2014 IEEE Congress on Evolutionary Computation (CEC)
July 6-11, 2014, Beijing, China

Cartesian Genetic Programming as Local Optimizer of Logic Networks

Lukas Sekanina, Ondrej Ptak and Zdenek Vasicek
Brno University of Technology, Faculty of Information Technology, IT4Innovations Centre of Excellence
Bozetechova 2, 61266 Brno, Czech Republic
Email: sekanina@fit.vutbr.cz, ondrej.ptak@gmail.com, vasicek@fit.vutbr.cz

***Abstract*—Logic synthesis and optimization methods work either globally on the whole logic network or locally on preselected subnetworks. Evolutionary design methods have already been applied to evolve and optimize logic circuits at the global level. In this paper, we propose a new method based on Cartesian genetic programming (CGP) as a local area optimizer in combinational logic networks. First, a subcircuit is extracted from a complex circuit, then the subcircuit is optimized by CGP and finally the optimized subcircuit replaces the original one. The procedure is repeated until a termination criterion is satisfied. We present a performance comparison of local and global evolutionary optimization methods with a conventional approach based on ABC and analyze these methods using differently pre-optimized benchmark circuits. If a sufficient time is available, the proposed locally optimizing CGP gives better results than other locally operating methods reported in the literature; however, its performance is significantly worse than the evolutionary global optimization.**

I. Introduction

By *logic synthesis* we understand a process transforming a behavioral circuit description to a logic network, i.e. network of gates. This logic network is then *optimized* in order to obtain 'better' representation (under given criteria) of the same logic behavior. There are two main classes of synthesis and optimization methods. While *global* methods operate on the whole network, *local* methods are focused on improving preselected subnetworks. Local methods are typically capable of a small reduction of the network size in a very short time. However, by applying local algorithms many times, the scope of changes is no longer local [1]. Global and local methods are combined in commercial logic synthesis systems.

Despite the increasing quality of synthesis and optimization algorithms, numerous works have shown that these algorithms generate far from optimum solutions for some types of circuits [2], [3]. This is valid not only for open academia tools [4], [5] which are typically used for research purposes, but also for commercial software. One of the reasons is that all solutions reachable by the conventional methods are constrained by a very restrictive set of transformations which are allowed for a given circuit representation. For example, an expression $y = \overline{a}b \vee a\overline{b}$ (i.e. a 5 gate circuit) will never be transformed to an obvious form $y = a \oplus b$ (i.e. a single gate circuit) when the Sum-of-Products (SoP) representation is utilized, because the exclusive-or ($\oplus$) decomposition is not supported in SoP. On the other hand, the fast processing, good scalability and theoretical background are the main advantages of the conventional methods.

In order to perform unconstrained and unbiased circuit optimization and find more compact circuit implementations, *evolutionary* logic synthesis and optimization methods were introduced in recent years [6], [7], [8]. For the cost of runtime, evolutionary algorithms were able to improve some results of conventional synthesis methods. However, their scalability and time overhead remain a big issue. As these evolutionary optimization algorithms do not usually distinguish the *logic synthesis* from the *logic optimization*, we will use these terms interchangeably in this paper.

It is proposed in this paper to apply *Cartesian genetic programming* (CGP), a very popular method of the evolutionary circuit design [9], as a *local optimizer* of combinational logic networks. The method iteratively identifies subcircuits, which are extracted from a complex circuit. The subcircuits are then optimized by CGP and the optimized subcircuits replace the original ones. The main advantage of locally working CGP would consist in its applicability to very large circuits, where the global evolutionary optimization might fail. As CGP is intended to iteratively optimize small, randomly chosen subcircuits, the scalability problems known from the CGP-based circuit evolution will be eliminated. This work should be understood as a complementary approach to the already existing CGP-based global optimization methods developed for combinational circuits [7], [10].

We are primarily interested in minimizing the cost (network nodes count or area) which is an important task not only in the digital circuit design flow, but also in other applications such as query optimization. Another issue, which this paper is focused on, is exploring to what extent the logic optimization process is influenced by the level of pre-optimization of the original network. Hence we will compare global and local CGP-based optimization algorithms using differently pre-optimized benchmark circuits.

In summary, the main contributions of this paper are as follows: (1) We propose a new method for local evolutionary optimization of logic networks. (2) We give a performance comparison of local and global evolutionary optimization methods with a conventional approach based on ABC. (3) We analyze these methods using differently pre-optimized benchmark circuits.

The rest of the paper is organized as follows. Section II surveys relevant research. After introducing CGP in Section III, the proposed methods are presented in Section IV. Sections V, VI and VII contain the experimental setup, results and discussion. Conclusions are given in Section VIII.

978-1-4799-1488-3/14/$31.00 ©2014 IEEE

II. Previous Work

A. Logic Synthesis and Optimization

Various circuit representations such as Sum-of-Products and Binary Decision Diagrams (BDD) have been utilized in logic synthesis and optimization algorithms. The current state-of-the-art (academic) tool for logic synthesis and optimization is ABC which represents digital circuits using And-Inverter Graphs (AIG) [4]. An AIG is a network whose node is either a primary input or a 2-input AND, and any edge can contain a negation. An AIG is structurally hashed to ensure uniqueness of the nodes. AIGs are more scalable and uniform than previous representations and allow applying numerous optimization and mapping algorithms upon them directly. The logic synthesis and optimization process has a form of a script which contains various commands (*balance*, *rewrite*, *refactor* etc.) that are sequentially applied on the initial structure. In order to improve the result of optimization, the whole script can be applied *iteratively*. Another option is to iteratively call a subset of commands. As the whole logic network is being processed, we can speak about *global* (re)synthesis and optimization.

B. Rewriting and Local Optimization

On the other hand, *local* methods operate on a subnetwork of the whole logic network. By *rewriting* we mean rule-based methods which replace certain subnetworks by their optimized pre-computed versions. These subnetworks are called *cuts*. A cut of a node n is a set C of nodes such that any path from a primary input to n must pass through at least one node in C. A cut C is K-feasible if $|C| \leq K$. A cut C is called K-input cut if $|C| = K$. Current implementations utilize 4-input or 5-input cuts [11], [1]. While it is possible to store all 2^{16} pre-computed solutions (canonical forms and transformations) for 4-input cuts, the space for storing 2^{32} solutions is not available on a common computer when 5-input cuts are considered. Hence, for example, a Boolean matcher has to be employed to calculate corresponding solutions.

The original paper, which introduced 4-input cuts into ABC, reported 13% (7%, respectively) average area improvement after mapping the network on the standard cells (FPGA LUTs, respectively) with respect to *unoptimized* circuits [1]. Adding a new AIG rewriting algorithm which supported 5-input cuts to the ABC synthesis tool led to 5.57% area reduction on average for already *heavily optimized* circuits [11].

The idea of local optimization has been generalized in [12], [13]. Instead of 4- or 5-input cuts (i.e. *single-output* networks), an arbitrary subcircuit is chosen and optimized using a standard flow for global synthesis and optimization. The result of the optimization then replaces the original subcircuit. The process can be iterated for a series of randomly chosen subcircuits. The authors of [12], [13] proposed two methods for the subcircuit selection. The Random Extraction algorithm randomly adds nodes to the initial randomly selected node, while keeping the selected network connected. The algorithm is parameterized by the number of gates of the subcircuit. The second algorithm looks for the most connected subcircuit. It starts by selecting a pivot node. Then the nodes reachable in a given distance (radius) from the pivot are connected to the subcircuit. On a representative collection of benchmark circuits the most significant average improvement of 9% (w.r.t. circuits heavily optimized by ABC) was obtained for window sizes of 90% of the original circuit size. However, if the window size is 15%, the average improvement is only 2.5%. The best results were obtained for radii ranging from 5 to 7.

C. Evolutionary Circuit Design

Inspired by a seminal work of Higuchi and his collaborators [14], Miller et al. showed that CGP can be used to evolve new implementations of adders and multipliers from scratch and optimize their area [15], [6]. The average improvement of 18% gates was obtained for small (up to 4-bit) multipliers over conventional implementations. However, the method is not scalable because it requires testing all possible assignments to the inputs in each call of the fitness function and searching very hard search spaces. Despite a considerable effort, as reported, for example, in [16], [8], [17], no significant improvement in the scalability has been obtained.

In order to optimize complex circuits, the fitness function was later redesigned to perform functional equivalence checking by means of a SAT solver instead of testing all possible input combinations [7]. This method led to a 25% area improvement with respect to the circuits heavily optimized by ABC [10]. As far as we know, no local evolutionary circuit resynthesis/optimization method has been proposed so far.

III. Cartesian Genetic Programming

CGP is a form of genetic programming which has been intensively studied for digital circuit synthesis and optimization. It was shown that CGP can design and optimize various digital circuits [9] and improve results of conventional circuit synthesis and optimization tools [10]. In this work, we will utilize the standard CGP for combinational circuit evolution.

A. Circuit Representation

A candidate circuit is modeled by means of a directed acyclic graph using an array of processing nodes (gates) arranged in n_c columns and n_r rows. A set of functions implemented by processing nodes will be denoted Γ. The circuit utilizes n_i primary inputs and n_o primary outputs.

Primary inputs and processing node outputs are labeled $0, 1, \ldots, n_i - 1$ and $n_i, n_i + 1, \ldots, n_i + n_c.n_r - 1$, respectively. Each node input can be connected either to the output of a node placed in previous l columns or to one of the primary circuit inputs. The l parameter is called the l-back parameter and influences the level of connectivity in candidate circuits.

A candidate solution consisting of two-input nodes is represented in the chromosome by $n_c \cdot n_r$ triplets $(x_1, x_2, \underline{\psi})$ determining for each processing node its function $\underline{\psi}$, and addresses of nodes x_1 and x_2 which its inputs are connected to. The last part of the chromosome contains n_o integers specifying the nodes where the primary outputs are connected to. Figure 1 gives an example.

One important feature of CGP is that not all nodes (gates) have to be included in the phenotype (see gate 6 in Fig. 1). The CGP encoding is redundant which, according to some studies [18], enables to improve the quality of search. Hence it is useful to support more nodes in the genotype than it

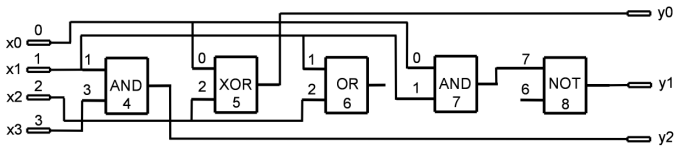

Fig. 1. A candidate circuit represented by CGP with parameters: $n_i = 4, n_o = 3$, $n_c = 5$, $n_r = 1$, $l = 4$, $\Gamma = \{0^{AND}, 1^{OR}, 2^{XOR}, 3^{NOT}\}$. Chromosome: 1, 3, 0; 0, 2, 2; 1, 2, 1; 0, 1, 0; 7, 6, 3; 5, 8, 4.

Algorithm 1: CGP

Input: CGP parameters, fitness function
Output: The highest scored individual p and its fitness

```
P ← randomly generate parent p and its λ offspring;
EvaluatePopulation(P);
while ⟨terminating condition not satisfied⟩ do
    α ← highest-scored-individual(P);
    if fitness(α) ≥ fitness(p) then
        p ← α;
    P ← create λ offspring of p using mutation;
    EvaluatePopulation(P);
return p, fitness(p);
```

might be needed for solving a given problem. The redundancy coefficient ρ quantifies this level of redundancy and $\rho \cdot n_c$ is then the actual node count (because $n_r = 1$ and $l = n_c$ is usually used in order to maximize inter-gate connection options).

B. Fitness Function

CGP can start either with a randomly generated initial population or be seeded by fully functional solution(s). In the former case, the fitness function is a two-phase process. Firstly, the circuit functionality is determined as $f_1 = b$, where b is the number of correct output bits obtained as the response for all possible assignments to the inputs. After reaching a fully functional solution ($b = b_{max} = n_o 2^{n_i}$) or when CGP is seeded by fully functional designs, the second phase is initiated in which the circuit size (or other objectives) can be optimized.

Because the evaluation time depends exponentially on the number of primary inputs, the approach is not scalable. A method capable of optimizing complex circuits will be described in Section IV-B.

C. Search Algorithm

CGP employs a (1 + λ) evolution strategy whose pseudocode is given in Algorithm 1. This search method is based on a point mutation operator which modifies h randomly selected genes (integers) of the parent circuit. Crossover is not utilized as explained in [9]. The termination criterion depends on a particular application.

IV. Local and Global Evolutionary Optimization

Local and global CGP-based optimization algoritms will be proposed in this section. Let circuit G be the subject of optimization. It is supposed that G has already been heavily optimized by ABC.

Algorithm 2: Subcircuit extraction

Input: Circuit G, maximum (sg^{max}) and minimum (sg^{min}) number of subcircuit gates, maximum (si^{max}) and minimum (si^{min}) number of subcircuit inputs
Output: Decomposition of G into a set of subcircuits S

```
S ← ∅;
X ← {all single-gate subcircuits of G};
while |X| > 0 do
    select s ∈ X randomly;
    E ← Expand(s, sg^max, sg^min, si^max, si^min);
    X ← X \ {all gates of E};
    S ← S ∪ E;
return S;
```

A. Local Optimization

The local optimization is performed over subcircuits of G which are obtained using Algorithm 2. Once a subcircuit is extracted, its truth table is computed and used in the fitness function which evaluates the subcircuit response to all possible input combinations. CGP attempts to minimize the number of gates of the subcircuit. When a predefined number of generations is exhausted, the highest-scored circuit replaces the original subcircuit in G. This subcircuit selection and optimization procedure is repeated when a given time or iteration count is not exhausted.

Algorithm 2 firstly creates a set X of single-gate subcircuits of G. A randomly chosen subcircuit $s \in X$ is then expanded by the *Expand* procedure in order to enlarge s as much as possible. This procedure iteratively includes neighboring gates of s from X into the subcircuit E. At the same time the *Expand* procedure ensures that the number of gates of E will stay between sg^{min} and sg^{max} and the number of inputs between si^{min} and si^{max}. No constraints are put on the number of outputs. Gates used in E are then removed from X and if X is still a nonempty set another iteration is performed. Note that each gate is assigned just to one subcircuit of G which allows for a parallel optimization of subcircuits.

Example 1: The basic steps of the proposed local optimization procedure are demonstrated on a circuit G (Fig. 2) whose one of possible decompositions (for parameters $sg^{max} = 8$ and $si^{max} = 4$) is depicted in Fig. 3. There are three subcircuits in G, marked blue, green and light grey. The dark grey I/O blocks represent auxiliary separators of these subcircuits. Fig. 4 shows that the blue subcircuit and the green subcircuit were later optimized by CGP. The single OR gate subcircuit (light gray) remains unmodified. Putting the optimized subcircuits and the single gate subcircuit together forms a new circuit G' which is functionally equivalent with G. Figure 5 shows next step of the optimization – a randomly chosen decomposition of G' into two subcircuits (light grey and blue). After optimizing the blue subcircuit by CGP, the resulting circuit is shown in Fig. 6.

B. Global Optimization

The global CGP optimization method is applied on the whole circuit. Because it is impossible to evaluate complex

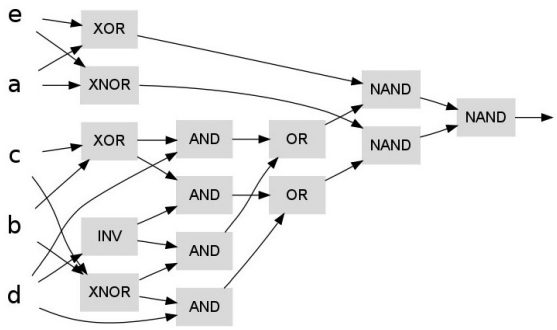

Fig. 2. Example 1-1: Initial circuit G

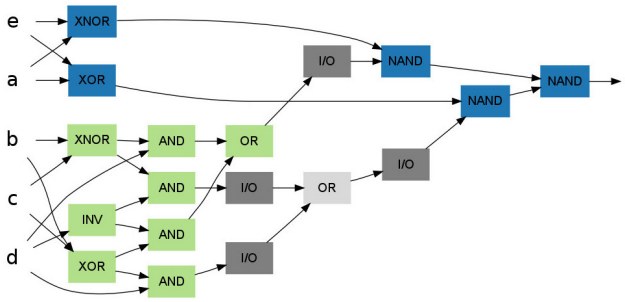

Fig. 3. Example 1-2: Decomposition of G according to Alg. 2.

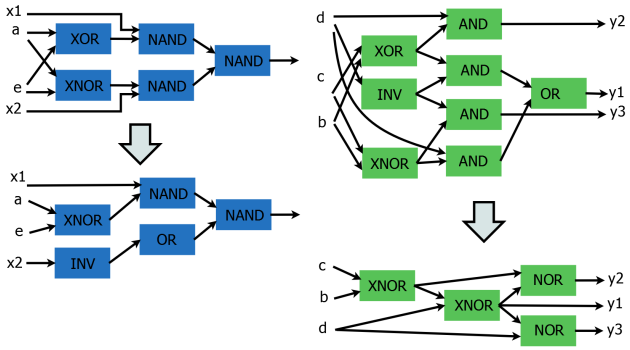

Fig. 4. Example 1-3: Subcircuits of G optimized by CGP.

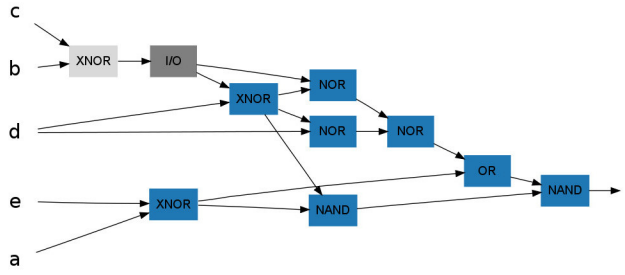

Fig. 5. Example 1-4: Optimized subcircuits are merged and another decomposition is applied.

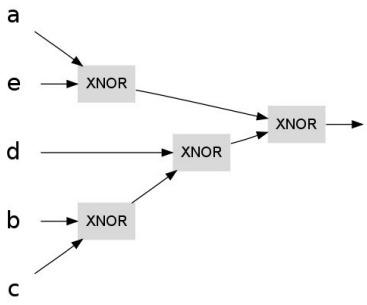

Fig. 6. Example 1-5: Resulting circuit.

candidate circuits by applying all possible input combinations, a formal functional equivalence checking algorithm is employed in order to determine whether a candidate circuit (created by mutation of the parent circuit) is functionally equivalent with the original circuit G. Functionally incorrect candidate circuits are discarded. In the case that the candidate circuit is fully functional then its fitness value is given by the number of gates (or the area) and the goal is to minimize the number of gates (or the area).

Since the satisfiability (SAT) solvers were significantly improved during last few years, the SAT-based equivalence checking was included into CGP and implemented in [7]. The idea is to create an auxiliary circuit H which is composed of the circuits to be checked (circuit G and candidate circuit) and a set of XOR gates connected to the corresponding outputs of the circuits to be checked. By means of the XOR gates it is possible to identify whether there is an input assignment (circuit G and candidate circuit share the inputs of H) for which the corresponding outputs provide different values. Circuit H is transformed into one Boolean formula in conjunctive normal form (CNF) and submitted to the SAT solver. The formula is unsatisfiable if and only if the circuits are functionally equivalent [19]. The transformation of H to CNF is conducted gate by gate using the Tseitin's algorithm [20].

In order to shorten the decision time, various methods can be applied to reduce the number of clauses for the SAT solver. Authors of [10] suggest to utilize knowledge of genes which have been modified by the mutation operator to calculate a 'difference' between the parent individual and its offspring, and to submit only the 'difference' to the SAT solver.

V. Experimental Setup

A. Common settings

The basic setting of CGP parameters which is valid for both local as well as global optimization is as follows: $\Gamma = \{\text{NOT}, \text{AND}, \text{OR}, \text{XOR}, \text{NAND}, \text{NOR}, \text{XNOR}\}$, $l = N_g$, $n_c = \rho.N_g$ and $n_r = 1$, where N_g is the number of gates of G (global optimization) or a subcircuit (local optimization). The role of other parameters will further be investigated in Section VI-A.

In order to reflect different implementation costs of gates at the transistor level, we will report the *relative area* which will be computed using weighted costs of gates. Table I shows the weights of gates with respect to the weight of a single NAND gate (according to [21]).

TABLE I. RELATIVE IMPLEMENTATION COST W.R.T. THE NAND GATE

Gate	Weight
NAND, NOR	1.00
AND, OR	1.33
XNOR	1.66
XOR	2.00
NOT	0.67

The MiniSAT 2 (version 070721) [22] was used as a SAT solver in the CGP implementation which calls the functional equivalence checking algorithm in the fitness function.

The experiments were carried out on a cluster consisting of Intel Xeon X5670@2.4 GHz processors and AMD FX-8120@2.8G processors.

B. Benchmark circuits

The proposed evolutionary optimization methods will be applied on 15 circuits from the LGSynth93 benchmark set. In order to show that the methods can improve results of a conventional approach, the benchmark circuits (.pla files) were pre-optimized. Firstly, 100 iterations of the following ABC script (according to [12]) were performed on each circuit to obtain the first benchmark set (BSA):
fraig_store; resyn
fraig_store; resyn2
fraig_store; resyn2rs
fraig_store; share
fraig_store; fraig_restore
map
The second benchmark set (BSB) contains the same circuits as BSA, but the quality of optimization was intentionally set lower just by removing the *share* command from the previous script. By means of BSB we will investigate the impact of using far from optimum circuits on the overall quality of the local as well as global optimization. Table II shows the number of primary inputs (PI) and outputs (PO) and the relative area of the benchmark circuits. The average area increment of BSB with respect to BSA is 40%.

VI. RESULTS

This section deals with experimental searching for the most suitable parameters of the local CGP optimizer. The proposed local as well as global optimization algorithms are then evaluated using the most suitable setting on benchmark sets BSA and BSB. If not explicitly stated, all results will be given for BSA circuits in this sections.

For the sake of clarity we have to emphasize that by *CGP run* we mean a single run of CGP on a particular subcircuit and by *LO run* we mean a single run of the whole local optimizer which can internally call multiple CGP runs for different subcircuits.

A. Initial Tests of Local Optimization

Using several circuits and multiple LO runs, we statistically analyzed the impact of parameters λ (the number of offspring), h (the number of mutations) and the subcircuit size on the overall performance. Setting $\rho = 2.5$ was fixed after several experiments (not discussed in the paper). Because of the page limit we will present just selected results, mainly for apex2 circuit.

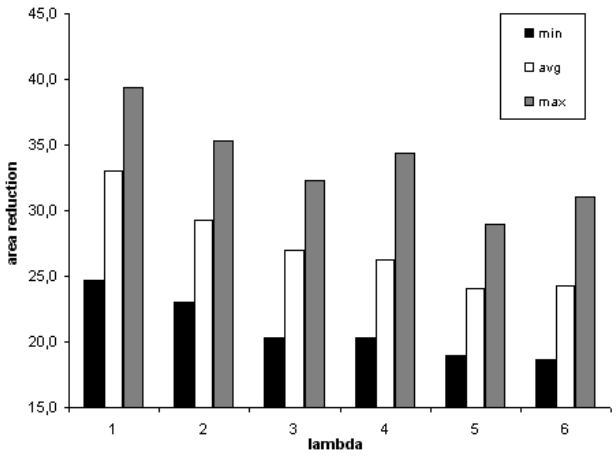

Fig. 7. Total area reduction obtained with various 'lambda' (λ) for apex2.

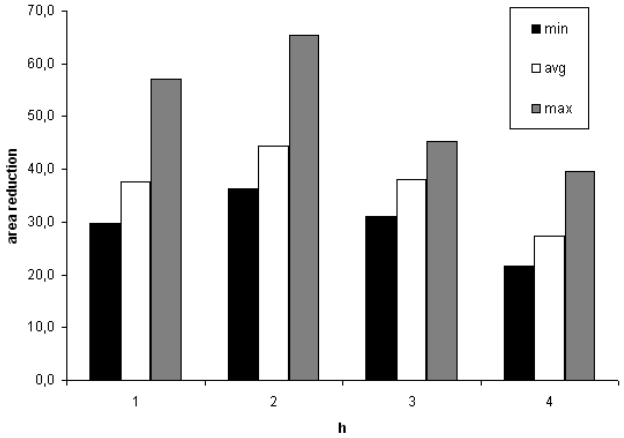

Fig. 8. Total area reduction obtained with a given number of mutations h for apex2.

Figure 7 shows that in the case of apex2, the most significant total area reduction is achieved when $\lambda = 1$. These results were obtained from 50 independent 30-min. LO runs with parameters $h = 1$, $\rho = 2.5$, $sg^{min} = 8$, $sg^{max} = 20$, $si^{min} = 1$ and $si^{max} = 10$. Only 1 CGP run with $g_{max} = 2 \cdot 10^5$ generations was allowed for a subcircuit.

In another experiment, two mutations per chromosome ($h = 2$) were identified as the most advantageous setting of CGP (Fig. 8) when $\lambda = 1$, $g_{max} = 15 \cdot 10^4$ (5 CGP runs allowed per a subcircuit), $\rho = 2.5$, $sg^{min} = 8$, $sg^{max} = 20$, $si^{min} = 3$ and $si^{max} = 9$ are kept unmodified in the experiments.

Then we analyzed the trade off between the number of CGP runs and generation count when a given time is available for the optimization of a single subcircuit. Three scenarios were statistically evaluated for four circuits (sqrt8, apex2, misex2 and cps):

- 16R: 16 runs, 10^5 generations each

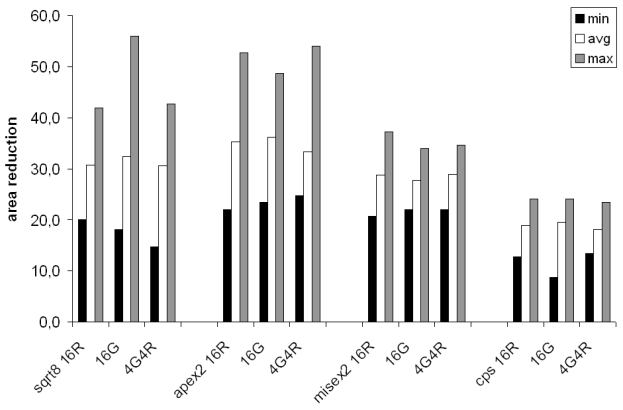

Fig. 9. Total area reduction obtained for four circuits using various settings of the number of independent CGP runs and generation counts

- 16G: 1 run, $16 \cdot 10^5$ generations
- 4G4R: 4 runs, $4 \cdot 10^5$ generations each

Fig. 9 indicates that the scenarios give very similar performance. These results were obtained from twenty 1-hour independent LO runs working with parameters $\lambda = 1$, $h = 2$, $sg^{min} = 8$, $sg^{max} = 40$, $si^{min} = 3$ and $si^{max} = 10$.

Logic networks optimized by the locally working CGP have to be relatively small in order to ensure a reasonable time of evolution. Hence we set the maximum subcircuit size sg^{max} to be 50 gates and allowed $si^{min} = 1$. Figure 10 shows the impact of two key parameters of subcircuits, the minimum number of gates sg^{min} and the maximum number of inputs si^{max}, on the total area reduction of apex2. On average the most useful setting for apex2 is $si^{max} = 10$ and $sg^{min} = 15$. These values were computed from 25 independent 1-hour LO runs, where $\lambda = 1$, $h = 2$ and $g_{max} = 5.10^5$ (1 CGP run allowed for a subcircuit).

B. Local Optimization

The proposed locally optimizing CGP was tested on both benchmark sets. We used the setup which was identified as the most advantageous during the initial experiments: $\lambda = 1$, $h = 2$, $\rho = 2.5$, 1 CGP run with $g_{max} = 10^6$ for a subcircuit, $sg^{min} = 10$, $sg^{max} = 50$, $si^{min} = 1$, and $si^{max} = 10$. As we prefer the quality of results over the time of optimization and we are interested in limits of the optimization methods, we allowed 12-hour LO runs, which is also a typical setup used for the global optimization in [10], [21]. Each experiment was repeated 5 times. Table II summarizes the average area reduction obtained by the local CGP-based optimization for both benchmark sets. A typical progress of additional 35 independent LO runs is given for apex2 in Fig. 11.

C. Global Optimization

The global resynthesis and optimization was implemented according to [10], [21] and executed using suggested parameters, i.e. $\lambda = 2$, $l = n_c = N_g.\rho$, $h = 2$, $\rho = 1$ and $n_r = 1$. The average area reduction obtained from ten 12-hour CGP runs is given and compared with the local optimizer in Table II. Convergence curves are shown for apex2 circuit in Fig. 12.

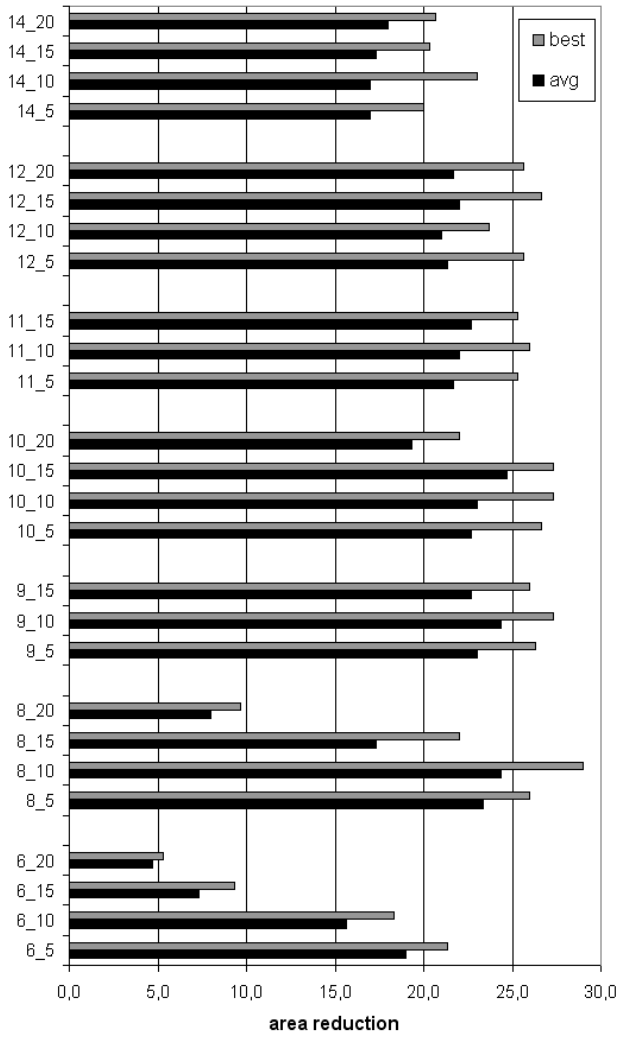

Fig. 10. Total area reduction obtained with various settings of subcircuit parameters (y-axis: $si^{max}\_sg^{min}$) for apex2.

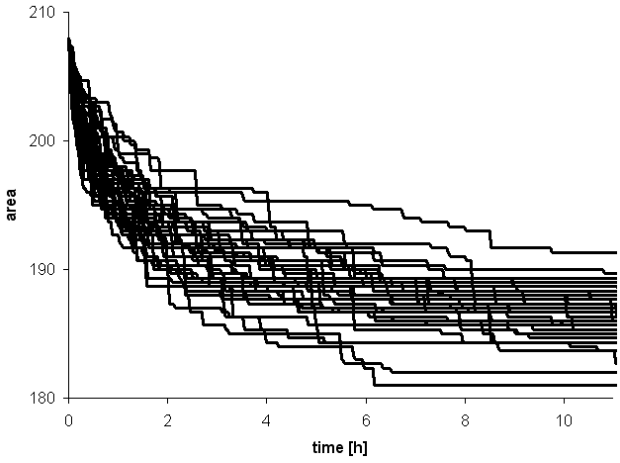

Fig. 11. Local optimization: apex2 convergence curves.

TABLE II. AVERAGE RELATIVE AREA REDUCTION FOR BENCHMARK SETS (BSA AND BSB) AND GLOBAL AND LOCAL CGP OPTIMIZER

Circuit	PI	PO	Area BSA	Area reduct. [%] Global	Area reduct. [%] Local	Area BSB	Area reduct. [%] Global	Area reduct. [%] Local
alu4	14	8	866.0	85.8	2.7	1113.7	89.4	2.7
apex1	45	45	1710.7	27.9	1.7	2125.3	22.3	2.4
apex2	39	3	208.0	28.4	10.4	273.3	47.5	13.4
apex3	54	50	1646.0	22.3	1.3	1795.7	21.1	0.9
apex5	117	88	763.3	3.5	8.3	889.3	6.4	5.6
b12	15	9	63.0	18.1	19.0	62.7	8.1	16.4
cordic	23	2	55.0	12.2	21.7	59.7	20.2	23.5
duke2	22	29	333.0	12.8	7.0	531.0	27.0	13.5
e64	65	65	374.7	22.2	25.4	532.7	37.1	20.9
misex3c	14	14	522.3	24.3	3.1	607.3	29.4	2.7
pdc	16	40	548.3	46.3	6.6	762.3	52.3	7.1
spla	16	46	554.0	47.6	5.6	868.0	60.2	6.8
t481	16	1	25.0	2.4	5.3	64.3	62.4	63.2
table5	17	15	1017.3	22.4	4.4	1505.7	35.9	2.9
vg2	25	8	96.7	16.9	20.2	165.0	38.4	49.5
Average			585.6	26.2%	9.5%	757.1	37.2%	15.4%

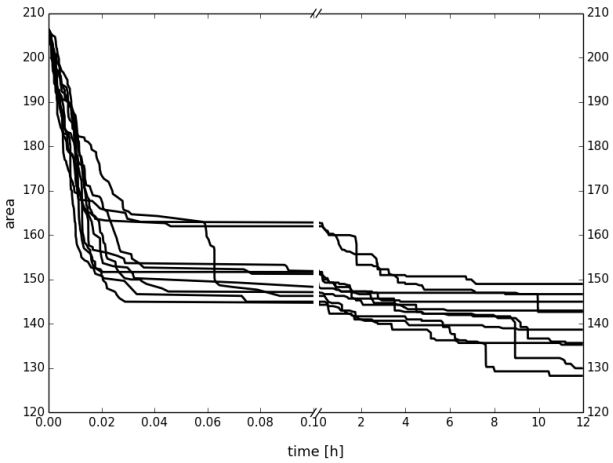

Fig. 12. Global optimization: apex2 convergence curves.

Contrasted to the local optimizer, a very fast and significant progress can be seen in the first two minutes.

VII. DISCUSSION

The proposed *local* CGP-based optimizer led to a 9.5% area reduction on average in comparison with the conventional (global) optimization employing 100 iterations of ABC. The minimum and maximum reduction was 1.3% (apex3) and 25.4% (e64). Only a small reduction of 2.7% was obtained for the alu4 benchmark while the global CGP-based optimization algorithm achieved 85.8%. It seems that the local method is insufficient for optimizing classic hard-to-synthesize circuits such as alu4. Fig 11 shows that even 12 hour runs are not enough to ensure stable results.

The authors of [13] reported a 9% average reduction for subcircuit size of 90% of the original circuit size (and stable results ensured by 5000 iterations of their algorithm). In our case, the maximum subcircuit size was 50 gates (28% of the original circuit size on average). However, for this subcircuit size, the authors of [13] obtained only a 6% area reduction on average. This comparison could be influenced by the fact that the authors of [13] utilized a more comprehensive benchmark set and optimized the number of gates instead of relative area. Both the proposed method and paper [13] give better results (the area reduction) than a common rewriting [11], [1].

Within the same time budget, we obtained significantly better results (26.2% average area reduction) using *globally optimizing CGP* which corresponds with results of previous studies [10], [21]. It is interesting to observe that subcircuit sizes 90–100% (which is, in fact, a global optimization!) led to almost a zero area reduction in the local optimizer from [13]. However, in 6 cases, the local CGP optimizer provided better average results than the global one.

The level of (pre)optimization of a circuit entering the proposed optimization methods significantly influences the quality of result. In order to illustratively present and summarize all results, we compared relations between the areas of original circuits (BSA, BSB) and circuits optimized by the proposed methods. Figure 13 shows the *average area increment* of circuits taken from the benchmark sets or resulting from the optimization process in comparison with the most compact implementations (which could be considered as an 'optimum' solution in this paper) that we have obtained from all the reported experiments. While circuits optimized by the globally working CGP seeded by BSA are 7% larger on average than the 'optimal' solutions, a 121% increment was obtained for locally optimizing CGP over BSB circuits. On the other hand, the locally optimizing CGP was still able to improve results of the globally working conventional ABC system.

Figure 13 also shows that a 14% improvement in area can be obtained when the global CGP optimizer is seeded by smaller benchmarks (BSA). In other words, the results differ only in 14% for considered seeding circuits and both results are relatively close to the 'optimal solution'. The local optimization is far more sensitive to the quality of the initial circuits. A 49% improvement in area can be observed for seeding the local CGP optimizer by BSA contrasted to BSB. Only a 15% improvement in area was obtained when the local CGP is applied on well pre-optimized circuits, which suggests that the local method is more useful when the original circuits are not pre-optimized.

Although the globally working CGP with the SAT solver in the fitness function provided the best results, it would be hardly applicable for really complex problem instances consisting of millions of gates because of the scalability limits of current SAT solvers. We expect that the proposed local method could be useful even for these complex instances because the optimization is always performed on very small subcircuits.

VIII. CONCLUSIONS

In this paper, we analyzed the impact of local and global evolutionary optimization on the area of combinational circuits under very relaxed optimization time constraints and without considering other circuit parameters. We also experimentally found the most useful setting of the parameters of the local CGP-based optimizer. The proposed locally optimizing CGP gives better results than relevant approaches from the literature. Globally optimizing CGP outperformed (on average) any local optimizer under our constraints; however, the local CGP optimizer was able to provide better results (on average) for 6 benchmark circuits.

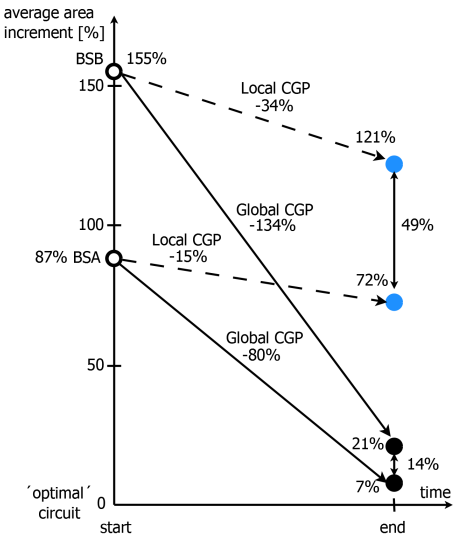

Fig. 13. Relative average area increment in comparison with the most compact implementation

Our future work will primarily be devoted to reducing the computational requirements of the proposed methods, considering more design objectives and evaluating the results on more comprehensive sets of circuits.

ACKNOWLEDGMENT

This work was supported by the Czech science foundation project 14-04197S – Advanced Methods for Evolutionary Design of Complex Digital Circuits.

REFERENCES

[1] A. Mishchenko, S. Chatterjee, and R. Brayton, "Dag-aware AIG rewriting a fresh look at combinational logic synthesis," in *Proceedings of the 43rd annual Design Automation Conference*, ser. DAC '06. ACM, 2006, pp. 532–535.

[2] P. Fiser and J. Schmidt, "Small but nasty logic synthesis examples," in *Proc. 8th Int. Workshop on Boolean Problems*, 2008, pp. 183–190.

[3] J. Cong and K. Minkovich, "Optimality Study of Logic Synthesis for LUT-Based FPGAs," *IEEE Transactions on Computer-aided Design of Integrated Circuits and Systems*, vol. 26, no. 2, pp. 230–239, 2007.

[4] A. Mishchenko, "ABC: A system for sequential synthesis and verification, Berkley logic synthesis and verification group," 2012. [Online]. Available: http://www.eecs.berkeley.edu/~ alanmi/abc/

[5] E. M. Sentovich, "Sis: A system for sequential circuit synthesis, University of California, Berkeley," 1992.

[6] V. Vassilev, D. Job, and J. F. Miller, "Towards the Automatic Design of More Efficient Digital Circuits," in *Proc. of the 2nd NASA/DoD Workshop on Evolvable Hardware*. IEEE, 2000, pp. 151–160.

[7] Z. Vasicek and L. Sekanina, "Formal verification of candidate solutions for post-synthesis evolutionary optimization in evolvable hardware," *Genetic Programming and Evolvable Machines*, vol. 12, no. 3, pp. 305–327, 2011.

[8] A. P. Shanthi and R. Parthasarathi, "Practical and scalable evolution of digital circuits," *Applied Soft Computing*, vol. 9, no. 2, pp. 618–624, 2009.

[9] J. F. Miller, *Cartesian Genetic Programming*. Springer-Verlag, 2011.

[10] Z. Vasicek and L. Sekanina, "A global postsynthesis optimization method for combinational circuits," in *Proc. of the Design, Automation and Test in Europe, DATE*. EDAA, 2011, pp. 1525–1528.

[11] N. Li and E. Dubrova, "AIG rewriting using 5-input cuts," in *Proceedings of the 2011 IEEE 29th International Conference on Computer Design*, ser. ICCD '11. IEEE Computer Society, 2011, pp. 429–430.

[12] P. Fiser and J. Schmidt, "It is better to run iterative resynthesis on parts of the circuit," in *Proc. of the 19th International Workshop on Logic and Synthesis*. University of California Irvine, 2010, pp. 17–24.

[13] ——, "Improving the iterative power of resynthesis," in *Proc. of the 15th IEEE Int. Symp. on Design and Diagnostics of Electronic Circuits Systems (DDECS)*. IEEE, 2012, pp. 30–33.

[14] T. Higuchi, T. Niwa, T. Tanaka, H. Iba, H. de Garis, and T. Furuya, "Evolving Hardware with Genetic Learning: A First Step Towards Building a Darwin Machine," in *Proc. of the 2nd International Conference on Simulated Adaptive Behaviour*. MIT Press, 1993, pp. 417–424.

[15] J. F. Miller, D. Job, and V. K. Vassilev, "Principles in the Evolutionary Design of Digital Circuits – Part I," *Genetic Programming and Evolvable Machines*, vol. 1, no. 1, pp. 8–35, 2000.

[16] E. Stomeo, T. Kalganova, and C. Lambert, "Generalized disjunction decomposition for evolvable hardware," *IEEE Transaction Systems, Man and Cybernetics, Part B*, vol. 36, no. 5, pp. 1024–1043, 2006.

[17] T. Aoki, N. Homma, and T. Higuchi, "Evolutionary Synthesis of Arithmetic Circuit Structures," *Artificial Intelligence Review*, vol. 20, no. 3–4, pp. 199–232, 2003.

[18] J. F. Miller and S. L. Smith, "Redundancy and Computational Efficiency in Cartesian Genetic Programming," *IEEE Transactions on Evolutionary Computation*, vol. 10, no. 2, pp. 167–174, 2006.

[19] E. Goldberg, M. Prasad, and R. Brayton, "Using SAT for combinational equivalence checking," in *DATE '01: Proceedings of the conference on Design, automation and test in Europe*. Piscataway, NJ, USA: IEEE Press, 2001, pp. 114–121.

[20] G. S. Tseitin, "On the complexity of derivation in propositional calculus," in *Studies in Constructive Mathematics and Mathematical Logic, Part II*, 1968, pp. 115–125.

[21] Z. Vasicek and L. Sekanina, "On area minimization of complex combinational circuits using cartesian genetic programming," in *2012 IEEE World Congress on Computational Intelligence, WCCI-CEC*. IEEE, 2012, pp. 2379–2386.

[22] N. Een and N. Sorensson, "An extensible SAT-solver," in *Theory and Applications of Satisfiability Testing*, 2004, pp. 333–336. [Online]. Available: http://minisat.se