EXPLORATION OF HARD TO SOLVE 3-SAT PROBLEMS

Robert Amador, Chen-Fu Chiang and Chang-Yu Hsieh

***Abstract*—We designed and implemented an efficient tough random symmetric 3-SAT generator and propose two deterministic algorithms that efficiently generate 3-SAT instances with a unique solution. We quantify the first algorithms 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 two deterministic algorithms generate 3-SAT instances with the number of clauses scaling as 4n, where n is the number of variables, and (n+6), respectively. By combining these two algorithms along with a simple padding algorithm, we prove a hybrid algorithm that can generate n-variable instances with the number of clauses that scale between (n+6) and 7n(n-1)(n-2). Overall, all proposed SAT generators seek to explore unique difficult to solve SAT problems.**

***Keywords*—3-SAT, Satisfiability, Efficient Tough Random Symmetric 3-SAT Generator, Tunable Unique-Solution, Critical Phase Transition**

I. 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. 3-SAT is a widely studied problem for a multitude of reasons. Primarily, it plays a crucial role in the historical development of theoretical computer science. For instance, it was the first identified Nondeterministic Polynomial complete (NP-complete)problem,[1], [2] and it is one of the most well-studied examples in the inter-disciplinary research program involving combinatorial optimization [3], [4], computer science, and statistical physics [5], [6]. Theoretical developments aside, 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.

Robert Amador was with the Department of Computer Science, State University of New York Polytechnic Institute, Utica, NY~13502, USA. (e-mail: amadorr@sunypoly.edu).

Chen-Fu Chiang is with the Department of Computer Science, State University of New York Polytechnic Institute, Utica, NY~13502, USA. (e-mail: chiangc@sunypoly.edu).

II. PROBLEM STATEMENT & MOTIVATION

By invoking statistical physics methods and concepts we have built a comprehensive picture of the complex structures that embody the classical 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,

$$(\alpha \equiv m/n) \tag{1}$$

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

$$\alpha_c = 4.2 \tag{2}$$

[7],[8]which clearly drawsa boundary in the space of all 3-SAT instances. On one side of this boundary where$\alpha > \alpha_c$, most instances are unsatisfiable. On the other side of this boundary, most problems are satisfiable.

A. *Tough Random Symmetric 3-SAT*

Via our Efficient Tough Random Symmetric 3-SATgenerator (ETRSG) we explore 3-SAT problems with a critical phase transition value of 4.24 in comparison to 3-SAT problems generated by a Tough Random K-SAT Generator (TSG) to better understand how the critical phase transition value effects solvability of symmetric 3-SAT Problems. Studying this specific subset of 3-SAT problems will allow for further research into solving SAT problems more efficiently.

B. *Tunable Unique-Solution 3-SAT*

The Unique-PT1 & Unique-PT4 algorithms can generate 3-SAT instances with a wide range of values for the order parameter α, i.e. we can have various ratios of number of clauses to the number of variables. This is a critical point as the earlier numerical investigations [9] did not conclude what could be an optimal value of α (with respect to n) to increase the chance that a random generation algorithm could output

Chang-Yu Hsieh worked on this while a postdoc at Singapore-MIT Alliance for Research and Technology, 1 CREATE Way, #10-01 & #09-03 CREATE Tower Singapore 138602

such a hard problem. For our ongoing investigations (to be reported in a separate study), we would attempt to clarify the optimality of α with respect to n (number of variables).

III. Background

A. ETRSG and TSG

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

Tough SAT Generator is one of the more competitive generators used 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 have at least one solution and by extension are solvable. 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.

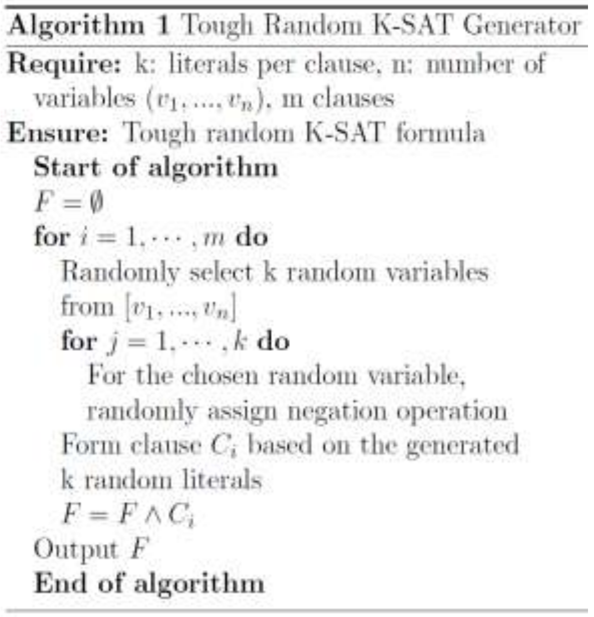
The contribution of this part of the work is to devise a way to generate harder 3-SAT problems and verify their hardness. We seek to generate harder instances more reliably and efficiently. The hardness is quantified by the measures given by the solver the instances require the solver to consume more resources and make more modifications. Our ETRSG algorithm is more reliable as it generates problems with a higher probability of being solvable. Our algorithm is also efficient as the generation process is almost linear time.

B. Unique-PT1 & Unique-PT4

Unique-PT1 & Unique-PT4, employ a naive approach that turns out to be a suitable algorithm to be run on an adiabatic quantum computer (AQC), such as D-Wave's quantum annealing approach [10], [11]. As such these algorithms will be considered separately in this work as they are inspired by our recent attempt to analyze the performance of AQC on solving 3-SAT problems.

For technical reasons, theoretical estimates of the AQC run-time can be more elegantly done if we are confined to unique-solution instances (i.e. if we only consider 3-SAT cases with a unique solution). We emphasize that the need to consider unique-solution instances is by no means a limitation on the computational capability of an AQC; rather it just makes the theoretical analyses easier. Nevertheless, this severe restriction has initially concerned us as the difficulty of 3-SAT instances surrounding a SAT-UNSAT phase transition is an “average" (or statistical) property of all problems close to this order parameter around 4:2 in the problem space [7], [8]. There is no theoretical analysis implying (with high probability) one would be able to randomly generate “difficult instances" when restricted to such a small subset of all problems surrounding the critical value ($\alpha_c = 4.2$) of the order parameter. This concern has apparently affected other researchers looking into similar issues with AQC. Much to our delight, in our quest to clarify the hardness of unique-solution problems, we have come across a set of recently identified “hard" problems [9] in this subspace. These problems can be colloquially described as “3-SAT instances with a unique solution while having many configurations violating just few clauses". In the energy-counting point of view, a large fraction of the configurations are squeezed to a small window of energy values right above the zero-energy state (the unique solution). This colloquial description certainly reminds one of the notorious protein folding problem [12]. Furthermore, to the contrary of conventional wisdom, extensive numerical investigations indicated these unique-solution hard instances are more easily found when the order parameter is smaller than the critical value (i.e. $\alpha<\alpha_c$).

IV. Algorithms



```
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
```

A. ETRSG & TSG 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.

The TSG algorithm 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 of negating the chosen variable. With the disjunction of the literals, a clause is formed, and the problem is complete.

Algorithm 2 Tough Random Symmetric 3-SAT Generator (TRSSG)

```
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
```

The ETRSG algorithm also 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, such as 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 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 clause.

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.

B. 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 4.24. A phase transition [5], [6] is a concept utilized in statistical physics but it can also be used to explain satisfiable and unsatisfiable transitions in 3-SAT problems. However, even instances within the critical phase transition number may be easy to solve when a modern SAT solver is used. overall, we use 4.24 as our phase transition number because that could be where more tough 3-SAT instances exist.

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
```

The subset of SAT problems with a critical phase transition number of 4.24 may 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 characteristics of hard to solve 3-SAT instances. In this experiment, we chose $\alpha = 4$, 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 likely 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.

C. Unique-Solution 3-SAT Instances

In the work [13], two algorithms, G and G2 are provided. We briefly describe the algorithms and the theorems. Interested readers can refer to the original work for further detailed analysis.

1) Naïve Instance Generation Algorithm G

Algorithm G is a naive approach that operates by randomly selecting a solution and then generating clauses that could be satisfied by the solution. It is shown that it requires at least

$$m = \theta(n \ln n) \tag{3}$$

to generate unique-solution instances with high probability.

Theorem 1.[13] *For any positive constant* $\delta < 1$ *if*

$$m \geq \frac{7}{3} n \ln n - \frac{7}{3} n \ln \frac{\delta}{2}, \tag{4}$$

then the formula F *generated by algorithm* G *has only one solution with probability at least* $1 - \delta$.

```
Algorithm 4 G: Naive instance
generation algorithm
Require: $(m, n)$ where $n$ is the number of
  variables and $m$ is the number of clauses
Ensure: A 3-SAT instance with certain
  probability that has a unique solution
  Start of algorithm
  Choose truth assignment
  $t \in \{0, 1\}^n$ randomly
  $F = \emptyset$
  for $i = 1, \cdots, m$ do
    Choose a clause $C_i$ that can be
    satisfied by $t$ randomly such that $F = F \wedge C_i$
  Output $F$
  End of algorithm
```

2) Modified Instance Generation Algorithm G2

Algorithm G2 randomly selects a solution and generates the first n clauses by choosing clauses violated by other assignments that are only 1 bit different from the solution. From the $n + 1$th to the mth clauses, G2 generates clauses that could be satisfied by the true assignment. It is shown that it requires approximately

$$m = 7n \tag{5}$$

to generate unique-solution instances with high probability.

```
Algorithm 5 G2: Modified Instance
Generation Algorithm
Require: $(m, n)$ where $n$ is the number
  of variables and $m$ is the number of clauses
Ensure: A 3-SAT instance with certain
  probability that has a unique solution
  Start of algorithm
  Choose a true assignment
  $t \in \{0, 1\}^n$ randomly
  let $t_1^i (1 \leq i \leq n)$ be an assignment
  that differs from $t$ at the $i$th bit
  $F = \emptyset$
  for $i = 1, \cdots, n$ do
    Choose a clause $C_i$ that the violation
    is caused by the $i$th bit in $t_1^i$ and $F = F \wedge C_i$
  for $i = n + 1, \cdots, m$ do
    Choose a clause $C_i$ that can be satisfied
    by $t$ randomly such that $F = |F \wedge C_i$
  Output $F$
  End of algorithm
```

Theorem 2. [13]*For any given fixed* δ *with* $0 < \delta < 1$ *and any fixed real* ϵ *with* $0 < \epsilon$, *when*

$$m \geq \max\left\{\frac{7}{6}(n+3)\ln\frac{2(1+\epsilon)}{\epsilon\delta}, \frac{7}{3} n \ln \frac{16(1+\epsilon)}{3}\right\}, \tag{6}$$

then the formula F *generated by algorithm* G *has only one solution with probability at least* $1 - \delta$.

We can see that G and $G2$ are algorithms that can generate unique solution 3-SAT instances with high success probability while the number of required clauses are high for G is as high as $\Theta(n \ln n)$ while it is $7n$ for $G2$.

3) Instance Generators: Uniqe-PT1 & Unique-PT4

In this section we discuss two main algorithms, Unique-PT1 and Unique-PT4, for generating unique-solution 3-SAT instances and one ancillary padding algorithm that adds more clauses into instances generated by Unique-PT1 or Unique-PT4 and still conforms to the given unique solution. Finally, we show a hybrid algorithm that combines the above three algorithms to generate unique-solution instances with the order parameter up to

$$((C(n,3) * 7)/n). \quad (7)$$

An n-variable 3-SAT instance is considered extremely difficult when there is only one unique solution. There are various tough SAT generators, but the uniqueness is not guaranteed. Some unique-solution 3-SAT instances generate algorithms [13] but the uniqueness is probabilistic (of high probability), but not guaranteed.

We will build our generator based on two approaches, Unique-PT4 and Unique-PT1, respectively and hybridly. Generate a random solution

$$s = x_1 x_2 \dots x_n, \quad (8)$$

where $x_i \in \{0,1\}$ is the value of variable v_i. We then proceed with either one of the following approaches:

```
**Algorithm 6** Unique-PT4: Unique-solution
3-SAT generator with the order parameter = 4
Require: $n \geq 3$ where $n$ is the number
  of variables in a desired 3-SAT instance
Ensure: A 3-SAT instance with one
  unique solution and the order parameter is 4.
  Start of algorithm
  Randomly select a $n$-bit
  solution $s = x_1x_2...x_n$, $F = \emptyset$
  for $i = 1, \cdots, n$ do
    Randomly select two distinct
    numbers, $j$ and $k$, from $\{1, 2, \cdots, n\}$, excluding $i$.
    Generate DO-NOT-CARE clause
    for $x_i$ by using $x_j$ and $x_k$ where the clause $C_i$ is
    if $x_i$ is 1 then
      $(vi \vee v_j \vee v_k) \wedge (v_i \vee v_j \vee \bar{v}_k)$
$\wedge(v_i \vee \bar{v}_j \vee v_k) \wedge (v_i \vee \bar{v}_j \vee \bar{v}_k)$.
    else
      $(\bar{v}i \vee v_j \vee v_k) \wedge (\bar{v}i \vee v_j \vee \bar{v}_k)$
$\wedge(\bar{v}_i \vee \bar{v}_j \vee v_k) \wedge (\bar{v}_i \vee \bar{v}_j \vee \bar{v}_k)$.
    Permute the order of the literals
    randomly within $C_i$ and $F = F \wedge C_i$
  Permute the order of the generated
  clauses in $F$ randomly and return $F$
  End of algorithm
```

Theorem 3. Each instance generated by algorithm Unique-PT4 is of one unique solution.

Proof. Assume there exists another solution

$$s_1 = y_1 y_2 \dots y_n \quad (9)$$

that differs from the true solution $s = x_1 x_2 \dots x_n$, by at least one variable where $i \in \{1,2,\dots n\}$ and $x_i, y_i \in \{0,1\}$. Suppose the value of variable v_i is on which s_1 and s disagree. If we collect the four DO-NOT-CARE clauses for variable v_i, we know that in order to satisfy all those four clauses, no matter what the value of v_j and v_k are, v_i must be of value x_i from s. Therefore, solution s_1 cannot satisfy all clauses in the instance. We can further conclude that there exists only one solution for instances generated by unique-PT4.

Theorem 4. *Each instance generated by algorithm Unique-PT1 is of one unique solution.*

```
**Algorithm 7** Unique-PT1: Unique-solution
3-SAT generator with the order parameter $\simeq 1$
Require: $n \geq 3$ where $n$ is the number
  of variables in a desired 3-SAT instance
Ensure: A 3-SAT instance with
  one unique solution with the order parameter $(n+6)/n$.
  Start of algorithm
  Part I
  Randomly select an $n$-bit
  solution $s = x_1x_2...x_n$.
  Create two lists, A and B, where
  initially A is an empty list and $B = \{1, 2, \cdots, n\}$.
  Randomly choose two distinct numbers
  $p$ and $q$ from B. With the randomly selected
  solution $s$, generate 8 clauses for variable $v_p$ and variable
  $v_q$ based on the DO-NOT-CARE approach
  mentioned in algorithm Uniqu-PT4.
  Move $p$ and $q$ from B to A.
  Define $OP(x_i, v_i) = v_i$ if $x_i = 0$ and
  $OP(x_i, v_i) = \bar{v}_i$ if $x_i = 1$ for all $i \in \{1, 2, \cdots, n\}$
  Part II
  for $i = 3, \cdots, n$ do
    Randomly choose two distinct numbers
    $r$ and $t$ in A and choose one number $w$ from B.
    Generate a clause based on $s, r, t$ and $w$
    if $x_w$ is 1 then
      $(v_w \vee OP(x_r, v_r) \vee OP(x_t, v_t))$
    else
      $(\bar{v}_w \vee OP(x_r, v_r) \vee OP(x_t, v_t))$
    Permute the order of the literals
    randomly within the clause.
    Move $w$ from B to A.
  Permute the order of the generated
  $n + 6$ clauses randomly.
  End of algorithm
```

Proof. Assume there exists another solution $s_1 = y_1 y_2 \dots y_n$ that differs from the true solution $s = x_1 x_2 \dots x_n$ by at least one variable where $i \in \{1,2,\dots n\}$ and $x_i, y_i \in \{0,1\}$. By Theorem 3, we know that

$$y_p = x_p \quad (10)$$

and

$$y_q = x_q \quad (11)$$

for s_1 to pass DO-NOT-CARE clauses. For the *i*th iteration of clause generation, where $i \geq 3$, we know the values of randomly selected variables v_s and v_t are already determined in previous iterations. The clause generated in the *i*th iteration specifically determines the value of v_w. Therefore, for s_1 to pass all the clauses, it must have exactly the same solution as s. If s_1 and s differs at variable v_p, then solution s_1 will definitely fail the clause generated during the iteration where value of v_p is specified. Therefore, solution s_1 cannot satisfy all clauses in the instance if it differs from s by at least one variable. We can further conclude that there exists only one solution for instances generated by Unique-PT1.

D. Tuning Tool Algorithms: Padding and Hybrid-1

Unique-PT1 and Unique-PT4, can generate 3-SAT instances with order parameter approximately 1 and 4, respectively. In order to increase the hardness of an instance, it is be desirable to tune the order parameter by adding more clauses into the instance that still obeys the original solution.

Corollary 1.*The Padding algorithm randomly generates β clauses that can be satisfied by the given solution.*

Proof. At the i_{th} iteration, a 3-CNF clause will violate the given solution $s = x_1x_2 \ldots x_n$, when the variables (v_p; v_q; v_r) are assigned the values (x_p; x_q; x_r). That means if

$$x'_p = x_p\ , x'_q = x_q\ , x'_r = x_r\ , \qquad (12)$$

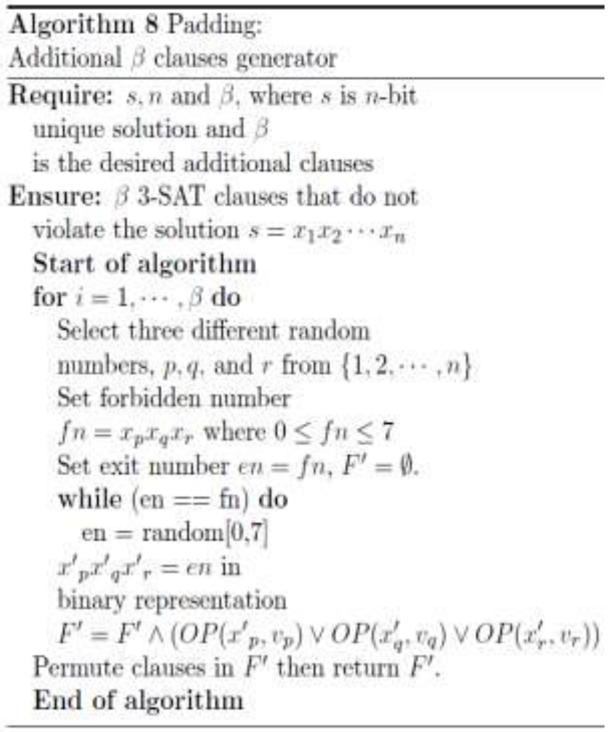
then the tuple

$$(v_p = x_p;\ v_q = x_q;\ v_r = x_r) \qquad (13)$$

will violate the clause

$$(\mathrm{OP}(x'_p, v_p) \vee \mathrm{OP}(x'_q, v_q) \vee \mathrm{OP}(x'_r, v_r)). \qquad (14)$$

Hence, we simply let f_n be the forbidden number we need to avoid when we randomly select a number between 0 and 7. By doing so, we avoid generating clauses that will conflict with the given solution s. Therefore, the clause $(\mathrm{OP}(x'_p, v_p) \vee \mathrm{OP}(x'_q, v_q) \vee \mathrm{OP}(x'_r, v_r)$ can always be satisfied by the tuple ($v_p = x_p$; $v_q = x_q$; $v_r = x_r$). It is clear to see that for each 3-bit representation, we have 7 possible outcomes (clauses) that would not violate the given solution, therefore, we could add at most $C(n,3) * 7 - (n + 6)$ clauses.



```
Algorithm 8 Padding:
Additional β clauses generator
Require: s, n and β, where s is n-bit
  unique solution and β
  is the desired additional clauses
Ensure: β 3-SAT clauses that do not
  violate the solution $s = x_1x_2\cdots x_n$
  Start of algorithm
  for $i = 1, \cdots, \beta$ do
    Select three different random
    numbers, p, q, and r from $\{1, 2, \cdots, n\}$
    Set forbidden number
    $fn = x_px_qx_r$ where $0 \leq fn \leq 7$
    Set exit number $en = fn$, $F' = \emptyset$.
    while (en == fn) do
      en = random[0,7]
    $x'_px'_qx'_r = en$ in
    binary representation
    $F' = F' \wedge (OP(x'_p, v_p) \vee OP(x'_q, v_q) \vee OP(x'_r, v_r))$
  Permute clauses in $F'$ then return $F'$.
  End of algorithm
```

Lemma 1. *The output instance of the Hybrid-1 algorithm has the order parameter α with exactly one unique solution.*

Proof. Option I: It is a simple combination of algorithm Unique-PT4 and algorithm Unique-PT1 and the purpose of each iteration is to nail down the selected undecided variable v_p.

Therefore, at the end of the iterations, the clauses must be satisfied by the unique solution s, according to theorem 3 and theorem 4. When

$$i = \left\lfloor \frac{n*(\alpha-1)}{3} \right\rfloor \qquad (15),$$

we have accumulated $4 * \left\lfloor \frac{n*(\alpha-1)}{3} \right\rfloor$ clauses and we accumulate anther $1 * (n - \left\lfloor \frac{n*(\alpha-1)}{3} \right\rfloor)$ clauses when i reaches n. That gives us αn clauses for running this algorithm through option 1.

Option II: By theorem 4 and corollary 1, we immediately know that we have an instance of αnclauses that can only be satisfied by the given unique solution s. Option II can have the ratio αup to approximately $7 * (n-1)(n-2)$ as explained in the Padding algorithm.

V. TOOLS AND EXPERIMENTS

A. *Tools*

1) Generators

The baseline generator we will use is the Tough Random K-SAT generator [14] that generates random K-SAT instances, which is built upon latest generating techniques up to 2017. The other generator is our ETRSG algorithm. Both algorithms are explained in section IV.A

Algorithm 9 Hybrid-1: Unique-solution 3-SAT generator

```
Require: $n \geq 3, \alpha$ where $n$ is the
  number of variables in a desired
  3-SAT instance, and $\alpha$ is the desired
  order parameter that $1 < \alpha \leq 4$
Ensure: A 3-SAT instance with one unique
  solution with a varying order parameter.
  Start of algorithm
  Repeat Part I of PT-1 algorithm
  Option I: $1 < \alpha \leq 4$
  for $i = 3, \cdots, n$ do
    Select a random number $p$ from B
    if $i \leq \lfloor \frac{n*(\alpha-1)}{3} \rfloor$ then
      Generate 4 clauses for $v_p$ based
      on DO-NOT-CARE in Unique-PT4
    else
      Generate 1 clause for $v_p$ based on
      Unique-PT1
    Permute the order of the literals
    randomly within each clause
    Move $p$ from B to A
  Permute the order of the
  generated $\alpha n$ clauses randomly
  Option II: $1 < \alpha \leq 7(n-1)(n-2)$
  Repeat Part II of PT-1 algorithm
  Invoke Padding algorithm to
  generate additional $(\alpha - 1)n - 6$ clauses
  End of algorithm
```

2) Solver and Platform

MiniSAT is a minimalistic, open-source SAT solver, developed to help researchers and developers alike get started on SAT. MiniSAT is released under the MIT license. MiniSAT utilizes Conflict Driven Clause Learning (CDCL) SAT solving with several other features such as dynamic variable ordering and clause deletion [15], [16]. A small glimpse into the inner workings of MiniSAT is provided as a basic introduction to conflict clause learning and to establish the basic idea behind CDCL SAT solvers.

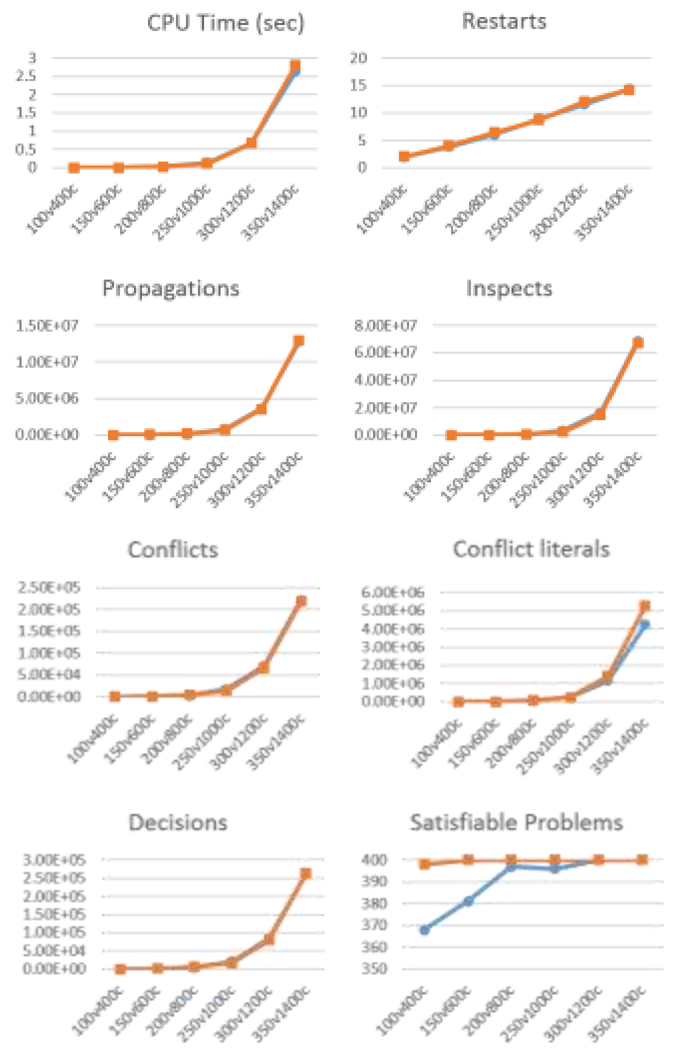


Fig. 1: $\alpha = 4$, 400 instances, Square: ETRSG, Circle: TSG. ETRSG problems and the TSG problems began to relate more directly to each other, and the advantageous difficulty of the ETRSG problem was deemed inconsequential

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 that we can use to gauge effectiveness. 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 basic operational functionality of the MiniSAT solver is as follows.

When MiniSAT is given a SAT problem, it solves the problem by choosing a primary 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 primary variable, it is then restarted with a different variable and propagation begins again. This is process is continually redone until a satisfying assignment to the problem is found and the problem is deemed satisfiable or until it is shown that no satisfiable solution can be made, thus deeming the problem unsatisfiable. These measures that MiniSAT uses are the metrics that we will use to gauge the difficulty of the ETRSG 3-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 more or less efficient hardware. 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.

B. 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 (α, 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.

VI. Discussion

A. 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 Fig. 1, we speculate the critical phase transition zone might lie beyond this point. Furthermore, with evidence from Fig. 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.

B. ETRSG 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], also, the harder instances may need finer characterization metrics. As shown in this experiment, 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.

C. Tunable Unique

We presented a set of tunable algorithms that can deterministically generate unique-solution instances of 3-SAT problems. These algorithms serve as valuable tools to help us better understand the computational hardness of 3-SAT from a numerical perspective. The generated problems can be used for benchmarking performances of different 3-SAT solving algorithms including even the novel quantum computing approach. However, we also expect additional applications of this algorithms such as by adding only one clause to suppress the unique solution, we can generate negative instances. This kind of problems can then be used to study the closely related MAX-3SAT problem and benchmark algorithms aiming to crack MAX-3SAT.

In comparison to the non-deterministic algorithm [13] proposed earlier in the literature, our algorithm can deterministically generate unique-solution instances (instead of being a probabilistic ones such that some generated instances admit multiple solutions). This deterministic property is extremely desirable as the number of "hard problems" (as defined in Ref. [9]) are extremely rare.

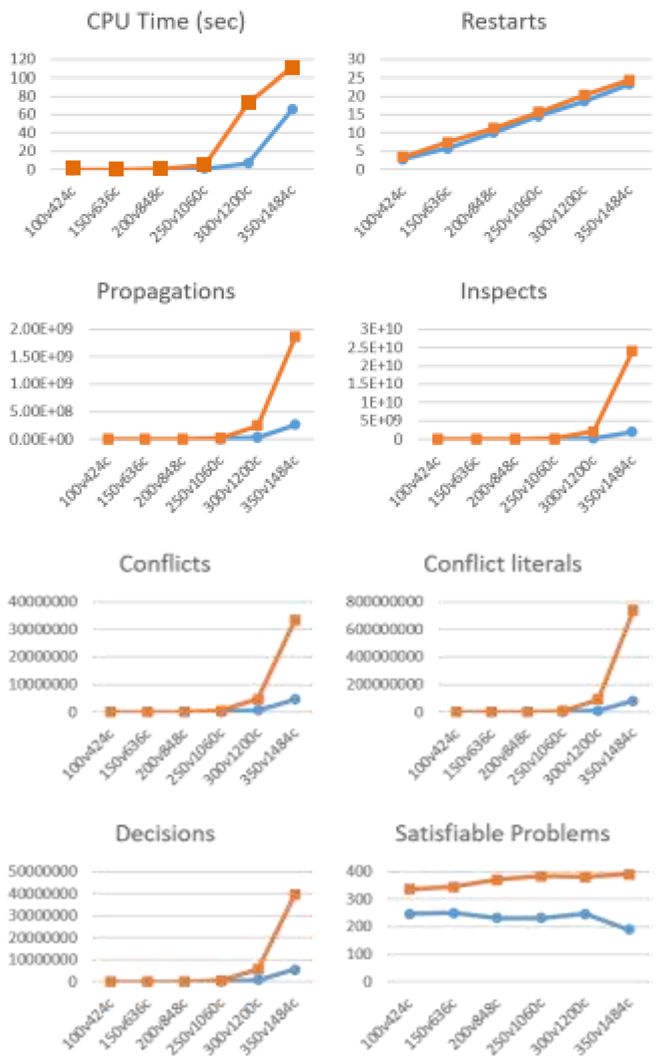


Fig. 2: $\alpha = 4.24$, 400 instances, Square: ETRSG, Circle: TSG. When more than 250 variables, ETRSG instances significantly outperform TSG instances in all aspects, except with slight outperformance in restart. ETRSG has a higher probability of generating solvable instances.

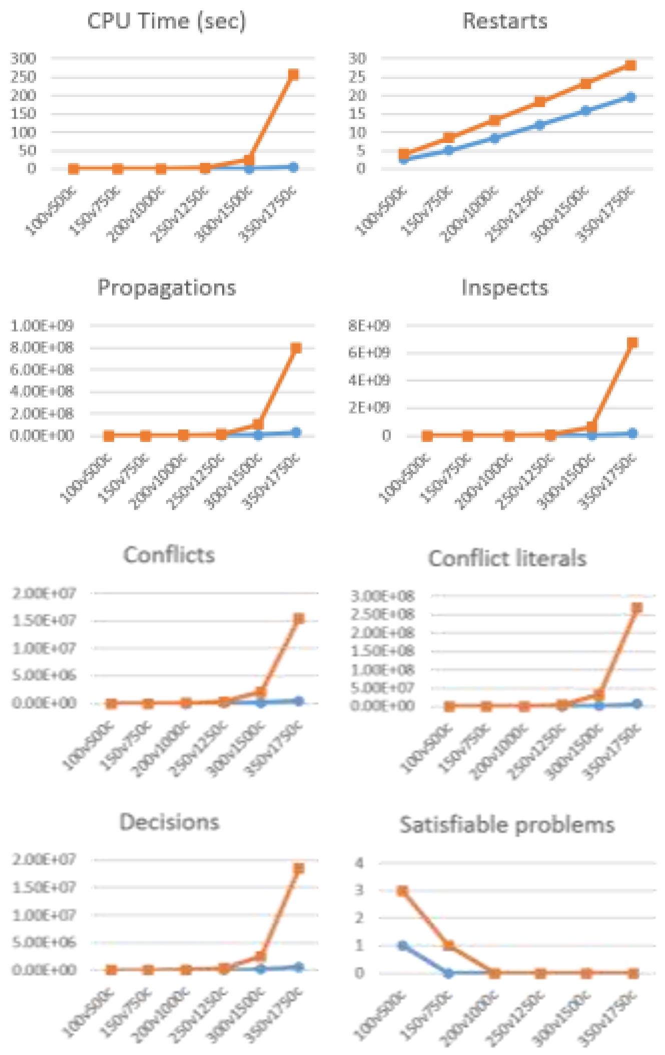


Fig. 3: $\alpha = 5$, 400 instances, Square: ETRSG, Circle: TSG. Similar to $\alpha =$ 4.24, but more significant in restart. The probability of generating solvable instances drops quickly to 0 for both since 5 is greater than the critical phase transition number.

VII. 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 many 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 VI.A a new phase transition number might exist for symmetric 3-SAT problems as the 4.24 ratio applies mainly 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.

As stated in section VI.C unique-solution algorithms serve as valuable tools to help us elucidate and quantify the computational hardness of 3-SAT. these problems could be used to develop a novel quantum computing solver and to study MAX-3SAT problems. Even the deterministic nature of our algorithm is highly desirable the number of "hard problems" (as defined in Ref. [9]) are extremely rare. We can explore more of these problems in a later study and even run comparison tests like what was done with our ETRSG algorithm. It would also be of interest to explore the landscape of these unique-solution problems in future work.

ACKNOWLEDGMENT

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

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] A. Aspuru-Guzik, A. D. Dutoi, P. J. Love and M. Head-Gordon, "Simulated quantum computation of molecular energies," Science, vol. 309, no. 5741, pp. 1704-1707, 2005.
[11] S. Boixo, V. N. Smelyanskiy, A. Shabani, S. V. Isakov, M. Dykman, V. S. Denchev, M. H. Amin, A. Y. Smirnov, M. Mohseni and H. Neven, "Computational multiqubit tunnelling in programmable quantum annealers," Nature Communications, vol. 7, 2016.
[12] B. Berger and T. Leighton, "Protein folding in the hydrophobic-hydrophilic (HP) model is NP-complete," Journal of Computational Biology, vol. 5, no. 1, pp. 27-40, 1998.
[13] M. Motoki and R. Uehara, "Unique solution instance generation for the 3-Satisfiability (3SAT) problem," SAT, pp. 293-305, 2000.
[14] https://toughsat.appspot.com/, "Tough SAT generation," 2017.
[15] 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.
[16] 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.

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.