

Generating “Random” 3-SAT Instances with Specific Solution Space Structure

Pushkin R. Pari, Jane Lin, Lin Yuan and Gang Qu

Department of Electrical and Computer Engineering
University of Maryland, College Park, Maryland 20742
{pushkin,janelin,yuanl,gangqu}@eng.umd.edu

Introduction

Generating good benchmarks is important for the evaluation and improvement of any algorithm for NP-hard problems such as the Boolean satisfiability (SAT) problem. Carefully designed benchmarks are also helpful in the study of the nature of NP-completeness. Probably the most well-known and successful story is the discovery of the phase transition phenomenon (Cheeseman, Kanefsky, and Taylor 1991). More recently, (Achlioptas et al. 2000) pointed out the importance of generating satisfiable problem instances.

In this abstract, we consider how to create 3-SAT formulas that look like random but with specific solution structures. In particular, we show that random 3-SAT formulas do not have their solutions distributed randomly in the solution space and we further develop generators to build random 3-SAT formulas with randomly distributed solutions or fixed number of solutions.

Is a 3-SAT Formula Random

There are two groups of SAT benchmarks: random SAT instances and those encoded from problems in other fields (Hoos and Stuzle, 2000). Arguably the most interesting and best studied one is the *random k-SAT* model, where each instance consists of clauses uniformly selected from all the possible clauses of the same length (Franco and Paull 1983).

Now we ask the reverse problem: *given a 3-SAT formula (or a family of formulas), determine whether it belongs to the random 3-SAT model.* Clearly it is impossible to repetitively generate random 3-SAT instances and check whether it matches the given one. Therefore, we need to extract properties from the random 3-SAT model and see whether the given 3-SAT formulas possess the same properties.

The ‘Suite I’ curve in Figure 1 represents the average occurrence of each variable from 50,000 50-variable 218-clause 3-SAT generated randomly based on (Franco and Paull, 1983). The other three curves represent 100 instances for three different families of 3-SAT formulas with the same number of variables and clauses. One can conclude that benchmark IV (represented by curve ‘Suite IV’) is not random while the other two seem to be random.

Other metrics for the randomness include: the distribu-

Copyright © 2004, American Association for Artificial Intelligence (www.aaai.org). All rights reserved.

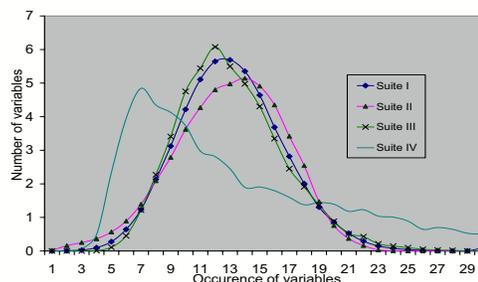


Figure 1: Clause randomness of 3SAT instances.

tion of average occurrence of each literal, each pair of variables/literals, total number of negated/unnegated variables, and the ratio of clauses with 0/1/2/3 negated variables. Such characteristics can be easily computed for the random 3-SAT model for a fixed clause-to-variable ratio and verified by extracting from large amount of random 3-SAT instances. Similar to Figure 1, benchmarks I, II and III have very similar behavior on all these criteria. This suggest to us that both II and III belong to the random 3-SAT model. However, they are created artificially to embed certain properties to the solution space.

Benchmarking with Structured Solution Space

We first create clauses based on the given solution space structure and then fine tune these clauses to make them “random” while preserving the desired properties. We show how this is performed by the example benchmark suites II and III in Figure 1. Recall that all the instances will have 50 variables and 218 clauses so they can be readily compared with the uniform random 3-SAT model.

SAT instance with given number of solutions

Our goal is to generate random-like 3-SAT formulas on n variables with m clauses and S solutions.

Phase I: generating a random core.

1. generate a uniform random 3-SAT formula with n variable and m clauses.
2. randomly select k variables, $\{v_1, v_2, \dots, v_k\}$ and delete the rest of the variables from the formula.
This gives us a random SAT with k variables consisting of 1-, 2-, and 3-literal clauses, which we refer to as random *core*.
3. find all the solutions to the core by either using a SAT solver or a simple exhaustive search. We can afford this by making k small ($k = 10$ in our experiments).

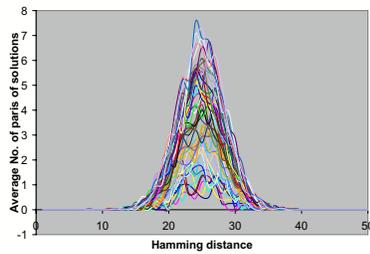


Figure 2: Solution randomness of randomly selected solutions.

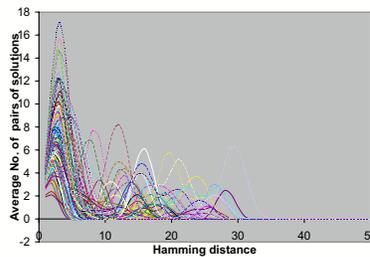


Figure 3: Solution randomness of uniform random 3-SAT instances.

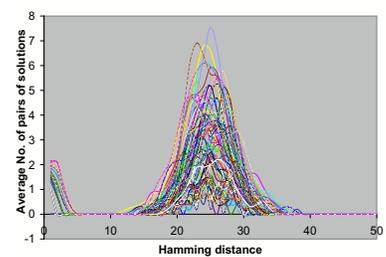


Figure 4: Solution randomness of benchmark suite III.

Phase II: adding the remaining variables to the core.

For each of the rest $n - k$ variables, we add them one by one to the core while tracking the solution database. For a variable $v_i (i = k + 1, k + 2, \dots, n)$, we explain how to add literal v_i into the core, the negated literal v_i' is added in the same fashion. When i is close to n , we pay special attention to make sure that the total number of clauses will be m and the total number of solutions will be S as required.

1. update the solution database. By introducing variable v_i , the number of solutions will be doubled.
2. based on the occurrence distribution of each literal, determine C , the occurrences of v_i ; C_2 (and C_1), the number of times v_i will appear in a 2-literal (and single literal) clause consists of the first $i - 1$ variables.
3. randomly select C_2 2-literal clauses and add literal v_i to make them 3-literal clauses. Eliminate solutions that fail to satisfy any of these newly created 3-literal clauses. If the size of the solution database is over $\frac{S}{2}$ or still far below $\frac{S}{2}$ when i is close to n , fine tune the selections of the 2-literal clauses.
4. add literal v_i to C_1 randomly selected single literal clauses; and introduce $C - C_2 - C_1$ single clauses with variable v_i only. Later variables will make these new 2-literal and single literal clauses eventually 3-literal clauses.

We have successfully generated 100 formulas using this approach with the total number of solutions varying from 1 to 5,000, the same range as we have observed from the 50,000 uniform random 3-SAT instances. These instances pass all the randomness test we listed in the previous section.

SAT instance with randomly distributed solutions

The uniform random 3-SAT model does not guarantee that the solutions are randomly distributed. One evidence is the so-called *backbone*, variables that remain constant in all the solutions (Singer, Gent, and Smaill 2000).

To get a quantitative measurement of the solution randomness, we solve the SAT formula for all the solutions and then compute the Hamming distance between each pair of solutions. Figures 2-4 plot the average number of pairs with the same Hamming distance for 100 sets of solutions, with each set contains 10-64 solutions, over three different families. In Figure 2, the solutions are selected randomly and we see the perfect bell shape of the normal Gaussian distribution. Figure 3 is based on uniform random 3-SAT formulas, where we need about 1,000 instances to find 100 instances with solutions in the range of 10-64. Figure 4 is drawn based on instances we generate to have the 10-64 solutions “randomly” distributed. Figure 4 looks much more similar to Figure 2

than Figure 3, which implies that the formulas we generate have random solutions in terms of the pairwise Hamming distance. Furthermore, the formula’s randomness has also been tested as shown in Figure 1. Due to space limitation, we briefly mention the procedure of how to build the benchmarks in Figure 4 without elaborating the details.

We start with k randomly selected solutions and then find *local clauses* that all the k solutions satisfy. By the term *local clauses*, we mean clauses consists of literals only from 10 pre-selected random variables. These clause will eliminate part of the non-solutions. We repeat this for about 100 times to ensure the randomness of clauses and the elimination of all non-solutions. Next, we remove the redundancy from clauses. A clause C in formula \mathcal{F} is *redundant* if $\mathcal{F} = \mathcal{F} \setminus C$, the formula obtained by removing C from \mathcal{F} . Whether C is redundant can be determined by checking the satisfiability of formula $C'(\mathcal{F} \setminus C)$. After removing all the redundancy, we observe that there are around 180 clauses left. We then gather the statistical information about this formula and add extra clauses to make the total number of clauses 218 and make the formula look as “random” as possible.

The existence of solutions with extremely small Hamming distance (around 2-4) in Figure 4 is due to the fact that it becomes almost impossible to eliminate all the non-solutions when large number of solutions are selected randomly. We compensate this by selecting 10-20 random solution and including their neighbors as solutions if the desired number of solutions is large.

References

- Achlioptas, D., Gomes, C. Kautz, H., and Selman, B. 2000. Generating satisfiable problem instances. In *Proceedings of the Tenth National Conference on Artificial Intelligence, Menlo Park, California*, 256–261.
- Cheeseman, P.; Kanefsky, B., and Taylor, W.M. 1991. Where the Really Hard Problems Are. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence, Sidney, Australia*, 331–337.
- Franco, J. and Paull, M. 1983. Probabilistic analysis of the Davis Putnam procedure for solving the satisfiability problem. In *Discrete Applied Mathematics*, 5:77–87.
- Hoos, H.H. and Stuzle, T. 2000. SATLIB: An Online Resource for Research on SAT. In *SAT 2000 (ed. I.P. Gent, H.V. Maaren, and T. Walsh*, pp. 283-292, IOS Press.
- Singer, J., Gent, I., and Smaill, A. 2000. Backbone Fragility and the Local Search Cost Peak. In *Journal of Artificial Intelligence Research*, 12: 235-270