Evaluating the Hardness of SAT Instances Using Evolutionary Optimization Algorithms

Alexander Semenov
ITMO University, St. Petersburg, Russia

Daniil Chivilikhin
ITMO University, St. Petersburg, Russia

Artem Pavlenko
ITMO University, St. Petersburg, Russia
JetBrains Research, St. Petersburg, Russia

Ilya Otpuschennikov
ISDCT SB RAS, Irkutsk, Russia

Vladimir Ulyantsev
ITMO University, St. Petersburg, Russia

Alexey Ignatiev
Monash University, Melbourne, Australia

Abstract

Propositional satisfiability (SAT) solvers are deemed to be among the most efficient reasoners, which have been successfully used in a wide range of practical applications. As this contrasts the well-known NP-completeness of SAT, a number of attempts have been made in the recent past to assess the hardness of propositional formulas in conjunctive normal form (CNF). The present paper proposes a CNF formula hardness measure which is close in conceptual meaning to the one based on Backdoor set notion: in both cases some subset B of variables in a CNF formula is used to define the hardness of the formula w.r.t. this set. In contrast to the backdoor measure, the new measure does not demand the polynomial decidability of CNF formulas obtained when substituting assignments of variables from B to the original formula. To estimate this measure the paper suggests an adaptive (ε, δ)-approximation probabilistic algorithm. The problem of looking for the subset of variables which provides the minimal hardness value is reduced to optimization of a pseudo-Boolean black-box function. We apply evolutionary algorithms to this problem and demonstrate applicability of proposed notions and techniques to tests from several families of unsatisfiable CNF formulas.

2012 ACM Subject Classification Theory of computation → Automated reasoning; Hardware → Theorem proving and SAT solving; Theory of computation → Optimization with randomized search heuristics; Mathematics of computing → Combinatorial optimization

Keywords and phrases SAT solving, Boolean formula hardness, Backdoors, Evolutionary algorithms

Digital Object Identifier 10.4230/LIPIcs.CP.2021.47

Supplementary Material *Software (Source Code)*: `https://github.com/ctlab/EvoGuess`

Funding This work was supported by the Ministry of Science and Higher Education of Russian Federation, research project no. 075-03-2020-139/2 (goszadanie no. 2019-1339). Ilya Otpuschennikov's research was funded by Ministry of Science and Higher Education of Russian Federation, project with no. of state registration: 121041300065-9. Artem Pavlenko was supported by JetBrains Research.

1 Introduction

Modern Boolean Satisfiability Problem (SAT) solving algorithms are de-facto a standard computational instrument used in many application domains including symbolic verification, software testing, bioinformatics, combinatorics, and cryptanalysis [10]. SAT solvers work with Boolean formulas, most often written in Conjunctive Normal Form (CNF). If determining

satisfiability of a CNF formula takes a SAT solver more than a preallocated amount of time, a natural question to ask is how *hard* this formula is for this specific solver? Hereinafter, it is convenient for us to follow the notation of [2] and use the concept of *hardness* of a SAT instance.

For some SAT solving algorithms and some families of formulas their hardness can be estimated analytically [1, 6, 7, 13, 15, 31, 55, 56]. However, to our best knowledge, the general case of the problem of estimating the hardness of a formula w.r.t. to practical SAT solving algorithms is yet to be resolved. The main reason for this is that a state-of-the-art SAT solver is a complicated piece of software, whose behavior depends on a vast number of various parameters [33, 34]. Smallest changes of parameter values may drastically affect the observable characteristics of the SAT solving process, e.g., the number of unit propagations or backtracks. This phenomenon is known as *heavy-tailed behavior* [30]. If a SAT solver demonstrates such behavior for a concrete CNF formula, then estimating how hard the formula is for this solver is hardly feasible with the existing methods, or such estimates will be extremely inaccurate. In light of the myriads of practical applications of modern SAT solvers, it is of unquestionable importance to propose a universal hardness measure for arbitrary CNF formulas that could be used in practice for any SAT solver.

Prior work proposed a few hardness measures of Boolean formulas w.r.t. specific SAT solving algorithms. One of the best-studied approaches estimates parameters of the search tree generated by the algorithm. This class of measures includes *space complexity of tree-like resolution* [27, 40], *width of formula* [27], and *space of formula* [2]. For some families of CNF formulas, e.g. pigeonhole principle formulas [18], such measures may be estimated analytically. However and as far as we know, there is no computationally efficient way of estimating any of said measures for an arbitrary CNF formula.

Another approach to estimating formula hardness builds on the concept of a *strong backdoor set* (SBS) introduced in [59]. An SBS is a subset of the set of variables of a CNF formula such that any assignment of the variables from this subset makes the whole formula polynomially decidable. Clearly, the set of all variables in the formula comprises a trivial SBS. If for formula C there exists a non-trivial SBS B w.r.t. some polynomial-time algorithm P, e.g. unit propagation [24], then the hardness of this formula w.r.t. B and P may be estimated as $\text{poly}(|C|) \cdot 2^{|B|}$. Thus, the notion of SBS gives us a way of estimating the hardness of a formula based on two main components: the strong backdoor set itself and the polynomial-time algorithm used for solving weakened SAT instances. Sadly, for an arbitrary Boolean formula there is no guarantee that a relatively small SBS exists. To check if a given set B of variables in formula C is an SBS, one has to run the algorithm P on all (in the worst case) $2^{|B|}$ CNF instances derived from C by substituting all possible assignments to the variables of B. The algorithm for solving SAT using backdoors described in [59] enumerates all subsets of the set of variables of a target CNF formula by iteratively increasing their size.

In this paper, we propose a novel hardness measure of an arbitrary CNF formula w.r.t. an arbitrary deterministic complete SAT solving algorithm, which may be estimated by applying standard methods of black-box optimization. Conceptually, the suggested hardness measure is in some sense similar to the aforementioned SBS-based hardness measure. For an arbitrary CNF formula over the set X of variables the proposed approach also uses two components: 1) a set $B \in 2^X$, and 2) an arbitrary complete but, most importantly, *not necessarily* polynomial SAT solving algorithm A.

The proposed decomposition hardness (or *d-hardness*) is defined for an arbitrary CNF formula C. More specifically, we first introduce measure $\mu_{B,A}(C)$ expressing the hardness of formula C w.r.t. a concrete set $B \in 2^X$ and a concrete deterministic complete SAT solving

algorithm A. Second, the d-hardness of C is defined as the minimum of $\mu_{B,A}(C)$ over all possible sets B. To estimate $\mu_{B,A}(C)$ in practice, we propose an adaptive probabilistic (ε, δ)-approximation algorithm. This algorithm uses ideas close to the ones from [36], and is based on the Monte Carlo method. But in contrast to many similar approaches, the proposed algorithm can adaptively tune the random sample size to achieve the required accuracy of $\mu_{B,A}(C)$ estimation. By estimating $\mu_{B,A}(C)$ with this algorithm we can reduce the problem of evaluating d-hardness of an arbitrary formula C to optimization of a stochastic pseudo-Boolean fitness function [22]. The latter problem is solved with approaches traditionally used in black-box optimization: namely, evolutionary algorithms [11, 41].

To illustrate the usefulness of the d-hardness concept suppose that we have some extremely hard CNF formula C. Consider the two following approaches. First, launch a SAT solver on C and wait as long as needed to decide satisfiability of C. There is no guarantee that the process will finish in any reasonable amount of time. Second, run algorithms that assess the d-hardness proposed in this article. After a fixed amount of time, say, 12 hours, we will get some set $B \in 2^X$ and a corresponding d-hardness estimate. By means of suggested methods one can compare different CNF formulas in the sense of their d-hardness.

Concrete contributions of this paper are the following.

1. We propose a new measure of hardness for a CNF formula w.r.t. an arbitrary complete deterministic SAT solving algorithm, and prove its theoretical soundness.
2. We develop an adaptive (ε, δ)-approximation algorithm for estimating this measure.
3. We conduct an experimental evaluation that demonstrates practical applicability of the proposed measure.

2 Preliminaries

Recall that Boolean variables have values from $\{0, 1\}$. A Boolean variable x and its negation $\neg x$ are called *literals*. Literals x and $\neg x$ are called *complementary*. A *clause* is a disjunction of literals, which does not include complementary ones. A Boolean formula in CNF is a conjunction of different clauses. Let C be an arbitrary CNF formula and X, $|X| = k$ be the set of variables encountered in C. An arbitrary total function $\alpha\colon X \to \{0, 1\}$ defines an *assignment* of variables from X. For an arbitrary assignment α the *interpretation* of formula C on α and the *substitution* of α to C are defined in a standard way, see e.g. [16]. Thus, a Boolean function $f_C\colon \{0, 1\}^k \to \{0, 1\}$ is defined. Assignment $\alpha \in \{0, 1\}^k\colon f_C(\alpha) = 1$ is called a *satisfying assignment* for C. If a satisfying assignment exists for C, formula C is called *satisfiable*. Otherwise, C is called *unsatisfiable*.

As in many other works on proof complexity and hardness of Boolean formulas, formulas are assumed to be in conjunctive normal form (CNF) and unsatisfiable, see e.g. [2, 18, 55], etc. It is justified by the fact that for the majority of satisfiable instances (especially with a large number of satisfying assignments) it is possible that the algorithm will get "lucky" and come across a short satisfiability certificate. This is not possible with unsatisfiable instances.

Let C be an unsatisfiable CNF formula over the set of variables X. For an arbitrary set $B \subseteq X$, denote the set of all possible assignments to variables of B as $\{0, 1\}^{|B|}$. Following [59], for an arbitrary $\beta \in \{0, 1\}^{|B|}$ denote $C[\beta/B]$ the CNF formula derived from C by substitution of the assignment β of variables B and consequent simplification of the resulting formula.

▶ **Definition 1** (Williams et al. [59]). *Set $B \subseteq X$ is called a strong backdoor set (SBS) for C w.r.t. a polynomial-time algorithm P if for any $\beta \in \{0, 1\}^{|B|}$ the CNF formula $C[\beta/B]$ is reported by P to be unsatisfiable.*

The article [2] studied a number of approaches to estimating hardness of Boolean formulas in CNF, and the main attention was paid to several similar tree-like metrics. However, for us the particular value are the conclusions made in [2] about the possibility to assess the hardness of a CNF formula using SBS. The following definition suggests itself as a consequence of the analysis of results from [2]. In fact, it uses SBS to evaluate the hardness of an arbitrary CNF formula and reduces this problem to an optimization problem.

▶ **Definition 2** (b-hardness). *Let C be an arbitrary unsatisfiable CNF formula and B be an arbitrary SBS for C w.r.t. polynomial-time algorithm P. Denote the total runtime of P on CNF formulas $C[\beta/B]$ for all $\beta \in \{0,1\}^{|B|}$ by $\mu_{B,P}(C)$. The backdoor-hardness (or* b-hardness*) of C w.r.t. P is specified as $\mu_P(C) = \min_{B \in 2^X} \mu_{B,P}(C)$, where the minimum is taken among all possible SBSes for C w.r.t. P.*

In [59], an algorithm for solving SAT using SBS is described: it enumerates sets $B \in 2^X$ by gradually increasing their cardinality. If for CNF formula C there exists a small-sized SBS, this algorithm may be quite efficient. Its complexity for an arbitrary C in the assumption that an SBS B exists such that $|B| < k/2$ is

$$O\left(p(|C|) \cdot \left(\frac{2k}{\sqrt{|B|}}\right)^{|B|}\right), \tag{1}$$

where $k = |X|$, $p(\cdot)$ is some polynomial and $|C|$ is the length of the binary encoding of C.

3 d-hardness: Decomposition Hardness of CNF Formula

There are two evident barriers for practical application of the b-hardness notion. First, to prove that an arbitrary $B \in 2^X$ is an SBS we have to construct (in the worst case) CNF formulas $C[\beta/B]$ for all $\beta \in \{0,1\}^{|B|}$. Second, the algorithm of [59] enumerates sets B of increasing cardinality ($|B| = 1, 2, \ldots$). Taking into account (1) we can conclude that if, e.g., the minimal backdoor B has cardinality $|B| = 20$ and $k = 100$, finding B with the aforementioned enumeration algorithm is unrealistic. A similar issue arises for the tree-like metric of hardness described in [2], where for formula refutation a variant of Beame-Pitassi algorithm [5] is used.

In this section we introduce a new hardness measure for CNF formulas. When formulating the main concept we pursue the next two goals: 1) to avoid the barriers referred above, and 2) to suggest a measure that can be used for any complete SAT solving algorithm, considering it as a black-box function. Let us begin from the following definition.

▶ **Definition 3.** *For an arbitrary CNF formula C over the set of variables X consider any set B, $B \in 2^X$, and let A be an arbitrary deterministic complete SAT solving algorithm. Define the hardness of C w.r.t. B and A as $\mu_{B,A}(C) = \sum_{\beta \in \{0,1\}^{|B|}} t_A(C[\beta/B])$, where $t_A(C[\beta/B])$ is the running time of A on CNF formula $C[\beta/B]$.*

The value $t_A(C[\beta/B])$ may be expressed in any appropriate units. For example, if A is a solver based on Conflict-Driven Clause Learning (CDCL) [42], $t_A(C[\beta/B])$ may be defined as the number of unit clause propagations made by A in the process of proving the unsatisfiability of $C[\beta/B]$. Let us emphasize, that unlike P from Definition 2, in the general case A is not a polynomial-time algorithm. The following definition arises by analogy with the concept of b-hardness.

▶ **Definition 4** (d-hardness). *The decomposition hardness (or d-hardness) $\mu_A(C)$ of CNF formula C w.r.t algorithm A is defined as:*

$$\mu_A(C) = \min_{B \in 2^X} \mu_{B,A}(C).$$

The main question in the context of these definitions is as follows: is there a practical way to estimate the values $\mu_{B,A}(C)$ and $\mu_A(C)$? Below we give a positive answer to this question harnessing the idea from [50]: expressing $\mu_{B,A}(C)$ via a special random variable with finite expected value and variance.

Let C be an arbitrary CNF formula over the set of variables X and A be an arbitrary deterministic complete SAT solving algorithm. Consider an arbitrary $B \in 2^X$ and specify a uniform distribution on $\{0,1\}^{|B|}$. Define a random variable ξ_B in the following way: for any $\beta \in \{0,1\}^{|B|}$ the value of ξ_B equals to the running time of algorithm A on CNF formula $C[\beta/B]$. Since algorithm A is complete, the random variable ξ_B has spectrum $S(\xi_B) = \{\xi_1, \ldots, \xi_M\}$, where $\xi_i \colon 0 < \xi_i < \infty$, $i \in \{1, \ldots, M\}$, and ξ_B has the following probabilistic distribution:

$$P(\xi_B) = \left\{ \frac{s_1}{2^{|B|}}, \ldots, \frac{s_M}{2^{|B|}} \right\},$$

where by s_i, $i \in \{1, ..., M\}$ we denote the number of such $\beta \in \{0,1\}^{|B|}$ that ξ_B has the value ξ_i. From the above, random variable ξ_B has an expected value $E[\xi_B] \colon 0 < E[\xi_B] < \infty$. It is not hard to verify the correctness of the following expressions:

$$\sum_{\beta \in \{0,1\}^{|B|}} t_A(C[\beta/B]) = \sum_{i=1}^{M} \xi_i \cdot s_i = 2^{|B|} \cdot \sum_{i=1}^{M} \xi_i \cdot \frac{s_i}{2^{|B|}}.$$

From the above, we can conclude that

$$\mu_{B,A}(C) = 2^{|B|} \cdot E[\xi_B]. \tag{2}$$

The equation (2) is quite important because it expresses $\mu_{B,A}(C)$ via finite expected value of some random variable and, hence, allows estimating the value using the Monte Carlo method [43]. In more detail, our nearest goal is to construct such an evaluation $\tilde{\mu}_{B,A}(C)$ of the value $\mu_{B,A}(C)$ that for any fixed $\varepsilon > 0$, $\delta > 0$ the following condition holds:

$$\Pr[(1-\varepsilon) \cdot \mu_{B,A}(C) \leq \tilde{\mu}_{B,A}(C) \leq (1+\varepsilon) \cdot \mu_{B,A}(C)] \geq 1 - \delta. \tag{3}$$

Parameters ε and $1 - \delta$ from (3) in a number of similar cases are named *tolerance* and *confidence level*, respectively.

Now, fix some natural number N. Given C, B, and A, let us carry out N independent observations of random variable ξ_B introduced above. We may consider these N observations as one observation of N independent random variables with the same probability distribution (remind, that we assume A to be deterministic). Denote these random variables $\xi^1, \ldots, \xi^N$. Define $\tilde{\mu}_{B,A}(C)$ as:

$$\tilde{\mu}_{B,A}(C) = \frac{2^{|B|}}{N} \cdot \sum_{j=1}^{N} \xi^j. \tag{4}$$

The sense of the fact that will be established below is close to one of the so-called zero-one estimator theorem from [36], but in our case ξ_B is not a Bernoulli variable and we cannot avoid the presence of $Var(\xi_B)$ in the resulting lower bound for N.

▶ **Theorem 1.** *Let C be an arbitrary CNF formula over variables X, A be a deterministic complete SAT solving algorithm, and B be an arbitrary subset of X. Then for $\tilde{\mu}_{B,A}(C)$ specified by (4) and for any $\varepsilon > 0$, $\delta > 0$, the condition (3) holds for any $N > \frac{Var(\xi_B)}{\varepsilon^2 \cdot \delta \cdot E^2[\xi_B]}$.*

Proof. Due to the assumptions on A, the random variable ξ_B has a finite expected value $E[\xi_B] > 0$ and finite variance $Var(\xi_B)$. If $Var(\xi_B) = 0$ then $S(\xi_B) = \{a\}$, where a is some constant: $a > 0$. In this case the claim of the theorem is trivially satisfied. Below let us assume that $Var(\xi_B) > 0$. Next we use the Chebyshev's inequality [28]:

$$\Pr\left[|\zeta - E[\zeta]| \leq k \cdot \sqrt{Var(\zeta)}\right] \geq 1 - \frac{1}{k^2} \tag{5}$$

which holds for any $k > 0$ and any arbitrary random variable ζ such that $Var(\zeta) > 0$. Fix an arbitrary $\varepsilon > 0$ and select k such that $k \cdot \sqrt{Var(\zeta)} = \varepsilon \cdot E[\xi]$. With this in mind, transform (5) to the following form:

$$\Pr\left[|\zeta - E[\zeta]| \leq \varepsilon \cdot E[\zeta]\right] \geq 1 - \frac{Var(\zeta)}{\varepsilon^2 \cdot E^2[\zeta]}. \tag{6}$$

Due to considering N independent observations of ξ_B as a single observation of N independent random variables with the same distribution the following holds: $E[\xi_1] = \ldots = E[\xi_N] = E[\xi_B]$, $Var(\xi_1) = \ldots = Var(\xi_N) = Var(\xi_B)$. Consider the random variable $\zeta = \sum_{j=1}^{N} \xi_j$. If we apply inequality (6) to it we get (taking into account elementary transformations):

$$\Pr\left[(1-\varepsilon) \cdot E[\xi_B] \leq \tfrac{1}{N} \cdot \textstyle\sum_{j=1}^{N} \xi_j \leq (1+\varepsilon) \cdot E[\xi_B]\right] \geq 1 - \tfrac{Var(\xi_B)}{\varepsilon^2 \cdot N \cdot E^2[\xi_B]} \ . \tag{7}$$

With respect to (2) and (4), the last inequality may be rewritten as:

$$\Pr\left[(1-\varepsilon) \cdot \mu_{B,A}(C) \leq \tilde{\mu}_{B,A}(C) \leq (1+\varepsilon) \cdot \mu_{B,A}(C)\right] \geq 1 - \tfrac{Var(\xi_B)}{\varepsilon^2 \cdot N \cdot E^2[\xi_B]}. \tag{8}$$

The validity of Theorem 1 directly follows from (8). ◀

4 Estimation of d-Hardness via Evolutionary Optimization Algorithms

As follows from the results of the previous section, for exact calculation of d-hardness of an arbitrary CNF formula C it is required to find the set B with the minimum value of $\mu_{B,A}(C)$ over all $B \in 2^X$. For any B, instead of trying out all vectors $\beta \in \{0,1\}^{|B|}$ as is necessary when we work with the b-hardness concept, we may compute the estimation $\tilde{\mu}_{B,A}(C)$ using the following Monte Carlo scheme:

- let us carry out N independent observations of random variable ξ_B: $\xi^1, \ldots, \xi^N$;
- calculate the value $\tilde{\mu}_{B,A}(C)$ specified by (4).

Due to Theorem 1, $\tilde{\mu}_{B,A}(C)$ is an (ε, δ)-approximation of $\mu_{B,A}(C)$ for a proper value of N.

4.1 (ε, δ)-approximation algorithm for d-hardness estimation

In theory, since $E[\xi_B]$ and $Var(\xi_B)$ are finite, we can estimate $\mu_{B,A}(C)$ with any accuracy specified beforehand. However, it may be not achievable for real cases: for example, when $Var(\xi_B)$ is too large. Therefore, in experiments when selecting N to achieve the required values of ε and δ we have to replace $E[\xi_B]$ and $Var(\xi_B)$ with their statistical counterparts. This practice is generally accepted in mathematical statistics. In the experimental part we will give a number of examples when the estimates obtained in this way are accurate enough.

Following e.g. [58] we use for estimating $E[\xi_B]$ the sample mean $\overline{\xi}_B$, constructed for a concrete random sample $\xi^1, \ldots, \xi^N$: $\overline{\xi}_B = \frac{1}{N} \cdot \sum\limits_{j=1}^{N} \xi^j$. The unbiased sample variance is used to estimate $Var(\xi_B)$: $s^2(\xi_B) = \frac{1}{N-1} \cdot \sum\limits_{j=1}^{N} (\xi^j - \overline{\xi}_B)^2$. Taking into account Theorem 1, for some fixed ε and δ, we select any such N that the following condition holds:

$$N > \frac{s^2(\xi_B)}{\varepsilon^2 \cdot \delta \cdot (\overline{\xi}_B)^2}. \tag{9}$$

More concretely, we use the following variant of this approach. At the starting point we choose some relatively small N (say, $N = 100$), construct a random sample and calculate $\overline{\xi}_B$ and $s^2(\xi_B)$. Using fixed values of ε and δ (say, $\varepsilon = 0.1$, $\delta = 0.05$) we check if condition (9) is satisfied. If not, we augment our current random sample by N new observations of ξ_B, thus doubling the random sample size; after this we recalculate $\overline{\xi}_B$ and $s^2(\xi_B)$. These steps are repeated until condition (9) is satisfied.

Note that in the general case we cannot efficiently calculate the value of ξ_B (and, accordingly, $\tilde{\mu}_{B,A}(C)$): for example, for B of a small cardinality this problem may be comparable in complexity with solving SAT for the initial CNF formula C. However, most importantly, there always exists such a set B that for any $\beta \in \{0,1\}^{|B|}$ the corresponding value of ξ_B is calculated efficiently, e.g. in the case when $B = X$. Another example in this context is when B is some Strong Unit Propagation Backdoor Set (SUPBS): a type of backdoor in which the unit propagation rule is used as the polynomial algorithm P [59].

The next important point is that unlike the algorithm from [59] or the Beam-Pitassi algorithm, we apply computational schemes used in metaheuristic optimization [41] to find a set B with a good value of $\tilde{\mu}_{B,A}(C)$. In such schemes, the objective function (*fitness function*) is calculated efficiently at some starting point, and then attempts are made to consistently improve the values of this function in other points of the search space w.r.t. some general search strategy, e.g. local search [14] or evolutionary algorithms [41].

So, in the context of all the concepts introduced above, let $B_0 = \{x_1^0, \ldots, x_n^0\}$, $B_0 \subseteq X$ be an initial subset for which $\tilde{\mu}_{B_0,A}(C)$ can be calculated efficiently (e.g. $B_0 = X$ or B_0 is some SUPBS). We will look for B with a good value of $\tilde{\mu}_{B,A}(C)$ as some $B \in 2^{B_0}$. Define B using a Boolean vector $\lambda_B \in \{0,1\}^n$, assuming that $\lambda_i = 1$ if $x_i^0 \in B$ and $\lambda_i = 0$ if $x_i^0 \notin B$, $\lambda_B = (\lambda_1, \ldots, \lambda_n)$. Fix N and consider the multivalued function

$$F_{A,C,N} \colon \{0,1\}^n \to \mathbb{R}_+ \tag{10}$$

defined as follows: for vector $\lambda_B \in \{0,1\}^n$ we build the set B, then we generate (in accordance with a uniform distribution on $\{0,1\}^{|B|}$) vectors $\beta_j \in \{0,1\}^{|B|}$, $j \in \{1, \ldots, N\}$ and, using these vectors as a random sample, construct corresponding values of ξ_B: $\xi^1, \ldots, \xi^N$. Then the value of function (10) for λ_B is $\frac{2^{|B|}}{N} \sum_{j=1}^{N} \xi^j$. Note that in the general case for different random samples the values of (10) can differ, thus this function is multivalued.

4.2 Used evolutionary optimization algorithms

In the experimental part of the article we use evolutionary algorithms for optimizing function (10): in more detail, we apply an algorithm from the family of $(1+1)$ Fast Evolutionary Algorithms, $(1+1)$ FEA [23] with parameter β, and one special modification of a genetic algorithm. Below we give a brief description of these algorithms.

First, consider the ordinary $(1+1)$ Evolutionary Algorithm (EA) [44]. It uses the simplest implementation of the concept of random mutation: one random mutation of an arbitrary $\alpha \in \{0,1\}^n$ is implemented by a series of n independent Bernoulli trials with success probability $p = 1/n$. If $i \in \{1, \ldots, n\}$ is the index of a successful trial, then the i-th bit in α is flipped. The $(1+1)$ EA has an extremely high worst-case complexity [25], but demonstrates good results in many practical cases. As mentioned in [57], this is mostly because on average $(1+1)$ EA behaves similarly to the Hill Climbing algorithm [49] (for a single random mutation, the expected value of the number of flipped bits equals one), but with a non-zero probability can move from α to any point in $\{0,1\}^n$.

There are ways of reducing the worst-case estimation of $(1+1)$ EA if we imply the complexity measure proposed in [25]. One of these ways is changing the mutation rate in the original $(1+1)$ EA. The $(1+1)$ FEA$_\beta$ described in [23] is a good example. The core of

this algorithm is the so-called heavy-tailed mutation operator: it flips bits of the considered Boolean vector with probability Λ/n (instead of $1/n$ in standard $(1+1)$ EA), where Λ is the value of a random variable with Power-law distribution $D^{\beta}_{n/2}$ with parameter β [23]. The worst-case estimation of this algorithm is $O(n^{\beta} \cdot 2^n)$ instead of n^n for the original $(1+1)$ EA. In computational experiments we used the $(1+1)$ FEA$_\beta$ with parameter $\beta = 3$, because it is the minimal integer value of this parameter for which the expected value of the number of flipped bits tends to some constant with the increase of n: according to [23], this constant is approximately 1.3685.

We also experimented with generating a new vector λ_B on the basis of several existing vectors, using a special variant of a genetic algorithm which was used in [47]. Several vectors λ_B with already calculated values of the considered objective function (10) form a *population* in terms of the genetic algorithm [41]. In one iteration, the new population (offspring) is formed from the current one.

Denote the current population as P_{cur} and the new population as P_{new}, $|P_{\text{cur}}| = |P_{\text{new}}| = R$ for some fixed R. Let $P_{\text{cur}} = \{\lambda_{B_1}, \ldots, \lambda_{B_R}\}$. P_{cur} is associated with a distribution $D_{\text{cur}} = \{p_1, \ldots, p_R\}$, where

$$p_i = \frac{1/F_{A,C,N}(\lambda_{B_i})}{\sum\limits_{j=1}^{R} \left(1/F_{A,C,N}(\lambda_{B_j})\right)}, i \in \{1, \ldots, R\}.$$

To form the new population P_{new}, we first select G individuals from P_{cur} w.r.t. the distribution D_{cur}, and apply the standard two-point crossover [41]. Second, we select H individuals from P_{cur} with respect to the distribution D_{cur} without changes. Finally, we apply to each $G + H$ selected individuals the standard $(1+1)$ random mutation, flipping each bit with probability $1/n$. We ensure $G + H = R$ and compute the value of the objective function for new individuals in P_{new}. Then, we choose R best individuals from $P_{\text{cur}} \cup P_{\text{new}}$, and the resulting set becomes P_{cur} for the next iteration. In the experiments, we used $R = 8$ and $G = 4$.

5 Experimental Evaluation

Here we demonstrate that the proposed approach allows practically estimating the d-hardness of unsatisfiable CNF formulas with sufficiently high precision. As concrete examples, we consider equivalence checking encodings and crafted tests. For the value of $t_A(C[\beta/B])$ we select the number of unit propagations made by algorithm A while solving CNF formula $C[\beta/B]$. This choice, in contrast with using solving time, together with fixing the random seed of the solver, facilitates reproducibility of our results. We also show that sometimes our approach discovers sets B that may be used to solve SAT formulas in parallel with super-linear speedup.

5.1 Benchmarks

We consider two classes of CNF formulas or tests. The first class is comprised of so-called crafted tests. These are synthetic tests, constructed with the aim to generate formulas that are as hard as possible with as few variables as possible.

Quite a few generators of such tests are available. In this work we used the `sgen` generator [54] version 6. Only unsatisfiable instances were generated using `sgen`, instances are denoted $\texttt{sgen}^{\text{seed}}_{\#\text{variables}}$, describing the number of variables in the CNF formula and the random seed used to generate it, e.g. $\texttt{sgen}^{101}_{150}$. Search for the set B with the minimal value of function (10) was done on the entire set of variables of the CNF formula.

We also considered a class of tests related to equivalence checking [39]. Consider two Boolean circuits S_1 and S_2 over any complete basis, e.g. $\{\neg, \wedge\}$. We assume that each circuit has n inputs and m outputs. Thus, circuits S_1 and S_2 define functions

$$f_1 \colon \{0,1\}^n \to \{0,1\}^m, f_2 \colon \{0,1\}^n \to \{0,1\}^m$$

respectively. We need to prove that $f_1 \cong f_2$ (pointwise equality), in this case the circuits S_1 and S_2 are equivalent ($S_1 \cong S_2$). It is known [38] that this problem can be efficiently (in time linear of the number of elements in S_1 and S_2) reduced to SAT for a CNF formula C: $S_1 \cong S_2$ if and only if C is unsatisfiable. CNF formula C is constructed from circuits S_1 and S_2 using Tseitin transformations [55]. Bits of vectors from $\{0,1\}^n$ are encoded with variables forming the set $X^{\text{in}} = \{x_1, \ldots, x_n\}$, associated with inputs of S_1 and S_2.

The CNF formula constructed in this way exhibits the following important property. For a Boolean variable x and an arbitrary $\alpha \in \{0,1\}$ let us denote by $l_\alpha(x)$ the literal $\neg x$ if $\alpha = 0$ and literal x if $\alpha = 1$. Consider an arbitrary $\alpha = (\alpha_1, \ldots, \alpha_n)$, $\alpha_i \in \{0,1\}$, $i \in \{1, \ldots, n\}$ and the following CNF formula:

$$l_{\alpha_1}(x_1) \wedge \ldots \wedge l_{\alpha_n}(x_n) \wedge C. \tag{11}$$

It is known (see e.g. [8]) that (un)satisfiability of formula C can be determined by solely applying exhaustive unit propagation to CNF formulas (11) obtained across all possible assignments $\alpha \in \{0,1\}^{|X^{\text{in}}|}$. In other words, set X^{in} is a SUPBS for C. Then from the above it follows that we can search for B with a good value of $\mu_{B,A}(C)$ among the subsets of X^{in}. For this purpose we will launch a methaheuristic search minimizing the function (10) on the Boolean hypercube $\{0,1\}^{|X^{\text{in}}|}$.

We applied the described approach to equivalence checking of circuits S_1, S_2 representing two different algorithms which perform sorting of any d l-bit natural numbers. We considered the following sorting algorithms: bubble sorting, selection sorting [20], and pancake sorting [29]. Corresponding SAT encodings can be constructed using any software applied in symbolic verification, e.g. CBMC [17]; in this work we use Transalg [46, 51], which better suits our purposes. We conducted a substantial amount of experiments where equivalence of such circuits was checked. Below we present a few of these results. The SAT instances are denoted by $\texttt{BvS}_{l,d}$, $\texttt{BvP}_{l,d}$, and $\texttt{PvS}_{l,d}$ for Bubble vs Selection, Bubble vs Pancake, and Pancake vs Selection, respectively.

5.2 Experimental setup and implementation details

The proposed approach has been implemented in Python, using PySAT [35] for SAT solving with backend solvers Glucose 3 [3] and CaDiCaL [9], referred to as *g3* and *cd* respectively. The implementation of black-box optimization makes use of distributed computation. Experiments were run on a computing cluster using up to 5 nodes, each node includes two 18-core Intel Xeon E5-2695 2.1 GHz processors and 128 GB of RAM. Each experiment consisted of estimation optimization phase (looking for a set B with minimal estimation value $\tilde{\mu}_{B,A}(C)$) and estimation checking phase (exact calculation of $\mu_{B,A}(C)$). We used the evolutionary algorithms described above for traversing across the search space. We denote the (1+1) FEA_3 algorithm as "FEA", and the Genetic Algorithm from [47] as "GA". For calculating the value $\tilde{\mu}_{B,A}(C)$ the adaptive probabilistic (ε, δ)-algorithm presented above was applied. For each B such that $|B| \leq 9$ we directly calculated $\mu_{B,A}(C)$ instead of its estimation $\tilde{\mu}_{B,A}(C)$. In each case the optimization process of function (10) was run with a time limit of 12 hours, using confidence level $1 - \delta = 0.95$. Depending on the concrete CNF formula, we used different values of N ranging from 500 to 40000.

The goal of estimation checking was to assess the efficiency of the decomposition using the found set B. To achieve this we solved instances $C[\beta/B]$ for all $\beta \in \{0,1\}^{|B|}$ for several described benchmarks (in cases when $|B| \leq 17$), and thus calculated the exact value $\mu_{B,A}(C)$.

5.3 Main experimental results on d-hardness estimation

As mentioned above, we have performed a substantial amount of experiments on different SAT formulas. Here we only report on experiments with tests whose dimensionality allows explicitly checking the precision of resulting d-hardness estimations by exact calculation of $\mu_{B,A}(C)$. In the experiments with formula $\texttt{PvS}_{4,7}$ we found with our algorithm sets B consisting of three and fewer variables. In order to evaluate the quality of the corresponding decompositions we traversed through all possible sets B of sizes 1, 2, and 3. The corresponding problems are relatively simple, however, to solve them all we used about 3 days of runtime of a single cluster node (36 cores of Intel Xeon E5-2695) in total. Note that finding a set via solving an optimization problem for function (10) took up to 12 hours. The results are presented in Fig. 1 in the form of boxplot diagrams (whiskers span is 1.5 of interquartile range). The lower bound (in the number of propagations) of the diagrams corresponds to the best (smallest) value of function $\mu_{B,A}(C)$ over all possible B: $|B| \in \{1,2,3\}$. For several sets with the best values of $\mu_{B,A}(C)$ found by the proposed approach, these values are represented in the diagram: red dots correspond to sets found by FEA and blue crosses to the ones found by GA. Note that in every case our algorithms managed to find a set B for which the value of $\mu_{B,A}(C)$ is between the zeroth and first quartiles of the distribution depicted by the diagram. This proves that our algorithms can find good sets B.

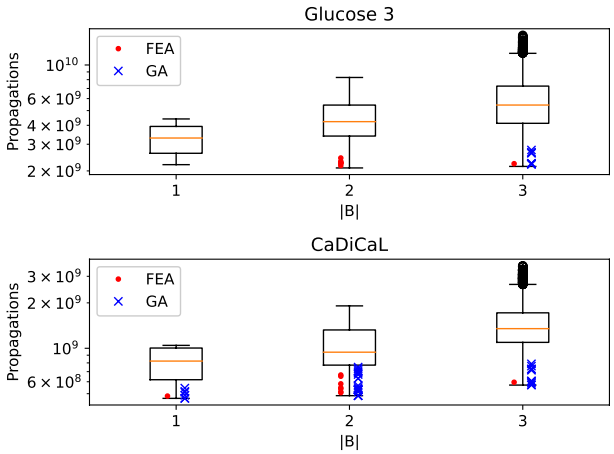

Figure 1 Boxplots for $\mu_{B,A}(C)$ of all sets B for CNF formula $\texttt{PvS}_{4,7}$, $|B| \leq 3$ and solvers `g3` (top) and `cd` (bottom), and examples of found estimations: our approach allows finding sets B allowing near-optimal hardness estimations.

Table 1 shows experimental results for several SAT instances. For each instance, SAT solver, and evolutionary algorithm, the table shows the cardinality of the found set B, the value $\mu_{B,A}(C)$, and the *decomposition rate* $r_{B,A}(C)$ calculated as $\mu_{B,A}(C)/t_A(C)$. Note that in most cases $r_{B,A}(C)$ is smaller than one, and thus, in these cases the corresponding slicing of formula C using the found set B yields a super-linear speedup when weakened formulas are solved in parallel. Also note that for equivalence checking tests $\texttt{PvS}_{4,7}$, $\texttt{BvS}_{4,7}$, $\texttt{BvP}_{4,7}$ search was done over SUPBSes consisting of $4 \times 7 = 28$ variables. For `sgen` search was done

Table 1 d-hardness estimations for different CNF formulas: most of the found sets B have decomposition rate $r_{B,A}(C) = \mu_{B,A}(C)/t_A(C) < 1$, making it possible to solve the $2^{|B|}$ weakened CNF formulas in parallel with super-linear speedup.

Instance	$\|X\|$	Solver A	Algorithm	$\|B\|$	$\mu_{B,A}/10^3$	$r_{B,A}$
$\texttt{PvS}_{4,7}$	3244	g3	FEA	3	2,190,213	0.792
		g3	FEA	4	2,250,504	0.814
		g3	GA	5	3,319,314	1.201
		g3	GA	6	3,333,915	1.206
		cd	FEA	3	595,695	1.043
$\texttt{sgen}^{1001}_{150}$	150	g3	FEA	5	101,371	0.424
		cd	FEA	6	244,191	0.763
		g3	GA	6	114,821	0.480
		cd	GA	7	247,947	0.775
$\texttt{sgen}^{101}_{150}$	150	g3	FEA	8	122,796	0.438
		cd	GA	7	131,557	0.470
$\texttt{sgen}^{200}_{150}$	150	g3	GA	7	151,275	0.569
		cd	GA	6	229,705	0.541
$\texttt{BvS}_{4,7}$	2134	g3	GA	3	460,944	1.140
		g3	FEA	3	449,325	1.112
$\texttt{BvP}_{4,7}$	2060	g3	FEA	3	726,080	1.049
		g3	GA	3	771,521	1.115

over the entire set X. Overall, we see that FEA performs slightly better than GA in terms of resulting $r_{B,A}(C)$ values. We can partially explain this by the fact that the GA uses more computational resources in one iteration in comparison with FEA.

We also performed experiments on searching for non-trivial SUPBSes in the sense of [59] among subsets of X^{in} for the $\texttt{PvS}_{4,7}$ example. Essentially, we implemented a variant of the algorithm from [59], enumerating subsets of X^{in} ($|X^{\text{in}}| = 28$) of gradually increasing cardinality. If for some $B \in 2^{X^{\text{in}}}$ the algorithm found such an assignment $\beta \in \{0,1\}^{|B|}$ that the application of the unit propagation rule to $C[\beta/B]$ was not enough to decide the satisfiability of $C[\beta/B]$, we concluded that B is not a SUPBS, and switched to the next candidate set B. As a result of these experiments, we have confirmed that for $\texttt{PvS}_{4,7}$ there is no such SUPBS B that $B \subset X^{\text{in}}$ (except X^{in} itself).

In all experiments we used the technique of dynamic adaptation of sample size described in Section 4.1. The plots in Fig. 2 show the dependence of ε on the iteration number for the instance $\texttt{sgen}^{1001}_{150}$: the purple plot does not adapt N (the initial value of N is 5000), while the blue, yellow, and green plots may increase N by up to a factor of two, four, and eight respectively. One may notice that the described strategy allows keeping ε below 0.1 most of the time, until finally the set B becomes small enough for switching to direct computation of $\mu_{B,A}(C)$, thus reducing ε to zero.

We also studied the accuracy of our estimation $\tilde{\mu}_{B,A}(C)$ with respect to its exact value $\mu_{B,A}(C)$. For this purpose we considered several intermediate sets B found by our approach for the $\texttt{PvS}_{4,7}$ formula and SAT solver `g3`. For each B we first calculated $\mu_{B,A}(C)$ by solving all $2^{|B|}$ weakened CNF formulas. Second, we calculated $\tilde{\mu}_{B,A}(C)$ using a sample size $N = \frac{1}{100}2^{|B|}$, and repeated this calculation 100 times with different random samples. The result is a distribution of values of $\tilde{\mu}_{B,A}(C)$. In Fig. 3 we depict these distributions

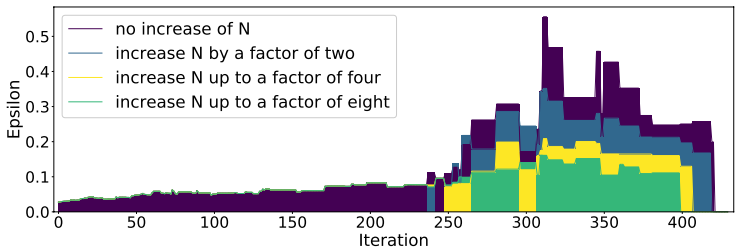

Figure 2 Dependence of ε from iteration number for $\texttt{sgen}_{150}^{1001}$ and g3: when N may be increased up to a factor of eight, the value of ε is below 0.1 most of the time.

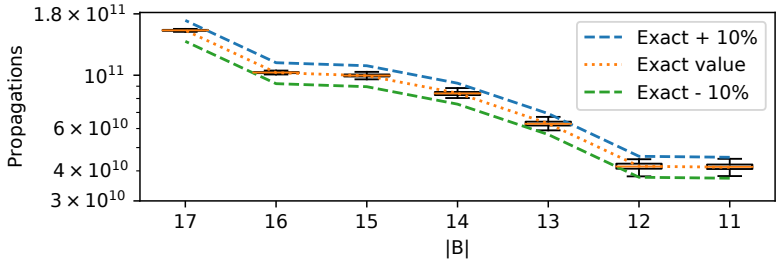

Figure 3 Accuracy of $\tilde{\mu}_{B,A}(C)$ for $\texttt{PvS}_{4,7}$ and g3: distributions of estimation values $\tilde{\mu}_{B,A}(C)$ remain within 10% of the exact value $\mu_{B,A}(C)$.

by boxplots for sets with $|B| \in \{17, 16, \ldots, 11\}$, in the order they were discovered by the evolutionary algorithm. Fig. 3 also shows with the dotted line the exact value $\mu_{B,A}(C)$ for each backdoor, and the +/- 10% interval around the exact value with dashed lines. As we can see, the distributions of $\tilde{\mu}_{B,A}(C)$ values remain within 10% of the exact value $\mu_{B,A}(C)$. Also, most importantly, the median value of $\tilde{\mu}_{B,A}(C)$ for each set B is almost exactly equal to the exact value $\mu_{B,A}(C)$ (the dotted line goes through horizontal lines in boxplots that depict the medians). This indicates that the approximation is quite accurate: if for set B the value of $\tilde{\mu}_{B,A}(C)$ is calculated once (as done during the optimization process), there is a high chance that the result will be very close to $\mu_{B,A}(C)$.

5.4 Hardness deviation of weakened CNF formulas

If hardness of weakened formulas differs drastically, one cannot achieve good speedup when solving them in parallel: if, e.g., solving one weakened formula requires, say, 95% of all propagations, then it would not be possible to get even a speedup that is linear in the number of used parallel threads. Thus, in order to use the sets B found by the proposed approach for parallel SAT solving, the corresponding sub-problems (weakened CNF formulas) need to be roughly equally hard. To check if the found sets B have this desired property, we have performed an experimental study regarding the variation of hardness of sub-problems.

More specifically, we measured the relative standard deviation of hardness of all $2^{|B|}$ sub-problems for each set B considered in Fig. 3 for CNF formula $\texttt{PvS}_{4,7}$ and SAT solver *g3*, and also for sets B of sizes from 12 to 20 for CNF formula $\texttt{sgen}_{150}^{200}$ and solver *g3*. Results are presented in Fig. 4 and Fig. 5 respectively.

As seen from the plots, for $\texttt{PvS}_{4,7}$ the relative standard deviation of sub-problem hardness does not exceed 0.003%, and for $\texttt{sgen}_{150}^{200}$ it is within 0.3%. This indicates that for these instances the weakened CNF formulas derived from the corresponding sets B are more or less of equal hardness, so there would be no issues during parallel solving.

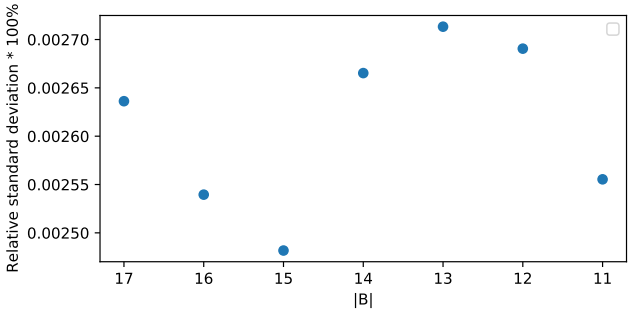

Figure 4 Relative standard deviation of sub-problem hardness for several sets B found for $\texttt{PvS}_{4,7}$ and *g3*.

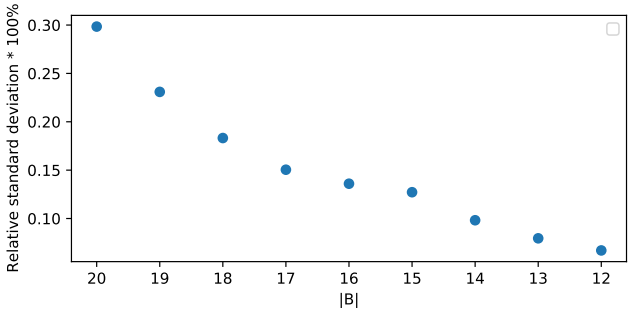

Figure 5 Relative standard deviation of sub-problem hardness for several sets B found for $\texttt{sgen}_{150}^{200}$ and *g3*.

5.5 Speedup in parallel solving

Here we show some results on solving the original CNF formulas in parallel by means of solving all $2^{|B|}$ weakened formulas derived from the set B generated by our approach. Table 2 shows values of speedup for several CNF formulas and sets B measured for 1..36 parallel threads. The speedup was evaluated as follows. In case of a single thread the speedup is $1/r_{B,A}(C)$, where $r_{B,A}(C)$ is the decomposition rate defined above. In case of $q, q \geq 2$ threads we first accumulated the total number of propagations made by A at each thread. Then we took the maximum value of the number of propagations across all threads and divide $t_A(C)$ by this value to compute the speedup. Thus, in the latter case we take into account the situation, when some threads have finished their work earlier than the others. Note that in the majority of cases, the speedup is indeed super-linear.

Of course, our approach does not and cannot guarantee that the speedup will be super-linear or even linear: apart from the set B itself, it depends on the properties of the CNF formula, the used strategy of parallel task distribution. However, practical results illustrated in Table 2 give reason to be optimistic.

5.6 Correspondence between the number of unit propagations and solving time

As noted above, in this paper for the value of $t_A(C[\beta/B])$ we select the number of unit propagations made by algorithm A while solving CNF formula $C[\beta/B]$. The reason for choosing this metric instead of just the running time (in seconds) is that the propagations metric is independent of the hardware platform, and the results can be replicated easily.

Table 2 Speedup when using set B to solve weakened CNF formulas on a single core and in parallel (using 2..36 threads).

Instance	$\lvert B\rvert$	Solver	1 thread	2 threads	4 threads	8 threads	16 threads	32 threads	36 threads
$\texttt{sgen}_{150}^{101}$	8	g3	2.3	4.6	8.8	16.8	31.3	37.0	37.0
	13	cd	1.9	3.9	7.7	14.9	29.4	56.9	62.6
$\texttt{sgen}_{150}^{200}$	8	g3	1.6	3.3	6.2	12.2	22.3	29.9	29.9
$\texttt{sgen}_{150}^{200}$	8	g3	1.8	3.6	7.1	13.3	25.5	36.2	36.2
	7	g3	2.2	4.4	8.5	15.8	28.0	28.8	28.8
	8	cd	1.3	2.6	5.0	9.6	19.1	22.8	22.8

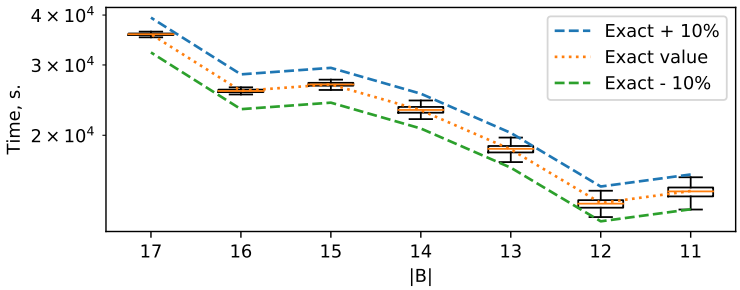

Figure 6 Accuracy of $\tilde{\mu}_{B,A}(C)$ for $\texttt{PvS}_{4,7}$ and g3, where $t_A(C[\beta/B])$ is the running time of the SAT solver in seconds: values of time and unit propagations are sufficiently, though not ideally, correlated.

However, in a practical application we would be interested in sets B that provide a speedup not only in the number of propagations, but also in terms of the running time. Therefore, we replicated results depicted in Fig. 3, measuring $t_A(C)$ and $t_A(C[\beta/B])$ in seconds (for a single thread).

Results are depicted in Fig. 6. Let us compare this plot with Fig. 3. Ideally (if the number of unit propagations exactly correlates with solving time), these plots should be quite the same, except for absolute values of propagations and time. Here, instead, we see that sometimes a decreased value of $\tilde{\mu}_{B,A}(C)$ (and also $\mu_{B,A}(C)$ for that matter) when $t_A(C[\beta/B])$ is measured in propagations corresponds to slightly increased values of $\tilde{\mu}_{B,A}(C)$ and $\mu_{B,A}(C)$ when $t_A(C[\beta/B])$ is measured in seconds: for example, this is the case for pairs ($|B| = 16$, $|B| = 15$) and ($|B| = 12$, $|B| = 11$). Despite this, the main trends of both plots are the same, indicating that when estimating the decomposition hardness the number of unit propagations can be considered as an adequate deterministic analog of a SAT solver running time.

6 Related Work

There have been a number of attempts to define hardness measures of Boolean formulas. Some of them are purely theoretical, others can be used in practical applications. For the most part, existing works appeal to the peculiarities of specific algorithms and do not consider the SAT solver as a black-box function, as it is done in our approach.

The relationship between the various measures of hardness is demonstrated in [2]. The key motivation for our work was the idea from [2] to determine the hardness of a Boolean formula, starting from the concept of the Backdoor Set introduced in [59]. This measure

(b-hardness) is determined only for Strong Backdoor Sets (SBS) of the function. The value of b-hardness on a particular SBS is equal to the total time required to solve all formulas obtained from the original CNF formula when partitioning it according to this SBS. Also recall that the b-hardness definition implies that weakened formulas are solved in polynomial time.

In our case, unlike [2] and [59], we use an arbitrary set of Boolean variables and an arbitrary (not necessarily polynomial) complete algorithm for solving SAT. In all other aspects our definition of decomposition hardness is similar to the definition of backdoor hardness. Actually, similar ideas have been used to evaluate the effectiveness of SAT Partitionings, mainly as applied to formulas arising in algebraic cryptanalysis: see e.g. [21, 26, 37, 50, 53, 60], etc. However, we emphasize that we are not aware of any works in which these ideas would be used to specify and estimate hardness of CNF formulas in general. Also, none of mentioned papers consider accuracy of obtained estimates or ways of improving this accuracy (such as our (ε, δ)-algorithm).

7 Discussion & Conclusion

Let us emphasize again that for any CNF formula there always exists such a set B that $\tilde{\mu}_{B,A}(C)$ can be calculated efficiently. Thus, for any extremely hard CNF formula we can always obtain some d-hardness estimation. It can be useful in cases when it is necessary to understand whether there is any practical sense in trying to solve the corresponding problem. One can argue that the accuracy of such estimates is questionable (e.g. due to transition from expectation and variance to their statistical counterparts), but our computational results show that they quite often turn to be accurate in practice. Sometimes, obtained preliminary estimates are not precise, but the resulting set B can give a very efficient decomposition (with rate $r_{B,A} < 1$).

As shown above, the cases when $r_{B,A} < 1$ are quite frequent in the studied classes of tests. In such a situation solving all CNF formulas $C[\beta/B]$ is cheaper than solving the original CNF formula without decomposition. Thus, if B has reasonable size, we may use a distributed computational platform to solve all weakened CNF formulas in parallel, and the corresponding speedup will be super-linear.

Recall that in this paper we only addressed hardness estimation for unsatisfiable CNF formulas. In the case of satisfiable formulas our estimation measure does not provide good accuracy guarantees: if a formula has many satisfying assignments, the SAT solving algorithm can get "lucky" or "unlucky", which would require other estimation measures, e.g. such as the one proposed in [52].

In conclusion, in this paper we have proposed a novel approach to evaluating the hardness of unsatisfiable SAT formulas w.r.t. a deterministic SAT solver. The new hardness measure, d-hardness, is computed w.r.t. a subset B of formula's variables, and corresponds to the minimal total computation effort needed to solve $2^{|B|}$ weakened CNF formulas across all possible subsets B. To illustrate the practical applicability of the new measure we proposed and developed an adaptive probabilistic (ε, δ)-approximation algorithm based on evolutionary optimization and demonstrated its effectiveness on tests from several families of SAT formulas.

We believe that the concept which lies in the base of the decomposition hardness can be useful in SAT solving strategies aimed at hard SAT instances. In the future, we plan to use the ideas which are close to the ones considered above to estimate the usefulness of cubes in the context of the Cube and Conquer approach [32].

Finally, although the paper argues that decomposition hardness can be effectively estimated with respect to any complete deterministic SAT solving algorithm, the presented experimental study focuses solely on a few SAT solvers based on conflict-driven clause learning (CDCL) [42]. As a result and given that the proof system of CDCL is known to be as strong as general resolution [4, 48], an interesting line of future work will be to extend the proposed ideas to existing algorithms that build on the proof systems strictly stronger than resolution, including cutting planes [19, 45] and dual-rail based MaxSAT [12], among others.

References

1 Michael Alekhnovich, Jan Johannsen, Toniann Pitassi, and Alasdair Urquhart. An exponential separation between regular and general resolution. In *STOC*, pages 448–456, 2002.
2 Carlos Ansótegui, Maria Luisa Bonet, Jordi Levy, and Felip Manyà. Measuring the hardness of SAT instances. In *AAAI*, pages 222–228, 2008.
3 Gilles Audemard, Jean-Marie Lagniez, and Laurent Simon. Improving Glucose for incremental SAT solving with assumptions: Application to MUS extraction. In *SAT*, pages 309–317, 2013.
4 Paul Beame, Henry A. Kautz, and Ashish Sabharwal. Towards understanding and harnessing the potential of clause learning. *J. Artif. Intell. Res.*, 22:319–351, 2004.
5 Paul Beame and Toniann Pitassi. Simplified and improved resolution lower bounds. In *FOCS*, pages 274–282, 1996.
6 Eli Ben-Sasson, Russell Impagliazzo, and Avi Wigderson. Near optimal separation of tree-like and general resolution. *Comb.*, 24(4):585–603, 2004.
7 Eli Ben-Sasson and Avi Wigderson. Short proofs are narrow - resolution made simple. *J. ACM*, 48(2):149–169, 2001.
8 Christian Bessiere, George Katsirelos, Nina Narodytska, and Toby Walsh. Circuit complexity and decompositions of global constraints. In *IJCAI*, pages 412–418, 2009.
9 Armin Biere. CaDiCaL at the SAT Race 2019. In *SAT Race*, pages 8–9, 2019.
10 Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors. *Handbook of Satisfiability (Second Edition)*, 2021.
11 Christian Blum and Andrea Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Comput. Surv.*, 35(3):268–308, 2003.
12 Maria Luisa Bonet, Sam Buss, Alexey Ignatiev, Antonio Morgado, and Joao Marques-Silva. Propositional proof systems based on maximum satisfiability. *Artif. Intell.*, 300:pages to appear, 2021.
13 Maria Luisa Bonet, Juan Luis Esteban, Nicola Galesi, and Jan Johannsen. Exponential separations between restricted resolution and cutting planes proof systems. In *FOCS*, pages 638–647, 1998.
14 Edmund Burke and Graham Kendall. *Search Methodologies*. Springer, 2014.
15 Samuel R. Buss and György Turán. Resolution proofs of generalized pigeonhole principles. *Theor. Comput. Sci.*, 62(3):311–317, 1988.
16 Chin-Liang Chang and Richard Char-Tung Lee. *Symbolic Logic and Mechanical Theorem Proving*. Academic Press, Inc., 1973.
17 Edmund M. Clarke, Daniel Kroening, and Flavio Lerda. A tool for checking ANSI-C programs. In *TACAS*, pages 168–176, 2004.
18 Stephen A. Cook and Robert A. Reckhow. The relative efficiency of propositional proof systems. *J. Symb. Log.*, 44(1):36–50, 1979.
19 William J. Cook, Collette R. Coullard, and György Turán. On the complexity of cutting-plane proofs. *Discret. Appl. Math.*, 18(1):25–38, 1987.
20 Thomas Cormen, Charles Leiserson, and Ronald Rivest. *Introduction to Algorithms*. MIT Press, 1990.
21 Nicolas T. Courtois and Gregory V. Bard. Algebraic cryptanalysis of the data encryption standard. In *IMACC*, pages 152–169, 2007.

22 Guoli Ding, Robert F. Lax, Jianhua Chen, Peter P. Chen, and Brian D. Marx. Transforms of pseudo-boolean random variables. *Discret. Appl. Math.*, 158(1):13–24, 2010.

23 Benjamin Doerr, Huu Phuoc Le, Régis Makhmara, and Ta Duy Nguyen. Fast genetic algorithms. In *GECCO*, pages 777–784, 2017.

24 William F. Dowling and Jean H. Gallier. Linear-time algorithms for testing the satisfiability of propositional horn formulae. *J. Log. Program.*, 1(3):267–284, 1984.

25 Stefan Droste, Thomas Jansen, and Ingo Wegener. On the analysis of the (1+1) evolutionary algorithm. *Theor. Comput. Sci.*, 276(1-2):51–81, 2002.

26 Tobias Eibach, Enrico Pilz, and Gunnar Völkel. Attacking bivium using SAT solvers. In *SAT*, pages 63–76, 2008.

27 Juan Luis Esteban and Jacobo Torán. Space bounds for resolution. *Inf. Comput.*, 171(1):84–97, 2001.

28 William Feller. *An Introduction to probability theory and its applications*, volume 2. John Wiley & Sons, Inc., 2 edition, 1971.

29 William H. Gates and Christos H. Papadimitriou. Bounds for sorting by prefix reversal. *Discret. Math.*, 27(1):47–57, 1979.

30 Carla P. Gomes and Ashish Sabharwal. Exploiting runtime variation in complete solvers. In *Handbook of Satisfiability (Second Edition)*, pages 463–480. IOS Press, 2021.

31 Armin Haken. The intractability of resolution. *Theor. Comput. Sci.*, 39:297–308, 1985.

32 Marijn Heule, Oliver Kullmann, Siert Wieringa, and Armin Biere. Cube and conquer: Guiding CDCL SAT solvers by lookaheads. In *HVC*, pages 50–65, 2011.

33 Frank Hutter, Holger H. Hoos, Kevin Leyton-Brown, and Thomas Stützle. Paramils: An automatic algorithm configuration framework. *J. Artif. Intell. Res.*, 36:267–306, 2009.

34 Frank Hutter, Marius Lindauer, Adrian Balint, Sam Bayless, Holger H. Hoos, and Kevin Leyton-Brown. The configurable SAT solver challenge (CSSC). *Artif. Intell.*, 243:1–25, 2017.

35 Alexey Ignatiev, Antonio Morgado, and Joao Marques-Silva. PySAT: A Python toolkit for prototyping with SAT oracles. In *SAT*, pages 428–437, 2018.

36 Richard M. Karp, Michael Luby, and Neal Madras. Monte-carlo approximation algorithms for enumeration problems. *J. Algorithms*, 10(3):429–448, 1989.

37 Stepan Kochemazov and Oleg Zaikin. ALIAS: A modular tool for finding backdoors for SAT. In *SAT*, pages 419–427, 2018.

38 Daniel Kroening. Software verification. In *Handbook of Satisfiability (Second Edition)*, pages 791–818, 2021.

39 Andreas Kuehlmann and Florian Krohm. Equivalence checking using cuts and heaps. In *DAC*, pages 263–268, 1997.

40 Oliver Kullmann. Upper and lower bounds on the complexity of generalised resolution and generalised constraint satisfaction problems. *Ann. Math. Artif. Intell.*, 40(3-4):303–352, 2004.

41 Sean Luke. *Essentials of Metaheuristics*. Lulu, second edition, 2013.

42 Joao Marques-Silva, Ines Lynce, and Sharad Malik. Conflict-driven clause learning SAT solvers. In *Handbook of Satisfiability (Second Edition)*, pages 133–182. IOS Press, 2021.

43 Nicholas Metropolis and S. Ulam. The Monte Carlo Method. *J. Amer. Statistical Assoc.*, 44(247):335–341, 1949.

44 Heinz Mühlenbein. How genetic algorithms really work: Mutation and hillclimbing. In *PPSN*, pages 15–26, 1992.

45 Jakob Nordström. On the interplay between proof complexity and SAT solving. *SIGLOG News*, 2(3):19–44, 2015.

46 Ilya V. Otpuschennikov, Alexander A. Semenov, Irina Gribanova, Oleg Zaikin, and Stepan Kochemazov. Encoding cryptographic functions to SAT using TRANSALG system. In *ECAI*, pages 1594–1595, 2016.

47 Artem Pavlenko, Alexander A. Semenov, and Vladimir Ulyantsev. Evolutionary computation techniques for constructing SAT-based attacks in algebraic cryptanalysis. In *EvoApplications*, pages 237–253, 2019.

48 Knot Pipatsrisawat and Adnan Darwiche. On the power of clause-learning SAT solvers as resolution engines. *Artif. Intell.*, 175(2):512–525, 2011.

49 Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach (4th Edition)*. Pearson, 2020.

50 Alexander Semenov and Oleg Zaikin. Algorithm for finding partitionings of hard variants of boolean satisfiability problem with application to inversion of some cryptographic functions. *SpringerPlus*, 5(1), 2006. Article no. 554.

51 Alexander A. Semenov, Ilya V. Otpuschennikov, Irina Gribanova, Oleg Zaikin, and Stepan Kochemazov. Translation of algorithmic descriptions of discrete functions to SAT with applications to cryptanalysis problems. *Log. Methods Comput. Sci.*, 16(1), 2020.

52 Alexander A. Semenov, Oleg Zaikin, Ilya V. Otpuschennikov, Stepan Kochemazov, and Alexey Ignatiev. On cryptographic attacks using backdoors for SAT. In *AAAI*, pages 6641–6648, 2018.

53 Mate Soos, Karsten Nohl, and Claude Castelluccia. Extending SAT solvers to cryptographic problems. In *SAT*, pages 244–257, 2009.

54 Ivor T. A. Spence. Weakening cardinality constraints creates harder satisfiability benchmarks. *ACM J. Exp. Algorithmics*, 20:1.4:1–1.4:14, 2015.

55 Grigoriy Tseitin. On the complexity of derivation in propositional calculus. *Studies in Constr. Math. and Math. Logic*, pages 115–125, 1970.

56 Alasdair Urquhart. The complexity of propositional proofs. *Bull. Symb. Log.*, 1(4):425–467, 1995.

57 Ingo Wegener. Theoretical aspects of evolutionary algorithms. In *ICALP*, pages 64–78, 2001.

58 Samuel S. Wilks. *Mathematical statistics*. John Wiley and Sons, 1962.

59 Ryan Williams, Carla P. Gomes, and Bart Selman. Backdoors to typical case complexity. In *IJCAI*, pages 1173–1178, 2003.

60 Oleg S. Zaikin and Stepan E. Kochemazov. On black-box optimization in divide-and-conquer SAT solving. *Optimization Methods and Software*, pages 1–25, 2019.