Impact of subcircuit selection on the efficiency of CGP-based optimization of gate-level circuits

Jitka Kocnova, Zdenek Vasicek
Brno University of Technology, Faculty of Information Technology, IT4I Innovation Centre of Exellence
Czech Republic
ikocnova@fit.vutbr.cz,vasicek@fit.vutbr.cz

ABSTRACT

Various EA-based methods have been applied to design and optimize logic circuits since the early nineties. The unconventional methods, however, typically suffer from various scalability issues preventing them to be adopted in practice. Recent improvement in the fitness computation procedure connected with the introduction of formal methods in the fitness evaluation such as SAT solvers or BDDs enabled pushing of the limits forward and approaching the complexity of industrial problems. It was demonstrated that EAs can be applied to optimize gate-level circuits consisting of thousands of gates without introducing any decomposition technique. Despite that, the efficiency decreases with increasing the circuit complexity. This problem can be managed by adopting the concept of the so-called iterative resynthesis based on the extraction of smaller sub-circuits from a complex circuit, their local optimization followed by the implantation back to the original circuit. Recently, a method based on the computation of so-called cuts was proposed. In this paper, we propose an alternative approach which is able to select more complex sub-graphs consisting of more nodes and more inputs. Compared to the previous method, the proposed approach allows to improve the efficiency of the optimization. More than 9% and 20% reduction was observed on the highly optimized logic and arithmetic circuits, respectively.

KEYWORDS

Cartesian Genetic Programming, Logic Synthesis, Combinational Circuits

ACM Reference Format:
Jitka Kocnova, Zdenek Vasicek. 2019. Impact of subcircuit selection on the efficiency of CGP-based optimization of gate-level circuits. In *Proceedings of the Genetic and Evolutionary Computation Conference 2019 (GECCO '19 Companion).* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3319619.3321926

1 INTRODUCTION

Logic synthesis, as understood by hardware community, is a process transforming a high-level description into a gate-level or transistor-level implementation. Due to the complexity, the synthesis process is typically broken into a sequence of steps. An important part of the whole process is logic optimization. Its goal is to transform a suboptimal solution into an optimal (or, at leasts near-optimal) gate-level implementation with respect to the given synthesis goals. Due to the scalability issues, the problem is typically represented using a suitable internal representation. Current state-of-the-art logic synthesis tools, such as ABC, represent circuits using a directed acyclic graph denoted as and-inverter graph (AIG) [1]. This representation is simple and scalable and leads to simple algorithms, but suffers from an inherent bias in representation. While eight of ten possible two-input logic gates may be represented by a single AIG node, XOR and XNOR gate require three AIG nodes each. Efficiency of synthesis is then limited as it mostly fully relies on transformations disallowing to increase the number of AIG nodes. Unfortunately, the ability to capture XOR gates is essential for efficient representation of arithmetic and XOR-intensive circuits. It has been shown that there exists a huge class of real-world circuits for which the synthesis fails and provides very poor results [3].

GECCO '19 Companion, July 13–17, 2019, Prague, Czech Republic
2019. ACM ISBN 978-1-4503-6748-6/19/07...$15.00
https://doi.org/10.1145/3319619.3321926

1.1 Evolutionary synthesis of logic circuits

Various evolutionary approaches have been recently successfully applied to optimize logic circuits [3, 4]. In [4], Cartesian Genetic Programming (CGP) was used to optimize large combinational circuits. The evolution was conducted directly at the level of common gates. Substantially better results were obtained compared to the state-of-the-art synthesis working on AIGs. On average, the method enabled a 34% reduction in gate count on when evaluated on an extensive set of benchmark circuits and executed for 15 minutes. Efficiency of this method, however, deteriorates with the increasing number of gates due to the bad scalability of representation – substantially more generations are required to reduce circuits consisting of more than 10^4 gates. To address this problem, the concept of so-called logic resynthesis was applied in [2]. The approach is based on iterative optimization of smaller portions of the original circuit. The optimized circuit is mapped to standard gates and optimized using the proposed method that extracts a relative small sub-circuits that are subsequently optimized by CGP. The original sub-circuit is then replaced by its optimized variant provided that there is any improvement at the global level and the whole process is repeated. These steps are iterated until either a predefined amount of time is exhausted or a required improvement is observed, depending on the design scenario. It was shown in [2] that this method naturally improves scalability of CGP especially in the case of large circuits consisting of thousands of gates. Compared to the direct application of CGP [4], substantial improvement was achieved.

1.2 Goal of this work

This work is focuses on the further improvement of the algorithm used to extracts sub-circuits from the original circuit proposed in [2]. We hypothesize that the optimization of much complex sub-circuits may lead to better improvements.

Algorithm 1: Evolutionary optimization

```
Input: Combinational circuit $G$, set of constraints $E$
Output: Optimized circuit $G'$
1 while termination condition not satisfied do
2     $S \leftarrow$ GetSubcircuit($G$, $E$) ;
3     $S' \leftarrow$ OptimizeByCGP($S$);
4     if $cost(S') < cost(S)$ then
5         $G \leftarrow (G \setminus S) \cup S'$ ;
6 return $G$
```

2 EVOLUTIONARY RESYNTHESIS

The principle of the evolutionary resynthesis of logic circuits is shown in Algorithm 1. The algorithm inputs a combinational circuit at the level of common gates represented by an acyclic graph G consisting of $|G|$ nodes.

At first, a subcircuit S is selected from the original circuit G. This subcircuit is then optimized by means of CGP independently on its context. In case of an improvement, i.e. the cost of S' is better compared to the S, the S' replaces the subcircuit S. This procedure is repeated until a termination condition is satisfied. The set of constrains E is used to setup the parameters of GetSubcircuit procedure.

Cuts were applied to select the subcircuits in [2]. This, however, produces subcircuits with a relatively small volume. In this paper, we propose to implement GetSubcircuit procedure similarly as proposed as in [3]. The selection starts with a randomly chosen node $g \in G$ that is included in S. Then, the neighboring nodes of the nodes already included in S are iteratively added into S. Two parameters are used to restricting the size of S, s_{min} and s_{max} ($s_{min} \leq |S| \leq s_{max}$). The process ends when s_{max} is exceeded or no more nodes remain.

3 RESULTS

The CGP is initialized as proposed in [4] to fairly evaluate the results. The termination condition is the number of iterations, in our case the resynthesis was applied $2 \cdot 10^4$ times. The parameters of OptimizeByCGP are as follows: l-back is equal to the number of gates of S, $5 \cdot 10^5$ generations are used, CGP topology: one row, the number of columns equals to $|S|$, the population size is 2, common 2-input gates are used. The proposed method is compared with [2] and the CGP-based global optimization method [4]. The total number of evaluations, i.e. the number of generated and evaluated candidate solutions, is the same in all cases and equals to 10^{10}. s_{min} was set to 5 and s_{max} was limited to 10^4. The experiments were executed on a set of benchmark circuits from IWLS 2005. For each circuit, five independent evolutionary runs were executed.

Table 1 summarizes the obtained results. It is shown that the proposed method performs substantially better compared [2] to especially for circuits having large depth. As evident, the resynthesis-based approach is able to improve the scalability of CGP. CGP working at global level (the last two columns) have problems with some instances and no reduction was achieved.

4 CONCLUSIONS

Proposed method gives better results than the previously published method. Both methods substantially outperform the CGP applied globally. Future work will be mainly devoted to a further improvement of selection of cuts.

5 ACKNOWLEDGMENTS

This work was supported by The Ministry of Education, Youth and Sports of the Czech Republic – INTER-COST project LTC18053 and by the Brno University of Technology project FIT-S-17-3994.

REFERENCES

[1] Robert Brayton and Alan Mishchenko. 2010. ABC: An Academic Industrial-Strength Verification Tool. In *Computer Aided Verification*. Springer Berlin Heidelberg, Berlin, Heidelberg, 24–40.

[2] Jitka Kocnova and Zdenek Vasicek. 2019. Towards a Scalable EA-based Optimization of Digital Circuits. In *EuroGP'19 (LCNS 11451)*. 81–97.

[3] Lukas Sekanina, Ondrej Ptak, and Zdenek Vasicek. 2014. Cartesian Genetic Programming as Local Optimizer of Logic Networks. In *2014 IEEE Congress on Evolutionary Computation*. IEEE CIS, 2901–2908.

[4] Zdenek Vasicek. 2015. Cartesian GP in Optimization of Combinational Circuits with Hundreds of Inputs and Thousands of Gates. In *EuroGP'15 (LCNS 9025)*. 139–150.

Table 1: Evaluation of the proposed method on a set of industrial benchmarks optimized by the state-of-the-art synthesis (column best ABC). The performance is given as the best as well as average achieved reduction, i.e. the number of removed gates, for the proposed method and two methods available in the literature.

			input (best ABC)		resynthesis [2]		resynthesis – proposed		global method [4]	
Benchmark	inputs	outputs	gates	depth	average	best	average	best	average	best
aes_core	789	532	21128	20	2.9%	2.9%	**4.7%**	**5.5%**	0.6%	1.7%
ethernet	10672	10452	60413	23	0.5%	0.5%	**0.9%**	**1.1%**	0.0%	0.0%
i2c	147	127	1161	12	9.2%	9.2%	**16.9%**	**17.8%**	10.0%	10.7%
systemcdes	314	126	2601	25	4.8%	5.0%	**9.2%**	**10.7%**	9.1%	9.9%
usb_phy	113	73	452	9	13.9%	14.0%	**16.7%**	**17.6%**	12.2%	12.2%
average (logic benchmarks)			15620	20	6.3%	6.4%	**9.7%**	**10.6%**	6.3%	6.5%
sqrt32	32	16	1462	307	**22.3%**	**24.3%**	12.6%	15.4%	3.0%	3.0%
diffeq1	354	193	20719	218	11.5%	11.5%	**13.1%**	**15.7%**	0.0%	0.0%
div16	32	32	5847	152	15.7%	15.8%	**20.5%**	**27.9%**	0.0%	0.0%
hamming	200	7	2724	80	28.6%	30.1%	**40.1%**	**40.9%**	14.6%	14.6%
revx	20	25	8131	171	14.5%	14.5%	**16%**	**17.8%**	0.0%	0.1%
average (aritmetic benchmarks)			8956	148	18.2%	19.2%	**20.5%**	**23.5%**	3.5%	3.5%