Satisfiability Solving Meets Evolutionary Optimisation in Designing Approximate Circuits

Milan Češka, Jiří Matyáš(✉), Vojtech Mrazek, and Tomáš Vojnar

FIT, Brno University of Technology, Brno, Czech Republic
imatyas@fit.vutbr.cz

Abstract. Approximate circuits that trade the chip area or power consumption for the precision of the computation play a key role in development of energy-aware systems. Designing complex approximate circuits is, however, very difficult, especially, when a given approximation error has to be guaranteed. Evolutionary search algorithms together with SAT-based error evaluation currently represent one of the most successful approaches for automated circuit approximation. In this paper, we apply satisfiability solving not only for circuit evaluation but also for its minimisation. We consider and evaluate several approaches to this task, both inspired by existing works as well as novel ones. Our experiments show that a combined strategy, integrating evolutionary search and SMT-based sub-circuit minimisation (using quantified theory of arrays) that we propose, is able to find complex approximate circuits (e.g. 16-bit multipliers) with considerably better trade-offs between the circuit precision and size than existing approaches.

1 Introduction

Approximate circuits are digital circuits that trade functional correctness (precision of computation) for other design objectives such as chip area or power consumption. Such circuits play an important role in development of resource-efficient systems, including applications such as image and video processing [10] or neural networks [14,17]. Designing approximate systems, i.e. finding optimal trade-offs between the approximation error and resource savings is, however, a complex and time-demanding process. Automated methods allowing one to develop high-quality approximate circuits are thus in high demand, especially when a bound on the approximation error is to be guaranteed.

There exists a vast body of literature (see, e.g. [13,16,18,19,26]) demonstrating that evolutionary-based algorithms are able to automatically design innovative approximate circuits providing high-quality trade-offs among the different design objectives. There are two main challenges related to the

This work was supported by the Czech Science Foundation grant GJ20-02328Y, the JCMM Brno Ph.D. Talent scholarship program, and the BUT project FIT-S-20-6427.

L. Pulina and M. Seidl (Eds.): SAT 2020, LNCS 12178, pp. 481–491, 2020.
https://doi.org/10.1007/978-3-030-51825-7_33

evolutionary-driven circuit approximation: (1) Finding a fast and reliable evaluation of candidate solutions. (2) Designing a quickly converging search strategy that drives the exploration towards high-quality solutions.

Concerning the first challenge, several circuit evaluation techniques have been proposed including parallel circuit simulation [27] and various formal methods [5,9,25,28]. In our recent work [3], we proposed and implemented a new *miter* construction together with a resource-limited verifier for SAT-based evaluation of the worst-case error. This approach has made feasible approximation of complex circuits, going beyond 16-bit adders and 12-bit multipliers, which were the limits of previously known techniques. In this paper, we aim at the second challenge. Inspired by recent advances in SAT-based exact synthesis [11,24] (the problem of finding the optimum logic representation of a given Boolean function), we investigate whether a search strategy based on *satisfiability solving* (StS)—i.e. SAT or SMT solving—can improve state-of-the-art methods for designing complex approximate circuits.

We emphasize that complex circuits typically have more than a thousand gates and thus a monolithic approach, i.e. representing the circuit approximation problem as a single StS query, is not tractable. Instead, we build on an iterative approach where sub-circuits are optimised (i.e. the sub-circuit logic is minimised while the original functionality is preserved) [23] or approximated (i.e. the functionality of the sub-circuit is not preserved and the error of the whole circuit is increased—to our best knowledge the iterative approximation has not been considered yet).

Despite the enormous progress in satisfiability solving, our experiments clearly show that the purely StS-based approximation significantly lags behind the standard evolutionary approximation. Although the StS-based approximation performs informed (and thus in some sense more useful) changes in the candidate circuits, the overhead caused by calling the solver does not pay off compared to the uninformed but very cheap genetic mutations. Put differently, the evolution can perform over 100-times more approximation attempts which is enough to overcome the benefit of the informed changes.

In order to leverage the benefits of the informed changes, we propose a *combined approach*. We interleave the evolutionary approximation and the StS-based optimisation. The evolutionary approximation typically quickly converges to a sub-optimal solution. After the progress of the evolution decreases below a certain threshold, we run the StS-based optimisation. It further reduces the circuit size, but, more importantly, it introduces new reconnections in the circuit causing that the subsequent evolution is able to escape a local minimum and further explore the design space.

2 Designing Approximate Circuits

Technology-independent functional approximation is the most preferred approach to approximation of digital circuits. The goal is to replace the original accurate circuit (further denoted as the *golden circuit*) by a less complex circuit which exhibits some errors but improves non-functional circuit parameters such

as power, delay, or chip area. Fully-automated functional approximation methods typically employ various heuristics to simplify the circuit logic and reduce its area approximated by the sum of the sizes of the gates used—this sum is further denoted as the *circuit size*.

The circuit size can be reduced either by replacing a gate by a smaller one or by disconnecting a gate. The gate is disconnected if there is no connection between its output and the primary outputs of the circuit. The essential operation in the approximation process is thus gate reconnection allowing one to disconnect some gates. We stress that the space of possible reconnections grows exponentially with the circuit size, and each reconnection typically causes a non-trivial change in the overall circuit functionality.

To overcome this complexity, existing approximation techniques leverage various forms of greedy algorithms, such as ABACUS [18], or genetic algorithms, such as Cartesian Genetic Programming [13,26], to identify suitable reconnections. The approximation then boils down to iteratively generating candidate solutions and evaluating their quality, i.e. the obtained trade-off between the circuit area and error. Circuit approximation can be naturally formulated as multi-objective optimisation, but most works consider single-objective optimisation of the circuit size for several predefined target errors—this is motivated practically as the required error levels are typically known in advance and single-objective optimisation is computationally less demanding.

There exist several metrics to quantify the error [8] and different techniques allowing one to evaluate these metrics. For small circuits (up to 12-bit inputs), parallel circuit simulation [27] provides the best performance. For larger circuits, various formal verification techniques have been proposed [5,9,25,28]. In this paper, we build on the SAT-based technique we proposed in [3] allowing one to verify whether a given candidate circuit meets the required bound T on the worst-case absolute error (WCAE), i.e. whether the difference between the candidate and the golden circuit is smaller than T for every input. The technique constructs a *miter* [28], an auxiliary circuit interconnecting the golden and candidate circuit and allowing one to check their approximate equivalence given by the bound T. The technique allows us to approximate complex circuits (16-bit multipliers and beyond).

Recent advances in exact SAT-based synthesis of Boolean chains [24], providing efficient implementation of a given Boolean function, have opened new avenues for automated circuit design and optimisation. In this paper, we investigate whether these advances can improve circuit approximation too.

3 SAT-based Circuit Approximation

We propose three different approaches for SAT-based circuit approximation.

3.1 A Monolithic Approach

The monolithic approach builds a single formula encoding the following synthesis problem: *For a given golden circuit GC, the size S of its currently*

best-known approximation, and an error bound T*, synthesize an approximating circuit* AC *whose size is smaller than* S *and that satisfies the constraint that* $error(GC, AC) < T$.

The formula has to encode the following features: (1) possible designs of the circuit (i.e. possible interconnections and functionality of the gates), which must be encoded using free variables whose suitable values are to be found by the solver, thus fixing a certain design of the circuit; (2) the way the error of the circuit is to be checked; and (3) the way the circuit size is to be evaluated.

In our approach, we use a forward-propagating network of two-input gates to represent the designed circuit. We represent each gate by three integers. The first two represent the inputs of the gate, and they can refer to some of the primary inputs or to the output of one of the gates (which we identify with the gate itself). The third integer then encodes the gate's functionality that is chosen from a predefined set of operations. A gate implementing each possible operation has a predefined size given by the target chip architecture. To ensure that the size of the synthesized circuit AC is smaller than the size S of the currently best approximation, we add a constraint on the sum of the sizes of the gates forming AC. We investigate and compare (cf. Section 4) the below presented three ways of encoding the structure and functionality of C.

The first encoding is purely *SAT-based* although we present it using both Boolean and integer variables—those are, however, bit-blasted away. Assume we have k types of (binary) gates, use l gates, and have m/n primary input/output bits, respectively. For each gate $g \in G = \{1, ..., l\}$, we use the integer variables $in_{g,1}$ and $in_{g,2}$ to denote the first and second input of g. These variables range over the domain $W = \{1, ..., m + l\}$ of all wires in the circuit where the first m wires carry the primary inputs and the next l wires carry the outputs of the different gates. For $g \in G$, we also use the integer variable f_g to denote its type with the domain $F = \{1, ..., k\}$. Let $\mathbb{I} = \{0, 1\}^m$ denote the different input combinations. For $u \in W$, we use the Boolean variable b_u^I to hold the value of the wire u for a primary input $I \in \mathbb{I}$. We encode all possible circuits by the conjunction of the formulae $(in_{g,1} = u \wedge in_{g,2} = v \wedge f_g = f) \rightarrow \bigwedge_{I \in \mathbb{I}}(b_{m+g}^I = b_u^I \; op_f \; b_v^I)$ that are generated for all gates $g \in G$, all possible types $f \in F$ of g, and all wires $u, v \in W$ that may be used as the inputs of g. In particular, we require that $u < g$ and $v < g$ to prevent backward connections in the circuit (e.g. the input of g_4 cannot be connected to the output of g_6). In the formula, op_f denotes the Boolean operation implemented by gates of the type $f \in F$.

We also need to link the concrete input combinations with the input wires, which is done by the conjunction $\bigwedge_{I \in \mathbb{I}} \bigwedge_{j \in \{1,...,m\}} b_j^I = I[j]$ where $I[j]$ denotes the j-th bit of I. Finally, for each output $o \in O = \{1, ..., n\}$, we introduce the integer variable out_o, which ranges over the domain of wires W and says from where the output o is taken, and the Boolean variable out_o^I carrying the value of the output o for the primary input $I \in \mathbb{I}$ (this variable will be compared with the appropriate output of the golden circuit). These variables are connected with the rest of the circuit using the conjunction of the formulae $out_o = u \rightarrow \bigwedge_{I \in \mathbb{I}} out_o^I = b_u^I$ generated for every output $o \in O$ and every wire $u \in W$. The solver then

chooses a concrete circuit by fixing the values of the variables $in_{g,1}$, $in_{g,2}$, and f_g for every $g \in G$ as well as the values of the variables out_o for every $o \in O$.

Second, using a *theory of arrays*, we simplify the above encoding by using an array $\overline{b}^I : W \to \{0,1\}$ for each $I \in \mathbb{I}$ to hold the values of the wires in W for the input I. Then, the conjuncts describing the structure of the circuit may be simplified to $f_g = f \to \bigwedge_{I \in \mathbb{I}}(\overline{b}^I[m+g] = \overline{b}^I[in_{g,1}]\ op_f\ \overline{b}^I[in_{g,2}])$. The input formula is changed to $\bigwedge_{I \in \mathbb{I}} \bigwedge_{j \in \{1,\ldots,m\}} \overline{b}^I[j] = I[j]$ and similarly for the output. Finally, third, using a *theory of arrays with quantifiers*, one suffices with a single array $\overline{b}$, simplifying the formulae describing the structure of the circuit to $f_g = f \to \overline{b}[m+g] = \overline{b}[in_{g,1}]\ op_f\ \overline{b}[in_{g,2}])$, adding the universal quantification $\forall i_1, ..., i_m$ over the entire formula, using the input formula $\bigwedge_{j \in \{1,\ldots,m\}} \overline{b}[j] = i_j$, and handling the output accordingly.

Using our encodings of AC, we can easily add a constraint on the required error that compares the WCAE between the result coming from AC (using the out_o variables) and the expected result for all input combinations.

A comparison to existing encoding schemes for exact synthesis. Our encoding of circuits is quite similar to other works such as [24]. The authors of [24] do not consider a predefined set of gates. Instead, they synthesize the internal functionality of the gates too. The work [24] and other existing approaches use SAT based encodings only. Further, they consider uniform gate sizes only (the circuit size is given by the number of gates). Our more general formulation using non-uniform gate sizes leads to more complex problems. As in [22], we use simplifications and symmetry-pruning to reduce the complexity of the StS queries.

3.2 Sub-circuit Approximation

As discussed in [11], the monolithic approach for exact synthesis is feasible only for small circuits up to 8 input bits (depending on the complexity of the synthesized function). Our experiments confirm similar scalability limits also for circuit approximation (cf. Sect. 4), and thus we focus on an iterative approach that approximates selected sub-circuits. We focus on approximation wrt. Hamming Distance as arithmetic metrics are not suitable for sub-circuits. Note that there is no effective method allowing us to determine how the error introduced in the sub-circuit affects the overall circuit error.

In every iteration, we select a single gate (either randomly or by enumeration, depending on the circuit size) and perform a breadth-first search starting from the selected gate to identify a sub-circuit of a suitable size. Note that, in our approach, we consider multi-input and multi-output sub-circuits. The size of the sub-circuits is indeed essential: Considering only very small sub-circuits prevents the approximation from doing more complicated and non-local changes that are crucial for finding high-quality approximate circuits. On the other hand, approximation of larger sub-circuits introduces a significant overhead causing that only a small number of iterations can be done within the given time limit. Regarding the encoding of sub-circuit approximation, we consider the same schemes as in

the monolithic approach discussed above. After every sub-circuit approximation, we need to evaluate the error of the whole circuit. If it satisfies the error bound, we accept the circuit as the new candidate solution, otherwise the next iteration continues with the circuit before the approximation.

3.3 Evolutionary Approximation with StS-Based Optimisation

Evolutionary algorithms, in particular Cartesian Genetic Programming (CGP), have achieved excellent results in approximation of complex circuits [3]. The key idea is similar to sub-circuit approximation, but here CGP performs random changes in the candidate solution instead of utilising satisfiability solving. Unlike finding an optimal sub-circuit approximation, random changes are very fast, and the success of CGP is typically achieved by a large number of small changes. We emphasize that CGP is also able to accumulate a large change in the candidate circuit via so-called *inactive mutations* [15]—a chain of changes where only the last change directly affects the circuit functionality. Although CGP usually quickly converges to a sub-optimum solution, it can get stuck in this solution for a long time. On the other hand, the StS-based approach is able to systematically search for improvements that are hard to find for CGP.

We hence propose a *combined approach* leveraging the benefits of both techniques. In particular, we interleave the evolutionary search by iterative StS optimisation. In contrast to StS-based approximation, StS-based optimisation minimises the size of the selected sub-circuit by changing the internal structure while preserving its functionality. The rationale behind this is based on the observation that a large portion of approximated sub-circuits are rejected as they cause that the WCAE error of the whole circuit gets above the allowed bound. Compared with CGP, the cost of each approximation operation is too high—in our scenarios, CGP is about 100-times faster. Therefore, the combined approach uses CGP to introduce changes affecting the functionality, and the StS-based optimisation to minimise the logic.Further, we also explore different encoding schemes for the optimisation problem.

The interleaving is controlled in the following way. If CGP gets stuck in a local optimum, we switch to the iterative StS optimisation that has a time budget depending on the given overall time for the approximation. Once the budget is spent, we continue with again CGP. Our experiments show that the optimisation helps CGP to escape the local optimum and to further effectively explore the space of candidate circuits.

4 Experimental Results

We ran all our experiments on a server with an Intel(R) Xeon(R) CPU at 2.40 GHz. Although search-based approximation can naturally benefit from a simple task parallelisation, we use a single-core computation to simplify the interpretation of the results.

A Comparison of different encoding schemes and satisfiability solvers. We consider a set of formulae relevant for the monolithic as well as for the iterative approach. The set includes both SAT and UNSAT instances including a full adder, 2-bit adder, 2-bit multiplier, 4-1 multiplexor, and some randomly generated 4-input functions. We compare the total time needed to solve all the formulae with a 3 hour time limit. We do not include any additional penalty for timing out. The Z3 solver [6] and the quantified array encoding proved to be the fastest combination of the encoding and the solver. Z3 with separate arrays for different input combinations is about 2 times slower, and the Glucose solver [1] with the purely SAT-based encoding is about 3 times slower. Z3 with the purely SAT-based encoding as well as its SMT variant without bit-blasting were roughly 5 times slower. Other tested solvers—MathSAT [2], Minisat [7], Sadical [12], and Vampire [20]—were all more than 5 times slower. Based on these observations, we use Z3 with the quantified array encoding in all further StS-based queries.

The monolithic approach. The monolithic approach was able to find optimal approximations of 2-bit adders and multipliers as well as randomly generated functions with 4 inputs and 2 outputs. Approximation of larger circuits proved to be infeasible, i.e. most instances timed out within the given limit of 3 h.

We compare the performance of our monolithic approach with Cirkit [21], a state-of-the-art tool for exact synthesis. As expected, Cirkit is able to achieve a better performance and scalability: It is significantly faster on 4-bit functions and it can also synthesize optimal solutions for some 6-bit and 8-bit functions. However, there are also some hard 6-bit instances that are infeasible for Cirkit.

The better performance of Cirkit is mainly caused by the following factors: (1) Our formulation of circuit approximation is more complicated due to the non-uniform gate sizes and the error quantification. (2) Cirkit uses different circuit representations (such as AIGs, MIGs, or n-bit look-up tables) that proved to be more efficient for some exact synthesis problems [22]. (3) Cirkit implements various optimisations and symmetry breaking methods [11]. Some of these methods are problem- and representation-specific and thus not directly applicable to our approximation problem. We are, however, aware that our current prototype implementation could be improved by adapting some of the methods. However, the improvements would not change the practical limits of the monolithic approach.

In the following subsections, we will examine three strategies for approximation of complex circuits: (1) CGP: the state-of-the-art evolutionary approximation [4]. (2) SMT: the sub-circuit approximation from Sect. 3.2. (3) COMB: the combined approach from Sect. 3.3 using the following interleaving strategy:

In each iteration, we run the CGP-based approximation until no improvement is found for 100 K generations. Then, for 10% of the overall time limit, we switch to the SMT-based optimisation. Afterwards, a new iteration starts.

Based on our preliminary experiments, we use sub-circuits with 5 gates (recall the discussion in Sect. 3.2) in all SMT-based sub-circuit approximation and optimisation queries. We also introduce a hard time limit on every such query.

Table 1. The resulting size of the approximate circuits, obtained using the proposed approximate strategies, expressed as the percentage of the size of the golden circuits (left) and of the size of the best known approximations presented in [3] (right).

	8-bit adders			4-bit multipliers		
Err	CGP	SMT	COMB	CGP	SMT	COMB
1 %	64.8	83.5	**54.5**	78.4	90.5	**74.6**
2 %	52.6	78.0	**44.9**	69.3	82.6	**67.1**
5 %	37.1	57.4	**32.3**	53.4	77.0	**49.7**

32-bit adders			16-bit multipliers		
Err[%]	CGP	COMB	Err[%]	CGP	COMB
10^{-5}	100.0	**81.5**	10^{-3}	97.9	**91.4**
10^{-4}	100.0	**81.3**	0.01	97.6	**91.1**
10^{-3}	100.0	**81.1**	0.1	95.0	**90.1**

It prevents the SMT solver to spend a prohibitively long time in complex queries and thus to significantly slow down the approximation process.

4.1 Performance on Small Circuits

We first consider small circuits (a 4-bit multiplier with 67 gates and an 8-bit adder with 49 gates) to understand performance aspects of the search strategies. We report the area savings (as the percentage of the size of the golden circuit) for selected WCAE error bounds and the approximation time limit of 1 h.

Table 1 (left) shows the results obtained from 15 independent approximation runs for each combination of the approximation method, circuit, and target error. For the 8-bit adder, the combined approach wins in all 45 evolutionary runs. On average, the combined approach saves 7.6% more than the pure CGP and 29% more than the pure SMT-based approach. For the 4-bit multiplier, the combined strategy provides 3.27% better savings than the pure CGP and 19,6% more than the pure SMT-based approach. It also wins 37 out of 45 comparisons.

These experiments show that the pure SMT approximation is not competitive, and it is not considered in the following approximation of complex circuits.

4.2 Performance on Complex Circuits

In this subsection, we focus on our key research question: *Can the combined strategy improve the performance of the approximation of complex circuits?*

We consider approximation of (1) a 32-bit adder (the golden model has 235 gates), and (2) 16-bit multiplier (the golden model has 1,534 gates). To evaluate the potential of the combined strategy, we start with state-of-art approximate circuits we obtained in our previous work by a pure evolutionary search strategy [3]. For each target error, we choose the best 32-bit adders and 16-bit multipliers, obtained by 2 and 8-h approximation runs respectively. From each of these circuits (seeds), we continue the approximation using pure CGP and combined strategy for 10 h (adders) and 75 h (multipliers).

32-bit adders. Each pure CGP run performs around 10 million iterations within the given 10 h but achieves no improvements at all. The sub-circuit optimisation, however, introduces changes in the circuit structure, which allow CGP to escape the local optimum and perform further improvements. In total, the combined strategy saves roughly 19% of the seeding circuit area—11% was achieved by the CGP approximation and 8% by SMT optimisation.

16-bit multipliers. As illustrated in Fig. 1, which shows the progress of the two approximation strategies for different target errors, the pure CGP approximation improves the candidate slowly and achieves only marginal improvements after 45 h. The combined strategy is able to improve the candidate solution during the whole 75-h run—after this time, it saves 4–6% more than the pure CGP. Recall that, compared to 32-bit adders, the approximation of the 16-bit multipliers is significantly more complex. The 8-h CGP run computing the seed performs around 230 K iterations, which is around 13-times less than the 2-h run for the 32-bit adder. Hence, the pure CGP run requires much more time to reach the local optimum.

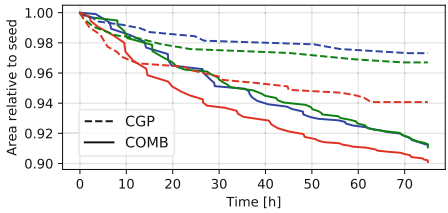

Fig. 1. Progress of the area reduction for the 16-bit multiplier and target WCAEs: red = 10^{-1}%, green = 10^{-2}%, blue = 10^{-3}%.

Conclusion. The proposed fusion of satisfiability solving and evolutionary optimisation leads to a new circuit approximation strategy that is able to effectively escape local optima and thus to explore the design space more effectively than pure evolutionary search strategies. The obtained approximate circuits provide the best known trade-offs between the precision and the chip area.

References

1. Audemard, G., Simon, L.: On the glucose SAT solver. Int. J. Artif. Intell. Tools **27**, 1840001 (2018)
2. Bruttomesso, R., Cimatti, A., Franzén, A., Griggio, A., Sebastiani, R.: The MathSAT 4 SMT Solver. In: Gupta, A., Malik, S. (eds.) CAV 2008. LNCS, vol. 5123, pp. 299–303. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-70545-1_28
3. Češka, M., Matyáš, J., et al.: Approximating complex arithmetic circuits with formal error guarantees: 32-bit multipliers accomplished. In: International Conference on Computer Aided Design (ICCAD'2017), pp. 416–423. IEEE (2017)
4. Češka, M., Matyáš, J., Mrazek, V., Sekanina, L., Vasicek, Zdenek, Vojnar, Tomáš: ADAC: Automated design of approximate circuits. In: Chockler, H., Weissenbacher, G. (eds.) CAV 2018. LNCS, vol. 10981, pp. 612–620. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-96145-3_35
5. Chandrasekharan, A., Soeken, M., et al.: Precise error determination of approximated components in sequential circuits with model checking. In: Design Automation Conference (DAC'2016), pp. 129:1–129:6. ACM (2016)
6. de Moura, L., Bjørner, N.: Z3: An efficient SMT solver. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 337–340. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-78800-3_24
7. Eén, N., Sörensson, N.: An extensible SAT-solver. In: Giunchiglia, E., Tacchella, A. (eds.) SAT 2003. LNCS, vol. 2919, pp. 502–518. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24605-3_37

8. Froehlich, S., Große, D., Drechsler, R.: One method - all error-metrics: a three-stage approach for error-metric evaluation in approximate computing. In: Design, Automation Test in Europe Conference Exhibition (2019)
9. Froehlich, S., Grosse, D., Drechsler, R.: Approximate hardware generation using symbolic computer algebra employing grobner basis. In: Design, Automation Test in Europe Conference Exhibition (DATE'2018), pp. 889–892. IEEE (2018)
10. Gupta, V., Mohapatra, D., et al.: Low-power digital signal processing using approximate adders. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **32**(1), 124–137 (2013)
11. Haaswijk, W., Soeken, M., et al.: SAT based exact synthesis using DAG topology families. In: Design Automation Conference (DAC'2018), pp. 1–6 (2018)
12. Heule, M.J.H., Kiesl, B., Biere, A.: Encoding redundancy for satisfaction-driven clause learning. In: Vojnar, T., Zhang, L. (eds.) TACAS 2019. LNCS, vol. 11427, pp. 41–58. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-17462-0_3
13. Lotfi, A., Rahimi, A., et al.: Grater: an approximation workflow for exploiting data-level parallelism in FPGA acceleration. In: Design, Automation Test in Europe Conference Exhibition (DATE'2016), pp. 1279–1284. EDA Consortium (2016)
14. Mahdiani, H.R., Ahmadi, A., et al.: Bio-inspired imprecise computational blocks for efficient VLSI implementation of soft-computing applications. IEEE Trans. Circuits Syst. I Regul. Pap. **57**(4), 850–862 (2010)
15. Miller, J.F., Thomson, P.: Cartesian genetic programming. In: Poli, R., Banzhaf, W., Langdon, W.B., Miller, J., Nordin, P., Fogarty, T.C. (eds.) EuroGP 2000. LNCS, vol. 1802, pp. 121–132. Springer, Heidelberg (2000). https://doi.org/10.1007/978-3-540-46239-2_9
16. Mrazek, V., Hrbacek, R., et al.: EvoApprox8b: library of approximate adders and multipliers for circuit design and benchmarking of approximation methods. In: Design, Automation Test in Europe Conference Exhibition (DATE'2017) (2017)
17. Mrazek, V., Sarwar, S.S., et al.: Design of power-efficient approximate multipliers for approximate artificial neural networks. In: International Conference on Computer Aided Design (ICCAD'2016), pp. 811–817. ACM (2016)
18. Nepal, K., Hashemi, S., et al.: Automated high-level generation of low-power approximate computing circuits. IEEE Trans. Emerg. Top. Comput. **7**, 18–30 (2018)
19. Reda, S., Shafique, M. (eds.): Approximate Circuits. Springer, Cham (2019). https://doi.org/10.1007/978-3-319-99322-5
20. Riazanov, A., Voronkov, A.: The design and implementation of vampire. AI Commun. **15**, 91–110 (2002)
21. Soeken, M.: Cirkit (version 3). https://github.com/msoeken/cirkit (2019)
22. Soeken, M., Amarù, L.G., et al.: Exact synthesis of majority-inverter graphs and its applications. IEEE Trans. Comput. -Aided Des. Integr. Circuits Syst. **36**(11), 1842–1855 (2017)
23. Soeken, M., De Micheli, G., Mishchenko, A.: Busy man's synthesis: combinational delay optimization with sat. In: Design, Automation Test in Europe Conference Exhibition (DATE'2017), pp. 830–835 (2017)
24. Soeken, M., Haaswijk, W., et al.: Practical exact synthesis. In: Design, Automation Test in Europe Conference Exhibition (DATE'2018), pp. 309–314 (2018)
25. Vasicek, Z., Mrazek, V.: Towards low power approximate DCT architecture for HEVC standard. In: Design, Automation Test in Europe Conference Exhibition (DATE'2017) (2017)
26. Vasicek, Z., Sekanina, L.: Evolutionary approach to approximate digital circuits design. IEEE Trans. Evol. Comput. **19**(3), 432–444 (2015)

27. Vašíček, Z., Slaný, K.: Efficient phenotype evaluation in cartesian genetic programming. In: Moraglio, A., Silva, S., Krawiec, K., Machado, P., Cotta, Carlos (eds.) EuroGP 2012. LNCS, vol. 7244, pp. 266–278. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-29139-5_23
28. Venkatesan, R., Agarwal, A., et al.: MACACO: Modeling and analysis of circuits for approximate computing. In: International Conference on Computer Aided Design (ICCAD'2011), pp. 667–673. ACM(2011)