Satisfaction Guaranteed?  
It's Sometimes Hard to Tell

Mathematical problems tend to fall into two categories: the unexpectedly easy and the unimaginably hard. Where is the line between the two? That's what Scott Kirkpatrick and Bart Selman have been studying. The two researchers—Kirkpatrick at IBM's T.J. Watson Research Center in
Yorktown Heights, New York, and Selman at Cornell University (previously at AT&T Bell Laboratories)—have found evidence that, much as water freezes at 32°F, a certain class of problems in logic undergoes a phase transition from easy to almost impossible when a "size" parameter crosses a threshold.

The problems Kirpatrick and Selman examined come from Boolean logic, and are known as "k-satisfaction," or k-SAT, problems. In Boolean logic, variables take only two values—True and False—and they are combined using the ordinary "AND," "OR," and "NOT" operations, typically represented by the symbols ∧, ∨, and ¬, respectively. Given an expression such as \((x \land y) \lor (y \land z)\), the satisfaction problem is to determine whether there is an assignment of True/False values to the variables (in this case x, y, and z) that will "satisfy" the expression—that is, make it true. This example allows lots of satisfying assignments, such as x True, y False, and z either True or False.

A k-SAT problem has a special form. For example, \((x \lor y \lor z) \land (y \lor z \lor w) \land (y \lor z \lor v) \land (y \lor z \lor w)\) is a 3-SAT expression. It consists of several "OR" clauses, each with 3 variables (or their negations), that are "AND-ed" together. In general, a k-SAT expression can have an arbitrary number of "OR" clauses "AND-ed" together, as long as each "OR" clause consists of k variables.

Any single clause of a k-SAT problem is easy to satisfy: Just pick any one of its variables to be True or False, depending on whether that variable appears with the "NOT" symbol. Such a selection often automatically makes other clauses True as well. When there are only a few clauses in the complete expression, or when many different variables are used, it's usually easy to pick assignments that satisfy the whole expression. On the other hand, when the number of clauses is much larger than the total number of variables, it can be difficult, and often impossible, to find a satisfying assignment: Choices that make one group of clauses True almost always make other clauses False.

Kirpatrick and Selman have found that for each value of \(k\), the relevant parameter is the ratio \(M/N\) of clauses to variables. The two researchers have run computer experiments on k-SAT problems with \(k\) from 2 to 6. In each simulation, they fix the total number of variables, \(N\). Then the comput-
er creates \( M \) \( k \)-variable clauses, choosing randomly from among the \( N \) variables and their negations, and checks—using a search algorithm—whether each randomly created Boolean expression is satisfiable. Checking thousands of such expressions for a given choice of \( M \) and \( N \), the computer obtains an estimate for the probability of a random \( k \)-SAT expression with \( N \) variables and \( M \) clauses being satisfiable. The researchers then plotted this probability as a function of the ratio \( M/N \) (see Figure 1).

What they find is a set of increasingly steep curves for each \( k \). Moreover, except for the 2-SAT case, each set of curves cross at a precise "critical ratio": \( M/N \approx 4.17 \) for \( k = 3 \), \( M/N \approx 9.75 \) for \( k = 4 \), \( M/N \approx 20.9 \) for \( k = 5 \), and \( M/N \approx 43.2 \) for \( k = 6 \). These values are increasingly close to the corresponding values of \( \log 2/\log(1 - 2^{-k}) \), which is a heuristic estimate for the value of \( M/N \) at which, on average, a random \( k \)-SAT expression is satisfied by just one assignment of True/False values to its variables. (On average, each clause reduces the number of satisfying assignments by a factor of \( 1 - 2^{-k} \), so the "expected number" of satisfying assignments for an \( M \)-clause, \( N \)-variable expression is \( 2^N(1 - 2^{-k})^M \). Setting this equal to 1 and solving for \( M/N \) gives the aforementioned estimate.)

The sudden change from predominantly satisfiable to predominantly unsatisfiable is strikingly similar to phase transitions that occur in physics. That's what got Kirkpatrick, a physicist, interested in the problem. The researchers have found that techniques borrowed from statistical mechanics closely model the way the probability curves steepen at the transition point as the number of variables and clauses is increased. In particular, a technique known as finite-size scaling can be used to bring all the curves into accordance (see Figure 2).

"The nice thing about finite-size scaling is that it 'couples' experimental data from many sizes of system onto a single, universal curve," Kirkpatrick explains. "At the same time, it proves that the curve sharpens up into either a step function or a sharp delta function spike."

Kirkpatrick and Selman found the \( k \)-SAT peak to occur at parameter values that produced equal fractions of
Figure 1. Phase transition for $k$-SAT, from $k = 2$ to $k = 6$. (Figure courtesy of Scott Kirkpatrick and Bart Selman.)

Figure 2. Finite-size scaling of the $k$-SAT transitions. (Figure courtesy of Scott Kirkpatrick and Bart Selman.)
Knowing where the hardest problems are likely to occur doesn't make them any easier to solve, but at least it gives you an idea what to expect.

satisfiable and unsatisfiable formulas. Another insight has come from the work of physicists Rémi Monasson at the Ecole Normale Superieure in Paris and Riccardo Zecchina at the International Centre for Theoretical Physics in Trieste, Italy. They found that for 2-SAT, the transition is "continuous," similar to the way iron stops being magnetic at its "Curie point" temperature, whereas for 3-SAT or higher, the transition is discontinuous, more like the way ice melts.

The k-SAT problem is important because many search algorithms can be formulated in terms of satisfying Boolean expressions. In general, k-SAT belongs to a class of problems that theoretical computer scientists call $\mathbf{NP}$-complete; this suggests there may be no efficient algorithm for determining whether any given expression can be satisfied. (One exception is 2-SAT, for which an efficient algorithm is known.) If there are inherently hard k-SAT problems, where are they?

Kirkpatrick and Selman think the hard cases are concentrated at the transition point. In earlier work with David Mitchell at Simon Fraser University and Hector Levene at the University of Toronto, Selman studied the difficulty that one commonly used search algorithms, known as the Davis–Putnam procedure, encounters with 3-SAT. The three computer scientists found a peak in the number of "calls" the procedure makes—a rough indication of the amount of work the algorithm is doing—that becomes increasingly prominent as the formulas grow in size (see Figure 3). The peak is positioned near what Kirkpatrick and Selman have identified as the transition point for 3-SAT.

More recently, Monasson, Zecchina, Kirkpatrick, Selman, and Lidoro Tougawsky, a computer scientist at Hebrew University in Israel, have investigated what happens "between" 2- and 3-SAT, studying formulas with a mixture of 2-clauses and 3-clauses. They've discovered another transition point: $k \approx 2.413$. For formulas with fewer than 43.3% 3-clauses, the cost of finding a solution increases linearly with the size of the formula; for those with more than 41.3%, the cost grows exponentially.

Knowing where the hardest problems are likely to occur doesn't make them any easier to solve, but at least it gives you an idea what to expect. That way you can plan how long a hunch to take while you let the computer plug away.
WHAT'S HAPPENING IN THE
MATHEMATICAL SCIENCES
1998 - 1999

BARRY CIPRA

Edited by
Paul Zorn