Estimating the hardness of SAT encodings for Logical Equivalence Checking of Boolean circuits

Alexander Semenov, Konstantin Chukharev, Egor Tarasov, Daniil Chivilikhin, and Viktor Kondratiev

ITMO University, St. Petersburg, Russia
alex.a.semenov@itmo.ru

Abstract. In this paper we investigate how to estimate the hardness of Boolean satisfiability (SAT) encodings for the Logical Equivalence Checking problem (LEC). Meaningful estimates of hardness are important in cases when a conventional SAT solver cannot solve a SAT instance in a reasonable time. We show that the hardness of SAT encodings for LEC instances can be estimated *w.r.t.* some SAT partitioning. We also demonstrate the dependence of the accuracy of the resulting estimates on the probabilistic characteristics of a specially defined random variable associated with the considered partitioning. The paper proposes several methods for constructing partitionings, which, when used in practice, allow one to estimate the hardness of SAT encodings for LEC with good accuracy. In the experimental part we propose a class of scalable LEC tests that give extremely complex instances with a relatively small input size n of the considered circuits. For example, for $n = 40$, none of the state-of-the-art SAT solvers can cope with the considered tests in a reasonable time. However, these tests can be solved in parallel using the proposed partitioning methods.

1 Introduction

Boolean circuits are widely used in theoretical computer science [1,18] as well as in numerous industrial applications. It would take too much space to list all the key references regarding the various practical applications of Boolean circuits. We only note that each hardware implementation of an arbitrary discrete function (*i.e.* function $f\colon \{0,1\}^n \to \{0,1\}^m$) can be viewed as some Boolean circuit, entailing the development of such a colossal industry as Electronic Design Automation (EDA).

One of the main problems related to Boolean circuits is the logical equivalence checking problem (LEC) [24,28]. This problem is posed as follows: there are two circuits S_f, S_h specifying some functions $f, h\colon \{0,1\}^n \to \{0,1\}^m$. The question is: "Is it true that f and h are equal, *i.e.* point-wise equality $f \cong h$ holds?". At the initial stage of development of formal verification methods, Binary Decision Diagrams (BDD) [9] were used to solve LEC. Works [4,5] argued in favor of solving LEC via applying complete SAT solvers based on the CDCL algorithm [25]; currently, LEC is mainly solved with such algorithms: a good example is the ABC [8] framework.

SAT solvers work with Boolean formulas in Conjunctive Normal Form (CNF). There is an algorithm linear in the size of circuits S_f, S_h that reduces LEC for these circuits to SAT for a CNF formula using Tseytin transformations [31].

Unfortunately, SAT for a CNF formula which encodes LEC for S_f and S_h can be difficult for state-of-the-art SAT solvers. If we use a sequential solver, in many cases we cannot even say how much time can be required for solving the corresponding SAT instance. Prediction of runtime for modern SAT solvers is very difficult in the general case due to their heavy-tailed behavior [19].

The main goal of this paper is to show that in many cases we can quite accurately estimate the hardness of a SAT instance which encodes some LEC problem by decomposing this instance into a family of significantly simpler SAT instances. Also, we demonstrate a connection between the hardness of LEC for two circuits and the number of satisfying assignments of some satisfiable CNF formula constructed in a special way for the considered pair of circuits. To estimate the said number of satisfying assignments we propose a probabilistic algorithm based on the Monte Carlo method [26].

2 Preliminaries

In this section, we introduce the necessary formal concepts and notation.

2.1 Satisfiability and Boolean circuits

We start from basic concepts related to SAT, the Boolean Satisfiability problem [7]. In the context of SAT one usually works with a Boolean formula in CNF.

Let C be an arbitrary CNF formula and X be the set of Boolean variables occurring in C. An *assignment* of variables from X is a mapping $\alpha\colon X \to \{0,1\}$. The set of all different assignments of variables from X is denoted as $\{0,1\}^{|X|}$ and called *Boolean hypercube* of dimension n, $n = |X|$.

In the context of SAT, for an arbitrary CNF formula C it is required to answer the following question: is it true that C is satisfiable? That is, is there as assignment of variables from X for which C is evaluated to *true*? In this formulation, SAT is NP-complete, and it is NP-hard when one has to detect the satisfiability of C and, in the case of a positive answer, to find some satisfying assignment. Despite the theoretical hardness of SAT, the last 20 years demonstrate impressive progress in the development of SAT solving algorithms with a wide spectrum of practical applications in symbolic verification, computational combinatorics, bioinformatics, cryptanalysis, etc. One of the most striking examples is hardware verification and, in particular, Logical Equivalence Checking (LEC). As it was said above, in LEC one has to answer the following question: is it true that two Boolean circuits are equivalent?

As in the majority of related articles, we regard a *Boolean circuit* as some directed acyclic graph. Consequently, we use the following standard graph theory definitions. A (directed) *graph* $G = (V, E)$ consists of a set of vertices V and a set of (directed) edges $E \subseteq V^2$. An *edge* is a pair of connected vertices. An *arc* is

a directed edge, *i.e.* an ordered pair of vertices. For each arc $(u, v) \in E$, vertex u is called a *parent* of v, and v is called a *child* of u. The set of all parents of a vertex v is denoted by P_v. The *indegree* of a vertex v is the number of parents of v, and the *outdegree* is the number of children. A vertex is called an *input* if it has no parents, and an *output* if it has no children. The sets of inputs and outputs are denoted as $V^{in} \subset V$ and $V^{out} \subset V$ respectively. A *path* is a sequence of arcs. A vertex u is called a *predecessor* of v if there is a path from u to v. A predecessor u which is also an input ($u \in V^{\text{in}}$) is called an *ancestor* of v. The set of all ancestors of a vertex v is denoted by A_v (*ancestor set*).

A Boolean circuit with n inputs and m outputs can be considered a natural way to specify some discrete function $f\colon \{0,1\}^n \to \{0,1\}^m$. Implying this, we will denote an arbitrary Boolean circuit defining a discrete function f as $S_f = (V, E)$.

Let S_f be an arbitrary Boolean circuit. Any vertex $v \in V \setminus V^{in}$ is called a *gate*. Each gate is associated with some *logical connective* from a predefined set called a *basis* (for example it can be $\{\wedge, \neg\}$, $\{\wedge, \vee, \neg\}$, $\{\wedge, \oplus, 1\}$, etc.). An example of a graphical representation of a Boolean circuit with $|V^{in}| = 3$ inputs and $|V \setminus V^{in}| = 8$ gates is shown in Fig. 1.

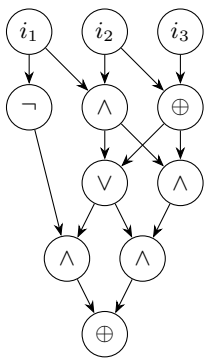

Fig. 1. An example Boolean circuit with three inputs (i_1, i_2, i_3) and eight gates

The set of vertices V of a circuit S_f can be naturally partitioned into subsets called "layers", which are defined inductively as follows.

Definition 1 (Circuit layers). *Let $V_0 = V^{in}$ denote the zeroth circuit layer. The k-th ($k \geq 1$) circuit layer V_k is defined inductively as the set of all vertices v satisfying the following two properties:*

1. $v \notin \bigcup_{j=0}^{k-1} V_j$;
2. $P_v \subseteq \bigcup_{j=0}^{k-1} V_j$.

Definition 2 (Associated functions). *With each gate $v \in V \setminus V^{in}$ let us associate a predefined Boolean function $g_v\colon \{0,1\}^{|P_v|} \to \{0,1\}$. The value of g_v is uniquely determined by the values of the functions g_w ($w \in P_v$) with respect to the semantics of the logical connective which is associated with gate v.*

Let us fix some order on set V^{in} and the same order will apply to the bits of an arbitrary word from $\{0,1\}^n$. Thus, each bit of an arbitrary word $\alpha \in \{0,1\}^n$ is uniquely connected with some vertex from V^{in}. Let us say that α is an *input word* of S_f, or simply input.

Definition 3 (Circuit interpretation). *Let $\alpha \in \{0,1\}^n$ be an arbitrary input of the circuit S_f. Begin traversing the circuit starting from the first layer V_1. For any $v \in V_1$ we suppose that the value of an arbitrary g_w ($w \in P_v$, $P_v \subseteq V^{in}$) is equal to the corresponding bit of α associated with w. For an arbitrary gate $v \in V_j$, $j > 1$, let us calculate the value of g_v on α using known values of g_w on this input word for all $w \in P_v$. We will also say that this value of g_v is* induced *by α. Continue the evaluation until the values of functions g_v are calculated for all gates of circuit S_f. Let us call the described process the* interpretation *of the circuit S_f on input word α.*

Let S_f be a Boolean circuit with n inputs and m outputs. Note that the interpretation of S_f specifies a total function $f\colon \{0,1\}^n \to \{0,1\}^m$. The value of this function on an arbitrary word $\alpha \in \{0,1\}^n$ is a Boolean vector $\gamma = (\gamma_1, \ldots, \gamma_m)$, where γ_k, $k \in \{1, \ldots, m\}$, are the values of functions g_v induced by α for all $v \in V^{out}$.

Definition 4 (Associated variables). *Let us associate with each vertex of circuit S_f a particular Boolean variable and denote the set of all such variables as X. Let X^{in} be the set of Boolean variables* associated *with the inputs of S_f; we will refer to these variables as to* input variables. *The variables assigned to gates will be called* auxiliary variables. *For an arbitrary $\widetilde{V} \subseteq V$, let $var(\widetilde{V})$ denote the set of Boolean variables assigned to nodes from $\widetilde{V}$. With a slight abuse of notation, we write $var(v) = x$ for a singleton vertex instead of $var(\{v\}) = \{x\}$.*

Let v be an arbitrary gate in S_f, and let $U_v = var(P_v)$, $u = var(v)$. Let g_v be a Boolean function corresponding to the gate v, and let $F(g_v)$ be an arbitrary Boolean formula over U_v (for example, a canonical CNF), which defines g_v. For a gate v, we denote by C_v the CNF representation of formula $(F(g_v) \equiv u)$.

Definition 5 (Template CNF formula). *Let S_f be some Boolean circuit which specifies the function $f\colon \{0,1\}^n \to \{0,1\}^m$. We will refer to the CNF formula $C_f = \bigwedge_{v \in V \setminus V^{in}} C_v$ as to the* template CNF formula *for function f.*

Note that C_f is in fact the CNF formula which can be obtained from S_f by applying Tseytin transformations [31].

Recall the following notation: $x^\sigma = \left\{ \begin{smallmatrix} x, \text{ if } \sigma=1 \\ \neg x, \text{ if } \sigma=0 \end{smallmatrix} \right.$. Let Φ be an arbitrary Boolean formula over the variables X. Denote by $\Phi|_{x=\sigma}$ the formula obtained by substituting x with σ in Φ [10]. It is clear that the formulas $x^\sigma \wedge \Phi$ and $\Phi|_{x=\sigma}$ are equisatisfiable. Thus, when working with the formula $x^\sigma \wedge \Phi$, we can regard the unit clause x^σ as the value σ of the variable x in the formula Φ.

Definition 6 (Cone). *Let v be an arbitrary gate in S_f, and Q_v be the set of all predecessors of v. The set $Q_v \cup \{v\}$ is called the* cone *of v.*

The following fact has been repeatedly established in the literature, see *e.g.* [3,15]. It uses a simple Boolean constraint propagation mechanism known as the Unit Propagation rule (UP) [14,25].

Lemma 1. *Let C_f be the template CNF formula for a circuit S_f. Let v be an arbitrary gate of S_f, the set Q_v be the cone of v, A_v, $A_v \subseteq Q_v$ be the ancestor set of v, and $X_v = \{x_{v,1}, \ldots, x_{v,r}\} = var(A_v)$ be the set of variables associated with A_v ($X_v \subseteq X^{in}$). Then, for each $(\alpha_1, \ldots, \alpha_r) \in \{0,1\}^{|X_v|}$, application of the UP rule to the CNF formula $x_{v,1}^{\alpha_1} \wedge \ldots \wedge x_{v,r}^{\alpha_r} \wedge C_f$ derives (in the form of unit clauses) the values of all variables from $var(Q_v) \setminus X_v$. Moreover, for the variable $u = var(v)$, the derived value is equal to the value of function g_v induced by any input word $\alpha \in \{0,1\}^n$ of S_f which contains (w.r.t. corresponding variables) the sub-vector $(\alpha_1, \ldots, \alpha_r)$. Note that the resulting set of unit clauses does not contain conflicting literals.*

Proof. The proof of this lemma uses the traversal of S_f by layers and the properties of Tseytin transformations.

Corollary 1 (of Lemma 1). *Application of UP to the CNF formula $x_1^{\alpha_1} \wedge \ldots \wedge x_n^{\alpha_n} \wedge C_f$ for any $(\alpha_1, \ldots, \alpha_n) \in \{0,1\}^{|X^{in}|}$ derives (in the form of unit clauses) the values of all variables associated with gates from $V \setminus V^{in}$, including the variables from $var(V^{out}) = \{y_1, \ldots, y_m\}$: $y_1 = \gamma_1, \ldots, y_m = \gamma_m$, $f(\alpha) = \gamma$, $\alpha = (\alpha_1, \ldots, \alpha_n)$, $\gamma = (\gamma_1, \ldots, \gamma_m)$.*

Note that from Lemma 1 and Corollary 1 it follows that the process of interpretation of circuit S_f on an arbitrary input word $(\alpha_1, \ldots, \alpha_n)$ is modelled by consecutive application of the UP rule to the CNF formula $x_1^{\alpha_1} \wedge \ldots \wedge x_n^{\alpha_n} \wedge C_f$.

2.2 SAT partitioning

As mentioned above, SAT is NP-hard, so some instances of SAT can be very difficult for conventional solvers. There are several approaches to parallelizing SAT solving [21], the main ones being the portfolio approach (*e.g.* [2]) and the partitioning approach (*e.g.* Cube and Conquer [20]). In this paper, we follow the partitioning approach.

Let us consider an arbitrary CNF formula C over the set of Boolean variables X and set $\Pi = \{G_1, \ldots, G_s\}$, where G_i ($i \in \{1, \ldots, s\}$) are some Boolean formulas. Let us say that the set Π yields a SAT partitioning of C if the following conditions hold:

- formulas C and $C \wedge (G_1 \vee \cdots \vee G_s)$ are equisatisfiable;
- for each $i, j \in \{1, \ldots, s\}$, $i \neq j$, formula $C \wedge G_i \wedge G_j$ is unsatisfiable.

For some set of variables $B \subseteq X$, $B = \{x_{k_1}, \ldots, x_{k_r}\}$, each formula $x_{k_1}^{\alpha_1}, \ldots x_{k_r}^{\alpha_r}$ for an arbitrary $(\alpha_1, \ldots, \alpha_r) \in \{0,1\}^r$ is called a *cube* (over X). For an arbitrary CNF formula C over the set of variables X, a simple example of a partitioning is generated by set $\Pi = \{G_1, \ldots, G_{2^r}\}$, which consists of all possible cubes over an arbitrary set B, $|B| = r$. In the following, we will use the term *partitioning* for both the set Π and for the set of CNF formulas generated by Π.

2.3 Background from probability theory

Below we will use some probabilistic reasoning to estimate the hardness of SAT encodings for LEC instances. Let us recall some relevant basic facts from probability theory.

Let ξ be some random variable with finite spectrum $\{\xi_1, \ldots, \xi_M\}$ and probability distribution $P_\xi = \{p_1, \ldots, p_M\}$. In the following, use assume that $0 < \xi_i < \infty$ for every $i \in \{1, \ldots, M\}$. Then, the expected value (expectation) of ξ is defined as $E[\xi] = \sum_{i=1}^{M} \xi_i p_i$. In many practical applications, knowledge of $E[\xi]$ turns out to be very important. However, it is often impossible to accurately calculate the exact value of $E[\xi]$ in a reasonable amount of time. In such cases, one can instead estimate $E[\xi]$ with some predetermined accuracy ε. The corresponding algorithms use random sampling and traditionally refer to the Monte Carlo method [26].

More precisely, let $\xi_1, \ldots, \xi_N$ be independent observations of the random variable ξ. Then, Chebyshev's inequality [16] implies:

$$Pr\left\{(1-\varepsilon)\cdot E[\xi] \leq \frac{1}{N}\sum_{j=1}^{N}\xi_j \leq (1+\varepsilon)\cdot E[\xi]\right\} \geq 1 - \frac{Var(\xi)}{\varepsilon^2 \cdot N \cdot E^2[\xi]}, \tag{1}$$

where $Var(\xi)$ denotes the variance of the random variable ξ. It follows from (1) that for finite $E[\xi]$ and $Var(\xi)$, the expectation $E[\xi]$ can be approximated (in the sense of (1)) by the value $\frac{1}{N}\sum_{j=1}^{N}\xi_j$ with any tolerance ε given in advance by increasing the number of observations N.

The following inequality is less accurate than (1) in the general case, but it is often more convenient to use for $E[\xi]$ estimation:

$$Pr\left\{\left|E[\xi] - \frac{1}{N}\sum_{j=1}^{N}\xi_j\right| \leq \varepsilon\right\} \geq 1 - \frac{Var(\xi)}{\varepsilon^2 \cdot N}. \tag{2}$$

If ξ is a Bernoulli variable, *i.e.* takes values in $\{0, 1\}$, then instead of (2) one can use the following variant of Chernoff bound (see *e.g.* [29]):

$$Pr\left\{\left|E[\xi] - \frac{1}{N}\sum_{j=1}^{N}\xi_j\right| \leq \varepsilon\right\} \geq 1 - 2 \cdot e^{-\frac{\varepsilon^2 \cdot N}{4}}. \tag{3}$$

The proof of (3) can be found in [22].

3 Estimating the hardness of SAT encodings of LEC instances using SAT partitioning

Let us return to LEC. Consider two Boolean circuits S_f, S_h defining functions $f, h\colon \{0,1\}^n \to \{0,1\}^m$. Let us construct a circuit which will be denoted by $S_{f\triangle h}$. This circuit is obtained from S_f and S_h by "gluing" together the input vertices (see Fig. 2). Thus, this circuit has the same V^{in} as S_f and S_h, and defines the function $f \triangle h : \{0,1\}^n \to \{0,1\}^{2m}$.

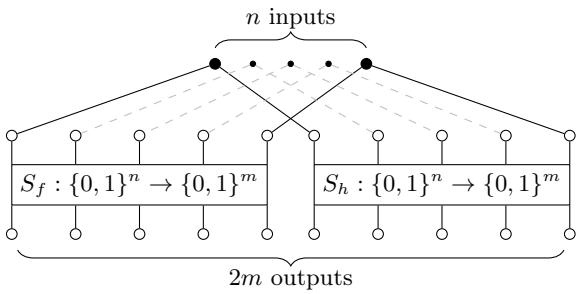

Fig. 2. Circuit $S_{f \triangle h}$ constructed by using the same set of inputs for two circuits S_f and S_h

Denote V_f^{out} and V_h^{out} the output sets of circuits S_f, S_h, and denote $Y_f = \{y_1^f, \ldots, y_m^f\}$ and $Y_h = \{y_1^h, \ldots, y_m^h\}$ the sets of variables assigned to vertices from V_f^{out} and V_h^{out} and ordered according to the semantics of the circuits. Consider the formula:

$$\left(y_1^f \oplus y_1^h\right) \vee \cdots \vee \left(y_m^f \oplus y_m^h\right). \tag{4}$$

Formula (4) defines a Boolean function $\mu\colon \{0,1\}^{2m} \to \{0,1\}$ called a *miter*. Let $C(\mu)$ be the CNF representation of μ. It follows directly from Lemma 1 that circuits S_f and S_h are equivalent if and only if the following CNF formula is unsatisfiable:

$$C_{f \triangle h} \wedge C(\mu),$$

where $C_{f \triangle h}$ is the template CNF formula for function $f \triangle h$. Consider below another corollary of Lemma 1.

Corollary 2. *For two arbitrary functions $f, h\colon \{0,1\}^n \to \{0,1\}^m$ specified by circuits S_f, S_h, the number of assignments satisfying template CNF formulas C_f, C_h, and $C_{f \triangle h}$, is 2^n.*

A rather interesting observation is the following. Modern CDCL-based SAT solvers, receiving a CNF formula of the form C_f or $C_{f \triangle h}$ as input, very quickly (usually within a fraction of a second) generate some satisfying assignment corresponding to some input/output pair. At the same time, CNF formulas (3) can be very hard. It is worth to note that formulas $C_{f \triangle h}$ and (3) differ from each other only by clauses corresponding to the miter (their fraction in (3) usually is extremely small).

Everywhere below, we assume that $\mathcal{O}$ is an arbitrary complete SAT solver. If formula (3) is difficult for $\mathcal{O}$, then often we cannot even say how much time $\mathcal{O}$ will take to solve this SAT instance. The difficulty of this kind of assessment is due to an effect that is known as the heavy-tailed behavior of CDCL-based SAT solvers [19]. However, in some cases, we can estimate the overall hardness of a SAT instance quite efficiently and accurately by estimating the hardness of its SAT partitioning.

Definition 7 (Hardness of SAT instance). *Let C be an arbitrary CNF formula, Π be some partitioning of C, and $\mathcal{O}$ be some complete SAT solver. The total runtime of solver $\mathcal{O}$ on instances $G \wedge C$ for all formulas $G \in \Pi$ is called the* hardness *of C w.r.t solver $\mathcal{O}$ and partitioning Π, and is denoted as $T_{\mathcal{O}}(C, \Pi)$.*

Below we show that $T_{\mathcal{O}}(C, \Pi)$ can be estimated using a probabilistic Monte Carlo algorithm. Let us describe the general scheme for constructing such estimates.

Let C is an arbitrary SAT instance and $\Pi = \{G_1, \ldots, G_s\}$ be some partitioning of C. If s is large, then it is possible to estimate the time needed to solve C *w.r.t.* Π through an estimate of the solution time of N SAT instances $G_k \wedge C$, $k \in \{1, \ldots, N\}$ chosen from Π according to some distribution. As a rule, at the initial stage we fix a uniform distribution on Π. Let us introduce a random variable ξ_Π whose values are equal to the running time of the SAT solver $\mathcal{O}$ on formulas $G_j \wedge C$, $j \in \{1, \ldots, s\}$. Let $Spec(\xi_\Pi) = \{\xi_1, \ldots, \xi_Q\}$ be the spectrum of ξ_Π, and each value ξ_r, $r \in \{1, \ldots, Q\}$, is assigned a probability $p_r = \frac{\#\xi_r}{s}$, where $\#\xi_r$ denotes the number of such G_j, $j \in \{1, \ldots, s\}$, that the running time of $\mathcal{O}$ on the formula $G_j \wedge C$ is ξ_r. Thus, ξ_Π has the distribution law $P(\xi_\Pi) = \{p_1, \ldots, p_Q\}$. Recall again that $\mathcal{O}$ is complete SAT solver, so ξ_Π has finite spectrum, expectation value, and variance. The following fact is true.

Theorem 1. *The hardness of SAT instance C w.r.t. solver $\mathcal{O}$ and SAT partitioning Π is $T_{\mathcal{O}}(C, \Pi) = s \cdot E[\xi_\Pi]$.*

Proof.

$$T_{\mathcal{O}}(C, \Pi) = \sum_{r=1}^{Q} \xi_r \cdot \#\xi_r = s \cdot \sum_{r=1}^{Q} \xi_r \cdot \frac{\#\xi_r}{s} = s \cdot E[\xi_\Pi].$$

The running time of $\mathcal{O}$ can be measured in any convenient units, for example, in seconds, the number of times the Unit Propagation rule is applied, or the number of conflicts generated by $\mathcal{O}$.

To estimate $E[\xi_\Pi]$, one can use the Monte Carlo method and specifically the formula (1). Despite the formal possibility of achieving any estimation accuracy by increasing the number of observations N of the value ξ_Π, in many practical cases, the obtained estimates may be inaccurate due to high variance $Var(\xi_\Pi)$, which, in turn, is a consequence of the effect of heavy-tailed behavior of CDCL SAT solvers. Thus, arises the problem of constructing such partitionings for which $Var(\xi_\Pi)$ would not exceed some reasonable limit: for example, the standard deviation $\sigma = +\sqrt{Var(\xi_\Pi)}$ should not exceed $E[\xi_\Pi]$. Below we describe two general SAT partitioning constructions for which σ has relatively small values on the LEC instances discussed below. The ideas underlying such constructions are based on the properties of CNF formulas C_f, C_h, $C_{f \triangle h}$, and $C_{f \triangle h} \wedge C(\mu)$.

Consider LEC for circuits S_f, S_h ($f, h\colon \{0,1\}^n \to \{0,1\}^m$). Let us once again focus on the fact that the CNF formula $C_{f \triangle h}$ has 2^n satisfying assignments, while the CNF formula $C_{f \triangle h} \wedge C(\mu)$ has none if $S_f \cong S_h$. Let $\Pi = \{G_1, \ldots, G_s\}$ be an arbitrary SAT partitioning of $C_{f \triangle h}$. Denote by $\#(G \wedge C)$ the number of

satisfying assignments of the formula $G \wedge C$ for an arbitrary $G \in \Pi$. It is easy to deduce the following fact from the general properties of the SAT partitioning and Lemma 1.

Proposition 1. *Let Π be an arbitrary SAT partitioning of $C_{f \triangle h}$. Then, the following equation holds:*

$$\sum_{G \in \Pi} \#(G \wedge C_{f \triangle h}) = 2^n.$$

Thus, an arbitrary formula $G \wedge C_{f \triangle h}$ has $\#(G \wedge C_{f \triangle h})$ satisfying assignments (and this number can be significantly larger than zero), but at the same time the formula $G \wedge C_{f \triangle h} \wedge C(\mu)$ is unsatisfiable if S_f and S_h are equivalent. Allowing a somewhat loose interpretation, we can say that by proving the unsatisfiability of $G \wedge C_{f \triangle h} \wedge C(\mu)$, we *block* $\#(G \wedge C_{f \triangle h})$ satisfying assignments of formula $G \wedge C_{f \triangle h}$. In total, when solving all problems in the SAT partitioning, we need to block all 2^n satisfying assignments of $G \wedge C_{f \triangle h}$ (the sets of assignments of different formulas $G \wedge C_{f \triangle h}$ are disjoint).

Taking into account all said above, there arises an attractive idea to link the hardness of formulas $G \wedge C_{f \triangle h} \wedge C(\mu)$ with the number of satisfying assignments of corresponding formulas $G \wedge C_{f \triangle h}$. Looking ahead, let us note that our computational experiments demonstrate this exact connection: the more satisfying assignments the formula $G \wedge C_{f \triangle h}$ has, the harder the formula $G \wedge C_{f \triangle h} \wedge C(\mu)$ is. Thus, if we want all problems $G \wedge C_{f \triangle h} \wedge C(\mu)$ in the SAT partitioning Π to have approximately equal hardness (which would correspond to a relatively low variance $Var(\xi_\Pi)$), we must ensure that all formulas $G \wedge C_{f \triangle h}$ have an approximately equal number of satisfying assignments. In the following, we describe two types of SAT partitionings that satisfy these requirements.

Construction 1 *Denote as $\widetilde{C}$ the CNF formula $C_{f \triangle h}$ or CNF formula $C_{f \triangle h} \wedge C(\mu)$. Consider the set of variables $X^{in} = \{x_1, \ldots, x_n\}$ assigned to the inputs of the circuit $S_{f \triangle h}$. Let us divide (generally speaking, in an arbitrary way) the set X^{in} into disjoint subsets of variables with k, $k \geq 1$ variables in each group. For simplicity, we will assume that n is divisible by k. We have sets $X^1, \ldots, X^{n/k}$. With each set X^j, $j \in \{1, \ldots, n/k\}$ we associate an arbitrary non-constant Boolean function λ_1^j, $\lambda_1^j \colon \{0,1\}^{|X^j|} \to \{0,1\}$, and the function $\lambda_2^j = \neg \lambda_1^j$. Let ϕ_1^j, ϕ_2^j be arbitrary formulas defining the functions λ_1^j and λ_2^j. Let the formula $\phi^1 \wedge \cdots \wedge \phi^{n/k}$ be an arbitrary formula in which ϕ^j denotes either the occurrence of ϕ_1^j or the occurrence of ϕ_2^j. It is easy to see that the following fact holds.*

Proposition 2. *The set of all $2^{n/k}$ possible formulas $\phi^1 \wedge \cdots \wedge \phi^{n/k} \wedge \widetilde{C}$ forms a SAT partitioning of $\widetilde{C}$.*

Note that if all functions λ_1^j, λ_2^j are *balanced* (take values 0 and 1 on an equal number of sets of values of variables from X^j), then for any set $\phi^1, \ldots, \phi^{n/k}$ the formula $\phi^1 \wedge \cdots \wedge \phi^{n/k} \wedge C_{f \triangle h}$ has $2^{(1-1/k)/n}$ satisfying assignments.

For example, let us suppose that n is even and consider the following case: $X^1 = \{x_1, x_2\}, X^2 = \{x_3, x_4\}, \ldots, X^{n/2} = \{x_{n-1}, x_n\}$. We choose the functions $\lambda_1^j(u, v) = (u \oplus v)$, $\lambda_2^j(u, v) = (u \equiv v)$. Then, each formula $\phi^1 \wedge \cdots \wedge \phi^{n/2} \wedge C_{f \triangle h}$ has $2^{n/2}$ satisfying assignments.

The second class of SAT partitionings which gives good results on LEC instances uses specially constructed cubes over subsets $B \subset X \setminus X^{in}$.

Construction 2 *Let us start again with LEC for two circuits S_f, S_h and consider CNF formulas $C_f, C_h, C_{f \triangle h}, C_{f \triangle h} \wedge C(\mu)$. Let X_f, X_h be the sets of variables occurring in formulas C_f, C_h accordingly. Consider the following sets: $B_f \subset X_f \setminus X^{in}$, $B_h \subset X_h \setminus X^{in}$, $B^* = B_f \cup B_h = \{x_1^*, \ldots, x_s^*\}$ (we assume that $|B^*| = s$). Obviously, the set of cubes $\Pi = \{(x_1^*)^{\alpha_1} \wedge \cdots \wedge (x_s^*)^{\alpha_s}\}_{\alpha \in \{0,1\}^{|B^*|}}$, $\alpha = (\alpha_1, \ldots, \alpha_s)$ yields a SAT partitioning of formulas $C_{f \triangle h}$ and $C_{f \triangle h} \wedge C(\mu)$.*

Our next task is to learn how to build sets B^* (we will refer to such a set as to a *decomposition set*) that would provide acceptable hardness *w.r.t.* Π for computationally hard LEC instances. Our approach to constructing sets B^* is based on the concept of a statistically balanced variable.

Below we consider the circuit S_f, implying that the results obtained are also applicable to S_h and $S_{f \triangle h}$. Keeping in mind all notions introduced above, we define a uniform distribution on $\{0, 1\}^n$, and for an arbitrary $v \in V \setminus V^{in}$ consider the events $g_v = 0$ and $g_v = 1$. Denote by $Pr\{g_v = 0\}$ and $Pr\{g_v = 1\}$ the probabilities of these events.

Definition 8 (Balanced gate). *Let $v \in V \setminus V^{in}$ be arbitrary gate. We call the gate v and the corresponding variable $var(v)$* balanced *if $Pr\{g_v = 0\} = Pr\{g_v = 1\} = 1/2$. If these probabilities deviate from 1/2 by no more than some $\varepsilon \in (0, 1)$, we call the gate v and variable $var(v)$* ε-balanced.

The balance of an arbitrary gate $v \in V \setminus V^{in}$ can be estimated efficiently using Chernoff bound (3). Indeed, let $p_v = Pr\{g_v = 1\}$, then v is associated with a Bernoulli random variable ξ_v, which takes the value 1 if g_v on the random input $\alpha \in \{0, 1\}^n$ takes the value 1. Since $E[\xi_v] = p_v$, then for any fixed $\varepsilon, \delta \in (0, 1)$ we can construct an (ε, δ)-approximation of p_v using some sample of random input words of S_f. Recall (see *e.g.* [22]) that an (ε, δ)-approximation of some parameter ν is some observable quantity $\tilde{\nu}$ such that $Pr\{|\nu - \tilde{\nu}| \leq \varepsilon\} \geq 1 - \delta$. Then it follows from (3) that for any fixed $\varepsilon, \delta \in (0, 1)$ to obtain the (ε, δ)-approximation of p_v, we only need to make $N \geq \left\lceil \frac{4 \cdot \ln(2/\delta)}{\varepsilon^2} \right\rceil$ independent observations $\xi_1, \ldots, \xi_N$ of random variable ξ_v, and compute the value $\frac{1}{N} \sum_{j=1}^{N} \xi_j$. This can be done efficiently. For example, for $\varepsilon = 0.05$ and $\delta = 0.01$, the value $\frac{1}{N} \sum_{j=1}^{N} \xi_j$ gives the required approximation for any $N \geq 8478$. As it will be shown further, one can construct a good SAT partitioning B^* using ε-balanced variables with $\varepsilon \leq 0.05$.

4 Experiments

4.1 Considered tests

In the role of f and h we considered functions defined by various algorithms that sort k arbitrary natural numbers represented by bit vectors of length l. Thus, we considered $f, h\colon \{0,1\}^n \to \{0,1\}^n$, where $n = k \cdot l$. More specifically, three sorting algorithms were used: Bubble Sorting, Selection Sorting [11] and Pancake Sorting [17]. The functions f, h corresponding to these algorithms were specified using And-Inverter Graphs. We applied ABC [8] to build a FRAIG (Functionally Reduced And-Inverter Graph) [27] for each considered circuit. The resulting circuits were used to construct the formulas $C_f, C_h, C_{f \triangle h}, C_{f \triangle h} \wedge C(\mu)$. The corresponding tests are denoted as follows: $\texttt{BvP}_{k,l}$ for Bubble vs. Pancake; $\texttt{BvS}_{k,l}$ for Bubble vs. Selection; and $\texttt{PvS}_{k,l}$ for Pancake vs. Selection. It should be noted that the resulting test classes scale very well and give complex instances even for relatively small input lengths. So, for example, the $\texttt{PvS}_{10,4}$ test instance is beyond the power of any of the conventional state-of-the-art SAT solvers. However, as we show below, they can be solved on a computing cluster in reasonable time using the partitionings described above.

4.2 Experimental setup and implementation details

In computational experiments, we used SAT solvers that ranked best in SAT competition and SAT Race of recent years: Kissat [6], CaDiCaL [6], and `MalpleLCMDistChronoBT-DL` [23]. To implement the SAT partitioning strategies described above, an MPI application was written in Python. This program was run on a computing cluster, each computing node of which is equipped with two 18-core Intel Xeon E5-2695 v4 Broadwell processors with 128 GB RAM (thus, 36 cores per one node were harnessed). When constructing the Monte Carlo estimates, random samples of 10000 were used. Up to ten computing nodes (360 cores in total) were used in the experiments.

We note here separately that apart from the solution time in seconds, we also measured the number of conflicts generated during SAT solving, since the number of conflicts can be considered as an estimate of the size of the search tree that the SAT solver explores when solving a specific instance. Indeed, the operation of the DPLL algorithm [12,13] corresponds to an ordinary binary tree, each branch of which, for an unsatisfiable test, ends in a conflict. In the case of CDCL, due to periodic restarts, instead of a tree, we are dealing with a forest. The number of paths in such a forest, in fact, can be considered as the complexity of a specific unsatisfiability proof that the SAT solver builds for the instance in question.

4.3 Main experimental results

For each series of tests `BvP`, `BvS`, and `PvS`, we generated and solved families of LEC instances of increasing complexity, corresponding to the following parameters: $k \in \{7, 8, 9, 10\}$, $l = 4$. Table 1 shows the time used to solve these instances

on one cluster one using one thread. We only included hard instances that were solved in more than three hours. The notation *>3d* means that the corresponding instance was not solved in three days and the computation was interrupted. Also, since the Maple solver, if interrupted, does not output the number of generated conflicts, the corresponding data is omitted. In the next series of experiments, SAT partitionings were built in accordance with Construction 1 and Construction 2.

Table 1. Time (in seconds) and number of conflicts used by sequential SAT solvers on considered instances

	Kissat		Cadical		Maple	
Instance	**Time**	**Confl.**	**Time**	**Confl.**	**Time**	**Confl.**
$BvP_{9,4}$	11 316	$9 \cdot 10^7$	25 710	$9 \cdot 10^7$	86 389	$18 \cdot 10^7$
$BvP_{10,4}$	154 410	$62 \cdot 10^7$	246 294	$47 \cdot 10^7$	>3d	—
$BvS_{9,4}$	3 054	$27 \cdot 10^6$	5 478	$28 \cdot 10^6$	8 564	$31 \cdot 10^6$
$BvS_{10,4}$	14 272	$97 \cdot 10^6$	36 048	$109 \cdot 10^6$	57 964	$130 \cdot 10^6$
$PvS_{9,4}$	64 437	$28 \cdot 10^7$	108 025	$26 \cdot 10^7$	>3d	—
$PvS_{10,4}$	>3d	$73 \cdot 10^7$	>3d	$40 \cdot 10^7$	>3d	—

In the case of Construction 1, we used a partitioning of the set X^{in} into disjoint pairs, triples, and quadruples of variables. Function λ_1^j used in Construction 1 was selected experimentally as follows ($\lambda_2^j = \neg\lambda_1^j$ in all cases):

- 2-XOR: for pairs, function λ_1^j, $j \in \{1, \ldots, n/2\}$ was defined by formula $\phi_1^j = (x_1^j \oplus x_2^j)$;
- 3-MAJ: for triples, function λ_1^j, $j \in \{1, \ldots, n/3\}$ was defined by formula $\phi_1^j = \texttt{majority}(x_1^j, x_2^j, x_3^j)$, where $\texttt{majority}(a, b, c) = a \wedge b \vee a \wedge c \vee b \wedge c$;
- 4-BENT: for quadruples, λ_1^j, $j \in \{1, \ldots, n/4\}$ was defined by the bent function [30] of four variables according to the formula $\phi_1^j = x_1^j \wedge x_3^j \oplus x_2^j \wedge x_4^j$.

Note that partitioning into pairs produces a large number of subproblems, which, although simple, result in much higher estimates of the total solving time than for triples and quadruples.

In Table 2, columns *Total subprobs* and *Solved subprobs* contain information about the total number of subproblems in the SAT partitioning (column *Decomposition Type*) and the number of subproblems solved in the experiment using five nodes of a computing cluster (180 cores). If the values in these columns are equal, then it means that all subproblems from the corresponding SAT partitioning have been solved. In these cases we compute the exact value of $E[\xi_\Pi]$ and the standard deviation of this value, both in seconds and in the number of conflicts. Otherwise, if the number of solved subproblems is smaller than the total number of subproblems, then we present statistical estimates of these values, calculated using the specified sample size. The column *Wall clock time* shows the

Table 2. Experimental results for solving decompositions of LEC instances $\texttt{BvP}_{9,4}$, $\texttt{BvP}_{10,4}$, $\texttt{PvS}_{9,4}$

Instance	Dec. type	Dec. size	Sample size	Solver	Avg ± sd time, s	Avg ± sd confl.	Wall time, s
$\texttt{BvP}_{9,4}$	2-XOR	262 144	10 000	Cadical	19 ± 4	$190 \cdot 10^3 \pm 34 \cdot 10^3$	—
			10 000	Kissat	21 ± 5	$259 \cdot 10^3 \pm 57 \cdot 10^3$	—
			10 000	Maple	114 ± 19	$411 \cdot 10^3 \pm 52 \cdot 10^3$	—
	3-MAJ	4 096	4 096	Cadical	355 ± 109	$2\,218 \cdot 10^3 \pm 542 \cdot 10^3$	8 087
			4 096	Kissat	276 ± 76	$2\,808 \cdot 10^3 \pm 709 \cdot 10^3$	6 286
			4 096	Maple	797 ± 216	$2\,592 \cdot 10^3 \pm 635 \cdot 10^3$	18 132
	4-BENT	512	512	Cadical	$2\,214 \pm 1\,149$	$10 \cdot 10^6 \pm 5 \cdot 10^6$	6 299
			512	Kissat	$1\,168 \pm 447$	$12 \cdot 10^6 \pm 5 \cdot 10^6$	3 323
			512	Maple	$4\,273 \pm 1\,923$	$11 \cdot 10^6 \pm 5 \cdot 10^6$	12 153
	4+4	256	256	Cadical	$1\,358 \pm 540$	$9 \cdot 10^6 \pm 3 \cdot 10^6$	1 931
			256	Kissat	884 ± 323	$11 \cdot 10^6 \pm 4 \cdot 10^6$	1 258
			256	Maple	$2\,286 \pm 836$	$9 \cdot 10^6 \pm 3 \cdot 10^6$	3 252
$\texttt{BvP}_{10,4}$	3-MAJ	16 384	10 000	Cadical	$1\,752 \pm 886$	$7 \cdot 10^6 \pm 3 \cdot 10^6$	—
			10 000	Kissat	$1\,072 \pm 476$	$9 \cdot 10^6 \pm 4 \cdot 10^6$	—
	4-BENT	1 024	1 024	Cadical	$22\,397 \pm 15\,010$	$63 \cdot 10^6 \pm 30 \cdot 10^6$	127 415
			1 024	Kissat	$10\,472 \pm 5\,667$	$77 \cdot 10^6 \pm 35 \cdot 10^6$	59 571
	4+4	256	256	Cadical	$45\,494 \pm 12\,845$	$102 \cdot 10^6 \pm 23 \cdot 10^6$	64 703
			256	Kissat	$18\,155 \pm 6\,451$	$124 \cdot 10^6 \pm 37 \cdot 10^6$	25 821
$\texttt{PvS}_{9,4}$	3-MAJ	4 096	4 096	Cadical	941 ± 317	$4 \cdot 10^6 \pm 1 \cdot 10^6$	21 422
			4 096	Kissat	766 ± 229	$6 \cdot 10^6 \pm 2 \cdot 10^6$	17 443
	4-BENT	512	512	Cadical	$6\,491 \pm 3\,512$	$20 \cdot 10^6 \pm 10 \cdot 10^6$	18 462
			512	Kissat	$3\,831 \pm 1\,641$	$27 \cdot 10^6 \pm 10 \cdot 10^6$	10 898
	6+6	4 096	4 096	Cadical	421 ± 492	$2 \cdot 10^6 \pm 2 \cdot 10^6$	9 588
			4096	Kissat	390 ± 465	$4 \cdot 10^6 \pm 4 \cdot 10^6$	8 869

time used to solve the corresponding partitioning: it corresponds to the time the user would need to wait in order to solve the LEC instance using the said partitioning. If the number of solved subproblems is smaller than the total number of subproblems, this value is omitted.

In the experiments for Construction 2, we used cubes built from variables corresponding to balanced gates (we refer to such variables and cubes as to *balanced* ones). More precisely, for each circuit S_f and S_h, the balance of each gate was calculated in the manner described above (using Chernoff bound). Then, from each circuit we chose q gates with the balance closest to 1/2, and built the decomposition set $B = \{\tilde{x}_1, \ldots, \tilde{x}_{2q}\}$ from the obtained variables. The considered SAT partitioning (denoted as q+q) is represented by all possible cubes $\tilde{x}_1^{\alpha_1} \wedge \cdots \wedge \tilde{x}_{2q}^{\alpha_{2q}}$. The experiments were carried out for $q \in \{4, 5, 6\}$.

In the context of all that has been said above, one of the main issues is the accuracy of the resulting estimates of $E[\xi_\Pi]$. The main factor that negatively affects the accuracy is the magnitude of $Var(\xi_\Pi)$. The data in Table 2 implies that the two proposed SAT partitioning constructions give a relatively small standard deviation and, as a result, the resulting estimates are very accurate.

Moreover, as shown below, in order to obtain relatively accurate estimates of $E[\xi_\Pi]$, it is sufficient to use samples whose size is significantly smaller than the total size of the considered SAT partitioning. The aforesaid is confirmed by the experimental data shown in Fig. 3 and Fig. 4, which demonstrate the dependence

of the accuracy of the estimate of $E[\xi_\Pi]$ on the size of the random sample. In Fig. 3 we present the plot for partitioning of the set X^{in} into triples (3-MAJ) for Construction 1, and in Fig. 4 into balanced cubes (4+4) for Construction 2. In both cases we used the LEC problem instance $\texttt{BvP}_{9,4}$ and the solver CaDiCaL.

For each value of the size of random sample N, we generated $P = 1000$ random samples of size N, and calculated the sample means $(\hat{\xi}^1, \ldots, \hat{\xi}^P)$, where each $\hat{\xi} = \frac{1}{N}\sum_{j=1}^{N} \xi_j$. Additionally, we calculated the mean of sample means $\Xi(N) = \frac{1}{P}\sum_{r=1}^{P} \hat{\xi}^r$, and also chose the minimal $M_*(N)$ and maximal $M^*(N)$ values. Next, we normalized all values by dividing them by $E[\xi_\Pi]$.

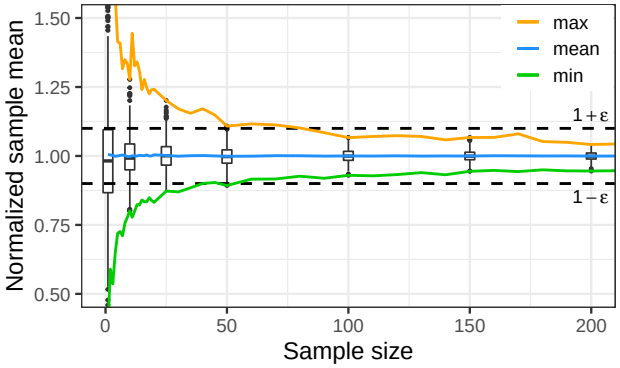

Fig. 3. Distributions of sample means for different sample sizes N on the 3-MAJ decomposition of $\texttt{BvP}_{9,4}$ instance

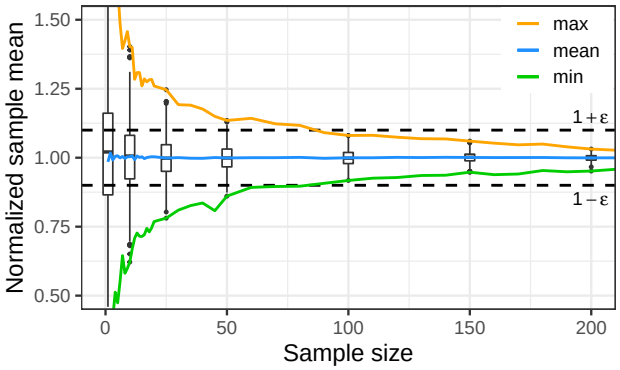

Fig. 4. Distributions of sample means for different sample sizes N on the decomposition into balanced cubes (4+4) of $\texttt{BvP}_{9,4}$ instance

Table 3. Experimental results for solving decompositions on $\texttt{PvS}_{10,4}$ instance

Dec. type	Dec. size	Sample size	Solver	Avg. ± sd time, s	Avg. ± sd conflicts	Wall clock time, s
2-XOR	1 048 576	10 000	Cadical	167 ± 57	$1011 \cdot 10^3 \pm 240 \cdot 10^3$	—
		10 000	Kissat	185 ± 64	$1520 \cdot 10^3 \pm 393 \cdot 10^3$	—
3-MAJ	16 384	10 000	Cadical	$5 \cdot 10^3 \pm 3 \cdot 10^3$	$15 \cdot 10^6 \pm 7 \cdot 10^6$	—
4-BENT	1 024	1 024	Cadical	$83 \cdot 10^3 \pm 48 \cdot 10^3$	$111 \cdot 10^6 \pm 47 \cdot 10^6$	474 922
		1 024	Kissat	$30 \cdot 10^3 \pm 13 \cdot 10^3$	$129 \cdot 10^6 \pm 43 \cdot 10^6$	171 182
6+6	4 096	4 096	Kissat	$3 \cdot 10^3 \pm 19 \cdot 10^3$	$10 \cdot 10^6 \pm 56 \cdot 10^6$	71 168
4+4	256	256	Kissat	$26 \cdot 10^3 \pm 74 \cdot 10^3$	$68 \cdot 10^6 \pm 59 \cdot 10^6$	37 606

In Fig. 3 and Fig. 4 the horizontal axis shows the varying size of random sample N. For some values of N, the corresponding distributions of sample means are shown using boxplots. Additionally, the plots contain the following normalized lines:

- $\Xi(N)/E[\xi_\Pi]$ (blue line, middle);
- $M_*(N)/E[\xi_\Pi]$ (green line, bottom);
- $M^*(N)/E[\xi_\Pi]$ (orange line, top);
- $(1 \pm \varepsilon)$ for $\varepsilon = 0.1$ (black dashed lines).

From the plots in Fig. 3–Fig. 4 it can be seen that on the considered class of tests, the calculated sample mean $\hat{\xi}$ gives a fairly accurate estimate of $E[\xi_\Pi]$ even when the sample size N is significantly smaller than the total size of the considered partitioning.

As mentioned above, problems from the considered class with input length $n = k \cdot l = 40$ are already extremely complex. However, problems $\texttt{BvP}_{10,4}$ and $\texttt{BvS}_{10,4}$ were solved using five nodes (180 cores) of the computing cluster in reasonable time (as can be seen in Table 2). Since the obtained estimates of hardness for $\texttt{PvS}_{10,4}$ were significantly higher than for $\texttt{BvP}_{10,4}$ and $\texttt{BvS}_{10,4}$, we used ten cluster nodes (360 cores), CaDiCaL and Kissat solvers to solve them (Maple showed significantly worse results in previous experiments). Results are shown in Table 3.

4.4 Experiments with unbalanced cubes

We emphasize that in Construction 2 we use cubes built from the most balanced variables, hoping that the corresponding SAT partitioning will have a small variance. And this hypothesis, as follows from Table 2, is generally confirmed. Of interest is the question of what will happen if we build cubes using the most *unbalanced* variables instead of balanced ones (*i.e.* unbalanced cubes)? On the one hand, $Var(\xi_\Pi)$ should be significantly higher, but, on the other hand, many subproblems in the SAT partitioning can be extremely simple.

We have carried out the corresponding experiments. It turned out that when using unbalanced cubes, in many cases even the CNF formulas $\tilde{x}_1^{\alpha_1} \wedge \cdots \wedge \tilde{x}_{2q}^{\alpha_{2q}} \wedge C_{f \triangle h}$ are unsatisfiable, *i.e.* formulas that do not even include the term $C(\mu)$

which encodes the miter. And the corresponding instances are easy for the SAT solver. However, the final SAT partitioning will necessarily contain abnormally hard formulas, the hardness of which is comparable with the hardness of SAT for $C_{f\triangle h} \wedge C(\mu)$ (*i.e.* for the case without using partitioning).

Let us denote as G^* such an abnormally hard cube. Using Chernoff bound, we estimated the number of satisfying assignments of CNF formula $G^* \wedge C_{f\triangle h}$. We conducted several such experiments with hard cubes, and the typical case is: for a hard cube G^* of realistic size (say, ≤ 40) the estimation of the number of satisfying assignments of formula $G^* \wedge C_{f\triangle h}$ was greater than $0.9 \cdot 2^n$ (with tolerance $\varepsilon = 0.01$ and confidence level $1 - \delta = 0.99$, *w.r.t.* Chernoff bound, $N = 211933$ was used). Thus, these results confirm the assumption put forward above about a direct relationship between the number of satisfying assignments of formula $G \wedge C_{f\triangle h}$ and the hardness of formula $G \wedge C_{f\triangle h} \wedge C(\mu)$.

5 Conclusion

In this paper, we explored how to estimate the hardness of SAT encodings for the Logical Equivalence Checking problem. One of our basic observations in this context is that we can estimate the hardness of a SAT encoding of LEC using some SAT partitioning. More specifically, we introduce the concept of hardness of a SAT instance *w.r.t.* a SAT partitioning and a SAT solver $\mathcal{O}$. We show that such estimates can be constructed using probabilistic algorithms based on the Monte Carlo method. The accuracy of this kind of estimates depends on the probabilistic characteristics of a specially defined random variable which is associated with a particular SAT partitioning. We propose two constructions of SAT partitionings, in relation to which we present arguments for the good accuracy of the obtained estimates of hardness. To carry out computational experiments, we use a class of LEC instances, where circuits are represented as And-Inverter Graphs which define various algorithms for sorting k natural numbers with bit length l. The hardness of such tests scales well due to the selection of values k, l, and one can generate extremely hard LEC instances already for circuits with $n = k \cdot l = 40$ inputs. In general, it is not possible to accurately predict the running time of a consecutive SAT solver on some of these tests. However, we estimate the hardness of such tests *w.r.t.* the proposed SAT partitioning. The estimates obtained indicate that the corresponding LEC instances can be solved in parallel using a reasonable amount of computational resources. We confirm these conclusions and the accuracy of the estimates obtained by solving the corresponding instances on a computing cluster. We also formulate a hypothesis about a direct relationship between the hardness of subproblems in the SAT partitioning and the number of satisfying assignments of special satisfiable CNF formulas associated with the original Boolean circuits, and we demonstrate that this hypothesis is true for circuits considered in our experiments.

Future work may include the development of optimization algorithms for finding cubes over auxiliary variables with good statistical estimation of hardness of LEC *w.r.t.* the corresponding SAT partitioning.

References

1. Arora, S., Barak, B.: Computational Complexity: A Modern Approach. Cambridge University Press (2009)
2. Balyo, T., Sanders, P., Sinz, C.: HordeSat: A massively parallel portfolio SAT solver. In: SAT. pp. 156–172 (2015)
3. Bessière, C., Katsirelos, G., Narodytska, N., Walsh, T.: Circuit Complexity and Decompositions of Global Constraints. In: IJCAI. pp. 412–418 (2009)
4. Biere, A., Cimatti, A., Clarke, E.M., Fujita, M., Zhu, Y.: Symbolic model checking using SAT procedures instead of BDDs. In: DAC. pp. 317–320 (1999)
5. Biere, A., Cimatti, A., Clarke, E., Zhu, Y.: Symbolic Model Checking without BDDs. In: TACAS. pp. 193–207 (1999)
6. Biere, A., Fazekas, K., Fleury, M., Heisinger, M.: CaDiCaL, Kissat, Paracooba, Plingeling and Treengeling entering the SAT Competition 2020. In: Proc. of SAT Competition 2020 – Solver and Benchmark Descriptions. Department of Computer Science Report Series B, vol. B-2020-1, pp. 51–53. University of Helsinki (2020)
7. Biere, A., Heule, M., van Maaren, H., Walsh, T. (eds.): Handbook of Satisfiability, Frontiers in Artificial Intelligence and Applications, vol. 185. IOS Press (2009)
8. Brayton, R., Mishchenko, A.: ABC: An academic industrial-strength verification tool. In: Computer Aided Verification. pp. 24–40. Springer Berlin Heidelberg, Berlin, Heidelberg (2010)
9. Bryant, R.E.: Graph-Based Algorithms for Boolean Function Manipulation C-35(8), 677–691 (1986)
10. Chang, C.L., Lee, R.C.T.: Symbolic Logic and Mechanical Theorem Proving. Computer Science Classics, Academic Press (1973)
11. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms. MIT Press, 3 edn. (2009)
12. Davis, M., Logemann, G., Loveland, D.: A machine program for theorem-proving. Commun. ACM 5(7), 394–397 (1962)
13. Davis, M., Putnam, H.: A computing procedure for quantification theory. J. ACM 7(3), 201–215 (1960)
14. Dowling, W.F., Gallier, J.H.: Linear-time algorithms for testing the satisfiability of propositional horn formulae 1(3), 267–284 (1984)
15. Drechsler, R., Junttila, T.A., Niemelä, I.: Non-Clausal SAT and ATPG. In: Handbook of Satisfiability (2009)
16. Feller, W.: An Introduction to probability theory and its applications, vol. 2. John Wiley & Sons, Inc., 2 edn. (1971)
17. Gates, W.H., Papadimitriou, C.H.: Bounds for sorting by prefix reversal 27(1), 47–57 (1979)
18. Goldreich, O.: Computational Complexity: A Conceptual Perspective. Cambridge University Press (2008)
19. Gomes, C., Sabharwal, A.: Exploiting runtime variation in complete solvers. In: Handbook of satisfiability. Frontiers in Artificial Intelligence and Applications, vol. 185, pp. 271–288 (2009)
20. Heule, M.J.H., Kullmann, O., Wieringa, S., Biere, A.: Cube and conquer: Guiding cdcl sat solvers by lookaheads. In: HVC. pp. 50–65 (2012)
21. Hyvärinen, A.E.J.: Grid Based Propositional Satisfiability Solving (2011), PhD thesis. Aalto University publication series
22. Karp, R.M., Luby, M., Madras, N.: Monte-carlo approximation algorithms for enumeration problems. Journal of Algorithms 10(3), 429–448 (1989)

23. Kochemazov, S., Zaikin, O., Kondratiev, V., Semenov, A.: MapleLCMDistChronoBT-DL, duplicate learnts heuristic-aided solvers at the SAT Race 2019. In: Proceedings of SAT Race 2019. vol. B-2019-1, p. 24 (2019)
24. Kuehlmann, A., Krohm, F.: Equivalence checking using cuts and heaps. In: DAC. pp. 263–268 (1997)
25. Marques-Silva, J., Lynce, I., Malik, S.: Conflict-driven clause learning SAT solvers. In: Handbook of Satisfiability, Frontiers in Artificial Intelligence and Applications, vol. 185, pp. 131–153. IOS Press (2009)
26. Metropolis, N., Ulam, S.: The Monte Carlo Method. J. Amer. Statistical Assoc. 44(247), 335–341 (1949)
27. Mishchenko, A., Chatterjee, S., Brayton, R.: Fraigs: A unifying representation for logic synthesis and verification. Tech. rep., Department of EECS, University of California, Berkeley (2005)
28. Molitor, P., Mohnke, J.: Equivalence Checking of Digital Circuits: Fundamentals, Principles, Methods. Kluwer Academic Publishers (2004)
29. Motwani, R., Raghavan, P.: Randomized Algorithms. Cambridge University Press (1995)
30. Tokareva, N.: Bent Functions: Results and Applications to Cryptography. Elsevier
31. Tseyitin, G.S.: On the complexity of derivation in propositional calculus. Studies in Constructive Mathematics and Mathematical Logic, Part II pp. 115–125 (1970)