EFFICIENT TOUGH RANDOM SYMMETRIC 3-SAT GENERATOR

Robert Amador[1], Chen-Fu Chiang[2], and Chang-Yu Hsieh [3]

[1,2]Department of Computer Science, State University of New York Polytechnic Institute, Utica, NY~13502, USA.
[3]This work was done while the author was a post doc at Singapore-MIT Alliance for Research and Technology, USA

ABSTRACT

We designed and implemented an efficient tough random symmetric 3-SAT generator. We quantify the hardness in terms of CPU time, numbers of restarts, decisions, propagations, conflicts and conflicted literals that occur when a solver tries to solve 3-SAT instances. In this experiment, the clause variable ratio was chosen to be around the conventional critical phase transition number 4.24. The experiment shows that instances generated by our generator are significantly harder than instances generated by the Tough K-SAT generator. The difference in hardness between two SAT instance generators exponentiates as the number of Boolean variables used increases.

KEYWORDS

3-SAT, Satisfiability, Efficient Tough Random Symmetric 3-SAT Generator, Critical Phase Transition

1. INTRODUCTION

The 3-satisfiability problem (3-SAT) can be succinctly summarized as follows: find an n-binary-variable configuration to satisfy a conjunction of clauses with each being a disjunction of three literals. It is a widely studied problem for several reasons. First, it plays a crucial role in the historical development of theoretical computer science. For instance, it was the first identified NP-complete problem, [[1], [2]] and one of the most well-studied examples in the inter-disciplinary research program involving computer science, combinatorial optimization [[3], [4]] and statistical physics [[5], [6]]. Besides these interesting developments on the theoretical front, the 3-SAT problem also plays a critical role in many applications such as model checking, planning in artificial intelligences and software verifications. Hence, for both theoretical and practical reasons, there are many strong motivations to devise more efficient algorithms to attack such a problem.

2. PROBLEM STATEMENT & MOTIVATION

The inter-disciplinary approach (especially invoking statistical physics methods and concepts) has certainly helped us to build a comprehensive picture of the complex structures of the 3-SAT problem. For instance, the concept of phase transitions in statistical physics has been adopted to elucidate the SAT-UNSAT phase transition of 3-SAT problems. In this statistical framework, the ratio parameter, ($\alpha \equiv m/n$)) for the phase transition is taken to be the ratio of the number of clauses (m) to the number of variables (n). The critical value of this order parameter is $\alpha_c = 4.2$ [[7], [8]] which clearly draws a boundary in the space of all 3-SAT instances. We explore 3-SAT problems with a critical value of 4.2 in comparison to 3-SAT problems generated by a Tough Random K-SAT Generator to better understand how the critical value effects solvability of 3-SAT Problems. Studying this specific subset of 3-SAT problems will enable further research into solving SAT problems more efficiently.

3. Background

In various SAT solvers/generators, the commonly used parameters are: n: the number of variables, m: the number of clauses, α: the ratio, which is determined by m/n. For an efficient tough random symmetric 3-SAT generator, a formula F is of n variables with the ratio number α that should have $\alpha * n$ clauses. In this work, we choose the ratio number close to the phase transition number 4.24. Particularly in 3-SAT, since each clause has 3 literals, each variable is expected to appear approximately $3 * \alpha$ times in F.

Tough SAT Generator is one of the competitive generators out there for generating tough SAT instances. We would like to compare the toughness of instances generated by our generator and TSG in the following categories: (a) frustrations caused by the generator to the SAT solver and (b) probability of generating instances that are solvable (that is there is at least one solution). The frustration rate can be quantified by the resources used by the solver, such as CPU time, restarts, conflicts and decisions. The probability can be quantified by the ratio between instances with solutions and the total instances generated by the generator.

The contribution of this work is to devise a way to generate harder instances and verify their hardness. We aim at generating those harder instances more efficiently and reliably. The hardness is quantified by the measures given in the solver that the instances require the solver to consume more resources and make more modifications. Our algorithm is more reliable as it generates with a higher probability of instances that have solutions. Our algorithm is also efficient as the generation process is almost linear time. With a verified efficient reliable algorithm that generates harder instances, in a later study we can characterize harder instances in another dimension, in addition to the conventional critical phase transition number.

3.1. Algorithms

In following paragraph, we describe the TSG algorithm (alg 1) and our efficient tough random symmetric 3-SAT generator (ETRSG) algorithm (alg 2). They can both generate SAT instances efficiently in almost linear time.

Algorithm 1 Tough Random K-SAT Generator

```
Require: k: literals per clause, n: number of variables $(v_1, ..., v_n)$, m clauses
Ensure: Tough random K-SAT formula
  Start of algorithm
  $F = \emptyset$
  for $i = 1, \cdots, m$ do
    Randomly select k random variables from $[v_1, ..., v_n]$
    for $j = 1, \cdots, k$ do
      For the chosen random variable, randomly assign negation operation
    Form clause $C_i$ based on the generated k random literals
    $F = F \wedge C_i$
  Output $F$
  End of algorithm
```

The TSG algorithm basically generates m clauses sequentially. In the 3-SAT case, each clause is generated by randomly picking 3 variables from the variable list and with 0.5 probability, the chosen variable is then assigned an negation operation. With the disjunction of the literals, a clause is formed.

The ETRSG algorithm generates m clauses sequentially. But initially it must generate a big sequence s_f that is of 3α subsequences. Each subsequence is a random arrangement of n variables. To avoid adjacent subsequences from forming an invalid clause, that is duplicated variables or literals, we must call the RndGen-Verif subroutine (alg 3) to ensure its validity. If two adjacent sequences are jointly required to produce a particular clause, the ETRSG algorithm checks the adjacent subsequences s_i and $s_{i+1}$to make sure a variable would not appear more than once in that particular clause.

Algorithm 2 Efficient Tough Random Symmetric 3-SAT Generator (ETRSG)

```
Require: n: number of variables $(v_1, ..., v_n)$, $\alpha$: desired ratio number number
Ensure: Tough random symmetric 3-SAT formula
  Start of algorithm
  $F = \emptyset$, $r = \lceil 3 * \alpha \rceil$, $m = \lceil \alpha n \rceil$, $s_f$ an empty string
  for $i = 1, \cdots, r$ do
    Call RndGen-Verif to generate a valid random sequence $s_i$ of n distinct variables
    $s_f$ = Concatenate$(s_f, s_i)$
  for $j = 1, \cdots, m$ do
    Pick 3 variables at position $(j-1)*3+1, (j-1)*3+2, (j-1)*3+3$ from $s_f$
    For the chosen random variables, randomly assign negation operation
    Form clause $C_i$ based on the generated 3 random literals
    $F = F \wedge C_i$
  Output $F$
  End of algorithm
```

Once s_f , of length $\lceil 3\alpha \rceil n$, is generated, each variable appears $\lceil 3\alpha \rceil$ times and then we can generate m clauses sequentially from position 1 until position $3m$ of s_f . For each position we also randomly assign the negation operation.

```
Algorithm 3 Random Generation Verification (RndGen-Verif)
Require: $s_f$: sequence generated so far, n: number of variables
Ensure: a valid sequence $s$ that would avoid invalid 3-SAT clauses
  Start of algorithm
  Generate a random sequence $s$ of n distinct variables
  if $((i-1)*n) \bmod 3 = 0$ then
    return $s$
  else if $((i-1)*n) \bmod 3 = 1$ then
    for Repetition of variables at the last position of $s_f$ and positions 1, 2 at $s$ do
      Regenerate $s$
  else if $((i-1)*n) \bmod 3 = 2$ then
    for Repetition of variables at the last two positions of $s_f$ and positions 1 at $s$ do
      Regenerate $s$
  Output $s$
  End of algorithm
```

3.2. Choice of Recurrence Number

The recurrence number r in ETRSG determines the number of times each variable must appear in the formula. In this paper, r is chosen based on selecting the ratio number α close to the well-known phase transition number. A phase transition [[5], [6]] is a concept utilized in statistical physics but it can also be used to better explain satisfiable and unsatisfiable transitions in 3-SAT problems. In reality, even instances with the critical phase transition number might be easy to solve for a modern solver. The 4.24 phase transition could be where more tough instances exist. In comparison to all possible instances with a critical phase transition number 4.24, this subset of tough instances might be exponentially rare among 3-SAT instances[9]. One of the major goals of this experiment is to figure out some of those exponentially rare instances and characterize them. The critical phase transition number is one of the characters for hard instances. In this experiment, we chose α= 4, 4.24 and 5. The rationale is that SAT instances with a ratio number greater than the critical phase transition number will almost always be rejected as there is no solution. SAT instances with a ratio number smaller than the critical phase transition number will likely have many solutions and therefore the SAT solvers can easily find the solution.

4. TOOLS AND EXPERIMENTS

4.1 Tools

4.1.1 Generators

The baseline generator is the Tough Random K-SAT generator [10] that generates random K-SAT instances, which is built upon latest techniques up to 2017. The other generator is our ETRSG algorithm. Both algorithms are explained in section 3.1.

4.1.2. Solver and Platform

MiniSAT is a minimalistic, open-source SAT solver, developed to help researchers and developers alike get started on SAT. It is released under the MIT license. MiniSAT deploys the Conflict Driven Clause Learning (CDCL) SAT solver with several other features such as clause deletion and dynamic variable ordering [[11], [12]]. A small glimpse into the inner workings of Minisat is provided as a basic introduction to conflict clause learning and to establish a small foothold on the basic idea of SAT solvers.

MiniSAT measures CPU time which, while valuable, is inconsequential as CPU time can change accordingly with better or worse hardware. It also provides other important measures. It stores the number of times the solver was forced to restart, conflicts, decisions, propagations, inspections and conflict literals deleted, which are all machine independent. The mechanism for the MiniSAT solver is as follows. When MiniSAT is given a SAT problem, it solves the problem by choosing a variable to begin propagation of other variables. When a conflict occurs, as in one literal is assigned both a positive and negative value, the solver will store this conflicting clause and begin propagation again from an older assignment but will avoid generating the prior conflicting clause. If the solver moves back to the original chosen variable, it is then restarted with a different variable and propagation begins again. This is redone until a satisfying assignment is found and the problem is deemed satisfiable or until it is shown that no satisfiable solution can be made, deeming the problem unsatisfiable. These are the results on which we will gauge the relative difficulty of the SAT instances.

The ETRSG algorithm was implemented in Python. The testing environment was created in cloud9, which is a cloud based ubuntu IDE. The environment has 512MB of available memory, 2GB of disk space which was more than enough for development and testing. In the case of MiniSAT, the CDCL algorithm used is ultimately machine independent because only CPU time will get better or worse with better or worse hardware respectively. Although, the times between the better and worse hardware can differ the algorithm will function the same way and have similar occurrences for restarts, conflicts, conflict literals, propagations, inspects, decisions and the rate of generating satisfiable instances.

4.2. Experiments

To compare the toughness of instances generated by TSG and ETRSG, we generate 3-SAT instances with test cases where $\alpha = 4$, 4.24 and 5. With each α, the number of variables n is set as 100, 150, 200, 250, 300 and 350. With each (a, n) pair we generate 400 instances for both TSG and ETRSG.

All the test problems were solved using the C instance of MiniSat V 1.4.1 and TSG version 1.1 K-SAT generator was used to generate the control problems.

5. RESULT

The experiment results were summarized in Figure 1 for $\alpha = 4$ case, Figure 2 for $\alpha = 4.24$ case and Figure 3 for $\alpha = 5$ case. What is worth noting is that the performance of TSG and ETRSG are almost similar when $\alpha = 4$. This could be explained that those instances are much easier to solve since there might exist multiple solutions. ETRSG remains a very stable high probability, almost 1, of generating solvable instances while TSG gradually catches up, from 0.925 to 1, as the number of variables increases. When we compare with the $\alpha = 4.24$ case and $\alpha = 5$ case, it is obvious the hardness measures, such as restart, conflict and decision, increase a couple orders of magnitude as α increases.

The experiment also yielded similar results for the $\alpha = 4.24$ and $\alpha = 5$ cases overall. Simply looking at each α case, we can conclude that *ETRSG* instances are more difficult than the *TSG* instances. The order of magnitude increases as α increases.

The more interesting phenomena we observed was that ETRSG problems retained their difficulty and overall solvability over their randomly generated counterparts. This implies there could exist a different phase transition number for ETRSG problems which can lead to further development of difficult symmetric 3-SAT problems. However, this result leads us to the conclusion that our ETRSG is more efficient in generating more *difficult* problems while maintaining solvability.

6. DISCUSSION

6.1. Critical Zone Exploration for ETRSG

With the speculation that the critical phase transition zone might be different for ETRSG problems, it might be worth discussing the exploration of this new hot and cold zone of satisfiability. Since when $\alpha = 4$, it yielded highly satisfiable problems as seen in Figure 1, we speculate the critical phase transition zone might lie beyond this point. Furthermore, with evidence from Figure 2, we speculate the crucial phase transition zone for ETRSG could be even beyond $\alpha = 4.24$ as the ETRSG problems were all still highly satisfiable. The critical zone must occur before 5 as nearly all symmetric and TSG problems were unsatisfiable. In short, this new number must occur after 4.24 but before 5 and the problem of searching for this number can be approached in a multitude of ways. This could be investigated in another study.

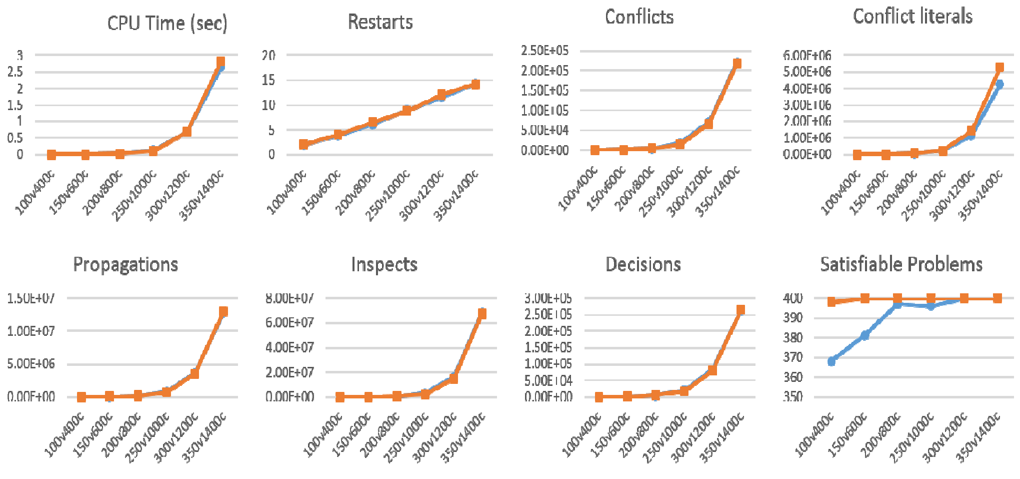

Figure 1: $\alpha = 4$, 400 instances, Red: ETRSG, Blue: TSG. ETRSG problems and the TSG problems began to relate more directly to each other, and the advantageous difficulty of the ETRSG problem was deemed inconsequential.

6.2. Toughness

As pointed out earlier, problems that occur with the typical critical phase transition number 4.24 might turn out to be easy to solve [9]. We might need a finer characterization for harder instances. As shown in this experiment, it is clear that an equal recurrence number for all variables could be one character that can be used to describe this set of harder problems. As described previously when $\alpha = 4.24$ ETRSG still generates with an increasingly high probability (0.75 to 1) solvable hard instances while TSG has a decreasing probability (0.63 to 1). The success rate drops almost to 0 when $\alpha = 5$. As for other measures, such as CPU time, restart, conflict and decision (and so on), are of a higher order of magnitude. A follow up study would focus on scaling α between 4.24 and 5 for ETRSG while keeping solvable probability high and the magnitude of difficulty increasing. Another investigation is needed to determine the cause of success probability dip for only 100 and 150 variables when $\alpha = 4$ transitions to $\alpha = 4.24$. It could be due to numerical fluctuation or some hidden factors to be discovered.

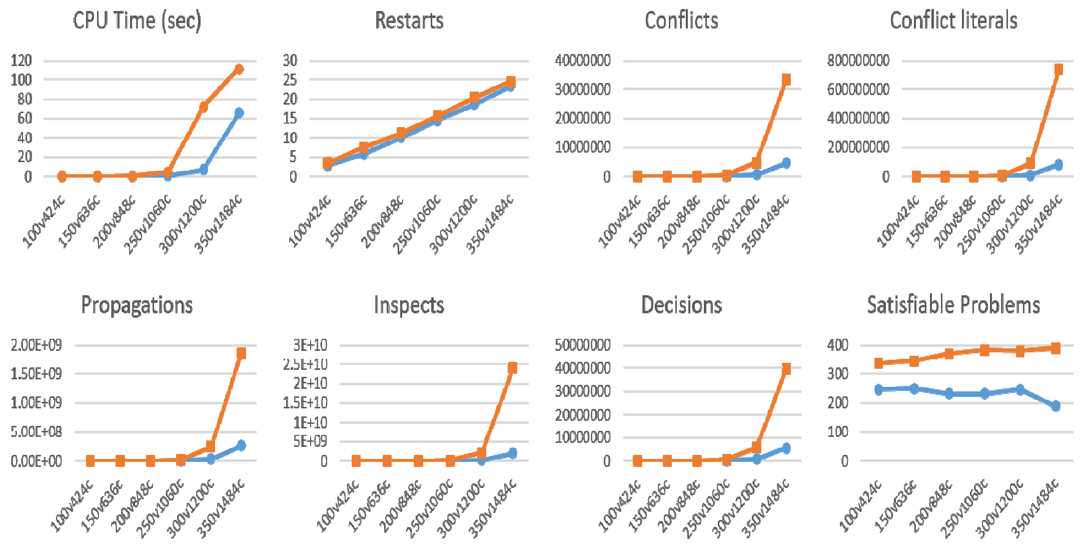

Figure 2: $\alpha = 4.24$, 400 instances, Red: ETRSG, Blue: TSG. When more than 250 variables, ETRSG instances significantly outperform TSG instances in all aspects, except with slight outperformance in restart. ETRSG has a higher probability of generating solvable instances.

7. CONCLUSION AND FUTURE WORK

As it shows in the experiment ETRSG 3-SAT problems tend to have a higher level of difficulty. This leads us to believe that the landscape of this type of problem might have many local minimums and only one unique global minimum. With such a landscape, a regular solver using Heuristics might be deceived to believe the local minimum is the global or it would take much more resources (time, space) for the solver to attack. To avoid bias, that is difficulty that has some solver dependency, we should translate the numerically-verified difficult problems into landscape problems.

Studying the landscape problem will allow us to better understand the difficulty of symmetric sat problems when compared to the relative ease of TSG 3-SAT problems. Also, as stated prior in section 5 a new phase transition number might exist for symmetric 3-SAT problems as the 4.24 ratio only applies to general 3-SAT problems. This new phase transition number will also help to shed light on difficulty and satisfiability bounds. Finally, a new partition-based solver that we are developing (for another study) can be used to tackle symmetric problems as it would be blind to the constraints of the problem as they would be broken down into smaller and more manageable problems.

Acknowledgements

R. A. and C. C. gratefully acknowledge the support from the State University of New York Polytechnic Institute.

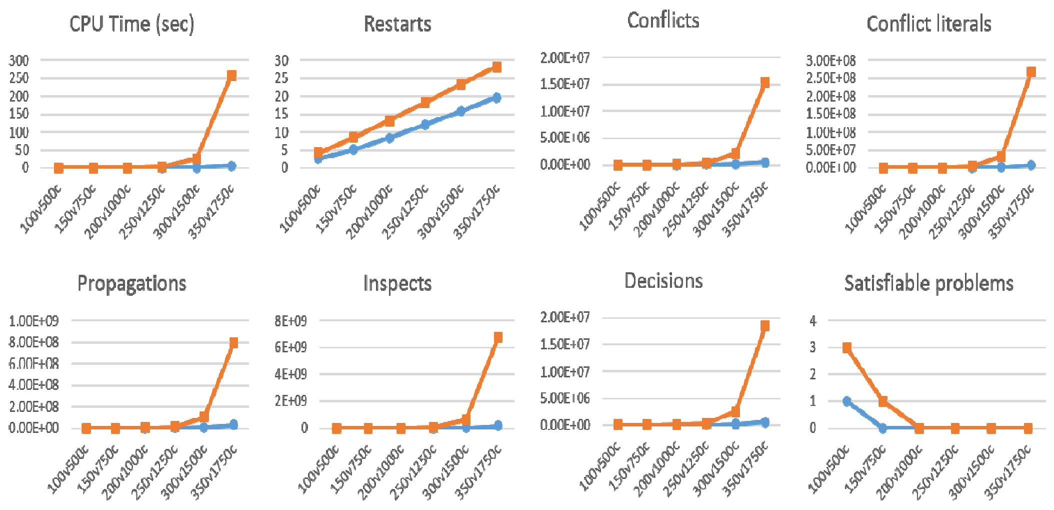

Figure 3: $\alpha = 5$, 400 instances, Red: ETRSG, Blue: TSG. Similar to $\alpha = 4.24$, but more significant in restart. The probability of generating solvable instances drops quickly to 0 for both since 5 is greater than the critical phase transition number.

References

[1] S. A. Cook, "The complexity of theorem-proving procedures," in Proceedings of the third annual ACM symposium on Theory of computing, 1971.

[2] R. M. Karp, "Reducibility among combinatorial problems," in Complexity of computer computations, Springer, 1972, pp. 85-103.

[3] C. H. Papadimitriou and M. Yannakakis, "Optimization, approximation, and complexity classes," Journal of computer and system sciences, vol. 43, pp. 425-440, 1991.

[4] R. Marino, G. Parisi and F. Ricci-Tersenghi, "The backtracking survey propagation algorithm for solving random K-SAT problems," Nature communications, vol. 7, p. 12996, 2016.

[5] S. Cocco and R. Monasson, “Statistical physics analysis of the computational complexity of solving random satisfiability problems using backtrack algorithms,” The European Physical Journal B-Condensed Matter and Complex Systems, vol. 22, pp. 505-531, 2001.

[6] A. Percus, G. Istrate and C. Moore, Computational complexity and statistical physics, OUP USA, 2006.

[7] B. A. Huberman and T. Hogg, “Phase transitions in artificial intelligence systems,” Artificial Intelligence, vol. 33, pp. 155-171, 1987.

[8] M. Mézard, G. Parisi and R. Zecchina, “Analytic and algorithmic solution of random satisfiability problems,” Science, vol. 297, pp. 812-815, 2002.

[9] M. Žnidarič, “Scaling of the running time of the quantum adiabatic algorithm for propositional satisfiability,” Physical Review A, vol. 71, p. 062305, 2005.

[10] https://toughsat.appspot.com/, “Tough SAT generation,” 2017.

[11] N. Een, “MiniSat: A SAT solver with conflict-clause minimization,” in Proc. SAT-05: 8th International Conference on Theory and Applications of Satisfiability Testing, 2005.

[12] N. Eén and A. Biere, “Effective preprocessing in SAT through variable and clause elimination,” in International conference on theory and applications of satisfiability testing, 2005.

AUTHORS

Robert Amador studies computer and information science and received his master’s from SUNY Polytechnic institute. His research interests include artificial intelligence and machine learning.

Dr. Chen-Fu Chiang studies computer science and received his master’s from the University of Pennsylvania and his PhD from the university of Central Florida. He is currently an assistant professor in the Computer Science department at the University of New York Polytechnic Institute. His research focus is on quantum computation, theoretical computation and artificial intelligence.

Dr. Chang-Yu Hsieh studies physics. He received his PhD from the University of Ottawa Canada. Upon his graduation, Dr. Hsieh had been conducting research in quantum system as postdocs in University of Toronto and MIT. His research focus is on complexity, near term quantum systems and quantum algorithms.