# CAN'T GET NO SATISFACTION

Brian Hayes

A reprint from

American Scientist

the magazine of Sigma Xi, the Scientific Research Society

Volume 85, Number 2
March–April, 1997
pages 108–112

# CAN'T GET NO SATISFACTION

Brian Hayes

You are chief of protocol for the embassy ball. The crown prince instructs you either to invite Peru or to exclude Qatar. The queen asks you to invite either Qatar or Romania or both. The king, in a spiteful mood, wants to snub either Romania or Peru or both. Is there a guest list that will satisfy the whims of the entire royal family?

This contrived little puzzle is an instance of a problem that lies near the root of theoretical computer science. It is called the satisfiability problem, or SAT, and it was the first member of the notorious class known as NP-complete problems. These are computational tasks that seem intrinsically hard, but after 25 years of effort no one has yet proved that they are necessarily difficult. It remains possible (though unlikely) that we are simply attacking them by clumsy methods, and if we could dream up a clever algorithm they would all turn out to be easy. Settling this question is the most conspicuous open challenge in the theory of computation.

SAT also has practical importance. In artificial intelligence various methods of logical deduction and theorem-proving are related to SAT. And similar issues arise in computer software for scheduling, such as assigning flight crews to aircraft or planning the production run of an automobile factory.

In recent years SAT has attracted further attention for another reason. Although the hardest SAT problems do seem very hard, many problem instances yield easily to elementary methods. If you make up thousands of SAT problems at random, simple algorithms quickly solve all but a few of them. Looking at these results more closely, investigators discovered a curious pattern. The hard and easy instances are not mixed up haphazardly; as a certain parameter is varied, the problems go from easy to hard and back to easy again. A physicist looking at this pattern would note a resemblance to the critical behavior observed near phase transitions in fluids and magnetic materials. And indeed there is a corresponding phase transition in the SAT system: In one phase almost all the propositions can be satisfied, but in another phase almost none can. The cases that are hardest to resolve lie near the transition between these regimes.

The connection between SAT and the physics of phase transitions strikes me as a surprising one—a classic who'd-have-thunk-it result. We are accustomed to using mathematics as a tool for interpreting the physical world, but not the other way around. And yet the phase-transition model of SAT works so well that it cannot be a mere metaphor, much less a coincidence.

*Brian Hayes is a former editor of* American Scientist. *Address: 211 Dacian Avenue, Durham, NC 27701. Internet: bhayes@amsci.org.*

### P and NP

The problem of the embassy ball is small enough to be solved by even the most plodding of methods. The problem is represented by the formula:

$$(p \text{ OR } \sim q) \text{ AND } (q \text{ OR } r) \text{ AND } (\sim r \text{ OR } \sim p)$$

Here $p$, $q$ and $r$ are Boolean variables, whose only possible values are *true* or *false*. The ~ symbol indicates negation, so that $\sim p$ is read "not $p$." The logical OR operation is defined so that ($p$ OR $q$) has the value *true* if either $p$ or $q$ is *true*, whereas ($p$ AND $q$) evaluates to *true* only if both $p$ and $q$ are *true*.

With three variables, each of which can take on either of two values, there are $2^3 = 8$ possible labelings, or ways of assigning values to the variables. Trying each of the labelings in turn reveals that two of them satisfy the formula, namely $p$ = *true*, $q$ = *true*, $r$ = *false* and $p$ = *false*, $q$ = *false*, $r$ = *true*. In other words, you can either invite both Peru and Qatar or you can invite Romania alone. Every other labeling violates at least one of the royal edicts.

The brute-force enumeration of labelings is not a practical approach to larger SAT problems. For a formula with $n$ variables, the number of possible labelings is $2^n$, a function that grows so fast the search becomes exhausting rather than exhaustive when $n$ is no more than 40 or 50. The presence of such exponential growth in a computational task is a telltale sign of a hard problem or an inefficient algorithm.

Algorithmic performance is measured as a function of problem size $n$. To compare two algorithms, you observe how their execution times change as $n$ becomes arbitrarily large. For example, logarithmic, linear, quadratic and cubic algorithms have running times proportional to log $n$, $n$, $n^2$ and $n^3$ respectively. All of these algorithms are classified as polynomial-time methods; so are those described by any higher power of $n$, such as $n^5$ or even $n^{500}$.

Another group of algorithms have running time characterized by an exponential function—that is, a function where the variable $n$ appears in the exponent, as in $n^{\log n}$, $2^n$ or $n^{n^n}$. The boundary between polynomial and exponential algorithms is a kind of continental divide in computational complexity theory. All exponential algorithms are slower than all polynomial ones for large enough values of $n$. For this reason polynomial algorithms tend to be seen as fast and practical, whereas exponential ones are dismissed as hopelessly inefficient.

The same scheme that rates the efficiency of algorithms can also evaluate the difficulty of problems. A problem is said to be in the class P if there is a polynomial-time algorithm for solving it. Unfortunately, the converse assertion is not so simple to establish. Just because no one has found a polynomial-time algorithm for a problem doesn't mean the problem is not in P. Perhaps some efficient algorithm exists, but we haven't been smart enough to figure it out. Hundreds of problems remain suspended in such a computational limbo. No polynomial-time algorithm is known for them, but neither is a proof that efficient algorithms do not exist. SAT is among these unsettled and unsettling problems.

Specifically, SAT is included in the class of problems designated NP, which stands for "nondeterministic polynomial." These are problems that cannot be solved in polynomial time (as far as anyone knows), but if you could guess the answer, you could efficiently check its correctness. For SAT the checking procedure is easy. Given a proposed labeling, merely substitute the specified *true* and *false* values for all $n$ variables and make sure the resulting formula is *true*. The time needed for this computation is a linear function of $n$.

SAT is a member not only of NP but also of the more exclusive club called NP-complete. An NP-complete problem is a master key to the entire set of NP problems. If a polynomial-time algorithm could be found for any one NP-complete problem, then it could be adapted to all problems in NP. SAT was the first problem shown to have this property (by Stephen Cook in 1971). Among other NP-complete problems are some celebrated ones such as graph coloring and the traveling-salesman problem. Significantly, evidence of phase transitions and critical points has turned up in some of these problems as well.

### Backtracking

Although we have no polynomial-time algorithm for SAT, we can do better than exhaustive search. One popular method of solving SAT problems is called backtracking. The basic strategy is to explore a branch of the tree of possible solutions until you come to a dead end, then back up to some earlier choice point and try another branch. If that path also fails, you back up further still, to an even earlier decision point, until eventually you either find a solution or run out of branches.

SAT algorithms are usually designed to work on Boolean expressions written in a format called conjunctive normal form (CNF). In CNF *literals* are grouped together to form *clauses*, which are assembled into a *formula*. A literal is just a variable in either affirmative or negative form; thus $p$ and $\sim p$ are both literals. A clause is a set of literals joined by OR; ($p$ OR $\sim q$) is a clause of length 2. In a formula, clauses are linked by AND, as in ($p$) AND ($q$ OR $r$). Note that the formula given above for the embassy-ball problem is in conjunctive normal form. Any Boolean expression can be converted into a semantically equivalent CNF formula, so there is no loss of generality in focusing on this one kind of expression.

The orderly structure of a CNF formula streamlines the search for a labeling. Because all the con-

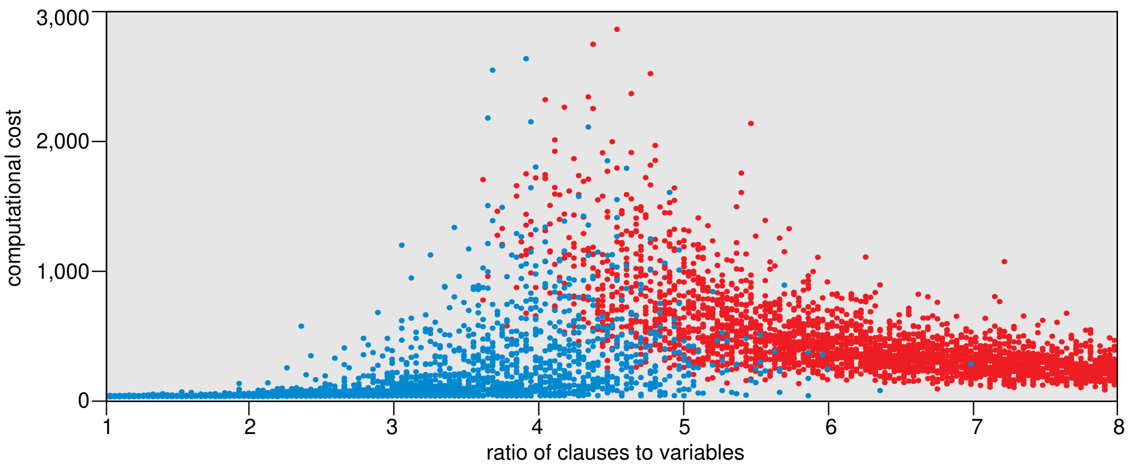


**Figure 1. Satisfiability problem, SAT, undergoes a phase transition. Each of the 5,000 dots represents a single instance of the problem; blue dots are satisfiable instances and red dots unsatisfiable. Height on the graph indicates the cost of finding a solution. The cost reaches a peak where the instances change from mostly satisfiable to most unsatisfiable.**

nectives inside each clause are OR operators, any *true* literal in a clause makes the entire clause *true*. On the other hand, because the clauses are linked by ANDs, any *false* clause makes the entire formula *false*. To put it another way, a *true* formula is any CNF expression that has no *false* clauses; note that this definition includes the empty formula, with no clauses at all. Conversely, a *true* clause must have at least one *true* literal, so that an empty clause is *false*.

The backtracking algorithm for CNF formulas can be imagined as a contest between two players, the Optimist and the Pessimist. The Optimist strives to find a satisfying labeling by looking into each clause for at least one *true* literal. If she finds one, she strikes out the entire clause. If she can eliminate all the clauses in this way, then the original formula is satisfiable. Meanwhile the Pessimist searches for *false* literals and removes them from the clauses in which they appear. If she can show that every labeling yields at least one empty clause, then the formula is unsatisfiable.

The algorithm is stated as a recursive procedure in Figure 2. Here is how it might be applied to the embassy-ball formula ($p$ OR ~$q$) AND ($q$ OR $r$) AND (~$r$ OR ~$p$). First, choose the variable $p$ and assign it a provisional value of *true*. The assignment makes the clause ($p$ OR ~$q$) true, so you can erase it; also, in the clause (~$r$ OR ~$p$), remove the ~$p$. These actions leave the reduced formula ($q$ OR $r$) AND (~$r$), to which you can now recursively apply the same procedure. Setting $q$ to *true* eliminates the first clause and leaves (~$r$) as the entire formula. Continuing in the same way, you set $r$ to *true*, but now you encounter a conflict: ~$r$ is false, and striking it leaves the empty clause ( ). You must therefore backtrack to the most recent decision point—namely the point where you set $r$ to true—and try the opposite choice. Now with the variable $r$ false, the literal ~$r$ becomes true, and you erase the entire clause (~$r$). The formula is empty. You have found the labeling $p$ = *true*, $q$ = *true*, $r$ = *false*.

The backtracking algorithm works no magic; like all other known solutions to SAT, it has exponential running time in the worst case. If you are unlucky, backtracking can take just as long as exhaustive search, but in practice it often runs much faster because it can prune whole limbs from the search tree without exploring their leaves. Performance is sometimes improved by heuristic rules for choosing which variable to label next. A particularly strong heuristic was noted by Martin Davis and Hilary Putnam as early as 1960. It suggests attending first to any variable that appears in a singleton clause—a clause with just one literal. The Davis-Putnam version of the backtracking algorithm has become a standard against which other methods are judged.

```
procedure Backtrack(formula, variables, labeling)
  If formula is empty, return labeling.
  Else if formula includes an empty clause, report failure.
  Else choose an unassigned variable, say x, and give it the value
       true. Throughout formula, if a clause includes the literal x,
       erase the entire clause; if a clause includes ~x, erase the
       ~x. Report the result of:
       Backtrack(formula, (variables – x), (labeling + (x = true))).
  Else set x to false. Throughout formula, if a clause includes the
       literal x, erase the x; if a clause includes ~x, erase the
       entire clause. Report the result of:
       Backtrack(formula, (variables – x), (labeling + (x = false))).
```

**Figure 2. Backtracking algorithm for satisfiability has exponential performance in the worst case but often does better in practice.**

## Phase Transitions

The classification of problems as P or NP is based entirely on worst-case analysis; a problem is banished from P if there is even one instance that requires an exponential solution time. But clearly the average case is also of interest. Recent statistical studies of SAT have focused on describing the distribution of hard and easy instances throughout the problem space. The paradigm is to generate a few thousand random CNF formulas, then set an algorithm churning away on them. You collect records of how many formulas can be satisfied, and how much effort is needed to find the solutions.

Most such studies are done with formulas made up of clauses that are all the same length. SAT problems with just one literal in each clause (known as 1-SAT problems) are not very interesting; a trivial linear-time algorithm solves them. 2-SAT problems also have a linear-time method, although it is less obvious. But 3-SAT—the set of formulas with three literals per clause—is NP-complete, and so it is just as hard as the more general SAT problem without restrictions on clause length.

To generate a clause in random 3-SAT, choose three distinct variables from the total set of $n$ variables, then either negate each one or leave it affirmative with probability $1/2$. To build a 3-SAT formula with $m$ clauses, repeat the process $m$ times.

It turns out the ratio of clauses to variables, $m/n$, is the crucial parameter for describing SAT statistics. Suppose you have classified a bunch of 3-SAT formulas as either satisfiable or not, and you graph the results as a function of $m/n$. One pattern you are sure to observe is that the proportion of satisfiable formulas decreases as $m/n$ increases. Formulas with only a few clauses and many variables can almost always be satisfied, since most of the variables appear only once or twice, and a conflict between them is unlikely; in this region the formulas are said to be underconstrained. At the other end of the spectrum, with many clauses and few variables, each variable can be expected to appear in many clauses, so that conflicts are frequent; here the formulas are overconstrained, and few of them are satisfiable.

This general trend from usually satisfiable to rarely so is easy enough to understand. What is harder to explain is the detailed shape of the curve *(see Figure 3)*. For small formulas (say $n$ = 10 variables) the transition is fairly gradual, but it becomes steeper as $n$ increases. At $n = 50$, the probability of satisfiability stays close to 1 for $m/n$ ratios up to about 4; then the probability falls steeply and remains close to 0 at all ratios greater than about 5. In other words, almost any formula with 50 variables and 200 clauses can be satisfied; but with 50 variables and 250 clauses, satisfiable formulas are rare. The abruptness of this transition is intriguing. And it gets even sharper, approaching the form of a step function as $n$ becomes arbitrarily large.

The steepness of the crossover is one reason for describing what happens in SAT as a phase transition. Changes of state in the physical world are similarly abrupt: Water is a liquid at 1 degree Celsius but a solid at –1 degree. The steepening of the SAT transition as the system gets larger is also a characteristic of phase changes, although a less-familiar one. When you measure size by counting atoms, just about anything is enormous, and so the "softer" phase transitions of small systems are seldom apparent in everyday experience. Nevertheless, experiments and simulations that vary the number of particles in a sample generate families of curves much like those in Figure 3.

Tabulating the effort needed to solve each problem instance brings further illumination *(see Figure 4).* At a low ratio of clauses to variables, the problems are mostly easy. At very high ratios, the effort per problem is only a little greater. In between is a hump in the curve where the average difficulty is much higher; this peak in solution cost corresponds to the crossover region in the probability graph. For any given value of $n$, the highest concentration of hard problems comes at an $m/n$ ratio near the point where 50 percent of the formulas are satisfiable. Also, as $n$ increases and the crossover becomes more abrupt, the peak in the cost curve grows dramatically taller.

Here is a qualitative explanation of the cost curve: In the underconstrained region (at a low $m/n$ ratio) a typical formula has many possible solutions, and so it takes little effort to find one. For example, the Davis-Putnam algorithm often proceeds straight to a satisfying assignment, with little or no backtracking. Overconstrained formulas, on the other hand, are almost all unsatisfiable, with dozens of literals in conflict; an algorithm will usually expose a fatal inconsistency after checking only a small fraction of the possible labelings. The middle of the curve is where problems are hard because this is the realm of just-barely-satisfiable and almost-satisfiable formulas. Here many partial labelings can be extended almost to completion before an inconsistency appears. Thus few branches of the solution tree are pruned away early.

Like the probability curve, the SAT cost curve will look familiar to students of phase transitions and critical phenomena. The canonical system for the study of critical behavior is a ferromagnet near its Curie point, which is the temperature where the material loses all magnetization. Above the

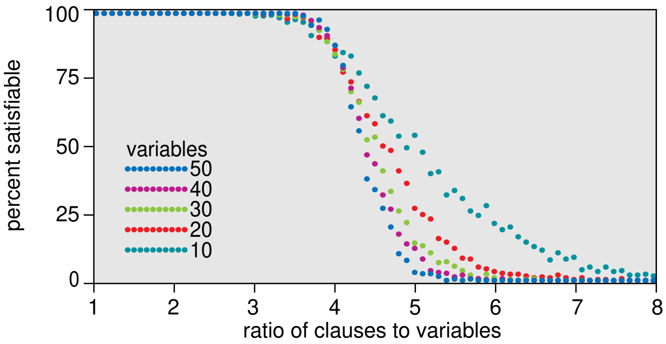


**Figure 3. Transition from satisfiable to unsatisfiable gets steeper as the number of variables increases. Each dot is the average of 300 instances.**

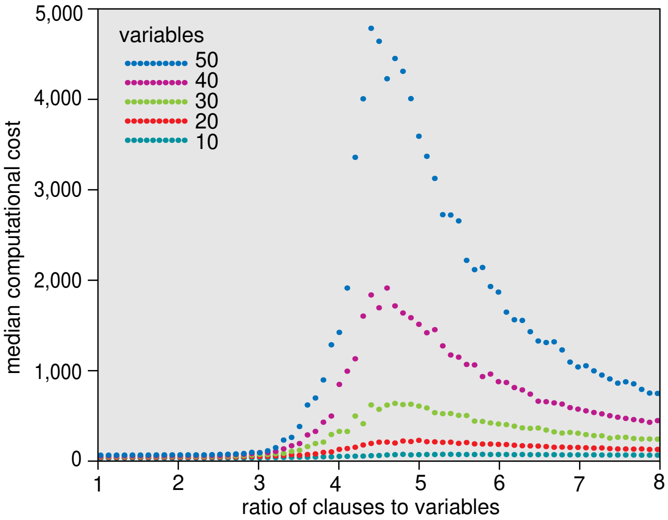


**Figure 4. Peak in the cost of finding solutions also gets sharper as the number of variables rises. Data are from the same instances as Figure 3.**

Curie temperature, the electron spins that give rise to ferromagnetism are randomly oriented, and so they cancel out and leave no net magnetization. As the material cools toward the Curie point, clusters of spins line up in parallel, and at the Curie point itself these clusters become effectively infinite in extent: A magnet is born. The Curie point also marks a sharp peak in the magnetic susceptibility—the material's sensitivity to a small external field. At high temperature, an applied field has little effect because thermal agitation disrupts any incipient magnetized regions. At low temperature the susceptibility is low again, but for a different reason: A weak external field cannot overcome the established magnetization. Near the Curie point the material is exquisitely sensitive; the smallest imposed field can reverse vast numbers of spins. A graph of the susceptibility near the Curie temperature looks just like the SAT cost curve, including a tendency for the peak to become steeper and to shift to slightly lower temperatures as the size of the system increases.

## Dissatisfactions

The idea of interpreting events in a purely mathematical system as phase transitions is not new. The earliest instance I know of was in the context of graph theory, and specifically in the study of graphs grown by randomly adding edges to connect a set of vertices. This was a field pioneered by the late Paul Erdős; in 1960 Erdős and Albert Rényi identified "sharp thresholds" in the connectivity of random graphs.

Later, in the 1980s, phase transitions in various computational systems (including SAT) were discussed by Richard Karp and Judea Pearl, by Paul W. Purdom, Jr., by Scott Kirkpatrick and R. H. Swendsen and by Bernardo A. Huberman and Tad Hogg. In 1991 a particularly influential paper was published by Peter Cheeseman, Bob Kanefsky and William M. Taylor. Titled "Where the *Really* Hard Problems Are," it reviewed evidence of phase transitions and critical points in several NP-complete problems. Cheeseman and his colleagues offered a conjecture: that phase transitions are not merely a common feature of NP-complete problems but in fact are a *defining* characteristic of all such problems.

The past six years have seen further exploration of these themes. Much of the recent work is summed up in a special issue of the journal *Artificial Intelligence*, edited by Hogg, Huberman and Colin Williams, titled *Frontiers in Problem Solving: Phase Transitions and Complexity*.

By now it seems well established that phase transitions in SAT are intrinsic to the problem itself; they are not an artifact of any particular algorithm. Furthermore, phase transitions exist not just in SAT but also in many other NP-complete problems. And yet the connection between NP-completeness and phase transitions is not a simple one. One might like to declare that if a problem has a phase transition, it must be in NP, but that is not so. There are problems in P that undergo phase changes and show the characteristic easy-hard-easy pattern; 2-SAT is among them. Conversely, there are problems in NP whose hard instances are not clustered at a phase boundary; the traveling salesman problem is an example.

In nature, phase transitions are classified as continuous or discontinuous; for example, the onset of magnetization is continuous, whereas the freezing and boiling of water are discontinuous. What about transitions in SAT? Recent work by Rémi Monasson and Riccardo Zecchina has shown that the 2-SAT transition is continuous, but the 3-SAT transition is discontinuous. Moreover, Monasson and Zecchina, together with Scott Kirkpatrick, Bart Selman and Lidror Troyansky, have devised a way of interpolating smoothly between these two regimes. Working with a model they call (2+$p$)-SAT, they generate formulas as a random mixture of clauses that have either two or three literals, in proportions determined by the parameter $p$. They find that their formulas retain the continuous transition characteristic of 2-SAT up to about $p = 0.4$, and thereafter act more like 3-SAT, with a discontinuous phase transition. This crossover point is quite different from the boundary between 2-SAT and 3-SAT in computational complexity theory. Given the worst-case assumptions of that discipline, (2+$p$)-SAT is necessarily in NP for any value of $p$ greater than zero. The average-case behavior is evidently different: Average running time grows polynomially for values of $p$ less than about 0.4, and exponentially for larger $p$.

But what *is* the average case of SAT, and how difficult is it? These questions have not yielded easy answers. They are really questions not about how to solve SAT problems but about how to generate representative sets of SAT instances. And even given a well-defined distribution of instances, measuring the average difficulty is not straightforward. The obvious measure is the mean difficulty, but mean values are so skewed by a few extremely hard formulas that most analysis has been done with medians.

Even if the measure is imperfect, knowing where the *really* hard problems are turns out to be useful, whether your aim is to find them (as in testing algorithms) or to avoid them (as in formulating real-world problems in need of solution).

## Bibliography


Cheeseman, Peter, Bob Kanefsky and William M. Taylor. 1991. Where the *really* hard problems are. In *Proceedings of the International Joint Conference on Artificial Intelligence*, Vol. 1, pp. 331–337. <http://ic-www.arc.nasa.gov:80/fia/projects/bayes-group/group/NP/>

Cook, Stephen A. 1971. The complexity of theorem-proving procedures. *Conference Record of the Third Annual ACM Symposium on the Theory of Computing*. Pages 151–158.

Davis, Martin, and Hilary Putnam. 1960. A computing procedure for quantification theory. *Journal of the Association for Computing Machinery* 7:201–215.

Erdős, Paul, and A. Rényi. 1960. On the evolution of random graphs. *Publications of the Mathematical Institute of the Hungarian Academy of Sciences* 5:17–61.

Garey, Michael R., and David S. Johnson. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. San Francisco: W. H. Freeman and Co.

Hogg, Tad, Bernardo A. Huberman and Colin Williams (eds.). 1996. *Frontiers in Problem Solving: Phase Transitions and Complexity*. A special issue of *Artificial Intelligence*, Vol. 81, Nos. 1–2, March 1996.

Huberman, Bernardo A., and Tad Hogg. 1987. Phase transitions in artificial intelligence systems. *Artificial Intelligence* 33:155–171.

Karp, Richard M., and Judea Pearl. 1983. Searching for an optimal path in a tree with random costs. *Artificial Intelligence* 21:99–116.

Kirkpatrick, Scott, and Bart Selman. 1994. Critical behavior in the satisfiability of random Boolean expressions. *Science* 264:1297–1301.

Kirkpatrick, Scott, and Robert H. Swendsen. 1985. Statistical mechanics and disordered systems. *Communications of the ACM* 28:363–373.

Mitchell, David G., Bart Selman and Hector Levesque. 1992. Hard and easy distributions of SAT problems. In *Proceedings, Tenth National Conference on Artificial Intelligence.* Pages 459–465. <http://www.cs.utoronto.ca/~mitchell/papers/ai92-hsat.ps>

Monasson, Rémi, and Riccardo Zecchina. 1996. Statistical mechanics of the random *K*-SAT model. <http://xxx.lanl.gov/abs/cond-mat/ 9606215>

Monasson, Rémi, Riccardo Zecchina, Scott Kirkpatrick, Bart Selman and Lidror Troyansky. 1996. Phase transition and search cost in the (2 + $p$)-SAT problem. In *Proceedings of the Fourth Workshop on Physics and Computation*, Boston University, pp. 229–232.

Purdom, Paul Walton, Jr. 1983. Search rearrangement backtracking and polynomial average time. *Artificial Intelligence* 21:117–133.