Genet Program Evolvable Mach (2016) 17:169–192
DOI 10.1007/s10710-015-9257-1

Evolutionary design of complex approximate combinational circuits

Zdenek Vasicek[1] · **Lukas Sekanina**[1]

Received: 30 June 2015 / Revised: 13 November 2015 / Published online: 12 December 2015

Abstract Functional approximation is one of the methods allowing designers to approximate circuits at the level of logic behavior. By introducing a suitable functional approximation, power consumption, area or delay of a circuit can be reduced if some errors are acceptable in a particular application. As the error quantification is usually based on an arithmetic error metric in existing approximation methods, these methods are primarily suitable for the approximation of arithmetic and signal processing circuits. This paper deals with the approximation of general logic (such as pattern matching circuits and complex encoders) in which no additional information is usually available to establish a suitable error metric and hence the error of approximation is expressed in terms of Hamming distance between the output values produced by a candidate approximate circuit and the accurate circuit. We propose a circuit approximation method based on Cartesian genetic programming in which gate-level circuits are internally represented using directed acyclic graphs. In order to eliminate the well-known scalability problems of evolutionary circuit design, the error of approximation is determined by binary decision diagrams. The method is analyzed in terms of computational time and quality of approximation. It is able to deliver detailed Pareto fronts showing various compromises between the area, delay and error. Results are presented for 16 circuits (with 27–50 inputs) that are too complex to be approximated by means of existing evolutionary circuit design methods.

✉ Lukas Sekanina
sekanina@fit.vutbr.cz

Zdenek Vasicek
vasicek@fit.vutbr.cz

[1] Faculty of Information Technology, IT4Innovations Centre of Excellence, Brno University of Technology, Brno, Czech Republic

Keywords Approximate circuit · Cartesian genetic programming · Binary decision diagram · Fitness function

1 Introduction

Reducing of energy consumption in integrated circuits is one of the key challenges of current chip design industry [4]. Hence, various approaches to energy consumption reduction have been developed. Energy consumption reduction can be tackled at different system levels (such as circuit, architecture, operating system, and software) with significantly different methodologies. One of them is *approximate computing* trying to exploit the error resilience which is displayed by many applications [11]. If one can relax the precision constraints, or tolerate some errors, hardware and software can be simplified and work with less energy. Suitable applications for approximate computing were identified in the areas of multimedia, database search, fault tolerant systems and others. They exploit the fact that human users, as major consumers of data outputs, have limited perception capabilities and no golden solution is usually available for validation of results [6]. An open question is how to automate the approximation of circuits and software in order to obtain desired quality of service (i.e. an acceptable error) and optimize available resources.

The *functional approximation* is one of methods allowing designers to approximate circuits at the level of logic behavior [40]. The idea behind the functional approximation is that a less complex function than the original one is implemented and used, providing that the error is acceptable and power consumption, area on the chip or other parameters are improved adequately. The approximations are obtained by a heuristic procedure which modifies the original, accurate circuit. Applying genetic programming as a heuristic method for circuit approximation has already led to finding high-quality compromises between key circuit parameters, see, for example [35, 36].

As the vast majority of approximation methods employ an arithmetic error metric, these methods are primarily suitable for the approximation of arithmetic circuits (adders, multipliers) and digital signal processing circuits. This paper deals with the approximation of general logic in which no additional information is usually available to establish a suitable error metric. Introducing approximations to general logic could be dangerous in many cases (e.g. for controllers), but there is still an important class of circuits (such as combinational logic of pattern matching circuits or complex encoders) in which the error can safely be exchanged for reducing the energy consumption or the area on a chip. For example, see an approximate pattern matching circuit optimized for fast classification of application protocols in high-speed networks [9]. In these cases, the error of approximation has to be expressed using a more general function, for example, as the average Hamming distance between the output values produced by a candidate approximate circuit and the accurate circuit.

The current literature describes various approaches to the digital circuit approximation. Regarding the methodological and evaluation approaches, two scenarios are dominating:

1. Ad hoc methods employed for the approximation of a (single) particular circuit. For example, see the approaches proposed to approximate multipliers [17] and adders [10].
2. Design automation methods developed for the approximation of a class of circuits (for example, SALSA [40], SASIMI [39] and ABACUS [23]).

In the first scenario, a lot of knowledge about a particular circuit and its typical utilization can be incorporated into the approximation method. However, it is difficult to apply the method for approximation of other circuits. In the second scenario, the approximations are performed using the same procedure for all problem instances of a given class. Approximate circuits showing different compromises between considered circuit parameters (area, delay, power consumption and errors of different types) are generated and presented to the user whose responsibility is to choose the most suitable approximate circuit for a given application. A detailed analysis of the impact of the approximation procedure on circuit parameters that were not considered during the approximation is also left on the user.

The goal of this work is to propose and evaluate an automated circuit approximation method (scenario 2) in which the error is expressed in terms of the average Hamming distance. We opted for the evolutionary approach based on genetic programming because it was capable of delivering high quality approximations in our previous work [35, 36]. In our method, gate-level circuits are evolved using Cartesian genetic programming (CGP) and internally represented using directed acyclic graphs. The method is thus suitable for approximation of combinational circuits, i.e. digital circuits in which the output values only depend on current input values. In the case of sequential circuits containing memory elements, the proposed method can be applied to a combinational part of the circuit.

The evolutionary circuit design methods in which candidate circuits are evaluated by checking their responses for all possible input combinations are not scalable. The main reason is that the evaluation time grows exponentially with the number of inputs. A naïve approach to evolve approximate circuits would be to identify a suitable subset of all possible input vectors, establish the fitness value using this subset and evolve a circuit showing a good trade-off between the error (for this subset) and the number of gates (or area). However, as it is reasonable to evaluate only up to about 2^{20} test vectors for each candidate circuit in a single CGP run on a common desktop computer [34], the resulting error would be extremely unreliable for circuits with, for example, 30 primary inputs.

In order to overcome this problem, we propose to determine the error of approximation by an equivalence checking algorithm operating over binary decision diagrams (BDD) representing the candidate approximate circuit and the accurate

circuit. The main advantage of BDDs is that the Hamming distance can be determined in linear time with respect to the BDD size. Converting a candidate circuit to BDD and performing the functionality comparison against the accurate circuit, expressed again as BDD, can be performed relatively quickly for many circuits relevant to practice. The proposed method is analyzed in terms of computational time and quality of approximation.

The method is evaluated using 16 benchmark combinational circuits which are difficult for the previous evolutionary approximation methods, because they have too many primary inputs (27–50 inputs) and gates. Pareto fronts showing obtained trade offs between the error, area and delay are also reported. Another contribution of our work is that it is focused on general (i.e. non-arithmetic) approximate circuits which has not been done before.

The rest of the paper is organized as follows. Section 2 summarizes relevant work in the areas of functional approximation and digital circuit evolution. The principles of BDD are defined in Sect. 3. The proposed method based on CGP is introduced in Sect. 4. The experimental setup, benchmark circuits and results of evolutionary design are presented in Sect. 5. Conclusions are given in Sect. 6.

2 Related work

This section briefly surveys conventional approaches introduced for functional approximation and evolutionary design methods developed for the design of common and approximate digital circuits. The survey is primarily focused on combinational circuits as no other circuits are relevant for this paper.

2.1 Functional approximation

The goal of functional approximation is to modify a given logic circuit in such a way that obtained error is minimal and key circuit parameters (such as delay, area and power consumption) are improved with respect to the original logic circuit. The approximations have been conducted manually or using systematic algorithmic methods. The *manual approximation* methods whose example results are approximate multipliers presented in [17] have recently been replaced by fully automated systematic methods in order to increase the design productivity as well as the quality and complexity of circuits that can be approximated.

The *systematic design automation methods* (such as SALSA [40], SASIMI [39] and ABACUS [23]) produce Pareto fronts showing various compromise solutions with respect to the optimized parameters (error, delay, and power consumption). It allows the user to select the best compromise solution for a given application.

A typical automated method starts with a fully functional circuit which is modified by means of a problem specific heuristic in order to improve key circuit parameters, and keep the error within predefined bounds. The Pareto front is obtained from multiple runs of a single-objective approximation (heuristic) algorithm initialized using different parameters (for example, five target errors are considered). Parameters of resulting approximate circuits are obtained by means

of professional design tools. As only tens to hundreds of design alternatives are generated, the resulting solutions do not cover the whole Pareto front and they are typically centered around a few dominant design alternatives (e.g. [23]). The available literature describing these methods does not present any detailed analyses of resulting Pareto fronts, i.e. it is unknown whether and how much the obtained results can be improved if, for example, more execution time were invested.

The key issue seems to be an efficient and reliable evaluation of candidate approximate circuits. Various error functions have been used, for example, worst error, relative error, average error magnitude, and error probability. While these errors can be computed for small circuits by analyzing circuit responses for all possible input vectors, formal methods have to be introduced to determine the error of complex arithmetic circuits. For example, an auxiliary circuit is constructed which instantiates the candidate approximate circuit and the accurate (golden) circuit and compares their outputs to quantify the error for any given input. In order to check whether a predefined worst error is violated by the candidate approximate circuit, Boolean satisfiability (SAT) solver is employed [41]. However, for example, no method capable of establishing the average error using a SAT solver has been proposed up to now.

Contrasted to the methods precisely calculating the error (which were described in the previous paragraph), the error of approximation is also often estimated using training data sets. This is typical for image and signal processing components (filters, classifiers) because suitable training data are usually available and calculating the exact error is intractable because of the overall complexity of these components [23].

Fault tolerant systems are another natural class of applications of approximate computing. Redundant circuits which are present in such systems can be approximated in order obtain a good trade off between dependability parameters and power consumption or area on the chip [25].

2.2 Evolutionary circuit design

The idea of evolvable hardware and digital circuit evolution was introduced by Higuchi et al. [12], in which the evolution of a six-input multiplexer using a circuit simulator was presented. Thompson reported first circuits evolved directly in the hardware in 1996 [29].

A significant development of evolutionary circuit design is connected to Cartesian genetic programming, a branch of genetic programming whose problem representation was inspired by digital circuits. In CGP, candidate circuits are encoded as arrays of integers and evolved using a simple search strategy. The standard CGP, its extensions (such as self-modifying CGP) and typical applications have been surveyed in a monograph [19]. Miller et al. demonstrated that CGP can improve results (in terms of the number of gates and delay) of conventional circuit synthesis and optimization algorithms in the case of small arithmetic circuits. A 4-b multiplier was the most complex circuit evolved in this category [38].

After the year 2000, various digital (predominately combinational) circuits were evolved. These circuits can be classified into two categories—completely specified and incompletely specified circuits. Completely specified circuits are arithmetic

circuits and general logic circuits in which a perfect response is requested for every legal input vector. On the other hand, incompletely specified circuits are used in applications such as classification, filtering, hashing and prediction in which the correctness can only be verified using a subset of all possible input vectors.

In comparison with conventional methods, the evolutionary design method is less scalable. It has several reasons. First, long chromosomes are needed to represent complex circuits, and consequently, huge search spaces have to be explored in which it is difficult to find useful designs. Second, the evaluation of complex circuits is very time consuming. In a typical approach, 2^n input vectors are applied (and simulated) to calculate the fitness of an n-input combinational circuit. In current practice, the maximum complexity of evolved circuits is low (about 20 inputs and 100 gates).

Several methods have been proposed to increase the complexity of circuits that can be obtained using evolutionary algorithms (EA). Functional level evolution [22] and decomposition methods [28, 30] enabled to reduce the search space. Combining functional level evolution with decomposition led to another increment in the complexity of evolved circuits [27]. Regarding the completely defined circuits, examples of the most complex circuits evolved so far are 22-b parity [24], 9-b adder [14] and 5-b multiplier [14]. The most complex circuit evolved using decomposition is a 135-b multiplexer which was obtained with a learning classifier system operating with complex building blocks. The correctness of resulting circuits was, however, estimated using simulation and manual inspection because it was impossible to get responses for all 2^{135} input vectors [15].

More promising results have been obtained by methods which try to reduce the fitness evaluation time using formal approaches in the fitness function.

In order to minimize the number of gates in fully functional circuits produced by well-tuned common synthesis and optimization tools, Vasicek and Sekanina [32] proposed to replace the circuit simulation by functional equivalence checking algorithms. For each candidate circuit and its parent, a SAT problem instance was created and solved using a SAT solver. If both circuits are functionally equivalent, the fitness of the candidate circuit is defined as the number of gates (with the aim to minimize them); otherwise, the candidate circuit is discarded. This approach led to a significant reduction in gate count for circuits having hundreds of inputs and containing thousands of gates [31], which is unreachable by the state of the art logic synthesis tools such as ABC [21]. The most complex circuit optimized using this method contains 16,158 gates and has 2176 inputs and 2136 outputs [31].

The SAT-based method is applicable only if a fully functional circuit is available. If a circuit has to be evolved from scratch (i.e. when no structural information about the circuit is provided, but responses are defined for all possible input combinations), Vasicek and Sekanina [34] combined CGP with BDD and developed a tool which allowed for evolving circuits with tens of inputs. The BDDs in the fitness function enable to effectively determine the Hamming distance between the output vectors of two circuits for many important problem instances (see Sect. 3). A 28-input circuit was successfully evolved from scratch without any kind of decomposition technique. In addition to that, the obtained circuit had less gates (a

57 % reduction) than the result of a conventional optimization conducted by the state-of-the-art tool.

2.3 Evolutionary circuit approximation

The use of evolutionary algorithms for functional approximation was surveyed in [26]. Employing evolutionary algorithms seems to be natural with respect to the goal of the approximation task. Small modifications introduced in the progress of evolution via genetic operators to a population of circuits and the principle of the survival of the fittest naturally lead to discovering such circuits which show very good compromises between the error and area (power consumption). Available evolutionary approximation methods employ CGP, which can operate either as a single-objective or multi-objective evolutionary optimizer. Within a given time which is available for the design, the single objective CGP provided more compact circuits than its multi-objective version [13, 35, 36].

Because of the scalability problems, the evolutionary approach allowed obtaining only relatively small approximate combinational circuits and arithmetic circuits (up to 8-b adders and multipliers [33, 35] when seeded by conventional implementations). More complex circuits, such as a 25-input median circuit, were then approximated by an evolutionary algorithm estimating the error of approximation using a small subset (10^5 vectors) of all possible input vectors (10^{60}) [36]. However, the method evaluating candidate circuits using a subset of input vectors is not applicable to approximate arithmetic circuits and other circuits that we treated as completely defined in Sect. 2.2.

In order to approximate complex circuits (belonging to the class of completely defined circuits) using the Hamming distance as a metric, we will use BDDs in the fitness function. This idea was initially proposed in our paper [37], but without a detailed experimental evaluation.

3 Binary decision diagrams in circuit design

3.1 Binary decision diagrams

A BDD is one of possible representations of logic functions. A BDD is a directed acyclic graph with one root, non-terminal nodes and two terminal nodes that are referred to '0' and '1'. Each non-terminal node is labeled by a primary input variable x_i. If $x_i = 0$ then the outgoing zero-edge is taken; if $x_i = 1$ then the outgoing one-edge is taken. By tracing a path from the root to terminal node '1' one obtains an assignment to input variables for which the function is evaluated to 1. An ordered binary decision diagram (OBDD) is a BDD where variables occur along every path from the root to a terminal node in strictly ascending order, with regard to fixed ordering. A reduced ordered binary decision diagram (ROBDD) is an OBDD where each node represents a unique logic function, i.e. it contains neither isomorphic subgraphs nor nodes with isomorphic descendants. Figure 1 shows a

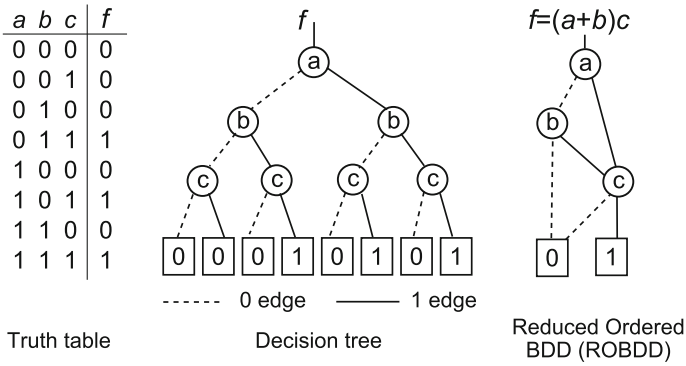

Fig. 1 Logic function $f = ac + bc$ expressed using truth table, BDD and ROBDD

Boolean function represented by truth table and corresponding BDD and ROBDD. Two or more logic functions can be represented by a single ROBDD (i.e. there are several root nodes) in which some subgraphs are shared by some of the functions.

ROBDDs are important because they are canonical, i.e. if two logic functions have the same truth table, their ROBDDs are isomorphic. Unfortunately, the size of ROBDD (i.e. the number of non-terminal nodes) for a given function is very sensitive to the chosen variable order; in some cases it is linear, in other cases is exponential with respect to the number of inputs [5]. Moreover, multipliers are known for their exponential memory requirements for any variable ordering [1]. In order to optimize the size of ROBDD, various minimization algorithms were proposed [5]. The most efficient method is sifting, an iterative algorithm which is based on finding the optimum position of each variable assuming that all other variables remain fixed.

BDDs and evolutionary computing have been combined in the past. For example, variable ordering of an BDD was optimized by EA [2], and an EA that learns heuristics for BDD minimization was proposed in [3]. Detailed survey is available in [5].

3.2 Operations over BDDs

ROBDDs are equipped with several operations. Let us mention two basic operations that are relevant to our paper: *apply* and *Sat-Count*.

The *apply*(op, f, g) operation enables to construct a ROBDD from existing ROBDDs. It takes a binary operator op and two ROBDDs f and g as arguments and returns a ROBDD corresponding with the result of f op g [18]. In fact, *apply* is a complex operation which can remove some nodes, add new nodes, and rearrange existing nodes to guarantee that the resulting BDD is a ROBDD.

The *Sat-Count* operation computes the number of input assignments for which f is evaluated to '1', i.e. it determines the number of elements in the so-called *Onset* of f. *Sat-Count* can be performed in time $O(|F|)$, where F is a ROBDD corresponding to f, just by following the leftmost path in F that leads to a non-

zero terminal. It means that the *Sat-Count* operator can be implemented in such a way that the number of input assignments for which $f = 1$ is obtained with linear time with respect to the size of ROBDD constructed for f. This is a very important feature in the context of evolutionary circuit design. Obtaining the same result using simulation requires $2^n|C|$ steps for n-input circuit C containing $|C|$ gates. On the other hand, it has to be noted that the worst case time complexity of BDD construction is exponential (see [5]), but it is not usually the case of circuits used in practice.

Several libraries have been developed to effectively construct (RO)BDDs and perform operations over them. In this work, Buddy package is employed [18].

3.3 Hamming distance using BDDs

BDDs are often used to decide whether two combinational circuits are functionally equivalent. Let us suppose that both circuits have k inputs denoted $x_1 \ldots x_k$ and m outputs denoted $y_1 \ldots y_m$ and $y'_1 \ldots y'_m$, respectively. Corresponding primary inputs of both circuits are aligned and corresponding primary outputs y_i and y'_i are connected using the XOR gates. The goal is to obtain one (auxiliary) circuit with k primary inputs $x_1 \ldots x_k$ and m primary outputs $z_1 \ldots z_m$, $z_i = y_i$ XOR y'_i. In order to disprove the equivalence, it is then sufficient to identify at least one output z_i whose $Onset(z_i)$ is not empty, i.e. to find an input assignment x for which the corresponding outputs y_i and y'_i provide different values. An example is given in Fig. 2 where two circuits C_A and C_B with four inputs and two outputs are checked for Boolean equivalence. Because y_2 and y'_2 capture the same Boolean function, the ROBDD constructed for z_2 consists of a single pointer to the zero node. The outputs y_1 and y'_1, however, represent different Boolean functions. The ROBDD constructed for z_1 thus consists of non-zero number of nodes and there exists at least one path from the root node determined by pointer z_1 to the node 1.

The auxiliary circuit used to perform the combinational equivalence checking can be applied to determine the Hamming distance between truth tables of circuit

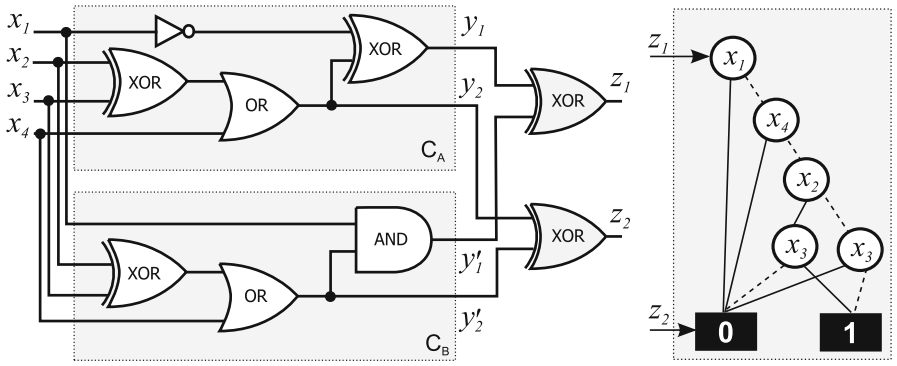

Fig. 2 Auxiliary circuit used to perform equivalence checking of two combinational circuits C_A and C_B (*left*) and ROBDD constructed for z_1 and z_2 (*right*)

C_A and C_B. The Hamming distance can be obtained by applying the *Sat-Count* operation on every output z_i and counting up all the results. In the example shown in Fig. 2, *Sat-Count* will return 2 for z_1 and 0 for z_2, i.e. the Hamming distance is $0 + 2 = 2$. It can easily be checked that if $x \in \{0000, 0110\}$, the circuits provide different output values. The Hamming distance is obtained in linear time with respect to the number of outputs.

4 Proposed method

The proposed method is based on the standard CGP [19]. The main contribution of this work is redefining the fitness calculation procedure in such a way that it can handle circuits with tens to hundreds of inputs, and showing how various compromises between the error, area and delay can be found.

Because many candidate approximate circuits have to be generated and evaluated during a typical CGP run, it is impossible to evaluate everyone using a professional design tool. Hence circuit parameters are estimated. This strategy was validated in [35].

It is assumed that the specification (i.e. an accurate circuit behavior) is given in a form of ROBDD (let us denote it σ). If not, a corresponding ROBDD is created from the accurate circuit using the *apply* operator as described in Sect. 3.2.

4.1 Circuit representation

A gate-level n_i-input/n_o-output circuit is represented using a directed acyclic graph which is encoded in a 1D array consisting of n_c gates. The number of rows, which is one of CGP parameters, is set to $n_r = 1$. This graph is internally stored using a string of integers, the so-called chromosome. The set of available logic functions is denoted Γ. The primary inputs are labeled $0 \ldots n_i - 1$ and the gates are labeled $n_i, n_i + 1, \ldots, n_i + n_c - 1$. For each gate, the chromosome contains three integers—two labels specifying where the gate inputs are connected to and a code of function in Γ. 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 directly be connected to the primary outputs. Example is given in Fig. 3.

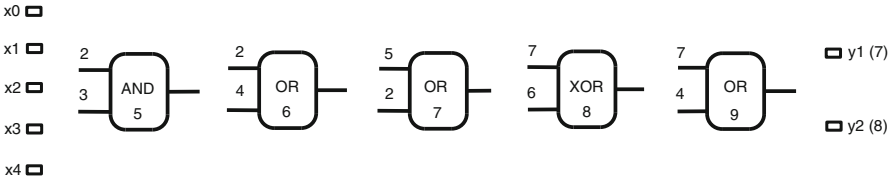

Fig. 3 Example of a circuit in CGP with parameters: $n_i = 5, n_o = 2,\ n_c = 5,\ \Gamma = \{0^{\text{and}}, 1^{\text{or}}, 2^{\text{xor}}\}$. Chromosome: 2, 3, 0; 2, 4, 1; 5, 2, 1; 7, 6, 2; 7, 4, 1; 7, 8. Gate 9 is not used. Its logic behavior is: $y_1 = (x_2 \text{ and } x_3)$ or x_2; $y_2 = y_1$ xor $(x_2$ or $x_4)$

The chromosome size is $3n_c + n_o$ genes (integers) if two-input gates are used. The main feature of this encoding is that the size of the chromosome is constant for a given n_i, n_o and n_c. However, the size of circuits represented by this chromosome is variable as some gates can remain disconnected. The gates which are included into the circuit after reading the chromosome are called the active gates.

4.2 From chromosome to ROBDD

Let circuit A be a candidate circuit represented using CGP. A new ROBDD, α, which is functionally equivalent with A has to be constructed. In order to do so, the number of BDD variables is defined firstly. Then, the *apply* function is called for every active gate N_j of circuit A. It consumes the logic function performed by N_j and two operands of N_j which are interpreted as pointers to appropriate ROBDD nodes or input variables. The function yields a pointer to a new ROBDD which represents the Boolean function at the output of gate N_j. Depending on the logic function of N_j, one or several new ROBDD nodes are thus included into α by means of one call of *apply*. The active nodes of A have to be processed from left to right in order to construct the ROBDD correctly.

4.3 Design objectives

There are three design objectives considered in this work: functionality (error), delay, and area.

4.3.1 Functionality

Circuit functionality is evaluated at the level of ROBDD and measured as the Hamming distance between the output bits generated by α and σ for all possible input combinations. The procedure follows the principle described and illustrated (Fig. 2) in Sect. 3.3. In particular, corresponding outputs of α and σ are connected to a set of exclusive-or gates, i.e. $z_i = y_i^\alpha$ xor y_i^σ for $i = 1, 2, \ldots, n_o$. By means of the *Sat-Count* operation, one can obtain the number of assignments b_i to the inputs which evaluate z_i to 1. Finally, the Hamming distance between α and σ, i.e. the fitness (functionality) of A, is the sum of b_i (see Eq. 1).

In order to accelerate this procedure, the ROBDD construction is optimized. We exploit the fact that the accurate circuit and candidate circuit (which was, in fact, created by a sequence of mutations over the accurate circuit) contain some identical subcircuits which can be removed for purposes of the Hamming distance calculation. ROBDD is then constructed using only those subcircuits which are not present in both circuits, i.e. the size of ROBDD is reduced.

4.3.2 Delay

In order to estimate the electrical parameters of circuit A, the area and delay are calculated using the parameters defined in the liberty timing file available for a

given semiconductor technology. Delay of a gate is modeled as a function of its input transition time and capacitive load on the output of the gate. Delay of the whole circuit is determined as delay along the longest path.

4.3.3 Relative area

The area of a candidate circuit is calculated relatively to the area of a nand gate. The following gates are considered in Γ: *and, or, xor, nand, nor, xnor, buf, inv*, with corresponding relative areas 1.333, 1.333, 2, 1, 1, 2, 1.333, and 0.667. It is assumed that power consumption is highly correlated with the area and hence it is sufficient to optimize for the area as proposed in [35].

4.4 Search method

The search method follows the standard CGP approach [19]. The initial population is seeded by the accurate circuit. In order to generate a new population, λ offspring individuals are created by a point mutation operator modifying h genes of the parent individual. The parent is either the accurate circuit (in the first generation) or the best circuit of the previous generation (in remaining generations).

One mutation can affect either the gate function, gate input connection, or primary output connection. A mutation is called neutral if it does not affect the circuit's fitness. If a mutation hits a non-used part of the chromosome, it is detected and the circuit is not evaluated in terms of functionality because it has the same fitness as its parent. Otherwise, the error is calculated. The best individual of current population serves as the parent of new population. The process is repeated until a given number of generations (or evaluations) is not exhausted.

The role of mutation is significant in CGP (see detailed analysis in [8, 20]). A series of neutral mutations can accumulate useful circuit structures in the part of the chromosome which is not currently used. One adaptive mutation can then connect these structures with active gates which could lead to discovering new useful circuits. It has to be noted that the mutation operates over chromosomes (not at the level of BDDs).

In order to construct Pareto front, we follow the approach in which a single-objective CGP (utilizing a linear aggregation of objectives) is executed multiple times with different target errors e_i ($e_i > 0$). It is assumed that Pareto front has to be constructed for v different errors $e_1 \ldots e_v$ (each expressed as a percentage of the average Hamming distance). An obvious criticism of this approach is that some solutions are never obtained and a classic multi-objective EA such as NSGA-II has to be used. Despite the fact that some hybridizations of NSGA-II and CGP have been proposed [13, 16], our previous studies in the area of evolutionary circuit approximation have shown that single-objective approaches provide better results [35, 36].

We propose a two-stage procedure for evolving an approximate circuit showing target error e_i using a single-objective CGP.

The first stage starts with a fully functional solution which is always available in practice. The goal is to gradually modify the accurate circuit and produce an

approximate circuit showing desired error e_i providing that a 5 % difference is tolerated with respect to e_i (tolerating a small error is acceptable; otherwise the search could easily stuck in a local extreme). The 5% error tolerance means that if, for example, $e_i = 0.3$ % then we accept all circuits showing the error 0.285...0.315 %. The fitness function fit_1 used in the first stage is thus solely based on the average Hamming distance (see Sect. 4.3.1),

$$fit_1 = Error(A) = \frac{\sum_{i=1}^{n_o} b_i}{2^{n_i}}. \quad (1)$$

In the second stage, which begins after obtaining a circuit with the target error, the fitness function reflects not only the error, but also the area and delay. Each objective is normalized to the interval $\langle 0, 1 \rangle$ and multiplied with weights w_e, w_a and w_d, respectively ($w_e + w_a + w_d = 1$). Then,

$$fit_2(A) = w_e Error(A) + w_a Area(A) + w_d Delay(A). \quad (2)$$

It is requested that the *Error* remains within 5 % tolerance with respect to e_i. Candidate circuits violating this hard constraint are discarded.

5 Results

This section firstly introduces benchmark circuits and CGP parameters used in all experiments. In order to check whether CGP can improve the results of conventional optimization and simultaneously obtain high-quality fully functional circuits which will be good starting points for subsequent approximations, CGP was employed to optimize the parameters of original (accurate) benchmark circuits. All results of approximations are represented as points in figures showing the objective space. Pareto fronts are constructed using the best obtained solutions.

5.1 Benchmarks

In order to evaluate the proposed approximation method which employs the Hamming distance to determine the error, we selected different types of combinational circuits from LGSynth, ITC and ISCAS libraries [7]. Even some arithmetic circuits (e.g. c3540) which should be approximated under an arithmetic error metric are included. The chosen circuits are difficult for the standard CGP, because $n_i > 25$ and more than 150 gates are involved [19].

At the beginning, all benchmark circuits were optimized using BDS [42] to get a reference solution from a "conventional" state-of-the art logic optimizer. Table 1 gives the number of primary inputs (n_i) and primary outputs (n_o), and then the number of gates (n_g) and delay (in terms of logic levels) after the optimization conducted by BDS. Table 1 also gives parameters of corresponding BDDs which serve as reference implementations used for Hamming distance calculations. The BDD size (see column $|BDD|$), obtained using [18], ranges from 321 nodes (itc_b10) to more than one million nodes (c3540). Because the size of BDD used as a

Table 1 Parameters of benchmark circuits

Circuit	Circuit parameters				Reference circuit			
	n_i	n_o	Gates	Levels	$\|BDD\|$	$\|BDD_{opt}\|$	$gain_{opt}$ (%)	t_{opt}
apex1	45	45	823	15	7073	1344	81	2.5
c1355	41	32	186	11	148,003	38,481	74	4.3
c3540	50	22	868	27	1,014,533	30,436	97	57.9
c432	36	7	159	25	167,300	1673	99	9.5
clmb[a]	46	33	641	19	6966	627	91	2.3
itc_b05[a]	34	56	427	24	18,788	1691	91	1.3
itc_b07[a]	49	49	312	25	11,055	995	91	3.1
itc_b10[a]	27	17	166	10	321	222	31	1.1
itc_b11[a]	37	31	421	18	1552	652	58	1.3
s1238[a]	31	31	483	18	1822	729	60	1.2
s635[a]	34	33	151	10	394	134	66	0.7
signet	39	8	630	17	11,471	1606	86	1.3
too_large	38	3	771	18	3508	807	77	3.3
x1dn	27	6	164	18	896	260	71	0.9
x6dn	39	5	318	14	3685	258	93	1.7
x9dn	27	7	168	20	484	218	55	0.5

The circuits that originally come from sequential benchmarks (i.e. represent a combinational subcircuit of a sequential circuit) are marked by superscript a

reference influences how fast the Hamming distance calculation and the whole evolutionary design can be, it is beneficial to optimize the BDD. The operations over small and optimized BDDs will then be performed faster than over original BDDs. Hence, we applyied *sifting* minimization algorithm to reduce the size of BDD. The results are reported in column $|BDD_{opt}|$. The improvement due sifting, which is 75.4 % on average, is given in the $gain_{opt}$ column. The last column t_{opt} is the time spent in the sifting procedure (in s). As it can be seen, the optimization of the variable order is able to ensure significant savings in the number of BDD nodes required to represent a reference circuit for a small cost of runtime. Note that this optimization is performed just once, before CGP is executed.

5.2 CGP setup

As the purpose of this paper is not to perform a detailed analysis of the CGP parameters setting, we used CGP with parameters that are usually reported in the literature. According to [26], the weights are chosen to be $w_e = 0.12$, $w_a = 0.5$, $w_d = 0.38$ and the CGP setup is: $\lambda = 4$, $h = 5$, Γ as defined in Sect. 4.3.3, and n_c is the number of gates in a particular benchmark circuit (according to Table 1). The experiments were conducted on a 64-b Linux machine running on Intel Xeon X5670 CPU (2.93 GHz, 12 MB cache) equipped with 32 GB RAM. CGP is implemented as a single-thread application.

5.3 Optimization of accurate circuits

First, CGP was employed to optimize the original benchmark circuits, i.e. no errors tolerated, $Error(A) = 0$. This step was performed because it has been known that a significant area reduction can be obtained by means of CGP [32].

A single CGP run was terminated after 30 min which seems to be a good compromise between requirements of practitioners expecting short optimization times and resources-demanding CGP. The number of generations was not specified as the termination criterion because the benchmark circuits have significantly different properties and, for example, different time is needed to process corresponding BDDs.

Table 2 gives parameters of the best and average circuit (obtained out of ten independent runs) with respect to parameters of original benchmarks: fitness (fit_2), the generation in which the best circuit was reached, and time to obtain the best circuit (runtime). It can be seen that the evolutionary optimization can lead in many cases to a significant delay, area (and, consequently, energy) reduction without introducing any error into the circuit. For example, an 80 % area improvement is reported for too_large benchmark circuit with respect to BDS. This particular circuit is hard for conventional optimization methods. Moreover, the parameters of the average circuits (determined as a median of all the runs) are close to the parameters of the best obtained circuit. C3540 and C1315 circuits represent the only exception where nearly none improvement is reported. A single gate was removed during optimization in both cases. Because the runtimes are close to the time limit, the majority of the benchmark circuits would be probably improved if more time is available to the evolution.

5.4 Evolutionary approximation

The circuits evolved in Sect. 5.3 will be used as (reference) accurate circuits in Pareto fronts. CGP-based approximations are performed from these accurate circuits for errors from $e_1 = 0.1$ to $e_9 = 0.9$ % given in terms of the Hamming distance. Results are presented from ten independent 30 min runs (one run one thread).

Firstly, we analyzed the first stage of the approximation. We calculated the time required to get an approximate circuit showing desired error e_i providing that an accurate circuit is used as a seed. In most cases, <1 s is required to find such a circuit. This corresponds with hundreds to few thousands of evaluated generations. Table 3 summarizes the cases in which more than 1 s is needed. The most difficult cases are c1355, c3540 and c432, and in particular 0.1–0.51 % error in case of c432, where some of the CGP runs spent more than 5 min. This was expected for c3540 and c1355 circuits because they are large and their BDDs are complex. A possible explanation for C432 is that the target error is too small. This findings is based on the fact that the number of evaluated generations is high (more than 15,000 genenerations) and that the achieved reduction in the area is low compared to the other benchmarks (see Fig. 4). We can, however, conclude that desired approximations can be reached relatively quickly if the other objectives are not taken into account. The chosen search strategy seems to be very efficient in this task despite

Table 2 Parameters of the best and average accurate circuits obtained using CGP

Circuit	Best obtained circuit						Average circuit (median)				
	fit_2	Generation	Runtime (min)	Improvement			fit_2	Runtime (min)	Improvement		
				Gates (%)	Area (%)	Delay (%)			Gates (%)	Area (%)	Delay (%)
apex1	0.96	1.6×10^6	28	3	3	5	0.96	26	2	3	5
c1355	1.00	7.6×10^3	28	<1	<1	0	1.00	21	0	0	<1
c3540	1.00	4.9×10^3	11	<1	<1	0	1.00	0	0	0	0
c432	0.97	63.4×10^3	18	2	3	2	0.97	17	1	2	3
clmb	0.65	2.9×10^6	29	36	36	26	0.70	29	31	32	20
itc_b05	0.94	0.5×10^6	29	5	6	1	0.95	28	4	5	1
itc_b07	0.97	0.4×10^6	7	<1	1	7	0.97	8	<1	1	6
itc_b10	0.91	3.7×10^6	13	7	6	18	0.91	21	7	6	16
itc_b11	0.96	2.5×10^6	26	4	4	1	0.97	29	3	3	1
s1238	0.85	2.8×10^6	28	13	15	12	0.86	29	13	14	13
s635	0.94	10.0×10^6	25	3	2	24	0.96	24	4	3	6
signet	0.58	1.2×10^6	29	43	45	30	0.62	28	41	42	18
too_large	0.30	1.0×10^6	29	79	80	28	0.33	25	76	77	26
x1dn	0.51	4.7×10^6	20	48	50	45	0.63	27	32	36	39
x6dn	0.90	2.9×10^6	25	11	12	3	0.92	26	8	9	3
x9dn	0.80	3.5×10^6	24	20	23	8	0.82	15	20	20	8

Table 3 Time (in s) spent in the first stage of the optimization

Circuit	$e_1 = 0.1$ %			$e_3 = 0.3$ %			$e_5 = 0.5$ %			$e_7 = 0.7$ %			$e_9 = 0.9$ %		
	t_{Q1}	t_{med}	t_{Q3}	t_{Q1}	t_{med}	t_{Q3}	t_{Q1}	t_{med}	t_{Q3}	t_{Q1}	t_{med}	t_{Q3}	t_{Q1}	t_{med}	t_{Q3}
c1355	9.2	27	82	32	58	100	12	48	383	12	19	30	3.2	3.2	3.2
c3540	5.9	86	168	17	70	145	7.2	17	36	18	24	49	7.7	29	56
c432	10	33	305	24	41	478	5.8	173	492	4.8	50	223	16	33	201
itc_b05	<1	<1	<1	<1	<1	2.2	<1	<1	1.6	<1	<1	1.3	<1	<1	2.3
itc_b07	<1	<1	1.3	<1	<1	1.3	<1	<1	<1	<1	<1	<1	<1	<1	<1
x1dn	<1	<1	<1	<1	<1	<1	<1	<1	4.1	<1	<1	<1	<1	<1	<1

The median value (t_{med}), the lower bound (t_{Q1}) and upper bound (t_{Q3}) of the interquartile range are calculated using all runs

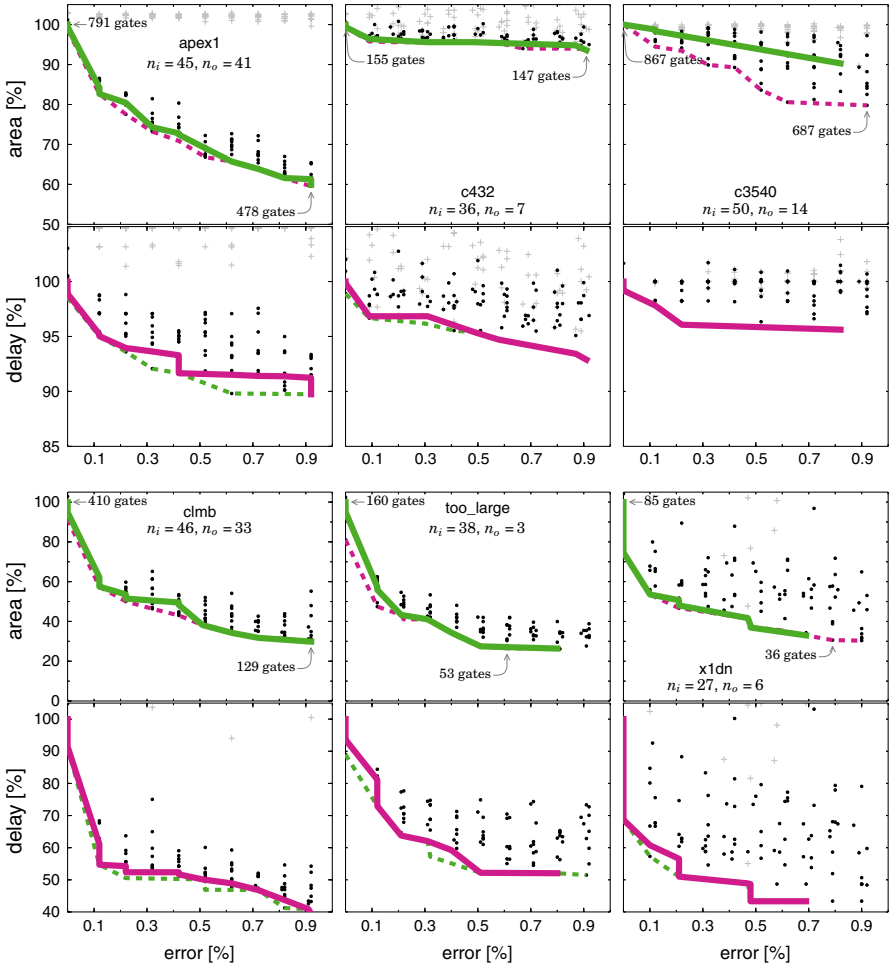

Fig. 4 Pareto fronts for benchmarks apex1, c432, c3540, clmb, too_large and x1dn

the fact that finding a circuit exhibiting a required error is in general a nontrivial task.

Next we measured the time needed to calculate the Hamming distance between two circuits with respect to a given error. This time (expressed as a median value) summarized from all the experiments is given in Table 4. This value influences how many candidate solutions can be evaluated within a period of time in average. The lower value, the higher number of evaluated candidate solutions. Stages 1 (searching for a circuit with a given error) and 2 (optimizing the area, and delay of the circuit) are handled separately. In most cases, the time is less than a few milliseconds. There are two cases (c1355, c3540) in which few 100s (1000s for c1355) of milliseconds are required to determine the Hamming distance. A consequence is that fewer generations can be produced within a given time for this circuit. This effect is clearly visible in Fig. 5 where the resulting circuit approximations are mostly far from the optimum (see black dots for errors higher than 0.4 %).

If we compare the mean time needed to determine the Hamming distance in the first and second stage (see last five columns of Table 4 showing the ratio between the first and second stage), it is evident that more evaluations per second can be performed in the second stage of the optimization. The reason is that BDDs are in average smaller than in the first stage. However, it can be also seen that the time needed to evaluate the Hamming distance of c1355 benchmark circuit increases with the increasing error. Thousands of milliseconds are needed in this particular case.

The resulting Pareto fronts are displayed in Figs. 4, 5, 6 and 7 (solid lines). For each circuit, two plots are presented: the best obtained area versus error and delay versus error, relatively to the fully functional circuit from Table 2 labeled by 100 %. The result of a single 30-min CGP run consisting of two stages is shown using a black dot. The plus symbol (+) indicates the results of the first stage. In several cases, the + symbols are not visible because they are outside the plotted areas. However, for example, the plot for s635 clearly shows that in most cases the first stage produced circuits within about 80–100 % in the area axis (corresponding to <1 s in average according to Table 3) while the second stage led in remaining 29.9 min (on average) to a significant improvement (about 30 % in the area axis).

Figures 4, 5, 6 and 7 contain the best compromises from independent CGP runs in which all circuits parameters (error, area and delay) are optimized together. For some applications it is interesting to know the best compromises for two objectives only (error vs. delay and area vs. delay). These compromises are plotted as additional Pareto fronts (dashed line). The number of gates is explicitly given for the biggest and smallest circuits.

For example, by increasing the error, the area was reduced by 70 % in the case of clmb circuit providing that the delay is adequately reduced. A general observation is that an improvement in delay is smaller than in area when the error is increasing. The reason is that fully functional benchmark circuits exhibit a relatively small delay and hence there is a little space to improve it. Two interesting cases are c432 and c3540 because their area requirements are high even for reduced accuracy. One reason could be that errors <1 % are too small to get a reasonable approximation.

Table 4 Time (in ms) required to calculate the Hamming distance in stage 1 (t_{med}, t_{Q3}) and stage 2 (t^*_{med})

Circuit	$e_1 = 0.1$ %		$e_3 = 0.3$ %		$e_5 = 0.5$ %		$e_7 = 0.7$ %		$e_9 = 0.9$ %		t_{med}/t^*_{med}				
	t_{med}	t_{Q3}	t_{med}	t_{Q3}	t_{med}	t_{Q3}	t_{med}	t_{Q3}	t_{med}	t_{Q3}	e_1	e_3	e_5	e_7	e_9
apex1	1.3	1.5	1.2	1.5	1.2	1.4	1.4	1.9	1.4	1.6	1.4	1.3	1.8	1.7	2.0
c1355	198	215	278	389	640	3083	583	704	665	665	1.5	4.6	0.3	0.1	0.1
c3540	162	200	173	192	155	161	195	215	197	216	1.0	1.1	1.1	1.3	1.3
c432	16	17	16	17	19	20	16	18	17	19	1.3	1.2	1.3	1.2	1.5
clmb	0.7	1.1	1.2	1.6	1.3	2.1	1.5	2.5	1.1	1.4	3.7	6.0	6.7	14	10
itc_b05	5.2	5.5	4.5	5.8	8.6	11	5.5	7.9	5.2	5.8	2.3	2.4	5.7	3.2	4.4
itc_b07	1.2	1.6	1.2	1.6	1.3	1.9	1.2	1.3	1.4	1.7	2.0	1.7	2.1	2.3	2.5
itc_b10	0.2	0.3	0.2	0.2	0.2	0.2	0.2	0.3	0.2	0.2	1.0	1.0	0.8	1.0	1.0
itc_b11	0.6	0.7	0.6	0.8	0.6	0.9	0.5	0.6	0.7	0.8	1.1	1.1	1.2	1.1	1.4
s1238	0.7	0.7	0.8	0.9	0.7	0.9	0.7	1.2	0.7	0.8	1.7	2.0	2.4	2.4	2.2
s635	0.2	0.3	0.1	0.3	0.2	0.3	0.3	0.8	0.2	0.3	2.2	1.3	2.2	2.6	2.0
signet	1.9	2.6	2.2	2.5	2.8	3.6	2.7	4.0	2.2	2.7	1.5	1.8	2.3	2.4	2.5
too_large	7.9	8.8	8.5	11	8.7	10	12	17	8.4	11	22	28	28	41	42
x1dn	0.4	0.6	0.3	0.4	0.3	0.4	0.8	1.1	0.3	0.6	2.7	3.1	3.5	5.1	3.5
x6dn	0.2	0.4	0.3	0.4	0.3	0.4	0.4	0.5	0.2	0.3	0.6	0.9	1.0	1.1	1.1
x9dn	0.5	0.7	0.5	0.5	0.3	0.5	0.6	1.2	0.4	0.5	1.6	1.8	1.7	3.2	3.9

The median value (t_{med}, t^*_{med}) and upper bound (t_{Q3}) of the interquartile range are calculated from run-time of all fitness function calls

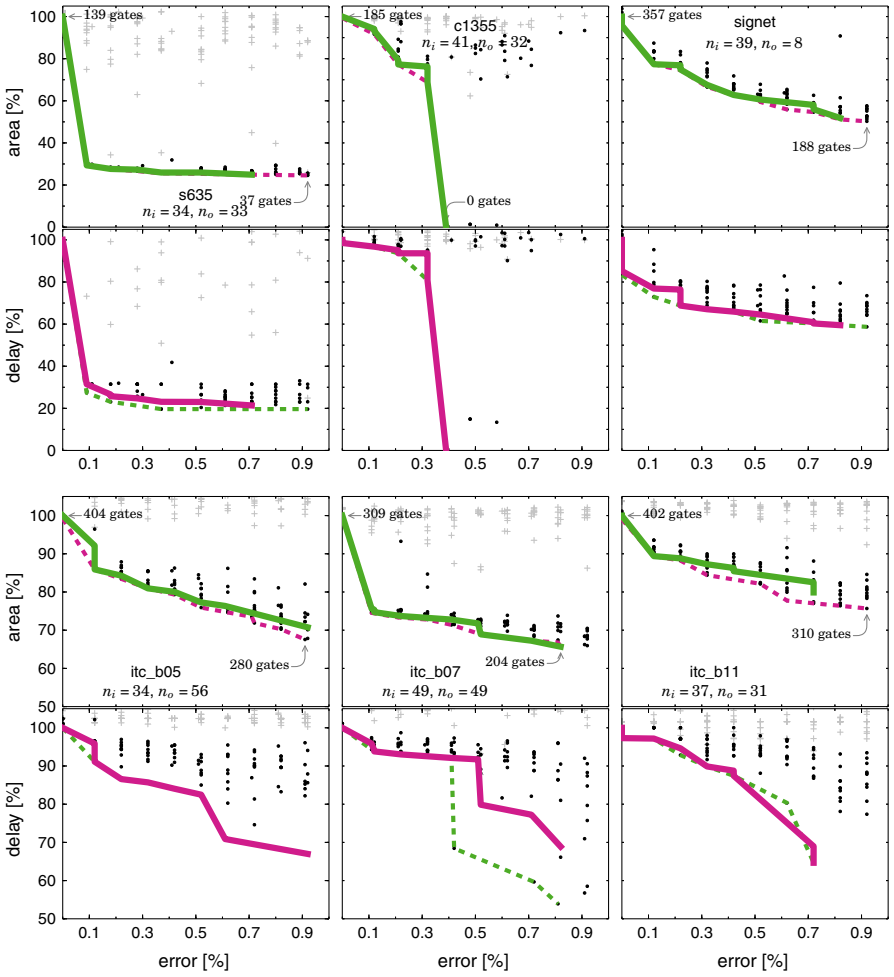

Fig. 5 Pareto fronts for benchmarks s635, c1355, signet, itc_b05, itc_b07 and itc_b11

Another reason could be that more generations are required to reduce the area. Hence we tried to prolong the evolution six times. However, no significant changes in the Pareto front have been observed (not shown in the paper). On the other hand, 0.4 % seems to be a huge error for c1355 because it allowed CGP to remove almost all gates from the circuit.

In some cases (e.g. apex1, c3540, s635), the independent CGP runs led to very similar results for a given e_i (see the black dots). In other cases (e.g. itc_b05, x1dn, c1355), the spread in the area and delay is quite large. This indicates that circuits have different structural properties and that their selection to our benchmark set is justified.

Figure 7 shows a detailed analysis of some of the discovered approximations for itc_b10 benchmark. For each of five target errors, three evolved circuits were

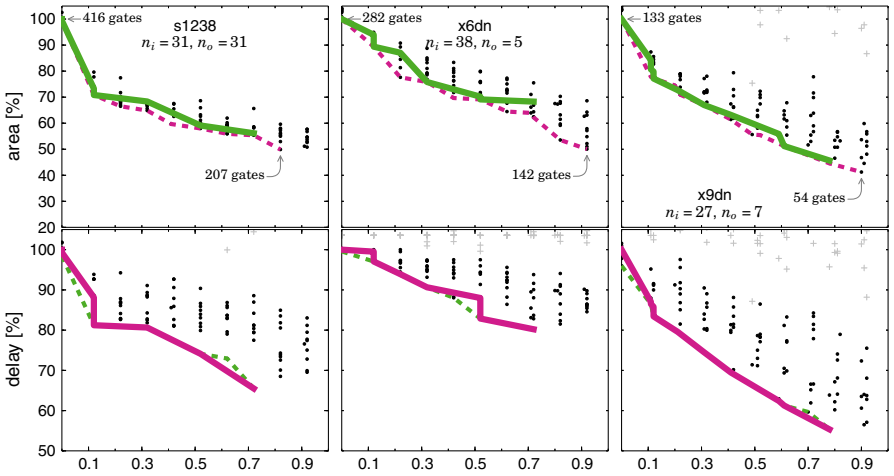

Fig. 6 Pareto fronts for benchmarks s1238, x6dn, and x9dn

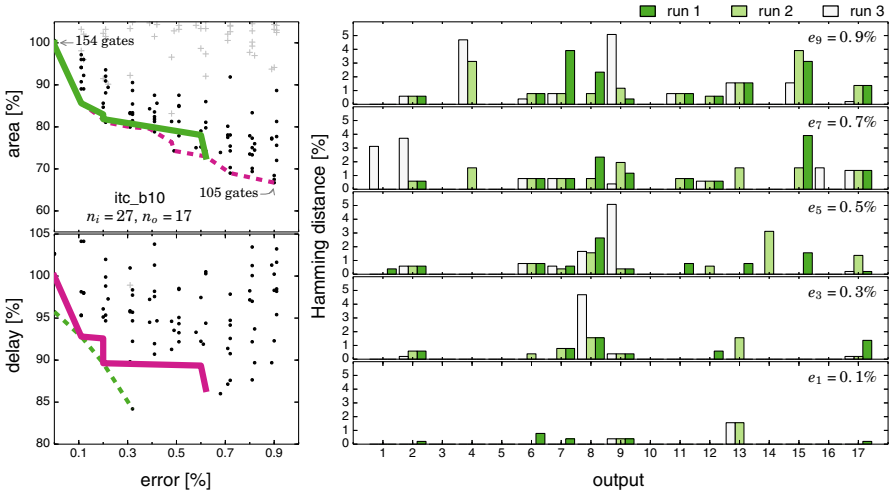

Fig. 7 Pareto front for benchmark itc_b10 (*left*) and the detailed analysis (Hamming distance of each output) of three evolved approximations for e_1, e_3, e_5, e_7 and e_9 (*right*)

chosen and their Hamming distances were calculated independently for all 17 outputs. It can be seen that the obtained solutions have in general different properties. The same solution was obtained only in the case of $e_1 = 0.1$ %, where run 2 and run 3 discovered a circuit with error in two outputs. The first run produced a completely different circuit. Although five output signals are affected by error in this case, the worst-case difference (Hamming distance) is not worse than 0.7 %.

Finally, we raised a question whether it is better to intensively optimize accurate circuits or introduce approximations in order to reduce the area. Table 5 shows how

Table 5 The number of gates when subsequent optimizations are applied

Method	BDS	CGP			
Error	0 %	0 %	0.1 %	0.5 %	0.9 %
clmb	641 (100 %)	410 (63 %)	250 (39 %)	167 (26 %)	129 (20 %)
s1238	483 (100 %)	416 (86 %)	298 (61 %)	241 (49 %)	213 (44 %)
signet	630 (100 %)	357 (56 %)	288 (45 %)	223 (35 %)	188 (29 %)
too_large	771 (100 %)	160 (20 %)	94 (12 %)	55 (7 %)	56 (7 %)
x1dn	164 (100 %)	85 (51 %)	61 (37 %)	44 (26 %)	36 (21 %)
x6dn	318 (100 %)	282 (88 %)	254 (79 %)	196 (61 %)	142 (44 %)
x9dn	168 (100 %)	133 (79 %)	103 (61 %)	72 (42 %)	54 (32 %)

subsequent optimizations reduced the number of gates in circuits displaying at least 10 % area improvement with respect to BDS. For example, CGP-based optimization of too_large circuit caused that 80 % gates were removed without any impact on the accuracy. A subsequent approximation (error = 0.1 %) removed only 8 % gates. It turns out that the impact of a proper logic optimization conducted in the standard scenario (no errors are allowed) can be, in fact, higher than when the approximations are introduced.

6 Conclusions

In this paper, we proposed a new CGP-based method which allowed us to approximate non-trivial combinational circuits. Employing a BDD package in the fitness function enabled to reduce the fitness evaluation time, which is the most contributing component to the total time of evolution. The error was expressed in terms of the Hamming distance—the error measure which can be applied for general logic approximation. Pareto fronts show reasonable tradeoffs between key circuit parameters which one would expect for combinational approximate circuits. Unfortunately, no results compatible with our scenario are available in the literature for comparison.

Our method consists of two stages. In the first stage, a circuit showing desired error is evolved from a fully functional solution. As the initial approximation is performed in order of seconds (10s–100s of seconds in the case of more complex circuits), the user thus quickly obtains a circuit with desired functionality. Additional optimizations of the area and delay are then performed in the second stage which can be terminated when a suitable tradeoff is reached. This approach allowed us to find high quality solutions in a relatively short time, which is important for practice. It was also shown that a significant area reduction can be obtained just by enabling the evolutionary optimization of the accurate circuit after performing its usual conventional optimization (Table 2).

Despite the fact that by means of BDDs we were able to approximate relatively complex circuits (10s of inputs, 100s of gates), the usage of BDDs represents an inherent weakness of the method. As we pointed out in Sect. 3, BDDs can grow exponentially for some functions. Hence it is important to find a more

suitable formal model and corresponding algorithms which will allow us to further extend the class of circuits that can be approximated by CGP.

In our future work, we also plan to combine our method with a truly multiobjective evolutionary algorithm in order to obtain a Pareto front in a single run. We will evaluate if the overhead associated with the multiobjective optimizer can lead to results which are competitive with the obtained ones.

Acknowledgments This work was supported by the Czech science foundation Project 14-04197S.

References

1. R.E. Bryant, On the complexity of VLSI implementations and graph representations of Boolean functions with application to integer multiplication. IEEE Trans. Comput. **40**(2), 205–213 (1991)
2. R. Drechsler, B. Becker, N. Göckel, Genetic algorithm for variable ordering of obdds. IEE Proc. Comput. Digit. Tech. **143**(6), 364–368 (1996)
3. R. Drechsler, N. Göckel, B. Becker, in *Learning Heuristics for OBDD Minimization by Evolutionary Algorithms*. Parallel Problem Solving from Nature—PPSN IV. Lecture Notes in Computer Science, vol. 1141 (Springer, Berlin, 1996), pp. 730–739
4. M. Duranton, K. DeBosschere, A. Cohen, J. Maebe, H. Munk, in *Hipeac Vision 2015*. Technical report (HiPEAC Network of Excellence, 2015). https://www.hipeac.net/publications/vision/
5. R. Ebendt, G. Fey, R. Drechsler, *Advanced BDD Optimization* (Springer, Berlin, 2000)
6. H. Esmaeilzadeh, A. Sampson, L. Ceze, D. Burger, Neural acceleration for general-purpose approximate programs. Commun. ACM **58**(1), 105–115 (2015)
7. P. Fiser, *Collection of Digital Design Benchmarks* (Czech Technical University in Prague, Prague). http://ddd.fit.cvut.cz/prj/Benchmarks
8. B.W. Goldman, W.F. Punch, Analysis of cartesian genetic programming's evolutionary mechanisms. IEEE Trans. Evol. Comput. **19**(3), 359–373 (2015)
9. D. Grochol, L. Sekanina, M. Zadnik, J. Korenek, V. Kosar, Evolutionary circuit design for fast FPGA-based classification of network application protocols. Appl. Soft Comput. **38**, 933–941 (2016). doi:10.1016/j.asoc.2015.09.046
10. V. Gupta, D. Mohapatra, A. Raghunathan, K. Roy, Low-power digital signal processing using approximate adders. IEEE Trans. CAD Integr. Circuits Syst. **32**(1), 124–137 (2013)
11. J. Han, M. Orshansky, in *Approximate Computing: An Emerging Paradigm for Energy-Efficient Design*. Proceedings of the 18th IEEE European Test Symposium (IEEE, 2013), pp. 1–6
12. T. Higuchi, T. Niwa, T. Tanaka, H. Iba, H. de Garis, T. Furuya, in *Evolving Hardware with Genetic Learning: A First Step Towards Building a Darwin Machine*. Proceedings of the 2nd International Conference on Simulated Adaptive Behaviour (MIT Press, Cambridge, 1993), pp. 417–424
13. R. Hrbacek, in *Parallel Multi-objective Evolutionary Design of Approximate Circuits*. Proceedings of the 2015 Conference on Genetic and Evolutionary Computation (GECCO '15) (ACM, 2015), pp. 687–694
14. R. Hrbacek, L. Sekanina, in *Towards Highly Optimized Cartesian Genetic Programming: From Sequential Via SIMD and Thread to Massive Parallel Implementation*. Proceedings of the Conference on Genetic and Evolutionary Computation (ACM, 2014), pp. 1015–1022
15. M. Iqbal, W.N. Browne, M. Zhang, Reusing building blocks of extracted knowledge to solve complex, large-scale Boolean problems. IEEE Trans. Evol. Comput. **18**(4), 465–480 (2014)
16. P. Kaufmann, T. Knieper, M. Platzner, in *A Novel Hybrid Evolutionary Strategy and Its Periodization with Multi-objective Genetic Optimizers*. IEEE Congress on Evolutionary Computation (CEC) (IEEE, 2010), pp. 1–8
17. P. Kulkarni, P. Gupta, M.D. Ercegovac, Trading accuracy for power in a multiplier architecture. J. Low Power Electron. **7**(4), 490–501 (2011)
18. J. Lind-Nielsen, H. Cohen, in *BuDDy—A Binary Decision Diagram Package*. http://sourceforge.net/projects/buddy/
19. J.F. Miller, *Cartesian Genetic Programming* (Springer, Berlin, 2011)
20. J.F. Miller, S.L. Smith, Redundancy and computational efficiency in cartesian genetic programming. IEEE Trans. Evol. Comput. **10**(2), 167–174 (2006)

21. A. Mishchenko, in *ABC: A System for Sequential Synthesis and Verification* (Berkeley Logic Synthesis and Verification Group, University of California, Berkeley, CA, US, 2012). http://www.eecs.berkeley.edu/~alanmi/abc/
22. M. Murakawa, S. Yoshizawa, I. Kajitani, T. Furuya, M. Iwata, T. Higuchi, in *Evolvable Hardware at Function Level*. Parallel Problem Solving from Nature (PPSN IV). LNCS, vol. 1141 (Springer, Berlin, 1996), pp. 62–71
23. K. Nepal, Y. Li, R.I. Bahar, S. Reda, in *Abacus: A Technique for Automated Behavioral Synthesis of Approximate Computing Circuits*. Proceedings of the Conference on Design, Automation and Test in Europe (DATE '14) (EDA Consortium, 2014), pp. 1–6
24. R. Poli, J. Page, Solving high-order Boolean parity problems with smooth uniform crossover, sub-machine code GP and demes. Genet. Program. Evol. Mach. **1**(1–2), 37–56 (2000)
25. A. Sanchez-Clemente, L. Entrena, M. Garcia-Valderas, in *Error Masking with Approximate Logic Circuits Using Dynamic Probability Estimations*. 20th International On-Line Testing Symposium (IOLTS) (IEEE, 2014), pp. 134–139
26. L. Sekanina, Z. Vasicek, in *Evolutionary Computing in Approximate Circuit Design and Optimization*. 1st Workshop on Approximate Computing (WAPCO 2015) (2015), pp. 1–6
27. A.P. Shanthi, R. Parthasarathi, Practical and scalable evolution of digital circuits. Appl. Soft Comput. **9**(2), 618–624 (2009)
28. E. Stomeo, T. Kalganova, C. Lambert, Generalized disjunction decomposition for evolvable hardware. IEEE Trans. Syst. Man Cybern. Part B **36**(5), 1024–1043 (2006)
29. A. Thompson, P. Layzell, S. Zebulum, Explorations in design space: unconventional electronics design through artificial evolution. IEEE Trans. Evol. Comput. **3**(3), 167–196 (1999)
30. J. Torresen, A scalable approach to evolvable hardware. Genet. Program. Evol. Mach. **3**(3), 259–282 (2002)
31. Z. Vasicek, in Cartesian GP in *Optimization of Combinational Circuits with Hundreds of Inputs and Thousands of Gates*. Proceedings of the 18th European Conference on Genetic Programming—EuroGP, LCNS no. 9025 (Springer, Berlin, 2015), pp. 139–150
32. Z. Vasicek, L. Sekanina, Formal verification of candidate solutions for post-synthesis evolutionary optimization in evolvable hardware. Genet. Program. Evol. Mach. **12**(3), 305–327 (2011)
33. Z. Vasicek, L. Sekanina, in *Evolutionary Design of Approximate Multipliers Under Different Error Metrics*. IEEE International Symposium on Design and Diagnostics of Electronic Circuits and Systems (IEEE, 2014), pp. 135–140
34. Z. Vasicek, L. Sekanina, in *How to Evolve Complex Combinational Circuits from Scratch?* IEEE International Conference on Evolvable Systems Proceedings (IEEE, 2014), pp. 133–140
35. Z. Vasicek, L. Sekanina, in *Circuit Approximation Using single- and Multi-objective Cartesian GP*. Proceedings of the 18th European Conference on Genetic Programming— (EuroGP), LNCS no. 9025 (Springer, Berlin, 2015), pp. 217–229
36. Z. Vasicek, L. Sekanina, Evolutionary approach to approximate digital circuits design. IEEE Trans. Evol. Comput. **19**(3), 432–444 (2015)
37. Z. Vasicek, L. Sekanina, in *Evolutionary Approximation of Complex Digital Circuits*. Genetic and Evolutionary Computing Conference (ACM, 2015), pp. 1505–1506
38. V. Vassilev, D. Job, J.F. Miller, in *Towards the Automatic Design of More Efficient Digital Circuits*. Proceedings of the 2nd NASA/DoD Workshop on Evolvable Hardware (IEEE Computer Society, Los Alamitos, 2000), pp. 151–160
39. S. Venkataramani, K. Roy, A. Raghunathan, in *Substitute-and-Simplify: A Unified Design Paradigm for Approximate and Quality Configurable Circuits*. Design, Automation and Test in Europe (DATE '13) (EDA Consortium, San Jose, 2013), pp. 1367–1372
40. S. Venkataramani, A. Sabne, V.J. Kozhikkottu, K. Roy, A. Raghunathan, in *SALSA: Systematic Logic Synthesis of Approximate Circuits*. The 49th Annual Design Automation Conference (DAC '12) (ACM, 2012), pp. 796–801
41. R. Venkatesan, A. Agarwal, K. Roy, A. Raghunathan, in *MACACO: Modeling and Analysis of Circuits for Approximate Computing*. IEEE/ACM International Conference on Computer-Aided Design (ICCAD) (IEEE, 2011), pp. 667–673
42. C. Yang, M. Ciesielski, BDS: a BDD-based logic optimization system. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **21**(7), 866–876 (2002)