Evolutionary Approach to Approximate Digital Circuits Design

Zdenek Vasicek and Lukas Sekanina

***Abstract*—In approximate computing, the requirement of perfect functional behavior can be relaxed because some applications are inherently error resilient. Approximate circuits, which fall into the approximate computing paradigm, are designed in such a way that they do not fully implement the logic behavior given by the specification and hence their accuracy can be exchanged for lower area, delay or power consumption. In order to automate the design process, we propose to evolve approximate digital circuits which show a minimal error for a supplied amount of resources. The design process which is based on Cartesian Genetic Programming (CGP) can be repeated many times in order to obtain various tradeoffs between the accuracy and area. A heuristic seeding mechanism is introduced to CGP which allows for improving not only the quality of evolved circuits, but also reducing the time of evolution. The efficiency of the proposed method is evaluated for the gate as well as the functional level evolution. In particular, approximate multipliers and median circuits which show very good parameters in comparison with other available implementations were constructed by means of the proposed method.**

***Index Terms*—Approximate Computing, Cartesian Genetic Programming, Digital circuits, Population Seeding.**

I. Introduction

Approximate computing is a new design paradigm emerging as a response to the never ending need for performance and energy efficiency of computing systems [1]. It exploits the fact that the requirement of perfect functional behavior (i.e. *accuracy*) can be relaxed because some applications are inherently error resilient. The errors are not recognizable as human perception capabilities are limited (e.g. in multimedia applications), no golden solution is available for validation of results (e.g. in data mining applications), or users are willing to accept some inaccuracies (e.g. when the battery of a mobile phone is almost depleted, but at least a basic functionality is still requested). Therefore, this accuracy can be used as a design metric, traded for area, delay, throughput or power consumption.

In approximate computing systems, approximations can be introduced at all design levels, starting from the circuit via the architecture and operating system to programming language. Examples of applications in which the principles of approximate computing are utilized range from inaccurate arithmetic circuits (e.g. adders [2], multipliers [3]) via high-level processing blocks (e.g. image compression [3], discrete cosine transform, finite and infinite impulse response filters [4]) to general purpose approximate computing machines [5] and programming languages [6]. The circuits which are intentionally designed in such a way that the specification is not met in terms of functionality and some savings are expected in terms of energy, performance or area are called *approximate circuits*.

Approximate computing as a field is in an early stage of development and without an established methodology. Approximate circuits have initially been constructed manually, by removing those parts of existing fully functional designs that did not contribute to the result significantly [3]. The current trend is to create general design methods (such as SALSA [4] and SASIMI [7]) capable of constructing approximate circuits which never exceed a predefined error. These '*error-oriented*' approaches, however, represent only one of the possible approaches in order to approximate circuits design.

Evolutionary circuit design techniques were successful in the task of designing a specific class of electronic circuits which has been documented in numerous survey articles (e.g. [8], [9]). The aim of this paper is to show that the approximate circuit design methodology based on principles of evolutionary design can produce efficient and competitive approximate gate-level as well as functional level combinational circuits. Because of the nature of approximate circuits (in fact, partially working circuits are sought) and principles of evolutionary circuit design (evolutionary-based improving of partially working circuits), we expect a synergy effect which could lead to establishing an evolutionary design as a competitive design method for approximate circuits.

In our previous work, we took advantage of the fact that the evolutionary design always provides a partially working solution even when resources needed for constructing a fully functional solution *are not* available [10]. It has to be noted that conventional methods do not usually provide any result when allocated resources are insufficient. As power consumption is often highly correlated with occupied resources, we can evolve a partially working circuit using constrained resources and assume that the circuit's power consumption will be reduced.

This idea is further elaborated as follows. Let n be the (minimum) number of gates required to implement a given logic circuit. The approximate circuit is created by means of randomly seeded Cartesian Genetic Programming (CGP) whose objective is to minimize a given error function and which can use up to m gates ($m < n$). If various other approximations are requested, CGP is executed multiple times with a gradually reduced amount of available gates. The user thus obtains a set of approximate combinational circuits, each of which typically exhibits different tradeoffs between the

Zdenek Vasicek and Lukas Sekanina are with Brno University of Technology, Faculty of Information Technology, IT4Innovations Centre of Excellence, Božetěchova 2, 61266 Brno, Czech Republic (email: vasicek@fit.vutbr.cz, sekanina@fit.vutbr.cz).

functionality and the number of gates. The proposed design approach can be considered as an '*area-oriented*' method because the user can control the used area (and so power consumption) more comfortably than by means of the error-oriented methods. Another important contribution of this article is a new method of seeding the initial population of CGP, which enables us to significantly reduce the time of evolution.

In order to demonstrate a wider applicability of our approach, the proposed method will be evaluated for gate-level as well as functional-level circuits. It should be noted that systematic methods have only been introduced for the bit (gate) level design of approximate circuits. Hence two case studies will be reported: the design of approximate combinational parallel multipliers (the gate level) and the design of a median computing circuit (the functional level). We will study the tradeoff between the correctness, area and power consumption for 2-bit, 3-bit and 4-bit multipliers. These small multipliers will be used as building blocks for larger multipliers, and again, the correctness will be traded for power consumption and area. The median computing circuit is a key component for median filters in image processing. It is expected that approximate median circuits can lead to a significant area reduction while the error of filtering remains small. In summary, the key contributions of this article are as follows:

- We propose a new methodology for approximate circuit design which exploits the area-oriented design approach and CGP seeded by heuristically created approximate circuits.
- We propose to extend the concept of approximate circuit evolution from the gate level to the functional level.
- We present novel implementations of approximate combinational multipliers created by CGP. These multipliers show very good parameters in comparison with similar multipliers reported in the literature.
- We present novel implementations of approximate median circuits created by CGP.

The rest of the paper is organized as follows. Section II surveys relevant research in areas of approximate circuits and evolutionary circuit design. The proposed design methodology is introduced in Section III. An experimental framework is presented together with obtained results in Section IV. After discussing the impact of this work, conclusions are given in Section V.

II. Related Work

Only a few papers on evolutionary circuit design have up to now directly or indirectly addressed the problem of approximate circuit design. Before introducing them in Section II-B we will give an overview of current (conventional) approximate circuit design techniques in Section II-A.

A. Approximate Circuits: Overview

Power consumption reduction is one of the key challenges of the current chip design industry. Conventional approaches to power reduction of digital circuits are applied at all design levels, starting from the architecture via the circuit to the technology [11]. Further reductions can be obtained by approximating the original circuit function by a new one whose implementation is more energy efficient. The requirement on functional equivalence between the specification and implementation is thus relaxed in order to minimize energy consumption, accelerate computations or reduce the area on a chip. The concept of approximate circuits is similar to *probabilistic circuits* which take into account the importance of bits of the circuit's output with respect to the complexity of their implementation [12]. However, approximate computing does not involve assumptions on the stochastic nature of any underlying processes implementing the system [1].

The next subsections will present basic design techniques (over-scaling and functional approximation), systematic design methodologies and error metrics used in approximate circuit designs.

1) Over-scaling: In the case of *over-scaling*, circuits are designed to be working perfectly under a normal environment. However, their energy consumption can be reduced by voltage over-scaling (i.e. using deliberately lower power supply voltage in which the circuit is known to occasionally produce erroneous outputs). Similarly, performance can be increased when the circuit is over-clocked. Timing induced errors are due to the fact that some paths in the circuit fail to meet the delay constraints. The combination of scaling the supply voltage and clock frequency is known as dynamic voltage scaling.

2) Functional Approximation: Functional approximation means that the circuit is designed in such a way that it does not fully implement the logic behavior given by the specification. A simple method is to reduce the precision of computations in the case of arithmetic circuits by ignoring the least significant bits. However, only insignificant area savings can be obtained for some key circuits such as multipliers. Other methods adopt logic synthesis scenarios in which implementations that satisfy the specification almost perfectly are sought, but the amount of resources is significantly reduced (see e.g. [2], [7]).

For example, a two-bit multiplier was manually constructed which consists of 5 gates only and exhibits a delay of $2d$, where d is a unit delay. Its output is correct for 15 out of 16 possible inputs. A usual conventional solution requires 8 gates and exhibits a delay of $3d$. This approximate multiplier has been used in larger approximate multipliers and then employed in approximate image processing applications [3].

3) Systematic Design Methodologies: As the manual redesign is not a universal and efficient method, systematic methods to synthesis of approximate circuits are currently being developed.

The Systematic methodology for Automatic Logic Synthesis of Approximate circuits (SALSA) starts with a RT level description of the exact version of the circuit and an error constraint that specifies the type and amount of error that the implementation can exhibit [4]. The methodology introduces the so-called Q-function which takes the outputs from both the original and approximate circuits and decides if the quality constraints are satisfied. The Q-function outputs a single Boolean value. The SALSA algorithm attempts to modify the approximate circuit with the goal of keeping the output of the Q-function unchanged.

Another systematic approach, Substitute-And-SIMplIfy (SASIMI), tries to identify signal pairs in the circuit that exhibit the same value with a high probability, and substitutes one for the other [7]. These substitutions introduce functional approximations. Unused logic can be eliminated from the circuit which results in area and power savings. The method is combined with technology-level optimizations such as down-sizing of gates (i.e. creating smaller than normally sized gates to reduce power consumption, in exchange for increased delay) on critical paths and voltage over-scaling which results in additional significant area and power savings.

SASIMI and SALSA are very new methods and, unfortunately, are not currently available to the public.

4) Error Metrics: The above methods are error-oriented in the sense that all logic optimizations leading to an approximate solution are constrained by a predefined *error criterion*. The error can be expressed by various metrics such as worst case error, average error, and error probability [13]. The design process has to be repeated when a new error criterion is established.

B. Evolutionary Circuit Design

Recent surveys on evolutionary circuit design (see, e.g. [8], [9]) clearly demonstrate that although some evolved implementations of target circuits can be considered as innovative, the evolutionary design approach fails in producing useful implementations of *complex circuits*. In order to at least partially eliminate this disadvantage, various approaches have been proposed to improve the problem representation and genetic operators (such as functional level representations [14], [15], decomposition [16], and developmental encodings [17]) and accelerate the fitness computation (such as partial evaluation [18], formal functional equivalence checking [19], and phenotype precompilation [20]).

1) Previous Works Related on Approximate Circuits: There are some examples of evolutionary circuit design that could be considered as approximate circuit design. For example, Miller evolved finite impulse response filters at the gate level where functionality was traded for area [21]. In fault tolerance applications, if a critical number of elements is damaged, the original function cannot fully be recovered; however, a partial functionality can be obtained by means of evolutionary design. This concept has been surveyed in [22]. In another research, Kneiper et al. investigated the robustness of evolved classifiers [23]. A classifier system was reported which is able to cope with changing resources at run-time. During optimization, the number of pattern matching elements was modified and its influence on classification accuracy was studied (i.e. there is a tradeoff between the classification accuracy and area).

Thompson's famous evolutionary design of a tone discriminator circuit in the XC6216 FPGA belongs to this class of applications too. Thompson's evolutionary algorithm discovered a tone discriminator requiring significantly less resources than usual solutions would occupy in the same FPGA [24]. Though the evolved discriminator was fully functional, its robustness was limited. Higher sensitivity to fluctuations in the environment (external temperature, power supply voltage) and dependability on a particular piece of FPGA were reported. Hence we can observe a tradeoff between the robustness and the amount of resources in the FPGA.

All these approaches and applications have something in common with approximate circuits. None of them, however, has fully exploited the capability of evolutionary design as a systematic method for an approximate circuit design.

2) Direct Evolution of Approximate Circuits: Finally, this section summarizes our previous work on evolutionary design of approximate circuits.

In [10], we evolved approximate implementations of small combinational circuits (3-bit and 4-bit adders and single output circuits) using randomly seeded CGP operating at the gate level. In order to provide solutions for every possible number of gates, CGP was repeatedly executed with gradually reduced resources available for implementation. The objective was to minimize the mean absolute error with respect to a fully functional circuit. Because the utilized power estimation algorithm (which is embedded into the SIS tool [25]) is very time consuming, it has not been included in the fitness function directly. Power consumption was calculated at the end of evolution for the best evolved approximate circuits.

An inherently *multiobjective approach* to evolutionary design of approximate multiplierless multiple constant multipliers (MCMs) was proposed in [26]. Three design objectives—accuracy, area and delay—were optimized by multiobjective CGP, where the area was inexpensively estimated as the number of utilized components and delay as the number of components along the longest path between the input and the output.

Both approaches utilized randomly generated initial populations which led to relatively time consuming evolutionary runs. Seeding the initial population by suitable pre-generated designs is one contribution of our work reported in the following sections. Another feature is that for circuits from papers [10], [26], we could check in the fitness function their responses for all possible input combinations, which is impossible for complex circuits such as median circuits.

III. Proposed Method

After emphasizing key features of the current approach to approximate circuit design, this section introduces the overall idea of the proposed method, the utilized evolutionary algorithm and the heuristic population seeding procedure.

A. Initial Considerations

Existing systematic approximate circuit synthesis methods (such as [4], [7]) always begin with a fully functional circuit C and a given quality constraint (acceptable error) e. Then C undergoes the "approximating procedure" and an approximate circuit C_1 is generated. It is ensured that the predefined error e is not exceeded by C_1. As the acceptable error (and a corresponding power consumption reduction) can be difficult to define for a given application in advance, the design process is usually repeated for several error values $e_2, e_3, \ldots, e_k$, yielding approximate circuits $C_2, C_3, \ldots, C_k$. The solution

which exhibits the most suitable tradeoff between design objectives is then the resulting approximate circuit. However, the area, power consumption and delay are *not* directly under the control of the "approximating procedure".

This is inherently a multiobjective circuit design problem which *could* be solved by a suitable multiobjective evolutionary algorithm (MOEA); for example, algorithms reported for evolution of conventional circuits in [26], [27] are based on NSGA-II [28]. It is expected that MOEAs will have difficulty with delivering really compact approximate circuits for complex problem instances because:

- Evolutionary design of non-trivial combinational circuits (e.g. 4-bit multipliers) from scratch is a difficult problem. Only a small fraction of runs usually produce a working circuit. The reason is that corresponding fitness landscapes are very rugged [29].
- It is even harder to evolve a working circuit (e.g. 4-bit multiplier) which is better than a conventional design according to a chosen criterion (i.e. the number of gates in our case) [30].
- A reasonably reliable estimate of power consumption which is important for building trustworthy Pareto fronts in MOEA can be very time consuming for complex circuits. For example, while the evaluation of a candidate 4-bit multiplier takes 35 μs, power consumption simulation by SIS requires 0.59 s (average numbers calculated on a 3 GHz processor are given).

Another difficulty lies in the scalability problem of the evolutionary circuit design. In this work, we adopted two approaches: (1) complex approximate median circuits are evolved by means of the functional-level evolution; (2) in the case of gate-level circuits, we focus on arithmetic circuits and adopt the approach introduced in [3] in which relatively small approximate circuits are used as building blocks of complex approximate circuits. In our case, these small approximate circuits are evolved by CGP.

B. Approximate Circuit Evolution

The main features of the proposed area-oriented method which addresses the above mentioned problems are as follows:

(1) The direct control of the resulting area (and possibly power consumption) could be very useful for some application scenarios (e.g. computing with the minimum error for a given power budget in a mobile phone). Hence the proposed method generates approximate circuits as a function of the area rather than the error. This 'area-oriented' approach cannot be accomplished by conventional circuit design tools because they do not provide any solution when available resources are insufficient.

(2) The proposed method works as follows. Let us suppose that P is a procedure capable of creating an approximate version of a fully functional circuit C which consists of n components (gates). P is employed to construct an approximate circuit C_1 using m_1 components with the aim of minimizing the predefined error criterion. This approximation exhibits the error e_1. Similar to error-oriented methods, such as SALSA and SASIMI, the design procedure can be repeated; however, here it is for various number of gates (not for various errors), in order to obtain different tradeoffs among design objectives. Approximate circuits $C_2, C_3, \ldots, C_k$ are then constructed by P wherein $m_2, m_3, \ldots, m_k$ gates are supplied; m_k is the number of gates in the smallest required approximation of C. It is expected that the resulting errors are $e_1 \leq e_2 \leq \ldots \leq e_k$. If m is successively $n-1, n-2, \ldots, 2$, and 1, an approximate circuit is constructed for every possible number of gates.

(3) In order to implement P, from available evolutionary circuit design methods we chose a single-objective CGP which enables the gate as well as functional level evolution [31]. Multiple runs of CGP are performed for a given amount of resources in order to find a circuit which exhibits the smallest possible error. Multiobjective NSGA-II-based CGP [26] will be used for comparative purposes in Section IV.

(4) The following features of the proposed method enable us to accelerate the whole design process:

- The initial population is seeded by approximate circuits (created according to Section III-E) in order to find much better solutions than a randomly seeded CGP.
- Power consumption is computed only for selected best circuits at the end of CGP runs.
- Fitness evaluation exploits the idea of parallel simulation of candidate circuits and circuit translation to the binary machine code [20].
- Multiple runs are executed on a computer cluster (p runs on p processors).

C. Cartesian Genetic Programming

CGP and its various versions are probably the most popular methods for the evolutionary circuit design [30], [31]. In this work, we utilize the standard CGP for combinational circuit evolution with a few modifications as explained in the following paragraphs.

1) Circuit Representation in the Chromosome: A candidate circuit is modeled by means of an array of processing nodes arranged in n_c columns and n_r rows. The processing elements can be either elementary gates or functional level components such as adders, comparators and shifters. The $n_c.n_r$ product is constrained by the maximum number of available nodes in the case of approximate circuit evolution.

The set of functions implemented by processing elements will be denoted Γ. The circuit utilizes n_i primary inputs and n_o primary outputs. All signals are defined over b bits, where $b = 1$ for the gate level evolution.

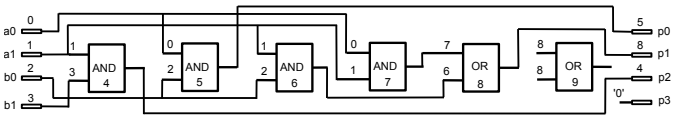

Fig. 1. A candidate 2-bit multiplier, with inputs $b_1 b_0 a_1 a_0$ and outputs $p_3 p_2 p_1 p_0$, represented by CGP with parameters: $n_i = n_o = 4$, $n_c = 6$, $n_r = 1$, $l = 4$, $\Gamma = \{0^{AND}, 1^{OR}\}$. Chromosome: 1, 3, 0; 0, 2, 0; 1, 2, 0; 0, 1, 0; 7, 6, 1; 8, 8, 1; 5, 8, 4, '0'.

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 gate placed in the previous l columns or to one of the primary circuit inputs. A candidate solution consisting of two-input nodes is represented in the chromosome by $n_c.n_r$ 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 either the nodes, where the primary outputs are connected to, or logic constants ('0' and '1'), which can be directly connected to the primary output. The support of logic constants at the primary outputs is crucial for evolving some approximate circuits.

In order to illustrate the CGP encoding in Figure 1, we chose the approximate 5-gate multiplier discussed in Section II-A2. One important feature of CGP is that not all gates have to be included in the phenotype (e.g. gate 9). The CGP encoding is redundant which, according to some studies [32], enables us to improve the quality of the search.

D. Fitness Function

The goal of evolution is to maximize the functionality of approximate circuits whose size is constrained by the $n_c.n_r$ product. The fitness is then defined as *error* to be minimized:

$$f = \sum_{j=1}^{K} |y(j) - t(j)|, \tag{1}$$

where y is candidate circuit's n_o-bit response and t is target response. The number of fitness cases is $K = 2^{n_i}$, because we have to evaluate circuit responses for all possible combinations of operands for arithmetic circuits. This definition of the fitness function is preferred over the Hamming distance based function because a better performance has been reported in in [10].

In the functional level evolution, the design problem is often understood as a symbolic regression problem. Then, K is the number of fitness cases in the training set.

1) Search Algorithm: We will use the $(1+\lambda)$ search method as recommended in [31].

1) The initial population of the size $1 + \lambda$ is created.
2) The fitness function f is called for each candidate circuit.
3) The highest-scored candidate circuit is selected as the new parent. It has to be noted that the previous parent α is never selected as the new parent if there are more individuals with fitness $f(\alpha)$ and $f(\alpha)$ is the best fitness value in a given population [31].
4) By applying a *point mutation*, λ offspring individuals are generated from the parent. In this type of mutation, h genes (integers) undergo a mutation.
5) Steps 2—4 are repeated until the termination condition is not satisfied.

E. Heuristic Population Seeding

Let C be a fully functional circuit consisting of n two-input gates. Let us suppose that CGP has to minimize the error (f) and only up to $n - 1$ gates can be utilized. The proposed heuristic for seeding the initial population is based on a local search and works as follows.

Every single gate of C is independently replaced by a wire connection (the upper input is connected to the output of the gate), which results in n approximate circuits consisting of $n - 1$ gates. The fitness values are then calculated for all n circuits. The whole procedure is repeated, but now the lower input is connected to the output for all the gates. In total, $2n$ new approximate versions of C, each of them containing $n-1$ gates, are obtained. The circuit producing the smallest error is taken as the seed for CGP.

A natural extension of this heuristic for a circuit in which n gates have to be reduced to $n - k$ gates consists of: (1) a random selection of k gates and their replacement by wire connections; (2) calculating the fitness value of the modified circuit; (3) repeating steps (1) and (2) N times (where N is a suitable constant); and outputting the circuit with the best fitness value. This approach is suitable for complex circuits (thousands of gates or more) in which modifying all the gates could be very time consuming.

F. Embedding the Heuristic into CGP

Providing a single approximate circuit is not usually the most valuable output of approximate circuit design methods. Designers are looking for various tradeoffs among the design objectives. In order to find approximate circuits for every possible number of gates, the proposed approximate circuit design flow will call CGP several times. We have developed two approaches for embedding the heuristic into CGP in order to obtain approximate circuits containing $n-1, n-2, \ldots, 2, 1$ gates. Together with the random population seeding, we thus propose and compare the following three scenarios for seeding the initial populations of CGP.

- RS – All initial populations are randomly generated.
- HS1 – Heuristic seeding, according to Section III-E, in which the best result of CGP containing m gates is used by the heuristic to build a new seed containing $m - 1$ gates. Applying HS1 means that each CGP run is, therefore, interleaved by a single run of the heuristic procedure removing just one gate from the best evolved solution.
- HS2 – Heuristic seeding, according to Section III-E, in which the heuristic is applied iteratively on its previous result in order to build a set of seeds containing $n-1, n-2, \ldots, 1$ gates. This means that all requested seeds are firstly generated by the heuristic and independent CGP runs are then initialized using the created seeds.

The initial, fully functional solution which the heuristics HS1 and HS2 begins with is a conventional implementation of target circuits.

IV. Experimental Results

Several papers have addressed the evolutionary design of small combinational parallel w-bit multipliers with the goal of minimizing the number of gates (see, e.g. [30], [33]). This task is considered as a very difficult benchmark for evolutionary circuit design methods; much harder than the evolution of

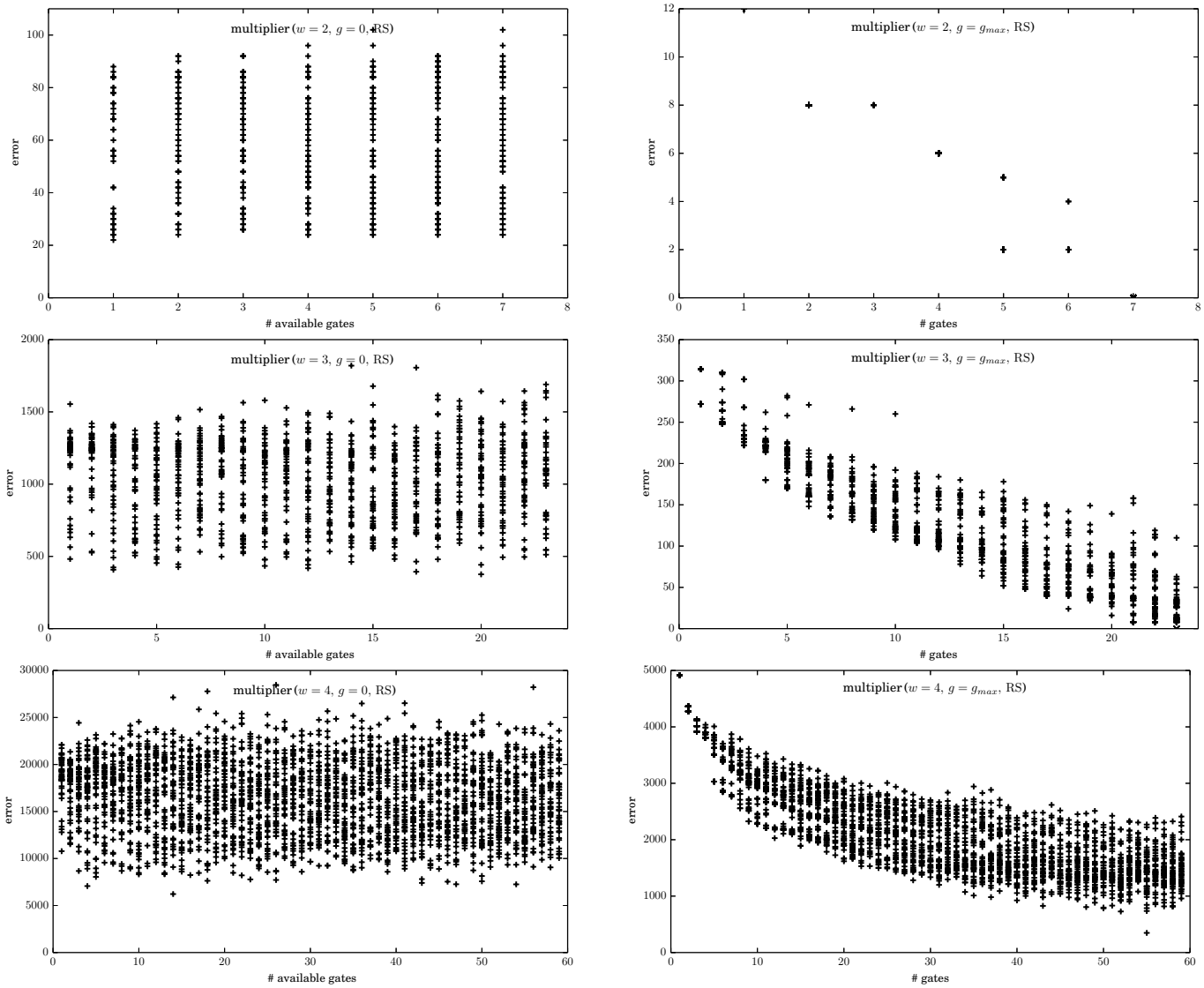

Fig. 2. Error of the randomly generated seeds (left column) and error of the evolved solutions (right column) for 2-bit, 3-bit and 4-bit approximate multiplier in the RS scenario.

adders, multiplexers or parity circuits. Hence results competitive with conventional synthesis algorithms were reported for up to 4-bit multipliers. This section extends these results by considering approximate versions of the multiplier circuits. Moreover, it presents a comparison of the proposed single objective CGP with MOEA. The second case study deals with the synthesis and optimization of approximate median circuits with 9 inputs (9-median, for short) and 25 inputs (25-median) working over 8 bits. Results will be reported for every possible number of gates (components) in order to show all available tradeoffs.

A. Approximate Multipliers

The goal of CGP is to design a multiplier showing the lowest possible error for a given number of gates. The error is expressed according to Eq. 1. The CGP parameters are initialized as follows: $n_r = 1$, $l = n_c$, $\lambda = 4$, $h = 5\%$, and $\Gamma = \{$BUF, NOT, AND, OR, XOR, NAND, NOR, XNOR$\}$, where BUF stands for an identity function. The setting of the CGP parameters is based on experiments conducted in our previous research [10]. The evolutionary algorithm stops when the predefined number of generations g_{max} is exhausted. All the experiments were performed on a cluster of computation nodes equipped with Intel Xeon processors running at 3 GHz.

CGP, seeded by the *RS strategy*, is applied as follows. Let n_{bst} be the number of two-input gates required to implement a conventional fully functional multiplier. All experiments were conducted for $n_{bst} = 7$, 23 and 59, corresponding to the 2-bit, 3-bit and 4-bit multiplier constructed according to the conventional Ripple-Carry-Array-Multipliers. For each w-bit multiplier, we performed n_{bst} independent experiments consisting of 50 independent CGP runs each. The parameter $n_c = n_{bst}, n_{bst}-1, \ldots, 1$ is used in these experiments. The initial population is always randomly generated. The maximum number of generations is limited to $g_{max} = 800 \cdot 10^6$, $500 \cdot 10^6$ and $350 \cdot 10^6$ for the 4-bit, 3-bit and 2-bit multiplier (which is

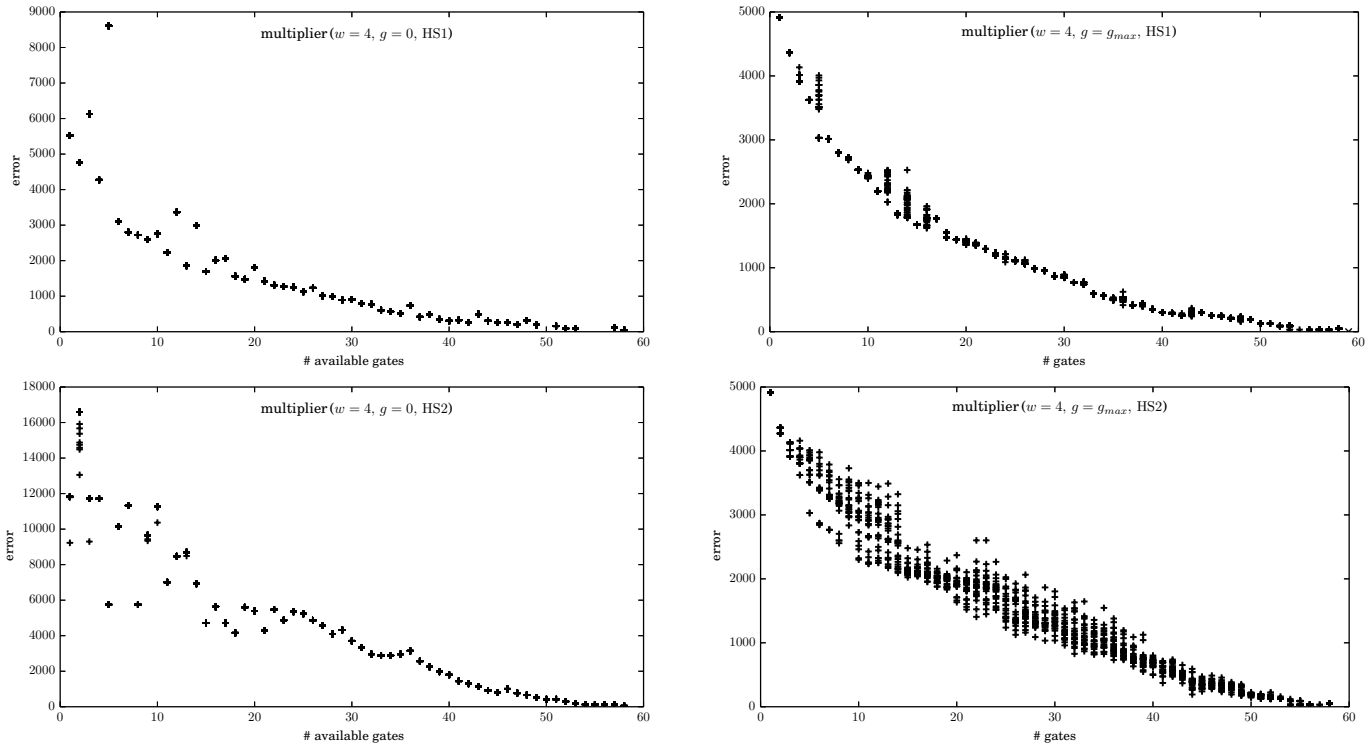

Fig. 3. Error of the seeds (left column) and error of the evolved solutions (right column) for 4-bit approximate multiplier in HS1 (top) and HS2 (bottom) scenarios.

consistent with [30]), corresponding to a single evolutionary run of 24 hours, 3 hours and 50 minutes, respectively.

In the case of the HS1 and HS2 strategies, all the evolutionary runs of the first experiment (when $n_c = n_{bst} - 1$) are seeded with the same initial circuit obtained from a conventional solution by removing exactly one gate. Seeding the initial population means that the number of generations can be reduced (see below). Hence we chose $g_{max} = 200 \cdot 10^6$, $100 \cdot 10^6$ and $100 \cdot 10^6$ for 4-bit, 3-bit and 2-bit multiplier, respectively. The corresponding runtime of a single CGP run is 2 hours, 30 minutes and 30 minutes, respectively.

1) Random Seeding: Figure 2 depicts fitness values of the randomly generated seeds and resulting fitness values at the end of evolution for all approximate multipliers in all runs. The column on the left in Figure 2 shows that the fitness values of seeds are distributed similarly for all problem instances, independently of the number of gates. The fitness values of evolved circuits (the right column) are one order of the magnitude smaller than in the case of the seeding circuits. However, the errors are still relatively high, especially for the 4-bit multiplier. With decreasing amount of resources, the spread of fitness values becomes smaller.

One can observe that the mean fitness f_{mean} of the initial seed (calculated over all runs) is practically independent of the number of available gates for a given multiplier. In additional experiments, we analyzed this phenomenon in detail for various multipliers and adders. Table I gives the *mean relative error*

$$\epsilon_{mrt} = \frac{f_{mean}}{2^{n_i}(2^{n_o} - 1)} \quad (2)$$

of randomly generated circuits consisting of one gate ($n_c = 1$) and n_{bst} gates. It seems that $\epsilon_{mrt} \approx 25\%$ is a reasonable error estimation, not only for multipliers, but also for other approximate arithmetic circuits such as adders that are randomly generated using the proposed method, independently of the number of used gates. This is an important experimental outcome which should help to establish the initial error of any approximation of small combinational circuits performed by means of CGP.

TABLE I
RELATIVE ERROR ϵ_{mrt} [%] FOR VARIOUS BIT WIDTHS w AND DIFFERENT NUMBER OF CGP COLUMNS n_c FOR TWO SELECTED ARITHMETIC CIRCUITS (200 INDEPENDENT RUNS)

multiplier			adder		
w	$n_c = 1$	$n_c = n_{bst}$	w	$n_c = 1$	$n_c = n_{bst}$
2	30.440	23.702	2	18.566	20.477
3	28.943	23.877	3	18.671	22.431
4	28.434	24.090	4	19.467	22.439
5	28.540	25.246	5	19.856	23.100
6	28.874	25.531	6	20.774	23.145
7	29.339	25.628	7	21.609	23.780
8	29.498	25.426	8	22.453	24.063

2) Heuristic Seeding: Because the HS1 strategy starts with already pre-optimized circuits, it can provide seeds which are very close to resulting circuits (Figure 3, above). Contrasted to a very large spread of error values in RS (Figure 2, right column), it can be seen in HS1 that the CGP runs often converge to one or two fitness values (errors). This is valuable for practice because it means that a single run almost always

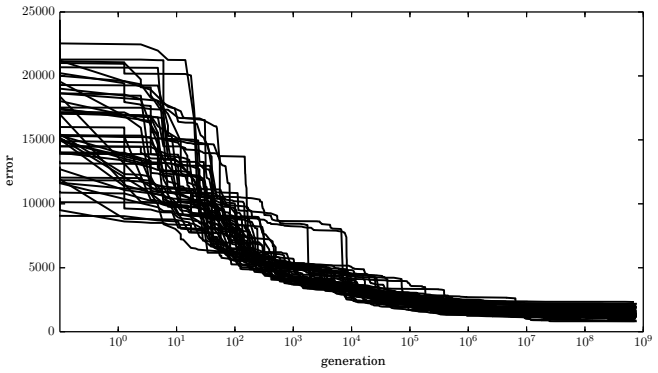

Fig. 4. Convergence curves for the best 4-bit multipliers in all 50 evolutionary runs ($n_c = 58$, RS strategy).

provides a high-quality solution. The quality of seeding by HS2 is 2–4 times worse as all seeds are generated before the CGP is employed and no intermediate results from CGP can influence the HS2 procedure (the y-axis in Figure 3, bottom left). The CGP runs converge to several solutions with different fitness values (errors). However, in both cases the error of the generated seeds is significantly lower than the error of the randomly generated seeds (see last row of Figure 2 and Figure 3).

3) Convergence Curves: Figures 4 and 5 show convergence curves of all runs in the case where the 4-bit multiplier can utilize 58 gates ($n_{bst} = 59$). Random seeding leads to long convergence times (the best fitness value f stagnates after 10^4 generations) and relatively high errors (see the y-axis of Figure 4). The HS1 strategy starts with error f=128 and ends up with error f=32 in most cases (see the y-axis of Figure 5). The average error at the end of evolution seeded by RS is approximately 50 times higher than in the case of the HS1 strategy.

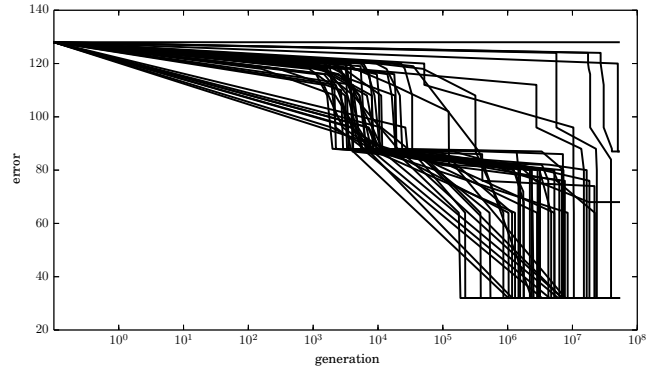

Fig. 5. Convergence curves for the best 4-bit multipliers in all 50 evolutionary runs ($n_c = 58$, HS1 strategy).

4) Overall Comparison: Figure 6 compares the best solutions obtained in scenarios RS, HS1, and HS2 for 3-bit and 4-bit approximate multipliers. We also included the best results obtained from 50 independent runs of MOEA which was seeded randomly (MOR) and, in another series of 50 runs, using conventional implementations of multipliers (MOB).

The utilized MOEA implements NSGA-II according to [26],

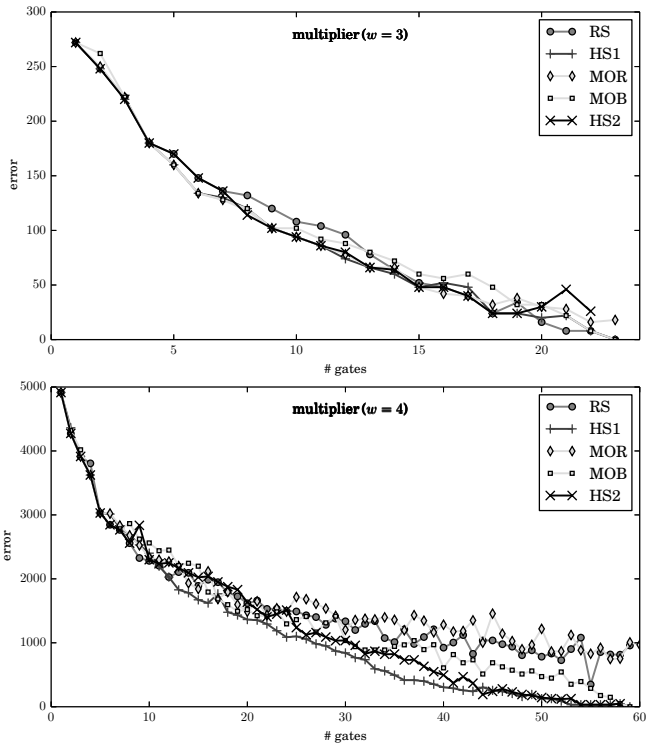

Fig. 6. Error of the best 3-bit (top) and 4-bit (bottom) approximate multipliers obtained by the proposed seeding strategies and in the multiobjective optimization scenario (MOR, MOB).

and employs a 50-member population. In order to allow the same number of evaluations as in the proposed CGP, $g_{max} = 40 \cdot 10^6$ and $64 \cdot 10^6$ for 3-bit and 4-bit multiplier, respectively, in the case of random seeding. The number of generations was decreased to $g_{max} = 8 \cdot 10^6$ and $16 \cdot 10^6$ in the case of seeding by conventional implementations.

While performance of all the methods is similar on the 3-bit multiplier, HS1 and HS2 seeding strategies clearly outperform RS and both MOEAs in terms of quality of results on the 4-bit multiplier. The gap is significant, especially when 60–90% gates remain in the circuit, which is a typical situation in practice.

Another improvement is in terms of time: RS requires 15 times more generations to reach a solution of the same quality as HS1 and HS2.

A detailed analysis of the best evolved approximate circuits revealed that a circuit containing k gates can exhibit a higher error than a circuit containing $k-1$ gates (see, for example, the small peak in the fitness function for circuits containing 16 gates and 17 gates in Figure 6, HS1, $w = 4$). In practice, the circuit containing 16 gates should be taken, even if 17 gates are allowed. There are two explanations for this behavior. Either the evolutionary algorithm did not find a better solution for 17 gates under our setup or a better solution for 17 gates does not exist at all.

5) Power Consumption vs. Error vs. Area: Using the SIS software [25] we calculated dynamic power consumption and delay for the best fully functional conventional as well as evolved multipliers (Table II) which will be serving as refer-

ence solutions in the following comparisons. The calculations are valid for the MCNC library [25], $V_{dd} = 5$ V and 20 MHz. The relative area of the used gates is: INV-A 0.67, BUF 0.0, NAND2 and NOR2 1.00, AND2 and OR2 1.33, XOR2 2.00, XNOR2 1.66. The sum of relative areas of gates connected into a particular circuit will be denoted 'area' and will represent the total circuit area *relatively* to the area of a single NAND gate in the following text.

TABLE II
PARAMETERS OF THE BEST FULLY FUNCTIONAL MULTIPLIERS

		power [uW]			area [-]			delay [ns]		
w	n_c	best	worst	mean	best	worst	mean	best	worst	mean
2	7	44.5	65.3	51.5 ± 5.2	8.3	10.7	9.4 ± 0.8	4.8	13.5	8.4 ± 1.8
3	23	220.4	248.9	235.9 ± 8.4	32.3	34.6	33.1 ± 0.7	20.9	26.8	24.3 ± 1.4
4	59	790.1	1425.4	1119.6 ± 193.3	83.9	87.5	86.2 ± 1.3	47.8	55.0	50.5 ± 2.5

Power consumption and error of the best evolved approximate 2-bit multipliers are analyzed for a given number of gates in Figure 7. Power consumption is given relatively to the best conventional solution from Table II. The 7-gate implementations are fully functional. It makes no sense to choose a 6-gate (3-gate, respectively) implementation because the same error can be obtained using a 5-gate (2-gate, respectively) implementation. The evolved 5-gate solution (error = 2) is identical (in terms of structure as well as parameters) with the approximate 2-bit multiplier discovered manually in [3] (see Section II-A2). Contrasted to 7-, 6-, 4- and 3-gate implementations, there is only one (we believe that truly optimal) unique solution in terms of power consumption and error composed of 5 gates.

Figures 8 and 9 show power consumption and error of the best evolved 3-bit and 4-bit approximate multipliers. A general observation is that the amount of different implementations (and spread of power consumption) decreases with reducing available resources.

6) Comparison With Other Approximate Multipliers: We rediscovered the manually created 2-bit approximate multiplier consisting of 5 gates (it is denoted M2 in Table III) [3]. Contrasted to the manual design we were able to find very good approximate 4-bit multipliers using CGP (see E4a and E4b in Table III). In order to demonstrate the quality of the evolved solutions, we composed (by the method introduced in [3]) larger approximate multipliers (4-bit, 8-bit and 16-bit) using M2, E4a and E4b. The approximate multipliers E4a and E4b were included in the table because they match the number of gates (47 in the case of E4a) and the average error (1.23% in the case of E4b) of the 4-bit approximate multiplier (C4) composed of M2 multipliers.

Table III gives the resulting area and error of the chosen approximate multipliers. The errors are given relative to the corresponding maximum values. The maximum value of the *worst case error* as well as *average error* is equal to $e_{max} = 2^{2w} - 1$. The average error is the total error (as defined in Eq. 1) averaged over all 2^{2w} inputs.

Approximate multipliers composed of evolved approximate 4-bit multipliers show a better tradeoff between the area and error than approximate multipliers composed of M2. For example, the 8-bit multiplier (E8a) composed of evolved 4-bit multipliers E4a exhibits the average error 0.32%, while the average error of the 8-bit multiplier (C8) composed of M2 is 1.38%. Moreover, the worst case error of E8a is 5 times lower. Both 8-bit multipliers, however, consists of 276 gates. A more compact implementation E8b (208 gates) shows an average error of 1.28%, which is even better than C8 can provide.

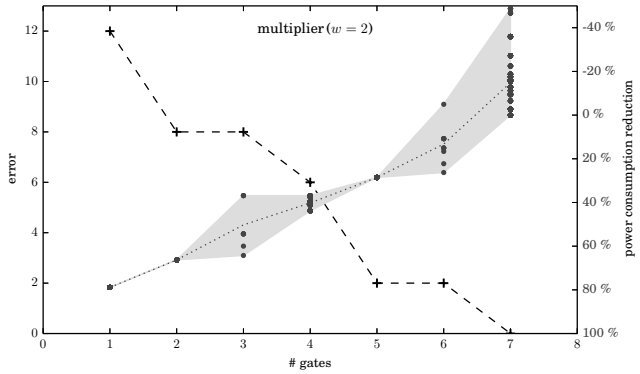

Fig. 7. Power consumption and error of the best evolved approximate 2-bit multipliers for a given number of gates. The mean power reduction is shown as dotted line.

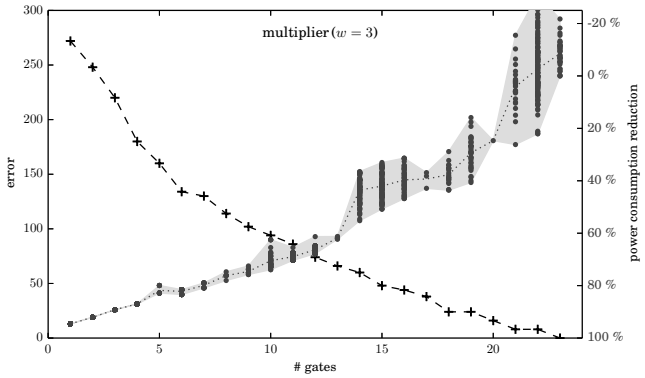

Fig. 8. Power consumption and error of the best evolved approximate 3-bit multipliers for a given number of gates.

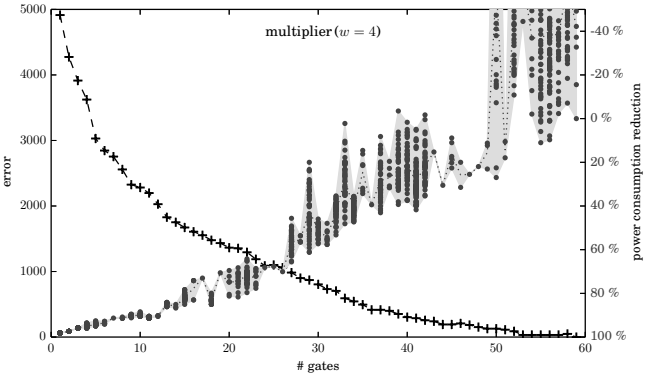

Fig. 9. Power consumption and error of the best evolved approximate 4-bit multipliers for a given number of gates.

Our results are hardly comparable with the SASIMI method, because SASIMI employs a different technology library

TABLE III
PARAMETERS OF MANUALLY CREATED AND EVOLVED APPROXIMATE MULTIPLIERS FOR LARGER BIT-WIDTHS (w)

Code	w	gates	worst err.	error prob.	average err.	area	delay [ns]
M2	2 [3]	5	13.33%	0.06	0.83%	6.7	5.3
C4	4	**47**	19.61%	0.19	**1.23%**	71.2	38.9
C8	8	276	22.05%	0.47	1.38%	427.4	93.5
C16	16	1288	22.22%	0.81	1.39%	2006.5	184.3

Code	w	gates	worst err.	error prob.	average err.	area	delay [ns]
E4a	4	**47**	3.92%	0.17	0.28%	66.2	34.4
E8a	8	276	4.41%	0.45	0.32%	407.3	87.8
E16a	16	1288	4.44%	0.81	0.32%	1926.3	178.7

Code	w	gates	worst err.	error prob.	average err.	area	delay [ns]
E4b	4	30	7.06%	0.77	**1.23%**	41.9	27.4
E8b	8	208	7.94%	0.98	1.28%	310.2	81.4
E16b	16	1016	8.00%	0.99	1.29%	1537.9	172.2

and applies various technology-dependent operations such as downsizing of gates, which allows for an additional area reduction. For example, an 8-bit multiplier initially consisting of 1055 gates was processed by SASIMI which resulted in a 37% area reduction (it roughly corresponds to an approximate multiplier consisting of 664 gates) and the average error of 0.32% [7]. For the same average error, our 8-bit approximate multiplier E8a consists of 276 gates only. It thus exhibits a reduction of 13% of gates in comparison with a *different* initial implementation containing 319 gates.

B. Approximate Median Circuits

As it is intractable to evaluate all possible input combinations (256^9 and 256^{25} vectors) for candidate median circuits, we randomly generated 10^4 training vectors for the 9-median circuit and 10^5 vectors for the 25-median circuit. These values were selected according to Table IV which shows the average deviation of the error if a certain number of randomly generated test vectors is applied to evaluate the quality of these circuits. In order to eliminate the dependency on a certain solution, 10 evolved median circuits utilizing 50% of the resources were used. Each circuit was evaluated using a set of 10 different randomly generated test vectors. It can be seen that if we apply multiple times at least 10^4 test vectors to evaluate the error of the 9-median circuit, the obtained deviation is less than 1%.

CGP operates with parameters $n_r = 1$, $l = n_c$, $\lambda = 4$, $h = 5\%$, and $\Gamma = \{\text{BUF, MIN, MAX}\}$. All components and connections are defined over 8 bits. Fully functional implementations with $n_{bst} = 37$ for the 9-median and $n_{bst} = 221$ for the 25-median were constructed using the bitonic sorter algorithm [34]. The number of generations of the RS-based

TABLE IV
RELATIVE ERROR DEVIATION OF 10 EVOLVED MEDIAN CIRCUITS FOR VARIOUS NUMBERS OF TEST VECTORS

	# test vectors						
problem	10	10^2	10^3	10^4	10^5	10^6	10^7
9-median	62.82%	13.24%	3.98%	0.98%	0.48%	0.15%	0.01%
25-median	46.41%	10.22%	2.28%	1.11%	0.30%	0.10%	0.01%

CGP is limited by $g_{max} = 3 \cdot 10^6$ for the 9-median and $g_{max} = 300 \cdot 10^3$ for the 25-median which corresponds to 3 hour CGP runs in both cases. CGP exploiting HS1 and HS2 utilized only 1/3 of the previously mentioned time budget. Each CGP run is repeated 50 times.

1) The Role of Seeding: The randomly seeded CGP led to fully functional solutions for the 9-median while no correct solution was discovered for the 25-median. It seems that solving the 25-median design problem from scratch is impossible for any evolutionary algorithm based on a direct encoding. Although CGP could utilize up to $n_{bst} = 221$ components, the most complex circuits only use 106 components (Figure 10). In order to investigate this phenomenon, we conducted an another experiment and seeded CGP by randomly created circuits that utilize all 221 components, but most of them were disconnected in the course of evolution, thus reaching 106 components again.

Figure 10 shows the fitness values of all randomly created circuits and the resulting evolved approximate median circuits. Compared to the multiplier problem, the error values of the randomly generated circuits are not distributed uniformly.

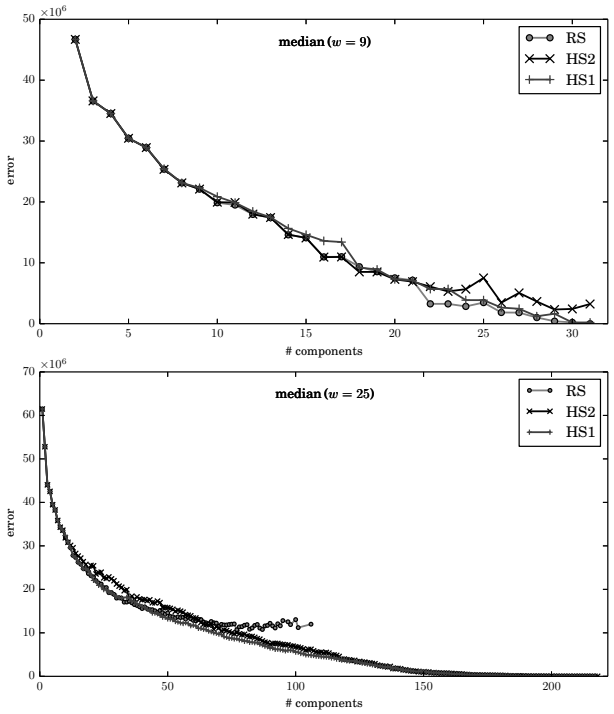

Fig. 11. Error of the best 9-input (top) and 25-input (bottom) approximate median circuits obtained by the proposed strategies.

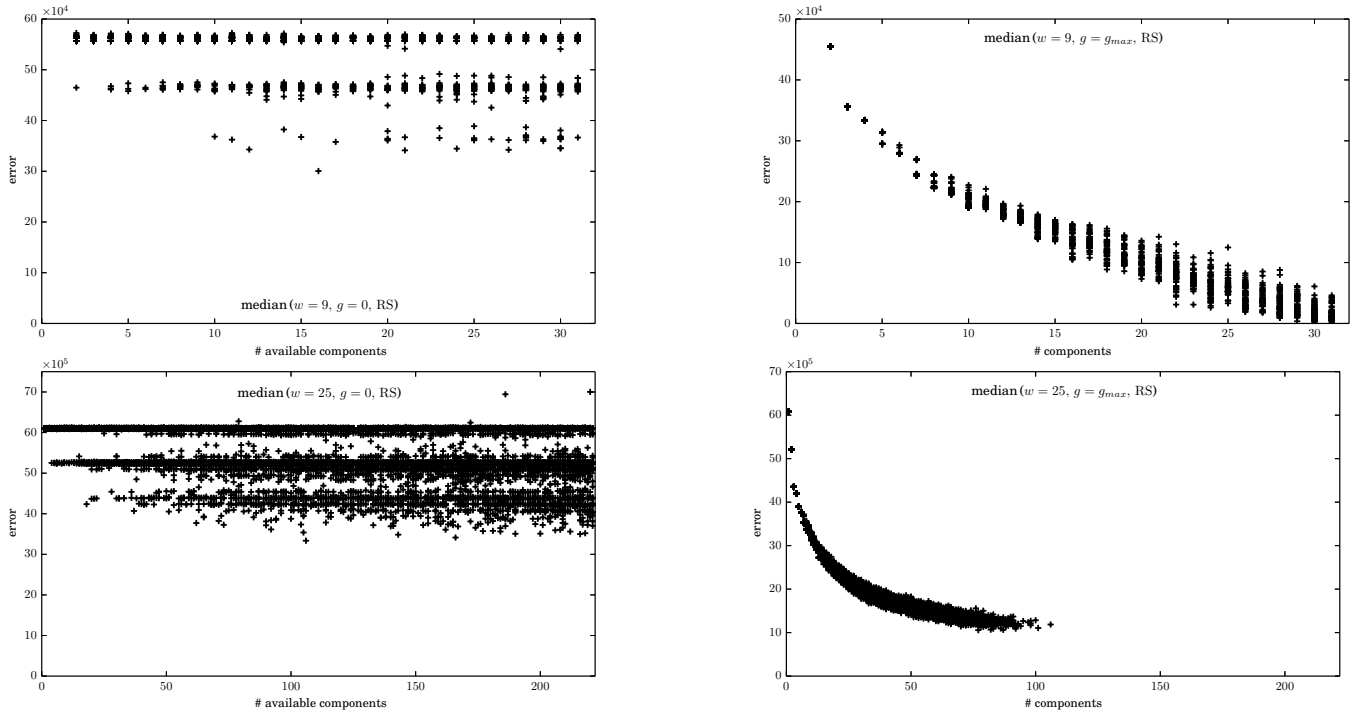

Fig. 10. Error of the seeds (left column) and error of the evolved solutions (right column) for 9-input (top) and 25-input (bottom) median circuit.

The effect of RS, HS1 and HS2 seeding strategies is compared in Figure 11 which gives the error of the best evolved solution for a given number of components. All strategies perform almost identically for the 9-median when the number of components is lower than 20. RS is clearly outperformed by HS1 for the 25-median. We can again observe the situation in which a circuit containing k components exhibits a higher error than a circuit containing $k-1$ components.

2) Best Approximate Median Circuits: Power consumption, area and delay of the best evolved fully functional solutions are summarized in Table V. These circuits are composed of 8-bit subcomponents: minimum and maximum. Each of them is represented as a netlist containing the gates presented in Section IV-A5. All the circuit parameters were obtained from the SIS tool. Power consumption is given for 320 thousand randomly generated input vectors.

TABLE V
PARAMETERS OF THE BEST CONVENTIONAL MEDIAN CIRCUITS

		power [mW]			area [-]			delay [ns]		
w	n_c	best	worst	mean	best	worst	mean	best	worst	mean
9	31	10.8	12.9	12.6	2314.2	2836.7	2750.8	285.9	429.7	295.1
25	221	72.4	72.4	72.4	16497.7	16497.7	16497.7	539.5	539.5	539.5

Figures 12 and 13 show power consumption and error of the best evolved approximate median circuits. The error is calculated using 10^6 test vectors. Power consumption is given relatively to the fully functional circuit showing the lowest power consumption.

Median circuits are very good examples of circuits for which it makes sense to introduce their approximate versions. The mean error remains relatively low, even for significant reductions of available gates. Hence significant improvements in energy consumption are obtained.

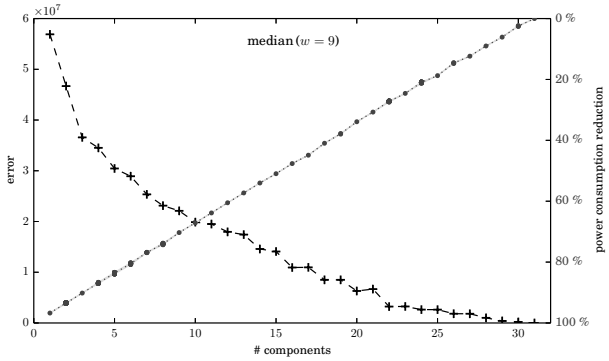

Fig. 12. Power consumption and error of the best evolved approximate 9-median for a given number of components.

V. Discussion and Conclusions

The proposed method and experimental results can be interpreted from several points of view. Firstly, the proposed method is a new systematic method for the design of approximate circuits. Its main contribution lies in the 'area' rather than the 'error' oriented approach to approximate circuit design which enables the user to comfortably control the used resources. It is useful, for example, when an image filter has to be approximated because the conventional implementation does not fit in the available space on a chip. The method works at the logic level and no special technology-oriented techniques (such as downsizing of gates) were considered

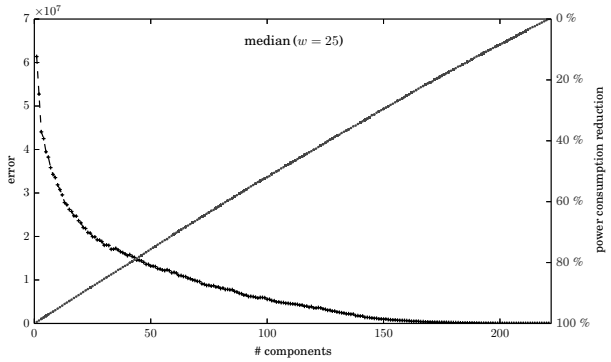

Fig. 13. Power consumption and error of the best evolved approximate 25-median for a given number of components.

during our experimental evaluation. Secondly, because the method is based on evolutionary computation, it can naturally provide more tradeoffs than current methods based on manual modifications of existing designs or reusing conventional synthesis tools. Thirdly, we evolved new approximate implementations of key circuits (multipliers, median computing circuits) which can immediately be used in various applications. These circuits, together with approximate adders and other approximate combinational circuits presented in our previous work [10], demonstrate that the CGP-based method is suitable for approximate circuits design. Fourthly, we have shown that the proposed method can easily be extended from the gate to the functional level evolution. Conventional methods (such as SALSA and SASIMI) work at the bit level only and hence they cannot be applied to directly approximate circuits such as the median. Fifthly, we provided detailed analyses of selected features of CGP, particularly the population seeding, which had not been done before.

Our initial assumption that the power consumption is highly correlated with area (see Section III-A) and that the proposed methodology can be based on reducing the number of gates was positively confirmed. Figure 14 shows the dependence for evolved multipliers. By considering the area (amount of gates) only, without calculating power consumption for every candidate circuit, we obtained very good approximations in a relatively short time.

Experimental results confirmed the superiority of heuristic seeding of the initial population over random seeding. The benefits are not only in improving the quality of evolved circuits, but also in reducing the time of optimization. Another advantage is that each run of CGP seeded by the HS1 strategy provides a high-quality solution. For more complex problem instances (such as the 25-median), the randomly seeded standard CGP did not provide satisfactory results. A suitable seeding approach thus remains a method to overcome this limitation.

The execution time is certainly the most critical disadvantage of the proposed method. It mainly depends on the number of inputs and size of the circuit. When a suitable seeding of the initial population is available, then CGP runtimes are typically in the order of tens of minutes on a common desktop computer. As no execution times were reported for SASIMI, we give the

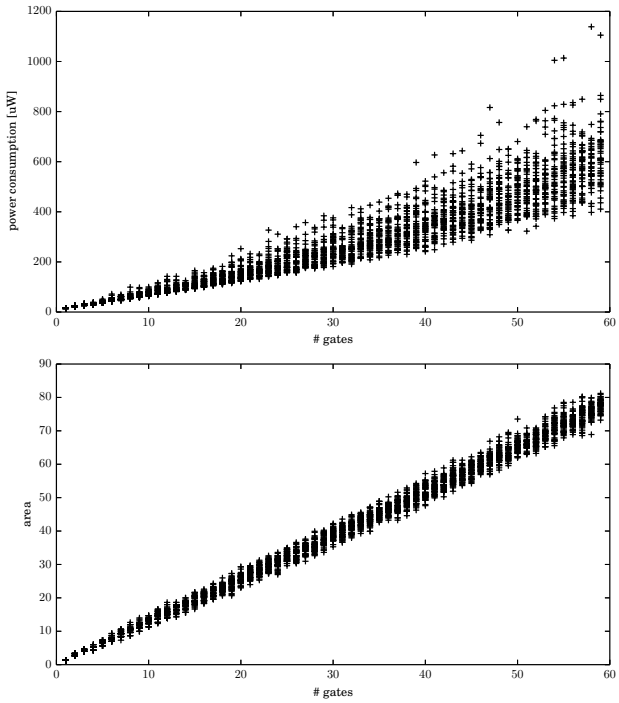

Fig. 14. Relation between the number of gates and power consumption (top) and delay (bottom) for all evolved approximate 4-bit multipliers.

execution times of SALSA which, on a server with an AMD Opteron 6176 (2.29 GHz) processor, ranged from 4 minutes to 2.5 hours depending on the circuit complexity.

Our future research will be focused on applying selected approaches (such as incremental evolution, functional equivalence checking) introduced to eliminate the scalability problems of evolutionary circuit design to evolutionary design methods intended for approximate circuits. We believe that the notion of approximate computing offers new applications for genetic programming (such as a design of underdesigned software for embedded systems) which should be explored in future research.

ACKNOWLEDGMENT

This work was supported by the IT4Innovations Centre of Excellence CZ.1.05/1.1.00/02.0070 and Brno University of Technology under number FIT-S-14-2297.

REFERENCES

[1] J. Han and M. Orshansky, "Approximate computing: An emerging paradigm for energy-efficient design," in *Proc. of the 18th IEEE European Test Symposium*. IEEE, 2013, pp. 1–6.

[2] V. Gupta, D. Mohapatra, A. Raghunathan, and K. Roy, "Low-power digital signal processing using approximate adders," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 32, no. 1, pp. 124–137, 2013.

[3] P. Kulkarni, P. Gupta, and M. D. Ercegovac, "Trading accuracy for power in a multiplier architecture," *J. Low Power Electronics*, vol. 7, no. 4, pp. 490–501, 2011.

[4] S. Venkataramani, A. Sabne, V. J. Kozhikkottu, K. Roy, and A. Raghunathan, "Salsa: systematic logic synthesis of approximate circuits," in *The 49th Annual Design Automation Conference 2012, DAC '12*. ACM, 2012, pp. 796–801.

[5] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger, "Neural acceleration for general-purpose approximate programs," in *Proc. of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Computer Society, 2012, pp. 449–460.

[6] A. Sampson, W. Dietl, E. Fortuna, D. Gnanapragasam, L. Ceze, and D. Grossman, "Enerj: Approximate data types for safe and general low-power computation," in *Proc. of the 32Nd ACM SIGPLAN Conference on Programming Language Design and Implementation*. ACM, 2011, pp. 164–174.

[7] S. Venkataramani, K. Roy, and A. Raghunathan, "Substitute-and-simplify: a unified design paradigm for approximate and quality configurable circuits," in *Design, Automation and Test in Europe, DATE'13*. EDA Consortium, 2013, pp. 1367 – 1372.

[8] J. D. Lohn and G. S. Hornby, "Evolvable hardware: Using evolutionary computation to design and optimize hardware systems," *IEEE Computational Intelligence Magazine*, vol. 1, no. 1, pp. 19–27, 2006.

[9] P. C. Haddow and A. M. Tyrrell, "Challenges of evolvable hardware: past, present and the path to a promising future," *Genetic Programming and Evolvable Machines*, vol. 12, no. 3, pp. 183–215, 2011.

[10] L. Sekanina and Z. Vasicek, "Approximate circuits by means of evolvable hardware," in *2013 IEEE International Conference on Evolvable Systems*, ser. Proceedings of the 2013 IEEE Symposium Series on Computational Intelligence (SSCI). IEEE CIS, 2013, pp. 21–28.

[11] V. Venkatachalam and M. Franz, "Power reduction techniques for microprocessor systems," *ACM Computing Surveys*, vol. 37, no. 3, pp. 195–237, 2005.

[12] N. R. Shanbhag, R. A. Abdallah, R. Kumar, and D. L. Jones, "Stochastic computation," in *The 47th Annual Design Automation Conference, DAC '10*. ACM, 2010, pp. 859–867.

[13] R. Venkatesan, A. Agarwal, K. Roy, and A. Raghunathan, "Macaco: Modeling and analysis of circuits for approximate computing," in *2011 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2011, pp. 667–673.

[14] M. Murakawa, S. Yoshizawa, I. Kajitani, T. Furuya, M. Iwata, and T. Higuchi, "Evolvable Hardware at Function Level," in *Parallel Problem Solving from Nature - PPSN IV*, ser. LNCS, vol. 1141. Springer Verlag, 1996, pp. 62–71.

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

[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. Gordon, "Exploiting Development to Enhance the Scalability of Hardware Evolution," Ph.D. dissertation, Department of Computer Science, University College London, 2005.

[18] K. Imamura, J. A. Foster, and A. W. Krings, "The test vector problem and limitations to evolving digital circuits," in *Proc. of the 2nd NASA/DoD Workshop on Evolvable Hardware*. IEEE Computer Society Press, 2000, pp. 75–79.

[19] 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.

[20] Z. Vasicek and K. Slany, "Efficient phenotype evaluation in cartesian genetic programming," in *Proc. of the 15th European Conference on Genetic Programming*, ser. LNCS 7244. Springer Verlag, 2012, pp. 266–278.

[21] J. F. Miller, "On the filtering properties of evolved gate arrays," in *1st NASA-DoD Workshop on Evolvable Hardware*. IEEE Computer Society, 1999, pp. 2–11.

[22] G. Greenwood and A. M. Tyrrell, *Introduction to Evolvable Hardware*. IEEE Press, 2007.

[23] T. Knieper, P. Kaufmann, K. Glette, M. Platzner, and J. Torresen, "Coping with resource fluctuations: The run-time reconfigurable functional unit row classifier architecture," in *Proc. of the 9th Int. Conf. on Evolvable Systems: From Biology to Hardware*, ser. LNCS, vol. 6274. Springer, 2010, pp. 250–261.

[24] A. Thompson, P. Layzell, and S. Zebulum, "Explorations in Design Space: Unconventional Electronics Design Through Artificial Evolution," *IEEE Transactions on Evolutionary Computation*, vol. 3, no. 3, pp. 167–196, 1999.

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

[26] J. Petrlik and L. Sekanina, "Multiobjective evolution of approximate multiple constant multipliers," in *IEEE International Symposium on Design and Diagnostics of Electronic Circuits and Systems*. IEEE Computer Society, 2013, pp. 116–119.

[27] J. Hilder, J. Walker, and A. Tyrrell, "Use of a multi-objective fitness function to improve cartesian genetic programming circuits," in *NASA/ESA Conference on Adaptive Hardware and Systems*. IEEE, 2010, pp. 179–185.

[28] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: Nsga-ii," *Evolutionary Computation, IEEE Transactions on*, vol. 6, no. 2, pp. 182–197, 2002.

[29] J. F. Miller, D. Job, and V. K. Vassilev, "Principles in the Evolutionary Design of Digital Circuits – Part II," *Genetic Programming and Evolvable Machines*, vol. 1, no. 3, pp. 259–288, 2000.

[30] ——, "Principles in the Evolutionary Design of Digital Circuits – Part I," *Genetic Programming and Evolvable Machines*, vol. 1, no. 1, pp. 8–35, 2000.

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

[32] J. F. Miller and S. L. Smith, "Redundancy and computational efficiency in cartesian genetic programming," *IEEE Trans. Evolutionary Computation*, vol. 10, no. 2, pp. 167–174, 2006.

[33] S. Zhao and L. Jiao, "Multi-objective evolutionary design and knowledge discovery of logic circuits based on an adaptive genetic algorithm," *Genetic Programming and Evolvable Machines*, vol. 7, no. 3, pp. 195–210, 2006.

[34] D. E. Knuth, *The Art of Computer Programming: Sorting and Searching (2nd ed.)*. Addison Wesley, 1998.