stay secure. Computational complexity can be a blessing as well as a curse. We want to make it provably hard for hackers to compute how to decrypt messages. Equally, we need to be able to encrypt those messages easily.

This example might remind you of the definition of NP: problems where it is easy to check answers but hard to find them. Cryptography is all about putting computational barriers in the way of the bad guys. If such barriers disappear, our modern world would be in big trouble.

Behind the Game

To show that Candy Crush is a mathematically hard problem, we could reduce to it from any problem in NP. To make life simple, my colleagues and I started from the granddaddy of all problems in NP, finding a solution to a logical formula. This is called the *satisfiability problem.* You will have solved such a problem if you ever tackled a logic puzzle. You have to decide which propositions to make true, and which to make false, to satisfy some set of logical formulae: The Englishman lives in the red house. The Spaniard owns the dog. The Norwegian lives next to the blue house. Should the proposition that the Spaniard owns the zebra be made true or false?

To reduce a logic puzzle to a Candy Crush problem, we exploit the close connection between logic and electrical circuits. Any logical formula can simply be represented with an electrical circuit. Computers are, after all, just a large collection of logic gates—ANDs, ORs, and NOTs—with wires connecting them together. So all we need to do is show that you could build an electrical circuit in a Candy Crush game.

First off, we need a board on which to build the circuit. This board needs to be a neutral pattern of candies where the order of the candy types in relation to the others never changes *(see figure at left).* The candy patchwork resembles traffic lights: In even columns, we alternate red jellybeans and yellow lemon drops, whereas in odd columns, we alternate orange lozenges and green gum pieces. With such a background, even if we move columns up or down, we will never create a chain of three identical candies.

Into this framework we insert the electrical components, which are made of purple cluster candies. The clusters push aside the other candies, rather than overwriting them. Connecting these clusters creates wires to carry signals around the circuit, and multiple wires can also be linked to make more complex configurations as needed (*see figure*). If we place a purple cluster on the input to the wire at the left, we will create a chain of three purple clusters. This chain gets deleted, part of the basic premise of the game, which moves down the candies in the affected columns and propagates the signal along the wire. Eventually, a purple cluster will appear in the output on the right. A signal is thus transmitted across the board.

A scene from the TV series *Elementary*, a modern Sherlock Holmes adaptation, illustrates how NP problems have permeated pop culture well beyond Candy Crush. In the show, two mathematicians conceal their work on P=NP from rivals by doing calculations in UV markers, as discovered by Holmes (*above*) after their murder for their groundbreaking results. (Photograph courtesy of CBS.)

We also need switches that the user can set to decide which wires are active. These switches represent the choice of whether a proposition in our Boolean formula is set to true or false. The user can move the middle purple cluster either up or down. This motion will set off a signal either to the left or to the right.

Finally, we can build logic gates such as AND, OR, and NOT out of other purple clusters using these basic components. We then just have to connect switches to these logic gates with long enough wires and we have an electrical circuit that simulates our logical formula. The electrical circuit has one output bit that represents the truth of the logical formula.

Puzzle Mapping

Expressed in terms of these electrical logic circuits, the puzzle in playing Candy Crush is deciding which switches to set so that the logic gates fire appropriately and the output bit is set to true. In this way, we reduce the problem of satisfying a logical formula to solving a Candy Crush problem. And as satisfying a logical formula is a hard problem, so must be solving a Candy Crush board.

You can also show the reverse. That is, you can reduce a Candy Crush problem to satisfying a logical formula. We simply need to write down a sequence of formulae that represent the play of a Candy Crush board. Essentially you find such a logical description of Candy Crush within any program that plays it. Hence, Candy Crush is no harder than any of the problems in NP, and the game is just as hard as solving all the other problems in NP. If we had an efficient way to play Candy Crush, we would have a provably efficient way to route trucks, roster staff, or schedule classes. Alternatively, if we had an efficient way to way to route trucks, roster staff, or schedule classes then we would have an efficient way to play Candy Crush. That's the power of a problem reduction.

Computational complexity can be a blessing as well as a curse.

The next time you fail to solve a Candy Crush board in the given number of moves, you can console yourself with the knowledge that it was a mathematically hard problem to solve. Indeed, that trait may be part of what makes the game so addictive; if it were as easy to solve as tic-tac-toe, for instance, it wouldn't be nearly as engaging.

At the heart of all this is the fundamental and beautiful idea of problem reduction, which has allowed computer scientists to simplify the maze of different computational problems into a smaller number of fundamental classes such as P and NP, which computer scientists call the complexity zoo. Currently there are about 500 problem classes in the zoo, including ones with exotic names such as Δ_2P, LogFew, NEEE, and P-close. (In case you haven't worked it out yet, computer scientists love acronyms.)

In the unlikely event that P is shown to be equal to NP, the number of distinct classes in the complexity zoo drops sharply. Many classes that are thought to be different would in fact map to each other. On the other hand, if P is not equal to NP, as most computer scientists believe, then the zoo rightly contains many distinct problem classes. In fact, the zoo continues to grow in size. Complexity zoologists have recently introduced new classes to describe the complexity of problems solved with quantum computers.

The idea of problem reduction offers an intriguing possibility for Candy Crush addicts. Perhaps we can profit from the millions of hours humans spend solving Candy Crush problems? By exploiting the idea of a problem reduction, we could conceal some practical computational problems within these puzzles. Other computational problems benefit from such interactions: Every time you prove to a website that you're a person and not a bot by solving a CAPTCHA (one of those ubiquitous distorted images of a word or number that you have to type in) the answer helps Google digitize old books and newspapers. Perhaps we should put Candy Crush puzzles to similar good uses.

Our studies of Candy Crush gave us deep respect for this seemingly innocuous pastime. It actually offers insight into one of the most important open questions today in mathematics, and the implications of this question extend to many practical applications such as the encryption algorithms used to keep your bank account safe. You might like to explain this bigger picture to your boss the next time you are caught trying to get *just one more level*.

For relevant Web links, consult this issue of *American Scientist Online:*

http://www.americanscientist.org/issues/id.111/past.aspx