The Eleventh International Symposium on Combinatorial Search
(SoCS 2018)

# Using Algorithm Configuration Tools to Generate Hard SAT Benchmarks

**Tomáš Balyo**
Karlsruhe Institute of Technology (KIT)
Karlsruhe, Germany
tomas.balyo@kit.edu

**Lukáš Chrpa**
Czech Technical University in Prague &
Charles University in Prague
chrpaluk@fel.cvut.cz

## Abstract

Algorithm configuration tools have been successfully used to speed up local search satisfiability (SAT) solvers and other search algorithms by orders of magnitude. In this paper, we show that such tools are also very useful for generating hard SAT formulas with a planted solution, which is useful for benchmarking SAT solving algorithms and also has cryptographic applications. Our experiments with state-of-the-art local search SAT solvers show that by using this approach we can randomly generate satisfiable formulas that are considerably harder than uniform random formulas of the same size from the phase-transition region or formulas generated by state-of-the-art approaches. Additionally, we show how to generate small satisfiable formulas that are hard to solve by CDCL solvers.

## Introduction

Algorithm configuration deals with automated tuning of free parameters that configure generic problem solvers (Hutter et al. 2009). In Automated Planning, algorithm configuration techniques were exploited for enhancing performance of planners such as LPG (Vallati et al. 2013) or FastDownward (Fawcett et al. 2011), or for configuring domain models by reordering their elements (actions, predicates) (Vallati et al. 2015). In boolean satisfiability (SAT), several state-of-the-art SAT solvers have been shown to significantly benefit from automated tuning (Hutter et al. 2009; Hutter, Hoos, and Leyton-Brown 2011)

Besides solving SAT formulas, research efforts also focus on generating formulas that are challenging for state-of-the-art SAT solvers. One of the most popular class of generated SAT formulas is uniform random 3SAT formulas with a variable-clause ratio corresponding to the phase-transition threshold (Mitchell, Selman, and Levesque 1992; Gent and Walsh 1994). This ratio causes that half of the generated formulas is satisfiable. These formulas are considered very hard to solve relative to their size and are often used to evaluate the performance of SAT solvers, in particular local search SAT solvers (Selman et al. 1992).

More advanced methods (e.g. (Barthel et al. 2002)) can generate even harder instances having the same or smaller size. On top of that the methods can guarantee to always generate a satisfiable instance and even the possibility to hide a predefined solution in the formula (Liu, Luo, and Yue 2015). Generators of hard satisfiable formulas with a predefined solution can also be used in cryptography as one-way functions (Papadimitriou 2003) (for example a password can be encoded as a solution, which is easy to verify but hard to find). Let us note that this is just a theoretical potential application and there are better ways of implementing one-way functions in practice.

In this paper, we describe a new method for generating hard satisfiable formulas with a predefined solution by exploiting algorithm configuration techniques. Specifically, we focus on 3SAT formulas, although our method can easily extended on formulas with arbitrary clause lengths. We conduct an empirical evaluation to demonstrate the hardness of the generated formulas for both CDCL and local-search SAT solvers. We show that we can generate instances especially difficult for stochastic local search algorithms (Selman et al. 1992). In particular, we generated satisfiable 3SAT formulas with as few as 60 variables that cannot be solved by state-of-the-art local search SAT solvers such as ProbSAT (Balint and Schöning 2012) or Dimetheus (Gableske 2013) in 10 minutes.

## Preliminaries

A *Boolean variable* is a variable whose domain consists of two values, *True* and *False*. By a *literal* we mean either $x$ (a *positive literal*) or $\overline{x}$ (a *negative literal*), where $x$ is a Boolean variable and $\overline{x}$ its negation. A *clause* is a disjunction (OR) of literals. A *conjunctive normal form* (CNF) *formula* is a conjunction (AND) of clauses. A clause can be also interpreted as a set of literals and a formula as a set of clauses. A 3SAT formula is a formula where each clause has exactly 3 literals. A (truth) assignment $\phi$ of a formula $F$ assigns a (truth) value to $F$'s variables. An *assignment $\phi$ satisfies* a literal $x$ ($\overline{x}$) if it assigns True (False) to the variable $x$. An *assignment $\phi$ satisfies* a clause if it satisfies at least one of its literals. Finally, *$\phi$ satisfies* a CNF formula if it satisfies all its clauses. A formula $F$ is said to be satisfiable if there is an assignment $\phi$ that satisfies $F$. Such an assignment is called a *satisfying assignment*. The satisfiability problem (SAT) is to find a satisfying assignment of a given CNF formula or determine that such an assignment does not exist (i.e., the formula is unsatisfiable).

```
cdc-generate (vars, φ, p1, p2, p3, r)
CDC0   F := ∅
CDC1   while |F| < r * vars do
CDC2     C = generateRandom3Clause(vars)
CDC3     i = numberOfSatisfiedLiterals(C,φ)
CDC4     if i > 0 then
CDC5       with probability p_i do F = F ∪ {C}
CDC6   return F
```

Figure 1: Pseudo-code of a CDC algorithm which has a set of variables, their truth assignment, and parameters $p_1, p_2, p_3$ and $r$ as an input. It produces a 3SAT formula from the given set of variables that is satisfied by the given assignment.

A *SAT solver* accepts a CNF formula as an input and returns its satisfying assignment if the formula is satisfiable, or returns that the formula is unsatisfiable otherwise. SAT solvers can be divided into two main categories:

- *Local Search Solvers* implement an incomplete stochastic local search algorithm such as walksat (Selman et al. 1993). These solvers usually perform well on randomly generated satisfiable formulas.
- *CDCL Solvers* implement the Conflict Driven Clause Learning (CDCL) algorithm (Marques-Silva and Sakallah 1999). These solvers perform well on structured problems such as those coming from real-world applications.

## Related Work

A commonly used method for generating hard satisfiable formulas (in SAT competitions for example) is to generate uniform random formulas from the SAT phase-transition region (Gent and Walsh 1994) and filter out those deemed as unsatisfiable (Kullmann 2006)[1] However, such methods cannot generate a formula with a predefined solution and the formulas need to be solved in order to guarantee their satisfiability.

A straightforward approach to generate a formula with a given satisfying assignment $\phi$ is to generate random clauses while filtering out those in which $\phi$ is not satisfied until we reach the desired number of clauses (Achlioptas, Jia, and Moore 2005). This approach, called the 1-hidden algorithm, has the disadvantage that generated formulas are easy to solve, especially by local search solvers (Achlioptas, Jia, and Moore 2005). The hardness can be increased by hiding 2 or more solutions (satisfying assignments) (Achlioptas, Jia, and Moore 2005).

Another approach is to use the clause distribution control (CDC) algorithm (Barthel et al. 2002). The CDC algorithm for generating $k$-SAT formulas has $k + 1$ parameters $0 < p_1, \ldots, p_k < 1$ and $r \in \mathbb{R}$. The parameter $r$ represents the clause/variable ratio and each $p_i$ is the probability that a clause which has exactly $i$ satisfied literals under a given assignment $\phi$ gets into the formula. A variant of the CDC algorithm for 3SAT is depicted in Figure 1. Implementations of the CDC algorithm define the values of $p_i$ and $r$ differently. For example the q-hidden algorithm defines $p_i = q^i$ for a parameter $q$ (Jia, Moore, and Strain 2005) and the generator of Barthel et al. (2002) uses a diluted spin-glass model to theoretically compute good values of $p_i$. A more detailed overview of algorithms for generating random formulas with hidden solutions can be found in a recent paper by Liu, Luo, and Yue (2015).

[1]The filtering is usually done by running a local search SAT solver with a large time limit and removing formulas it cannot solve. This has the obvious problem that only formulas that the solver can solve are selected as benchmarks (which are then mostly used to evaluate other solvers).

## Our Approach

Our method is based on the CDC algorithm, however, in contrast to existing approaches (Barthel et al. 2002; Jia, Moore, and Strain 2005; Liu, Luo, and Yue 2015) the values of the parameters $p_i$ and $r$ are tuned by exploiting automatic algorithm configuration tools.

Contrary to improving performance of SAT solvers (Hutter et al. 2009), we use algorithm configuration tools for the opposite purpose – to slow down SAT solvers (by generating hard benchmarks). For our experiments we use the 2.10.03 version of the parameter optimization tool SMAC (Hutter, Hoos, and Leyton-Brown 2011).

### Obtaining The CDC Parameters

In order to use SMAC, we developed a routine, outlined in Figure 2, that evaluates a given parameter configuration for a particular solver. The task is, in essence, to determine how many formulas can be solved until they become too hard to solve. The evaluation routine relies on the fact that formulas get harder with a larger size (more variables and clauses) for any given configuration.

Starting from 20 we incrementally increase the number of variables by 5 per iteration until 600. In each iteration, we generate 8 formulas using the CDC algorithm (Figure 1), where the planted solution is generated randomly.. These 8 formulas are then solved by a SAT solver (with a time limit of 1 minute). The solving process is parallelized[2]. If half (4) or more formulas are solved then we continue to the next iteration (unless we reached the limit of 600 variables) otherwise we return the total number of solved formulas so far as the score of this configuration. Lower score means the configuration gives harder formulas with smaller number of variables. Hence, such a configuration is better for our purposes. The SMAC tool is then used to find the values of $p_i$ and $r$ while minimizing the score.

We used four representative state-of-the-art SAT solvers for the parameter configuration process – two local search solvers: ProbSAT (Balint and Schöning 2012) (version SC13.2) and Dimetheus (Gableske 2013) (version 2.100.994) and two CDCL solvers: Lingeling (Biere 2013) (version bal) and Glucose (Audemard and Simon 2009) (version 4.0). We created seven optimization scenarios – one for each solver individually and three that combined all four solvers (the sum of scores was minimized). Two of the combined optimization scenarios optimize the parameters ac-

[2]Each core can accommodate one solver-benchmark pair.

```
evaluate-configuration (p1, p2, p3, r)
SC0   score := 0
SC1   for i := 20 to 600 step 5 do
SC2     solved := 0
SC3     repeat 8 times:
SC4       vars := generateVars(i)
SC5       φ := generateAssigment(vars)
SC6       F := cdc-generate(vars, φ, p1, p2, p3, r)
SC7       if the solver solves F in 1 minute
            then solved := solved + 1
SC8     score := score + solved
SC9     if solved < 4 then break
SC10  return score
```

Figure 2: Pseudo-code of a configuration evaluation algorithm which has 4 inputs – $p_1, p_2, p_3$ and $r$. Its aim is to estimate the size of the largest formula that can be solved under one minute for a given configuration using a particular SAT solver.

cording to Barthel et al.( 2002) and Q-Hidden approach (Jia, Moore, and Strain 2005) (more details below). The SMAC tool was given 15 hours to find the best parameter values in each scenario. SMAC was run on computers with two Octa-Core Intel Xeon E5-2670 2.6 GHz processors and 64GB of RAM.

The parameter values for the configuration of Barthel et al.( 2002) were chosen according to their specification:

$$\begin{aligned} r &> 4.25 \\ 0.077 < p_1 &< 0.25, \\ p_2 &= (1 - 4p_1)/6 \\ p_3 &= (1 + 2p_1)/6 \end{aligned} \tag{1}$$

For the "Combination Barthel" configuration, using automatic configuration we obtained only the values of $p_1$ and $r$ and then calculated $p_2$ and $p_3$ according to the above equations. For the "Combination Q-Hidden" configuration, we obtained the values of $q$ and $r$ using automatic configuration and then calculated $p_1$, $p_2$ and $p_3$ such that $p_i = q^i$.

Table 1 summarizes the values of $p_i$ and $r$ obtained by our configuration approach and compares them with the values used in previous works (Barthel et al. 2002; Jia, Moore, and Strain 2005). Since Barthel et al.( 2002) specify the range of values for $p_1$ and $r$ (as shown above), we needed to pick concrete values, so we selected $4.3$ for $r$ and $0.163$ for $p_1$ since it is exactly in the middle of the $(0.077, 0.25)$ range ($p_2$ and $p_3$ are calculated from $p_1$). For the Q-Hidden configuration we used $q = 0.3$ and $r = 5.5$ because these values were also used in the experimental section of the original paper (Jia, Moore, and Strain 2005). We did not include 1-Hidden and 2-Hidden formulas (Achlioptas, Jia, and Moore 2005) in our evaluation since the Q-Hidden formulas (that we included) are reported to be much harder (Jia, Moore, and Strain 2005)

## Hardness of The Generated Formulas

To evaluate the hardness of formulas generated by using our parameter configuration approaches we ran experiments with the already mentioned four state-of-the-art SAT solvers – ProbSAT, Dimetheus, Lingeling and Glucose (same versions as used for the parameter optimization).

For each of the nine configurations listed in Table 1 we generated 10 benchmark formulas for each of the following sizes (number of variables): $50, 60, \ldots, 290, 300, 320, 340, \ldots, 580, 600$. Additionally, we generated random "phase-transition" 3SAT formulas of the same sizes. We will refer to a set of 10 benchmark formulas of a particular size generated by using a particular configuration as a group of benchmarks.

The evaluation was done on computers with two Octa-Core Intel Xeon E5-2670 2.6 GHz processors (16 cores in total) and 64GB of RAM. The benchmark formulas were run in parallel (running 16 solver-benchmark pairs at the same time) with a time limit of 10 minutes per each.

The results of our experimental evaluation are summarized in Tables 2 and 3. Taking a closer look on the results we can make several interesting observations:

- The uniform and Barthel et. al configurations generate relatively hard instances for the CDCL solvers, however, they seem to be easy for the local search solvers.
- The Q-Hidden approach gives reasonably hard instances for the local search solvers.
- The ProbSAT, Dimetheus, Combination and Combination Q-hidden configurations yield hard instances for the local search solvers. Only a few instances with more than 100 variables are solved despite the fact that at SAT competitions these solvers routinely solve uniform random 3SAT instances from the phase-transition region with thousands of variables.
- The Lingeling and Combination Barthel configurations yield instances that are similarly hard for all the solvers.

In summary, our approach can generate hard SAT formulas, especially for the local search solvers, and therefore it might be a good candidate for being used for cryptographic applications.

## Tuning a Solver on the Generated Formulas

An interesting question is whether a SAT solver can be tuned (using an algorithm configuration tool) to perform well on formulas generated by our approach such that they are hard for that solver.

In the case, that the solver can be automatically tuned to perform well on the benchmarks tuned to be hard for that specific solver, it would be interesting to know what happens if we iterate the process of solver and benchmark generator tuning. The question is whether (and how soon) we arrive at a fixed point (where the solver and generator cannot be improved anymore) or whether the process does not converge.

The answers to the question depend on the used solver and the degree of its configurability. We run this experiment with the SAT solver Spear (Hutter et al. 2007), since it is

| Configuration | $p_1$ | $p_2$ | $p_3$ | $r$ |
|---|---|---|---|---|
| ProbSAT config. | 0.996 | 0.038 | 0.168 | 7.821 |
| Dimetheus config. | 0.855 | 0.063 | 0.384 | 5.146 |
| Lingeling config. | 0.414 | 0.028 | 0.503 | 4.410 |
| Glucose config. | 0.218 | 0.111 | 0.295 | 4.705 |
| Combination | 0.785 | 0.030 | 0.278 | 6.227 |
| Combination Barthel | 0.242 | 0.005 | 0.247 | 4.298 |
| Combination Q-Hidden | 0.360 | 0.130 | 0.047 | 5.699 |
| Barthel et al.( 2002) | 0.163 | 0.057 | 0.221 | 4.300 |
| Q-Hidden (Jia, Moore, and Strain 2005) | 0.300 | 0.090 | 0.027 | 5.500 |

Table 1: The values of the $p_i$ and $r$ parameters for the CDC algorithm evaluated in this paper. The first seven configurations come from our automatic configuration approach.

| | Number of Solved Instances (out of 410) | | | | Size of Largest Solved Formula (Max. 600) | | | |
|---|---|---|---|---|---|---|---|---|
| Benchmark Category | ProbSAT | Dimetheus | Lingeling | Glucose | ProbSAT | Dimetheus | Lingeling | Glucose |
| ProbSAT config. | 2 | 88 | 409 | 410 | 80 | 540 | 600 | 600 |
| Dimetheus config. | 52 | 46 | 345 | 391 | 170 | 170 | 600 | 600 |
| Lingeling config. | 245 | 245 | 289 | 293 | 600 | 600 | 420 | 400 |
| glucose config. | 410 | 410 | 310 | 293 | 600 | 600 | 460 | 420 |
| Combination | 24 | 31 | 392 | 410 | 80 | 100 | 600 | 600 |
| Combination Barthel | 218 | 222 | 288 | 301 | 600 | 600 | 460 | 460 |
| Combination Q-Hidden | 51 | 47 | 381 | 410 | 230 | 230 | 600 | 600 |
| Uniform 3SAT | 410 | 410 | 294 | 284 | 600 | 600 | 480 | 440 |
| Barthel et. al | 409 | 409 | 333 | 307 | 600 | 600 | 580 | 500 |
| Q-Hidden | 142 | 144 | 365 | 406 | 600 | 600 | 600 | 600 |

Table 2: The total number of solved instances and the size (the number of variables) of the largest solved formula for each benchmark category and solver.

a highly configurable SAT solver, which has been used to demonstrate the strength of the SMAC tool and the automatic algorithm configuration technique in general (Hutter, Hoos, and Leyton-Brown 2011). The available distribution of the SMAC tool contains a Spear configuration scenario as one of the examples, which considers 26 parameters. We utilized this scenario for our experiments while replacing the testing and training instances by our own benchmarks.

The experiment was performed as follows. Firstly, we used our approach to optimize the parameters of the CDC algorithm against the default configuration of Spear. Then, we generated 41 test and 41 training instances of different sizes. After that, Spear was tuned using these instances. Both tuning phases were run with a timeout of 15 hours using the same hardware configuration as in the other experiments.

The performance of the default Spear and tuned Spear on the generated instances was evaluated in the same way as for the other experiments. We observed a relatively small improvement for tuned Spear, 289 solved instances instead of 279 (an improvement of 3.5%). We consider this improvement not significant enough to continue the iteration process (tuning of the CDC algorithm and tuning of Spear) and thus we conclude that Spear cannot be satisfyingly tuned to perform well on the benchmarks generated for it by our approach. Noteworthy, such results are preliminary and the outcome might be different for other solvers.

## Conclusion

In this paper, we showed that the CDC algorithm used for generating challenging SAT formulas can be automatically configured to generate SAT formulas with a planted solution that are small in size and very hard for existing state-of-the-art SAT solvers. Besides benchmarking purposes, generating challenging SAT formulas with a planted solution is useful in cryptographic applications. The conducted experiments indicate that exploiting algorithm configuration approaches for configuring parameters for the CDC algorithm can generate harder formulas than the existing approaches.

The proposed approach is extendable to $k$-SAT (with $k > 3$) formula generation. We plan to investigate for which $k$ we can generate the smallest hard formulas for various kinds of solvers. A theoretical investigation of why the parameter values found by automatic configuration approaches produce such hard instances would be also beneficial.

The introduced benchmark generator has been used in the last three International SAT Competitions (SAT Race 2015, SAT Competition 2016 and SAT Competition 2017)[3].


## Acknowledgments

This research was partially funded by the Czech Science Foundation (project no. 18-07252S).


[3] http://www.satcompetition.org/

| Benchmark Category | Size of Hardest Solved Group (Max. 600) ProbSAT | Dimetheus | Lingeling | Glucose |
|---|---|---|---|---|
| ProbSAT config. | 0 | 500 | 600 | 600 |
| Dimetheus config. | 90 | 90 | 480 | 600 |
| Lingeling config. | 320 | 320 | 340 | 360 |
| Glucose config. | 600 | 600 | 400 | 360 |
| Combination | 70 | 70 | 600 | 600 |
| Combination Barthel | 560 | 560 | 360 | 380 |
| Combination Q-Hidden | 90 | 90 | 600 | 600 |
| Uniform 3SAT | 600 | 600 | 360 | 340 |
| Barthel et. al | 600 | 600 | 460 | 400 |
| Q-Hidden | 270 | 270 | 600 | 600 |

Table 3: The number of variables in the hardest (containing the highest number of variables) solved group of benchmarks. Each group contains 10 instances of the same size and category. A group is considered to be solved if at least 5 instances are solved under 10 minutes.

## References


Achlioptas, D.; Jia, H.; and Moore, C. 2005. Hiding satisfying assignments: two are better than one. *Journal of Artificial Intelligence Research* 623–639.

Audemard, G., and Simon, L. 2009. Predicting learnt clauses quality in modern sat solvers. In *IJCAI*, volume 9, 399–404.

Balint, A., and Schöning, U. 2012. Choosing probability distributions for stochastic local search and the role of make versus break. In Cimatti, A., and Sebastiani, R., eds., *SAT*, volume 7317 of *LNCS*, 16–29. Springer.

Barthel, W.; Hartmann, A. K.; Leone, M.; Ricci-Tersenghi, F.; Weigt, M.; and Zecchina, R. 2002. Hiding solutions in random satisfiability problems: A statistical mechanics approach. *Physical review letters* 88(18):188701.

Biere, A. 2013. Lingeling, plingeling and treengeling entering the sat competition 2013. In *SAT Competition 2013*, 51–52.

Fawcett, C.; Helmert, M.; Hoos, H.; Karpas, E.; Röger, G.; and Seipp, J. 2011. Fd-autotune: Domain-specific configuration using fast-downward. In *Workshop on Planning and Learning (PAL)*.

Gableske, O. 2013. Solver description of dimetheus v. 1.700 for the sat competition 2013. In *SAT Competition 2013*, 30.

Gent, I. P., and Walsh, T. 1994. The sat phase transition. In *ECAI*, volume 94, 105–109. PITMAN.

Hutter, F.; Babić, D.; Hoos, H. H.; and Hu, A. J. 2007. Boosting verification by automatic tuning of decision procedures. In *Formal Methods in Computer Aided Design, 2007. FMCAD'07*, 27–34. IEEE.

Hutter, F.; Hoos, H. H.; Leyton-Brown, K.; and Stuetzle, T. 2009. ParamILS: an automatic algorithm configuration framework. *Journal of Artificial Intelligence Research* 36:267–306.

Hutter, F.; Hoos, H. H.; and Leyton-Brown, K. 2011. Sequential model-based optimization for general algorithm configuration. In *Learning and Intelligent Optimization*. Springer. 507–523.

Jia, H.; Moore, C.; and Strain, D. 2005. Generating hard satisfiable formulas by hiding solutions deceptively. In *Proceedings of the National Conference on Artificial Intelligence*, volume 20, 384. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999.

Kullmann, O. 2006. The sat 2005 solver competition on random instances. *Journal on Satisfiability, Boolean Modeling and Computation* 2:61–102.

Liu, R.; Luo, W.; and Yue, L. 2015. Hiding multiple solutions in a hard 3-sat formula. *Data & Knowledge Engineering* 100:1–18.

Marques-Silva, J. P., and Sakallah, K. A. 1999. Grasp: A search algorithm for propositional satisfiability. *Computers, IEEE Transactions on* 48(5):506–521.

Mitchell, D. G.; Selman, B.; and Levesque, H. J. 1992. Hard and easy distributions of SAT problems. In *Proceedings of the 10th National Conference on Artificial Intelligence. San Jose, CA, July 12-16, 1992.*, 459–465.

Papadimitriou, C. H. 2003. *Computational complexity*. John Wiley and Sons Ltd.

Selman, B.; Levesque, H. J.; Mitchell, D. G.; et al. 1992. A new method for solving hard satisfiability problems. In *AAAI*, volume 92, 440–446.

Selman, B.; Kautz, H. A.; Cohen, B.; et al. 1993. Local search strategies for satisfiability testing. *Cliques, coloring, and satisfiability* 26:521–532.

Vallati, M.; Fawcett, C.; Gerevini, A. E.; Hoos, H. H.; and Saetti, A. 2013. Automatic generation of efficient domain-optimized planners from generic parametrized planners. In *Proceedings of the Sixth Annual Symposium on Combinatorial Search, SOCS*.

Vallati, M.; Hutter, F.; Chrpa, L.; and McCluskey, T. L. 2015. On the effective configuration of planning domain models. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, 1704–1711.