Speeding up the Solving of Logical Equivalence Checking Problems with Disjunctive Diagrams

Victor Kondratiev[1], Ilya Otpuschennikov[2], Alexander Semenov[1]
[1] ITMO University, St. Petersburg, Russia
[2] Matrosov Institute for System Dynamics and Control Theory SB RAS, Irkutsk, Russia
Emails: vikseko@gmail.com, otilya@yandex.ru, biclop.rambler@yandex.ru

***Abstract*—In the context of the problem of checking the equivalence of Boolean circuits (LEC), we propose an approach that increases the efficiency of modern SAT solvers on this problem by generating additional constraints of a special kind. To generate such constraints, we use a variant of decision diagrams called disjunctive diagrams. In contrast to well-known Binary Decision Diagrams these diagrams can be constructed effectively for an original formula and can also be used to extend the original CNF formula with new clauses which are its logical consequences. In computational experiments, we show that the resulting formulas, encoding difficult LEC variants, extended by the generated constraints are often significantly easier to solve for state-of-the-art SAT solvers compared to the original formulas.**

Keywords*—*Boolean satisfiability problem; Logical Equivalence Checking; Disjunctive Diagrams; Multipliers

I. Introduction

Boolean circuits are one of the basic objects of modern computer science in both theoretical [1], [2] and practical [3], [4] senses. Verification of Boolean circuits is considered to be a very important direction in such industrial area as Electronic Design Automation (EDA). In this paper we consider a well-known problem from EDA referred as Logical Equivalence Checking (LEC). One of the main computational tools for solving LEC are SAT solvers. Unfortunately, there are many examples of well-known functions that give LEC problems in SAT form which are very hard for all state-of-the-art SAT solving algorithms. In such situations the ways which provide opportunity to improve the efficiency of SAT solvers on such extremely hard instances are required. For instance, different preprocessing techniques can be applied for this purpose.

In this paper we propose a preprocessing technique based on the notion of Disjunctive Diagrams [5]. The main goal of our paper is to develop a corresponding technique and to argue its efficiency in application to some extremely hard LEC problems in form of SAT instances. We achieve this purpose, describing corresponding algorithms, and using equivalence checking problems for several different algorithms performing multiplication of integer numbers in the role of benchmarks.

Let us give a brief overview of the paper. The next section contains the basic definitions and results necessary for understanding the rest of the paper. Section III presents the description of main novel developed algorithms. In particular, we mean a basic preprocessing algorithm which constructs new additional constraints using which we decrease SAT solver runtime on considered benchmarks.

Also we discuss here, how we can use a special SAT partitioning based technique to estimate the hardness of extremely hard LEC instances in SAT form. In the experimental part we present the results of computational experiments and we can see, that the proposed method is able to cope with extremely hard LEC in SAT form outperforming the standard approach by $5-14\%$. It should be noted that these results are successful, taking into account the well-known extreme hardness of formulas encoding LEC for multipliers.

II. Preliminaries

By $\{0,1\}^K$ we denote the set of all binary words (also called Boolean vectors) of length K. Let F be an arbitrary Boolean formula [6] over set of variables X, $|X|=K$. The formula specifies a Boolean function $\phi_F : \{0,1\}^K \to \{0,1\}$, which is defined on whole set $\{0,1\}^K$. The function ϕ_F is defined by substituting values of Boolean variables from X into F in the standard sense [7]. Vectors from $\{0,1\}^K$, which are sets of values of variables from X, are called assignments of these variables. An arbitrary assignment $\alpha \in \{0,1\}^K$ such that the following holds: $\phi_F(\alpha)=1$, is called a satisfying assignment for formula F. If F has a satisfying assignment then F is called satisfiable, and the formula is unsatisfiable if does not exist any satisfying assignment for F. The Boolean Satisfiability Problem (SAT) requires for an arbitrary Boolean formula F to determine if this formula is satisfiable. This problem is NP complete and it is NP-hard if, besides checking satisfiability, we would like to find some satisfying assignment in case when the formula is satisfiable.

It is well-known that the problem of checking the satisfiability of an arbitrary Boolean formula F can be reduced in polynomial time to the problem of checking the satisfiability of some Conjunctive Normal Form (CNF) C, using Tseitin transformations [8]. Hereinafter, when mentioning SAT we will assume the problem of checking the satisfiability of an arbitrary formula in a CNF.

A huge amount of combinatorial problems from a variety of industrial areas: program analysis, computer security, cryptanalysis, bioinformatics, combinatorics, etc. are effectively reduced to SAT. However, one of the main applications of modern SAT solvers for more than 20 years continues to be Electronic Design Automation (EDA). Within EDA, SAT solvers are used to solve the Logical Equivalence Checking (LEC) problem and directly related Automated Test Pattern Generation (ATPG) problem. The LEC problem is usually considered for two Boolean circuits.

Let us remind [9], that Boolean circuit S_f is mathematically interpreted by some directed graph (V, A), V is set of vertices and A is set of arcs. V contains n vertices without parents, these vertices are called inputs of S_f and we denote their set by V^{in}, also V contains m vertices without children, these vertices form set V^{out} and are called outputs of S_f. Each vertex in $V \setminus V^{in}$ is assigned some Boolean connective from some basis. The vertices from $V \setminus V^{in}$ are called gates. Let us further consider the complete basis $\{\neg, \wedge\}$ everywhere and call the graph representing the corresponding circuit the And-Inverter Graph (AIG) [10].

On a circuit S_f we can define a procedure for its interpretation on an arbitrary input vector $\alpha \in \{0,1\}^n$, which consists of successive computation of values of Boolean functions associated with all gates, which are topologically sorted w.r.t. structure of corresponding graph. The result of interpreting S_f on α is a vector $\gamma \in \{0,1\}^m$ and thus S_f specifies a (total) discrete function of kind $f : \{0,1\}^n \to \{0,1\}^m$.

Let us return to LEC problem. This problem is formulated in the following manner: given two circuits S_f and S_h, both with n inputs and m outputs. Answer the question: is it true that S_f and S_h actually specify the same function, i.e. the following point-wise equality holds: $f \equiv h$? In case of answer 'yes' for this question, the circuits S_f and S_h are called equivalent ($S_f \equiv S_h$).

It is well-known fact that LEC problem is efficiently (in polynomial time of the length of descriptions of circuits S_f and S_h) reduced to SAT [4], [11]. To do this, a new circuit is constructed from circuits S_f, S_h: for each pair of identical inputs (with the same natural number) these inputs are glued together, and each pair of identical outputs is combined by additional XOR gate, after which the outputs from all m such XOR gates are fed to the gate linking them by disjunction. The block consisting of XOR gates and the resulting disjunction is called a miter [4], see Figure 1. The constructed circuit, denoted by $S_{M(f,h)}$, defines a Boolean function $\Phi_{M(f,h)} : \{0,1\}^n \to \{0,1\}$. The CNF $C_{M(f,h)}$ is constructed for $S_{M(f,h)}$ using Tseitin transformation. It can be shown that the CNF $C_{M(f,h)}$ is unsatisfiable if and only if the original circuits are equivalent.

Unfortunately, CNF $C_{M(f,h)}$ can turn out to be extremely hard for all state-of-the art SAT solvers. Such situations are typical for a number of circuits defining well-known arithmetic functions, e.g., multiplication functions for natural numbers. Such circuits are called multipliers. It is well-known fact that the LEC problem in SAT form for two multipliers is extremely hard for the best SAT which won in SAT solvers competitions conducted annually. As a consequence, in some recent works instead LEC the problems of identification of multipliers are considered: in the framework of this approach the problem to recognize if the considered circuit implements multiplier is studied. Such an approach gives the possibility to work with

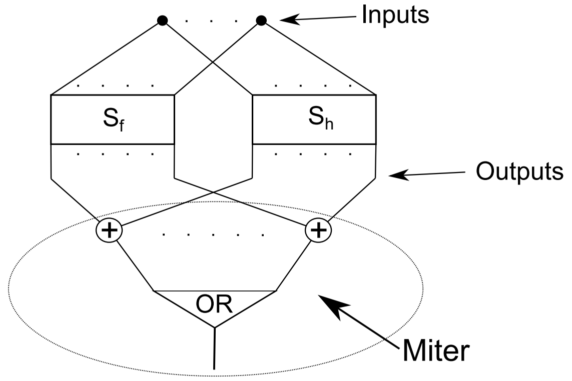

Fig. 1. LEC constructed by two circuits S_f and S_h with miter.

multipliers of significantly higher dimension comparing with LEC approach [12], [13], [14], [15]. In our study we consider only LEC approach to verification of multipliers.

Often when dealing with such hard problems it is not even possible to predict the SAT solver running time a priori, and this is a serious problem of its own. To solve this problem, in [16] was proposed an approach based on decomposition hardness notion. In [17] ideas from [16] were transferred to hardness estimation of hard LEC in form of SAT instances, exploiting information about structure of considered circuits. This technique from [17] will be used further to estimate hardness of LEC in SAT form for two multipliers when we augmented our CNF encoding of considered LEC with some additional information, obtained using preprocessing techniques, based on decision diagrams of special kind.

Specifically, we will use further so-called Disjunctive Diagrams, introduced in [5]. Let us briefly recap their main point. A DisJunctive Diagram (DJD) is constructed by an arbitrary formula in disjunctive normal form (DNF) and the diagram is essentially Shannon bracket representation of considered DNF in form of directed graph. Using arbitrary CNF C, we construct DNF $D = \neg C$, and then using D we construct DJD $R(D)$ representing it. It is important that $R(D)$ is constructed in polynomial time of length D. An arbitrary diagram $R(D)$ has two terminal vertices: vertex '1' and vertex '?'. The diagram $R(D)$ in contrast to the well-known ROBDD [18] and ZBDD [19] is not binary and can have more than one root.

Each path in $R(D)$ from some root to terminal vertex '1' corresponds to a particular conjunction of literals in D. An arbitrary path π from some root to terminal '?' defines an assignment α_π over some set of variables $X_\pi \subseteq X$ (hereinafter, X the set of variables over which the original CNF formula C is specified). Next, we show how the path properties in $R(D)$ in terminal vertex '?' can be used to generate additional constraints using which we can improve the performance of SAT solver in application to extremely hard LEC instances.

Figure 2 shows DNF and the corresponding disjunctive forest (left) and the disjunctive diagram (right) constructed

$$D = (x_1 \wedge \neg x_2 \wedge \neg x_3) \vee (x_1 \wedge \neg x_3 \wedge \neg x_4) \vee (\neg x_2 \wedge \neg x_3) \vee (x_2 \wedge x_4) \vee (\neg x_3 \wedge x_4) \vee x_4$$

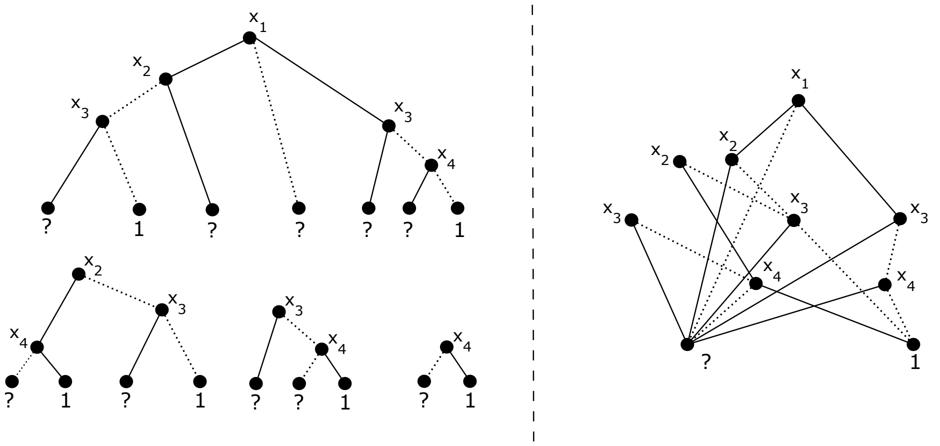

Fig. 2. DNF D with Disjunctive forest $F(D)$ (left) and DJD $R(D)$ (right) for it.

with the order $x_1 < x_2 < x_3 < x_4$.

III. Hardness estimations of Extremely hard LEC instance and disjunctive diagrams based preprocessing

So, let C be a CNF formula encoding some extremely hard LEC instance. As noted above, on such formula a modern SAT solver A can work indefinitely long and, unfortunately, this time cannot be predicted with the necessary accuracy in the general case. However, it is possible to construct an estimate of the hardness of formula C using the ideas from the article [16]. Let us recall that in [16] it was proposed to estimate the hardness of C w.r.t. A and some plain partitioning [20] of formula C, and this approach is close in sense of idea to one from [21] to use the notion of backdoor [22] to evaluate the hardness of arbitrary Boolean formula. Corresponding hardness measure is referred as decomposition hardness (d-hardness) or hardness w.r.t. SAT partitioning. In [17] we described one particular construction of SAT partitioning that can be used to evaluate the hardness of CNF formulas encoding namely some LEC instances. Let us give its brief description.

So, consider LEC for two Boolean circuits S_f and S_h, as well as miter-circuit $S_{M(f,h)}$ and construct for this circuit CNF formula $C_{M(f,h)}$ over set of variables X. Denote by X^{in}, $X^{in} \subseteq X$ set of variables associated with inputs of S_f,S_h and $S_{M(f,h)}$ when transition from $S_{M(f,h)}$ into $C_{M(f,h)}$ happens. For some natural $k : 2 \leq k < n$ let us split X^{in} into $q = \lceil \frac{n}{k} \rceil$ disjoint subsets, where:

$$n = k\lfloor \frac{n}{k} \rfloor + r, r \in \{0, 1, ..., k-1\}$$

Let us denote the corresponding subsets X^j, $j \in \{1, \ldots, q\}$, and suppose that for each $j \in \{1, \ldots, \lfloor \frac{n}{q} \rfloor\}$ we have $|X^j| = k$, and in case $r \neq 0$ the following condition holds: $|X^{\lceil \frac{n}{k} \rceil}| = r$. We associate two Boolean functions with each X^j: some function $\lambda^j : \{0,1\}^{|X^j|} \to \{0,1\}$ and its negation $\neg\lambda^j$. Let ${\phi_1}^j$ and ${\phi_2}^j$ be Boolean formulas in CNF specified functions λ^j and $\neg\lambda^j$ respectively, and ϕ^j be a formula which can be in different moments both formulas ${\phi_1}^j$ and ${\phi_2}^j$. It is obvious, that there exist $2^{\lceil \frac{n}{k} \rceil}$ different formulas of kind:

$$\phi^1 \wedge ... \wedge \phi^{\lceil \frac{n}{k} \rceil}$$

The main fact established in [17] is that this family consisting of $2^{\lceil \frac{n}{k} \rceil}$ formulas is SAT partitioning of formula $C_{M(f,h)}$. Experiments show that picking up functions of kind λ^j and $\neg\lambda^j$ carefully, we can have the situation when formulas of kind:

$$\phi^1 \wedge ... \wedge \phi^{\lceil \frac{n}{k} \rceil} \wedge C_{M(f,h)} \qquad (1)$$

are significantly simpler for SAT solver A comparing with original formula $C_{M(f,h)}$.Then hardness of $C_{M(f,h)}$ w.r.t. A and SAT partitioning

$$P = \left\{ (\phi^1 \wedge ... \wedge \phi^{\lceil \frac{n}{k} \rceil})_i \right\}_{i=1}^{2^{\lceil \frac{n}{k} \rceil}}$$

can be specified (by analogy with [16]) as summary runtime of algorithm A for all formulas of kind (1). Let us denote this measure $\mu_{A,P}(C_{M(f,h)})$. Arguing by analogy with [16], it is easy to show that following equation holds:

$$\mu_{A,P}(C_M(f,h)) = 2^{\lceil \frac{n}{k} \rceil} \cdot E[\xi_P]$$

where ξ_P is random variable associated with P in manner similar to one from [17]. But then we can estimate the value $\mu_{A,P}(C_{M(f,h)})$ estimating expectation $E[\xi_P]$ using the Monte Carlo method [16], [17].

The described method of estimating the hardness of formulas turns out to be very useful for understanding the effectiveness of various techniques that speed up SAT solvers in application to extremely hard SAT instances (when we cannot predict the runtime of conventional SAT solver A). We use the method of hardness estimation just described to investigate how efficient the preprocessing method can be for hard formulas of the kind $C_{M(f,h)}$, when one harnesses disjunctive diagrams to this preprocessing. Let us describe corresponding method.

Let us return to diagram $R(D)$ constructed for DNF formula $D = \neg C$, where C is some CNF formula over variables X, and consider an arbitrary path π which is passed from some root vertex to terminal vertex '?'. Use the notations introduces above: X_π and α_π. Denote Ω_π the set of all assignments of variables from X which coincide with α_π in values of variables from X_π, thus,

$$|\Omega_\pi| = 2^{|X|-|X_\pi|}$$

Let $\Pi_?$ be the set of all paths in terminal vertex '?' in $R(D)$. From the basic properties of disjunctive diagrams it follows that the value $|\Pi_?|$ is bounded above by some polynomial of size of formula C in general case.

Leading by [5], it is not hard to see that formula C is unsatisfiable if and only if formula of kind $C(\alpha_\pi)$ is unsatisfiable for any $\pi \in \Pi_?$, where by $C(\alpha_\pi)$ is denoted formula which is result when substituting assignment α_π of variables from X_π into C. Starting from this point, let us establish the following fact.

Statement 1. *Consider an arbitrary $\pi \in \Pi_?$. If formula $C(\alpha_\pi)$ is unsatisfiable, then clause obtained by negation of α_π is logical entailment of formula C.*

Proof sketch. Let $C(\alpha_\pi)$ be unsatisfiable and $\Delta(\alpha_\pi)$ be the clause obtained by negation of α_π: e.g. let α_π be $x_{10} = 1, x_{18} = 0, x_{27} = 1$, then we represent α_π as the following conjunction of literals: $x_{10} \wedge \neg x_{18} \wedge x_{27}$. Accordingly, $\Delta(\alpha_\pi) = (\neg x_{10} \vee x_{18} \vee \neg x_{27})$. If α is some satisfying assignment for C, then α can not coincide with α_π in corresponding literals (otherwise we have contradiction with the fact that α satisfies C). This fact means that $C \to \Delta(\alpha_\pi)$. $\square$

The established fact, despite of its simplicity, gives us a simple way to construct additional constraints which can be used to augment the original formula C, hoping to reduce the search space for algorithm A. We can consider formulas of kind $C(\alpha_\pi)$ for all $\pi \in \Pi_?$ (keeping in mind that fact, that $|\Pi_?|$ is bounded by polynomial of formula C description size). To each formula $C(\alpha_\pi)$ we will apply SAT solver A. Corresponding SAT problems can be extremely hard, and consequently we can launch A with some fixed time limit t. And from all said above it follows that clauses of kind $\Delta(\alpha_\pi)$, if it managed to construct them in time $\leq t$, can be very useful due to prohibition by them some prospective search directions for algorithm A.

IV. Computational experiments

This section presents the results of computational experiments on the application of DJD-preprocessing to hard LEC problems.

We conducted computational experiments on the "Academician V.M. Matrosov" cluster of Irkutsk Supercomputer Center [23].

TABLE I
COMPARISON OF DECOMPOSITION HARDNESS ESTIMATES OF INSTANCE CvK WITH DJD-PREPROCESSING BY DIFFERENT ORDERS

Order	Average time to solve subproblems (s.)	Estimate of d-hardness after preprocessing (s.)
Direct	30.29	1 985 085
Frequency	29.68	1 945 108
Random 1	30.06	1 970 013
Random 2	29.78	1 951 663
Random 3	29.95	1 962 804
Random 4	29.75	1 949 696
Random 5	30.02	1 967 391
Random 6	**29.53**	**1 935 279**
Random 7	29.93	1 961 493
Random 8	30.03	1 968 047
Random 9	29.96	1 963 459
Random 10	29.87	1 957 561

As benchmarks, as mentioned above, we considered problems encoding equivalence of different algorithms for multiplication of pairs of integers. Concretely, the following algorithms were considered: standard "Column" multiplier, "Wallace tree" [24], Dadda algorithm [25], as well as "Karatsuba decomposition" [26]. In all cases, we considered versions of algorithms that multiply two 16-bit numbers. Further we will refer to corresponding benchmarks as to `CvK`, `CvW`, `KvW`, `DvC`, `DvK` and `DvW`, by first letters of algorithms names whose equivalence is tested in each particular problem. All these algorithms were translated to And-Inverter Graphs, using Transalg program tool [27], [28]. Corresponding AIGs and supporting scripts are available in github repository[1].

The main results of the experiments demonstrate that the generating additional constraints using disjunctive diagrams can reduce the estimated time for solving extremely hard LEC instances in SAT form quite significantly. All estimates were constructed in the way described in Section III, and the following parameters of this method were used:

$$q = 2, X = \{x_1, ..., x_{32}\}, X^j = \{x_{2j-1}, x_{2j}\}, j = 1, ..., 16$$

The functions $\lambda^j, \neg\lambda^j$, where

$$\lambda^j = x_{2j-1} \oplus x_{2j}, j = 1, ..., 16$$

were used. Thus, each SAT partitioning consisted of $2^{32/2} = 2^{16}$ subproblems. The SAT solver Kissat [29] was used as Algorithm A when we constructed the estimates.

Let us remind [5] that when constructing some disjunctive diagram one can use different orders of variables. We generated additional constraints of kind $\Delta(\alpha_\pi)$ using different such orders. Specifically, the following orders were used:

- Frequency order — order according to the frequency of occurrence of variables in a formula;

[1]https://github.com/Vikseko/LEC_benchmarks

TABLE II
COMPARISON OF DECOMPOSITION HARDNESS BEFORE AND AFTER DJD-PREPROCESSING

Instance	d-hardness before prep. (s.)	d-hardness after prep. (s.)	% of speedup
CvK	2 178 663	1 929 530	11.4
CvW	1 418 200	1 298 746	8.4
DvC	1 302 856	1 118 082	14.1
DvK	2 301 015	2 165 651	5.8
DvW	1 595 147	—	0
KvW	2 344 223	2 215 808	5.4

- Direct order — the order that coincides with the numbers of variables in the formula and, respectively, gates in the circuit;
- Random order — random order of variables.

All work with disjunctive diagrams was performed using the program PyDJD [30], which uses the software complex PySAT [31]. The SAT solver MapleChrono [32] was used as SAT oracle in the PySAT. The time limit for checking one path was 0.01 seconds.

We constructed estimates of decomposition hardness before and after DJD preprocessing in 12 different ways for each formula under the following considerations: a disjunctive diagram by direct order, by frequency order, and by 10 different random orders. In all our experiments decomposition hardness equals the time of sequential solving of all weakened subproblems in corresponding partitioning.

Estimates were constructed from a random sample of 1000 subproblems. As an example, detailed results for instance `CvK` are shown in Table I. Decomposition hardness of the instance `CvK`, constructed by solving all 2^{16} subproblems, was 2178663 seconds, with an average runtime of one subproblem equals 33.24 seconds.

The first column of Table I shows the order in which the disjunctive diagram was constructed to perform preprocessing. The second column of the table gives the average time to solve the subproblems from the decomposition. The third column gives an estimate of the time it would take to completely solve the all sub-problems when performing order-specific preprocessing. As can be seen in the table, the best estimate was obtained for one of the random orders.

A similar study was performed for all of the problems under consideration. In the results for each CNF, except for `DvW`, an order was found whose preprocessing improves the decomposition hardness estimation more than the other orders.

In the next experiment, a complete solving of all subproblems in corresponding partitioning was performed to obtain the exact decomposition hardness for each problem before and after preprocessing with the best order w.r.t. found estimate for each considered LEC problem.

Let us briefly describe Table II. The first column gives the names of the problems under consideration. The second column gives the decomposition hardness of the problems. In this case, we are no longer talking about estimation, but about the exact time it takes to solve all subproblems in the decomposition (for corresponding calculations computing cluster was used). The third column gives the decomposition hardness of problems after preprocessing, in case it is better than that given in the second column. The last column of the table indicates how much faster the entire partitioning was solved when DJD-preprocessing was applied. As we can see, for 5 of the 6 pairs of multiplication algorithms considered, preprocessing improved the decomposition difficulty of the corresponding equivalence check problem by $5-14\%$. And this can be considered as significant achievement for such hard problem as LEC for two multipliers.

V. CONCLUSION AND FUTURE WORK

In the present paper we propose an approach to preprocessing of extremely hard SAT instances using disjunctive diagrams (DJD) introduced in [5]. The key feature of our approach is that we construct new constraints, which augment an original formula C, using different orders to construct DJD for C. Also we use the relatively new technique [17] to estimate the hardness of considered formulas. We demonstrate that proposed preprocessing technique allows to improve the state-of-the-art SAT solvers performance in application to extremely hard formulas encoding the Logical Equivalence Checking problem for two Boolean circuits implementing different algorithms of integer multiplication. In the nearest future we plan to develop preprocessing algorithms which use DJD structure in more complete manner, rearranging the diagram due to exclusion some paths in vertex '?' with help of SAT oracle.

ACKNOWLEDGMENTS

We would like to express our deep gratitude to anonymous reviewers for their valuable comments.

Victor Kondratiev and Alexander Semenov were supported by the Russian Science Foundation, project № 22-21-00583. Their contribution consists of theoretical base of proposed method and development of related algorithms.

Ilya Otpuschennikov was supported by the Ministry of Science and Higher Education of Russian Federation, research project no. 121041300065-9. His contribution is development of benchmarks sets, used in the experimental part of this research.

Victor Kondratiev was also supported by a personal scholarship from Huawei.

REFERENCES

[1] S. Arora and B. Barak, *Computational Complexity: A Modern Approach*. Cambridge University Press, 2009.
[2] O. Goldreich, *Computational Complexity: A Conceptual Perspective*. Cambridge University Press, 2008.
[3] A. Kuehlmann and F. Krohm, "Equivalence checking using cuts and heaps," in *DAC*, 1997, pp. 263–268.
[4] P. Molitor and J. Mohnke, *Equivalence Checking of Digital Circuits: Fundamentals, Principles, Methods*. Kluwer Academic Publishers, 2004.

[5] A. A. Semenov and I. V. Otpuschennikov, “On one class of decision diagrams,” *Automation and Remote Control*, vol. 77, no. 4, pp. 617–628, 2016.

[6] A. Biere, M. Heule, H. van Maaren, and T. Walsh, Eds., *Handbook of Satisfiability - Second Edition*, ser. FAIA. IOS Press, 2021, vol. 336.

[7] C.-L. Chang and R. C.-T. Lee, *Symbolic Logic and Mechanical Theorem Proving*, ser. Computer Science Classics. Academic Press, 1973.

[8] G. S. Tseitin, “On the complexity of derivation in propositional calculus,” *Studies in Constructive Mathematics and Mathematical Logic, Part II*, pp. 115–125, 1970.

[9] I. Wegener, *The Complexity of Boolean Functions*. John Wiley & Sons, 1987.

[10] A. Biere, “The AIGER And-Inverter Graph (AIG) format version 20071012,” Institute for Formal Models and Verification, Johannes Kepler University, Altenbergerstr. 69, 4040 Linz, Austria, Tech. Rep. 07/1, 2007.

[11] R. Drechsler, T. A. Junttila, and I. Niemelä, “Non-clausal SAT and ATPG,” in *Handbook of Satisfiability*, ser. Frontiers in Artificial Intelligence and Applications, A. Biere, M. Heule, H. van Maaren, and T. Walsh, Eds. IOS Press, 2009, vol. 185, pp. 655–693.

[12] D. Kaufmann, A. Biere, and M. Kauers, “Verifying Large Multipliers by Combining SAT and Computer Algebra,” in *2019 Formal Methods in Computer Aided Design (FMCAD)*, 2019, pp. 28–36.

[13] D. Kaufmann and A. Biere, “Amulet 2.0 for verifying multiplier circuits,” in *TACAS*, ser. LNCS, vol. 12652, 2021, pp. 357–364.

[14] D. Kaufmann and A. Biere, “Improving AMulet2 for verifying multiplier circuits using SAT solving and computer algebra,” *International Journal on Software Tools for Technology Transfer*, 2023.

[15] C. Yu, M. Ciesielski, and A. Mishchenko, “Fast algebraic rewriting based on And-Inverter Graphs,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 9, pp. 1907–1911, 2018.

[16] A. A. Semenov, D. Chivilikhin, A. Pavlenko, I. V. Otpuschennikov, V. Ulyantsev, and A. Ignatiev, “Evaluating the hardness of SAT instances using evolutionary optimization algorithms,” in *CP*, ser. LIPIcs, vol. 210, 2021, pp. 47:1–47:18.

[17] A. A. Semenov, K. Chukharev, E. Tarasov, D. Chivilikhin, and V. Kondratiev, “Estimating the hardness of SAT encodings for logical equivalence checking of boolean circuits,” *CoRR*, vol. abs/2210.01484, 2022.

[18] R. E. Bryant, “Graph-Based Algorithms for Boolean Function Manipulation,” *IEEE Transactions on Computers*, vol. C-35, no. 8, pp. 677–691, 1986.

[19] S.-I. Minato, “Zero-Suppressed BDDs for Set Manipulation in Combinatorial Problems,” in *Proceedings of DAC ’93*. ACM, 1993, p. 272–277.

[20] A. E. J. Hyvärinen, “Grid Based Propositional Satisfiability Solving,” 2011, PhD thesis. Aalto University publication series.

[21] C. Ansótegui, M. L. Bonet, J. Levy, and F. Manyà, “Measuring the hardness of SAT instances,” in *AAAI*, 2008, p. 222–228.

[22] R. Williams, C. P. Gomes, and B. Selman, “Backdoors to typical case complexity,” in *IJCAI*, 2003, pp. 1173–1178.

[23] “Irkutsk Supercomputer Center of SB RAS. URL: http://hpc.icc.ru.” [Online]. Available: http://hpc.icc.ru

[24] C. S. Wallace, “A suggestion for a fast multiplier,” *IEEE Transactions on Electronic Computers*, vol. EC-13, no. 1, pp. 14–17, 1964.

[25] L. Dadda, “Some schemes for parallel multipliers,” *Alta Frequenza*, vol. 34, no. 5, pp. 349–356, May 1965.

[26] D. Knuth, *The Art of Computer Programming, Volume 2: Seminumerical Algorithms*, ser. Addison-Wesley Series in Computer Science & Information Processing. Addison-Wesley, 1969.

[27] I. Otpuschennikov, A. Semenov, I. Gribanova, O. Zaikin, and S. Kochemazov, “Encoding Cryptographic Functions to SAT Using TRANSALG System,” in *ECAI 2016, Frontiers of Arificial Intelligence an Applications*, vol. 285, 2016, pp. 1594–1595.

[28] A. Semenov, I. Otpuschennikov, I. Gribanova, O. Zaikin, and S. Kochemazov, “Translation of Algorithmic Descriptions of Discrete Functions to SAT with Applications to Cryptanalysis Problems,” *Logical Methods in Computer Science*, vol. 16, no. 1, pp. 1–42, 2020.

[29] A. Biere, K. Fazekas, M. Fleury, and M. Heisinger, “CaDiCaL, Kissat, Paracooba, Plingeling and Treengeling entering the SAT Competition 2020,” in *Proc. of SAT Competition 2020 – Solver and Benchmark Descriptions*, ser. Department of Computer Science Report Series B, vol. B-2020-1. University of Helsinki, 2020, pp. 51–53.

[30] V. Kondratiev, “Using disjunctive diagrams for preprocessing of conjunctive normal forms,” in *2022 45th Jubilee International Convention on Information, Communication and Electronic Technology (MIPRO)*, 2022, pp. 883–887.

[31] A. Ignatiev, A. Morgado, and J. Marques-Silva, “PySAT: A Python toolkit for prototyping with SAT oracles,” in *SAT*, 2018, pp. 428–437.

[32] A. Nadel and V. Ryvchin, “Chronological Backtracking,” in *SAT*, ser. LNCS, vol. 10929, 2018, pp. 111–121.