2012 年臺灣國際科學展覽會

優勝作品專輯

國家：United States

編號：110008

作品名稱

Do SAT Problems Have Boiling Points？

得獎獎項

二等獎

作者姓名

Soumya C. Kambhampati

Abstract

The Boolean Satisfiability problem, called SAT for short, is the problem of determining if a set of constraints involving Boolean (True/False) variables can be simultaneously satisfied. SAT solvers have become an integral part in many computations that involve making choices subject to constraints, such as scheduling software, chip design, decision making for robots (and even Sudoku!).

Given their practical applications, one question is when SAT problems become hard to solve. The problem difficulty depends on the *constrainedness* of the SAT instance, which is defined as the ratio of the number of constraints to the number of variables. Research in the early 90's showed that SAT problems are easy to solve both when the constrainedness is low and when it is high, abruptly transitioning ("***boiling over***") from easy to hard in a very narrow region in the middle.

My project is aimed at verifying this surprising finding. I wrote a basic SAT solver in Python and used it to solve a large number of randomly generated 3SAT problems with given level of constrainedness. My experimental results showed that the percentage of problems with satisfying assignment transitions sharply from 100% to 0% as constrainedness varies between 4 and 5. Right at this point, the time taken to solve the problems peaks sharply. Similar behavior also holds for 2SAT and 4SAT. Thus, SAT problems do seem to exhibit phase transition behavior; my experimental data supported my hypothesis.

Background on SAT

A simple SAT problem is shown in Figure 1. A SAT problem is specified by listing a set of variables(**P, Q** and **R** in the example), and a set of clauses (or constraints) over them (**C1, C2** and **C3**). Each constraint says that at least one of the variables it names must have the value it requires them to have. For example, **C1** says that either **P** should be false or **Q** should be false. The solution (also called "*satisfying assignment"*) to a SAT problem is an assignment of True/False values to the variables that satisfies all constraints. For our problem, a solution is **P=False, Q=True** and **R=True**. The **constrainedness** of a SAT problem is defined as the number of clauses to the number of variables (1 for this example).

If there are *n* variables, there will be $\mathbf{2^n}$ possible complete assignments, and we need to search through them to find a satisfying assignment. A backtracking SAT Solver searches over a tree of partial assignments as shown in Figure 1. Here, we start with **P** set to **True** (left branch), and then **Q** set to **True.** At this point, constraint **C1** is violated, so we backtrack and set **Q** to **False.** Next, we find that setting **R** to either value violates a constraint. This necessitates further backtracking, and trying **P** with **False.** Next **Q** is set to **True** and at this point all constraints are satisfied (so the value of **R** doesn't matter!).

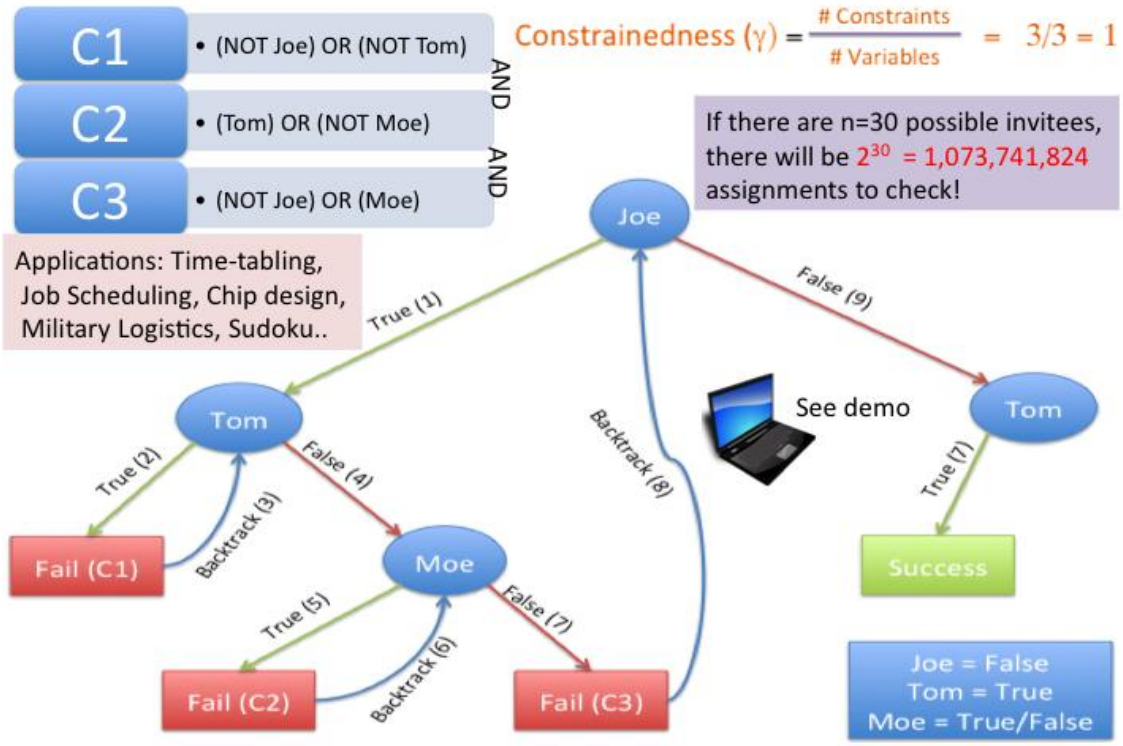

Figure 1 Boolean Satisfiability (SAT) Problem: Examples and Concepts

Many problems involving making choices subject to constraints--including how to schedule tasks, how to design chips and even how to solve a Sudoku problem--can be converted to SAT problems. Because a SAT solver has to search over $\mathbf{2^n}$ assignments, it can take a long time to solve large problems; in fact SAT is known to be an "NP-Complete" problem (in that there are no ways of solving them that can take less than exponential time in the worst case). In practice however, SAT problems have been found to be easy to solve in many cases. In fact, when constrainedness is very low, then almost any assignment is a solution, and when the constrainedness is very high, then the problems can be easily shown to be unsolvable (suppose you are trying to schedule when you will do your homework on the weekend; if there are very few assignments then it is easy to schedule them; if you have way too many assignments, you probably will give up.).

Question

The aim of my project is to find out where the hard SAT problems are. I assumed they will occur for middle values of constrainedness, but prior research says that they occur in a very narrow-band of critical constrainedness. I wanted to verify this.

Hypothesis & Variables

The hypothesis for this science project is that the hardness of solving SAT problems peaks sharply in the middle as the constrainedness increases.

My independent variable is constrainedness (γ). My dependent variables are median CPU time (in seconds) taken to solve a random SAT instance, and the percentage of instances that have satisfying assignments.

Approach

My approach consisted of writing a basic backtracking SAT solver in Python and using it to solve a large number of random SAT problems with specified degree of constrainedness. I instrumented the program to keep track of the amount of CPU time spent in solving each problem as well as the number of problems that were *satisfiable* and those that were shown to be *unsatisfiable.*

SAT Solver: I wrote a backtracking SAT solver as a recursive program in Python. The procedure is invoked with the SAT variables and constraints (clauses), and an empty assignment of variables as its input. At each recursive call, the procedure checks to see if the current assignment is violating any clause; if so, it returns with failure. Next it checks to see if the current assignment satisfies all clauses; if so, it returns the current partial assignment as the solution. If neither of the above cases hold, then it picks the next unassigned variable, and calls itself twice recursively, once with that variable

assigned True and once with it assigned false.

Generating Random SAT Problems: I wrote a program that takes **v,** the number of variables in the SAT instance, and the constrainedness parameter **γ** and makes **v* γ** random clauses. Each clause (constraint) consists of three randomly chosen variables, and each variable is randomly chosen to be either True or False. I first focused on 3SAT (all clauses of size 3), but then also experimented with 2SAT and 4SAT.

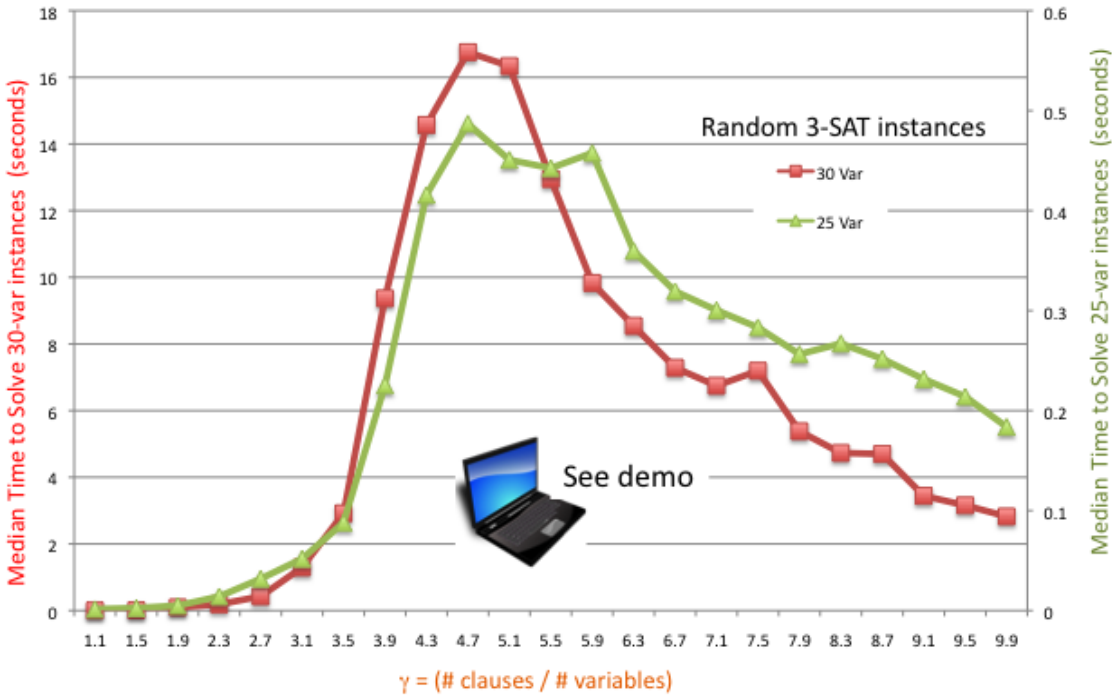

Figure 2. Hardness vs. Problem Size for random 3-SAT instances with 25 and 30 variables. We note that hardness peak for constrainedness between 4 and 5. The peak is more pronounced as number of variables increase.

Materials & Experiments

I experimented with kSAT (k=2,3,4) instances of 25 and 30 variables. In each case, I varied γ from 1 to 10 in increments of 0.4. For each value of γ, I generated 200 random SAT instances. These instances are then fed to the SAT solver and the time taken to solve the instance, as well as the fraction of the 200 instances that were found to be satisfiable were noted. (I describe results for 25 and 30 variable instances, as instances with fewer variables were too easy to solve and those with more were too hard for my

basic solver.) The experiments were all run on an iMac with 2.8 gigahertz quadcore processor and 8 gigabytes of RAM.

Results & Analysis

Figures 2-4 show the results in graphical form. I observed from the plots in Figure 2 that for both 25 and 30 variables instances, the random SAT problems' hardness peaks sharply when the constrainedness is between 4 and 5. I also observed that the transition is more abrupt for the harder 30-variable problems. From Figure 3, where the hardness and satisfiability results are juxtaposed, I observed that (i) the percentage of satisfiable problems falls sharply from 100% to 0% as constrainedness varies between 4 and 5 and (ii) the problem hardness peaks right around the place where fraction satisfiable is 50%. This shows that the hardest problems to solve are those where probability that the problem has a solution is 0.5 From Figure 4, I observed that 2SAT and 4SAT also have boiling points, and that the boiling point increases as the clauses get more loose (since more of them are needed to reach the same degree of tightness).

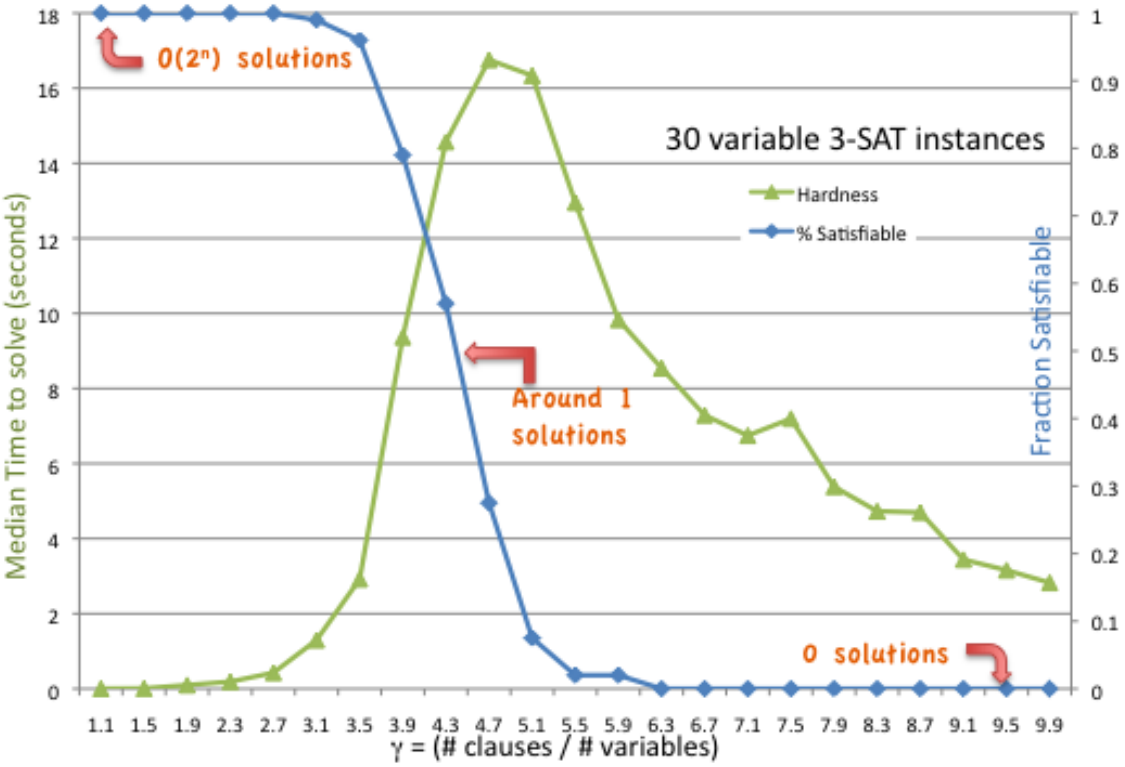

Figure 3. Plot showing how hardness and percentage satisfiability align as constrainedness increases.

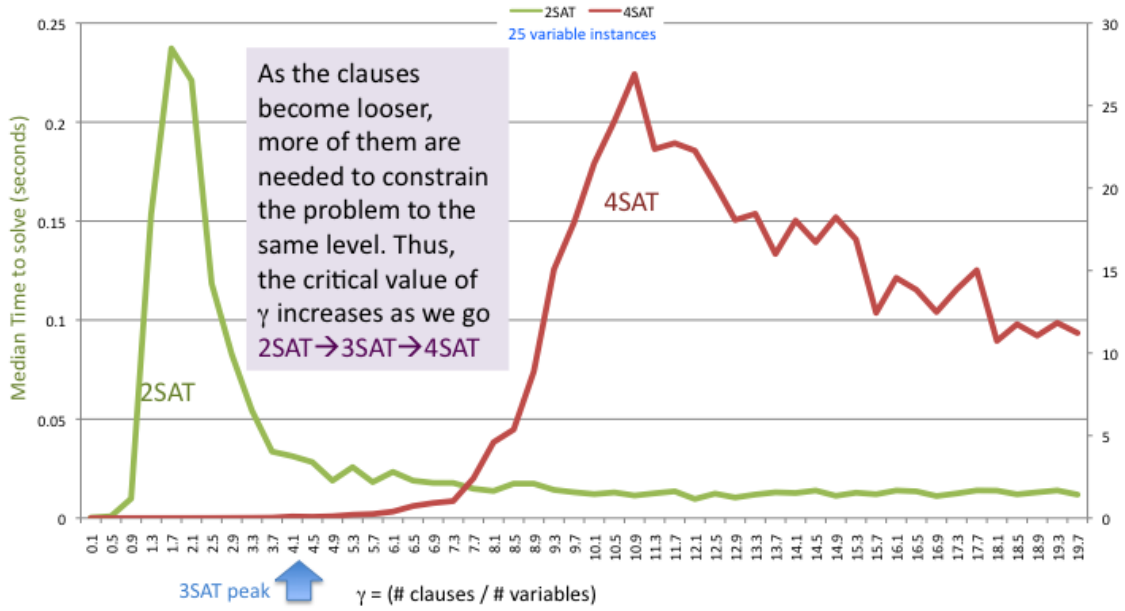

Figure 4. Plot showing that 2-SAT and 4-SAT problems boil too, but their transition points are to the either side of the transition point for 3-SAT.

Conclusion & Future Work

My hypothesis was that SAT problems would exhibit phase transition behavior. My results demonstrate that this is true. The percentage of problems with satisfying assignment transitions sharply from 100% to 0% as critical ratio varied between 4 and 5. Right at this point, the time taken to solve the problems peaked sharply. Thus, SAT problems do seem to exhibit phase transition behavior.

More recently, working with another student from my school, I have extended this work by investigating how phase transition varies with non-uniform random distributions of SAT problems. We have considered power-law (or “rich get richer” distributions), where some variables take part in clauses more often than others; and neighborhood distributions where clauses are comprised of variables that are more likely to come from the same neighborhood.

Bibliography

1. Cheeseman, Peter, Bob Kanefsky and William M. Taylor. "Where the Really Hard Problems Are." Proc. International Joint Conference on Artificial Intelligence (1991): 331-337.

2. Gomes, Carla P. and Bart Selman. "Can Get Satisfaction." Nature (2005): 751-752.

3. Gomes, Carla P., Henry Kautz, Ashish Sabharwal and Bart Selman. “Satisfiability Solvers,” Handbook of Knowledge Representation (2008):89:134

4. Lutz, Mark, “Learning Python,” O’Reilly Media Inc. (2009)

5. Hayes, Brian. "Can't Get No Satisfaction." American Scientist (1997): 108-112.

評語

Good theory work.

Good presentation.