On Evolutionary Synthesis of Compact Polymorphic Combinational Circuits[⋆]

Zbyšek Gajda[1†], Lukáš Sekanina[1‡]

Faculty of Information Technology, Brno University of Technology, Czech Republic

Polymorphic gates are unconventional circuit components that are not supported by existing synthesis tools. This article presents new methods for synthesis of polymorphic circuits. Proposed methods, based on polymorphic binary decision diagrams and polymorphic multiplexing, extend the ordinary circuit representations with the aim of including polymorphic gates. In order to reduce the number of gates in circuits synthesized using proposed methods, an evolutionary optimization based on Cartesian Genetic Programming (CGP) is implemented. The implementations of polymorphic circuits optimized by CGP represent the best known solutions if the number of gates is considered as the decision criterion.

Key words: polymorphic circuit, digital circuit synthesis, evolutionary computing, genetic programming

1 INTRODUCTION

Polymorphic electronics was introduced by A. Stoica's group at NASA Jet Propulsion Laboratory as a new class of electronic devices that exhibit a new

[⋆] This is authors' manuscript of the paper: Gajda Zbysek, Sekanina Lukas: On Evolutionary Synthesis of Compact Polymorphic Combinational Circuits. Journal of Multiple-Valued Logic and Soft Computing, Vol. 17, No. 6, 2011, p. 607-631. For the final version, see Old City Publishing, Inc. at http://www.oldcitypublishing.com/MVLSC/MVLSCcontents/MVLSCv17n5-6contents.html

[†] email: `gajda@fit.vutbr.cz`

[‡] email: `sekanina@fit.vutbr.cz`

style of (re)configuration [28]. Polymorphic gates play the central role in the polymorphic electronics. A polymorphic gate is capable of switching among two or more logic functions. However, selection of the function is performed unconventionally. Logic function of a polymorphic gate depends on some external factors, e.g. on the level of the power supply voltage (V_{dd}), temperature, light or some other external signals [28, 29, 27, 32, 20]. For example, Stoica's polymorphic bifunctional NAND/NOR gate controlled by V_{dd} operates as NOR for V_{dd} = 1.8 V and NAND for V_{dd} = 3.3 V [27]. In fact, polymorphic gates merge the capability of performing logic operations with sensing. Hence polymorphic gates would be also very useful in building the *embodied intelligence*—intelligent devices whose function emerges in an interaction with a physical environment [3]. Although polymorphic gates can be implemented relatively effectively using current CMOS technology, we can expect an expansion of polymorphic devices with further development of nanoelectronics and molecular electronics.

Having polymorphic gates, researchers have begun to develop new methods for synthesis of digital circuits that contain polymorphic gates [22, 15, 25, 7]. Main motivation is to obtain reconfigurable (and thus potentially adaptive) circuits for a very low cost and without the need to implement a reconfiguration infrastructure (switches, multiplexers, configuration registers etc.). The goal of the polymorphic circuit synthesis can be formulated as a problem of finding such a circuit which performs required functions $f_1 \dots f_k$ in modes $1 \dots k$ of polymorphic gates [22]. Figure 1 shows an example of polymorphic digital circuit and its equivalent behavior in both modes of the polymorphic NAND/NOR gate (i.e., $k = 2$, $f_1 = \overline{i_0 \wedge i_1} \oplus i_2$ and $f_2 = \overline{i_0 \vee i_1} \oplus i_2$). Note that the method used to physically control the mode of polymorphic gates is not important in the proposed synthesis problem formulation. Unfortunately, conventional synthesis algorithms are not directly applicable for solving the polymorphic circuit synthesis problem which is, in fact, a more difficult case of the classic digital circuit synthesis problem.

In general, the design of an efficient digital circuit synthesis algorithm is a well-known problem that has been approached by many researchers in the recent decades. A synthesis algorithm operates over a circuit representation. Various models have been devised to represent digital circuits in such a form which is suitable for synthesis algorithms. Among others, Boolean expressions, truth tables and binary decision diagrams (BDD) have been utilized. The synthesis algorithms are capable of transforming the initial circuit representation (which is derived from the behavioral specification) onto a circuit representation which is suitable for subsequent circuit fabrication. The circuit

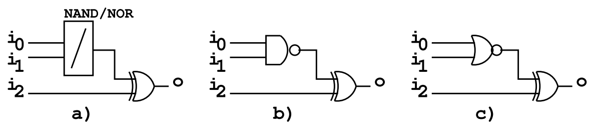

FIGURE 1
a) Example of a polymorphic circuit; b) Equivalent circuit in mode 1; c) Equivalent circuit in mode 2

representation together with the synthesis algorithm determines the space of possible implementations that one can obtain as a result of the synthesis process.

It remains unclear how to represent a gate-level polymorphic circuit and how to define such transformations over a chosen representation which will lead to an efficient implementation of the polymorphic circuit using a given set of ordinary and polymorphic gates. A partial success was achieved using evolutionary design methods [22, 14, 25, 7] which do not pose any requirements on the representation or the set of transformations. However, because the methods are search-based, they are not scalable and only relatively small polymorphic circuits were evolved (e.g., max. 7-input Multiplier/Sorter circuits [25]). Papers [7, 15] have exploited some conventional methods to synthesize polymorphic circuits, however, achieving relatively inefficient implementations.

In this article, we extend the concept of combining conventional synthesis with evolutionary synthesis that was developed in our previous work [7]. The goal is to propose a method that will be able to synthesize and optimize especially mid-size polymorphic circuits. The first problem (which this article deals with) is how to modify conventional circuit synthesis methods in order to allow them to operate with polymorphic gates. Two approaches are proposed and compared: polymorphic multiplexing and polymorphic BDDs. Unfortunately, these approaches produce area-inefficient solutions when applied solely. Hence the second problem targeted in this article is the optimization of the number of gates in polymorphic circuits. The proposed optimization algorithm is based on Cartesian Genetic Programming (CGP) [16]. Two different fitness functions will be compared for this task. As there is not available any set of benchmark polymorphic circuits, we have introduced a new set of benchmark circuits to evaluate the synthesis algorithms. The benchmark set consists of 14 circuits with 4–13 inputs and 4–20 outputs. In order to fairly compare the results of various methods, solutions will be sought in the form of circuits composed of two-input gates (inverters included). In addition to ordinary gates, we restrict ourselves to use only the NAND/NOR polymorphic gate controlled by V_{dd} because only this gate is currently available for a physical implementation [27, 24]. However, in general, proposed methods can utilize an arbitrary (but functionally complete) set of gates.

The rest of the article is organized as follows. Section 2 introduces the area of polymorphic electronics. Cartesian Genetic Programming, its advantages and limitations are surveyed in Section 3. Section 4 formally defines the polymorphic circuit synthesis problem. In Section 5, proposed methods to the synthesis of polymorphic circuits are described. In particular, we use Cartesian Genetic Programming, polymorphic multiplexing and polymorphic binary decision diagrams. Subsection 5.4 deals with the optimization of polymorphic circuits. The optimization is performed using CGP, i.e. CGP is 'seeded' with conventional designs. Section 6 presents the results obtained using proposed methods on a set of benchmark circuits. Discussion of obtained results is presented in Section 7. Conclusions are given in Section 8.

2 POLYMORPHIC ELECTRONICS

Current research in the field of polymorphic electronics can be split into three areas: (i) design of reliable polymorphic gates, (ii) development of synthesis algorithms and (iii) development of applications. As the problem of synthesis will be dicussed in the remaining parts of the paper we will briefly summarize (i) and (iii) in this section.

Table 1 surveys some polymorphic gates reported in literature. For each polymorphic gate, logic functions performed by the gate are given together with the values that represent recommended setting of the control signal variable. The number of transistors characterizes the size of polymorphic gates only partially as the transistors occupy different areas and the gates were fabricated using different fabrication technologies.

Only two of the polymorphic gates have been fabricated so far; remaining polymorphic gates were either simulated or tested in a field programmable transistor array (FPTA-2). For instance, the 6-transistor NAND/NOR gate controlled by V_{dd} was fabricated in a 0.5-micron HP technology [27]. Another NAND/NOR gate controlled by V_{dd} and introduced in [20] was utilized in the REPOMO32 chip which is an experimental reconfigurable platform for development of polymorphic circuits [24]. REPOMO32 consists of 32 two-input Configurable Logic Elements; each of them can be programmed to perform one of the following functions: AND, OR, XOR and polymorphic NAND/NOR (controlled by V_{dd}). When V_{dd} = 3.3V the NAND/NOR gate exhibits the NOR function and when V_{dd} = 5V the gate exhibits the NAND function. Remaining gates do not change their logic functions with the changes of V_{dd} (3–5 V). The chip was fabricated in a 0.7-micron AMIS technology.

TABLE 1
Existing polymorphic gates

Gate	control values	control	transistors	ref.
AND/OR	27/125°C	temperature	6	[28]
AND/OR/XOR	3.3/0.0/1.5V	ext. voltage	10	[28]
AND/OR	3.3/0.0V	ext. voltage	6	[28]
AND/OR	1.2/3.3V	V_{dd}	8	[29]
NAND/NOR	3.3/1.8V	V_{dd}	6	[27]
NAND/NOR/NXOR/AND	0/0.9/1.1/1.8V	etx. voltage	11	[32]
NAND/NOR	5/3.3V	V_{dd}	8	[20]

Papers [28, 29] indicate various areas in which polymorphic gates could be utilized. The applications of polymorphic electronics reported or proposed so far are given as references in the following summary.

- Automatic change of circuit behavior when a power supply is not sufficient [23].
- Implementation of low-cost reconfigurable/adaptive systems that are able to adjust their behavior inherently in response to certain control variables (e.g., multifunctional counters [32, 19]).
- Implementation of novel concepts for testing and diagnosing of electronic circuits (e.g., self-checking adders [20] and reduction of test vector volume [26]).
- Implementation of a hidden function, invisible to the user, which can be activated in a specific environment [28, 29].
- Intelligent sensors for biometrics, robotics and industrial measurement [28, 29].
- Reverse engineering protection [28, 29].

3 CARTESIAN GENETIC PROGRAMMING

Cartesian genetic programming was introduced by Miller and Thompson a decade ago [18]. It resembles the concept of genetic programming, but introduces some important modifications: (i) a candidate circuit is modelled using a directed acyclic graph; (ii) the graph is encoded as a fixed-size string of integers; and (iii) the search is performed using a mutation-based evolutionary strategy (no crossover is employed). The main advantage of CGP is that it generates very compact solutions, i.e. it can effectively reduce the total number of gates in the case of circuit evolution (see evolved multipliers in [30]).

In the basic version of CGP, a candidate circuit is modeled in a matrix of u (columns) $\times$ v (rows) of programmable n_e-input elements (two-input gates, in our case). The number of inputs, n, and outputs, m, of the circuit is fixed. Each gate input can be connected either to the output of a gate placed in the previous L columns or to one of the circuit inputs (n_e integers are devoted to encode the connections of a single gate). The L-back parameter, in fact, defines the level of connectivity and thus it reduces/extends the search space. Feedback is not allowed. Each gate can be programmed to perform one of logic functions defined in the set Γ which contains ordinary and polymorphic functions in our task. Figure 2 shows an example of the encoding used in CGP.

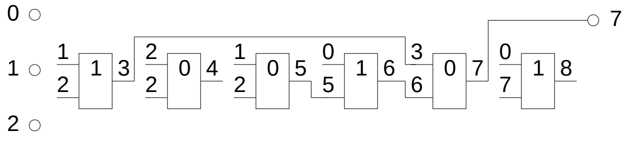

FIGURE 2
An example of a circuit in CGP with parameters: $n = 3$, $m = 1$, $L = 6$, $u = 6$, $v = 1$, $\Gamma = \{and(0), or(1)\}$. Elements 4 and 8 are not utilized. Chromosome: 1,2,**1**, 2,2,**0**, 1,2,**0**, 0,5,**1**, 3,6,**0**, 0,7,**1**, 7. Functions of elements are typed in bold. The last integer indicates the output of the circuit.

In case of the combinational circuit evolution, the fitness value of a candidate circuit is traditionally defined in CGP as [10]:

$$fitness = B + (uv - z) \tag{1}$$

where B is the number of correct output bits obtained as response for all possible assignments to the inputs, z denotes the number of gates utilized in a particular candidate circuit and uv is the total number of available gates. The last term $uv - z$ is considered only if the circuit behavior is perfect, i.e. $B = m2^n$.

CGP operates with the population of $1 + \lambda$ individuals (typically, λ is from 2 to 15). The initial population is either randomly generated or seeded using existing designs. Every new population consists of the best individual of the previous population and its λ offspring created by a point mutation (the number of mutated genes is a parameter of the mutation operator). In case when two or more individuals have received the same fitness score in the previous population, an individual which has not served as the parent in the previous population will be selected as the new parent. This strategy is used to ensure diversity of population.

CGP has been successfully utilized in many applications (e.g., [16, 6, 21, 13, 11]), investigated experimentally [17] and extended to support modularity, self-modification and other features [31, 9]. It has been shown that neutrality (i.e. a neutral effect of inactive genes on genotype fitness) which is inherent for CGP is very beneficial to the efficiency of evolutionary process [17].

However, the main problem is that in case of the combinational circuit evolution, the evaluation time of a candidate circuit grows exponentially with the increasing number of inputs (assuming that all possible input combinations are tested in the fitness function) [30]. Hence, the evaluation time becomes the main bottleneck of the evolutionary approach when complex circuits with many inputs are evolved.

4 POLYMORPHIC CIRCUIT SYNTHESIS PROBLEM

In this article, the polymorphic circuit synthesis problem is restricted to the gate level. Target circuits will consist of polymorphic and ordinary gates. The following problem formulation resembles the definition proposed in [22, 25].

Let $\Gamma^{(1)}$ denote a set of ordinary gates. Let $\Gamma^{(2)}$ denote a set of polymorphic gates. A polymorphic gate implements two[⋆] functions according to a control signal which can hold two different values. A gate is in *mode j* (and so performing the j-th function) in the case when j-th value of the control signal is activated. For purpose of this article, we denote a polymorphic gate as X_1/X_2, where X_i is its i-th logic function. For example, NAND/NOR denotes a gate operating as NAND in the *mode 1* and as NOR in the *mode 2*. Note

[⋆] This can be naturally extended for k different functions.

that ordinary gates can perform only one function, however, their functionality must be fully defined for each mode. For example, a conventional NAND gate considered for polymorphic circuits must perform the NAND function in both modes (denoted as NAND/NAND). Let Γ denote a set of all gates, $\Gamma = \Gamma^{(1)} \cup \Gamma^{(2)}$.

A polymorphic circuit can formally be represented by a graph $G = (V, E, \varphi)$, where V is a set of vertices, E is a set of edges between the vertices, $E = \{(a,b) | a, b \in V\}$, and φ is a mapping assigning a function (gate) to each vertex, $\varphi : V \rightarrow \Gamma$. Note that V models the gates and E models the connections of the gates. A circuit (and also its graph) is in the *mode j* in the case when all gates are in the *mode j*.

Given Γ and logic functions f_1 and f_2 required in different modes, the problem of the multifunctional circuit synthesis at the gate level is formulated as follows: Find a graph G representing the digital circuit which performs logic function f_1 in the *mode 1* and logic function f_2 in the *mode 2*. Additional requirements can be specified, e.g. to minimize delay, area, power consumption etc.

Unfortunately, this problem can not be approached by conventional synthesis methods directly since they do not allow representing polymorphic logic functions and manipulating with them.

Note that paper [15] also deals with the polymorphic circuits synthesis problem; however, the goal of the synthesis is different: a target circuit is constructed to perform a single function independently of the polymorphic gates mode.

5 PROPOSED METHODS

This section describes three approaches to the polymorphic circuit synthesis problem. The first one is based on the evolutionary design using CGP. The remaining approaches extend the conventional synthesis methods to be applicable for polymorphic circuits. While polymorphic multiplexing allows polymorphic gates to be included to the output part of the target circuit the polymorphic binary decision diagrams enable inserting the polymorphic gates to the input part of the target circuit. In both cases, CGP is used in the post-sythesis phase to reduce the number of gates.

5.1 Direct Circuit Evolution Using CGP

In our previous work, we have used CGP to synthesize polymorphic circuits [22, 25]. In contrast to equation 1 we have modified the fitness function so that a candidate circuit is evaluated in both modes. The new fitness value is defined as follows:

$$fit_1 = 1 + (B_1 + B_2)(u.v + 1) + z \tag{2}$$

where B_1 (resp. B_2) is the number of incorrect output bits for f_1 (resp. f_2) obtained as response for all possible assignments to the inputs, z denotes the number of gates utilized in a particular candidate circuit and $u.v$ is the total number of programmable gates available. The last term z is considered only if the circuit behavior is perfect in both modes ($B_1 + B_2 = 0$). Here, the goal of evolution is to minimize the output of Equation 2. However, this approach is not scalable and thus new synthesis methods have to be proposed.

5.2 Polymorphic Multiplexing

A straightforward approach to the implementation of polymorphic circuits is the utilization of a polymorphic multiplexer (*pmux*) which propagates signal A in the *mode 1* of polymorphic gates and signal B in the *mode 2* of polymorphic gates. One of possible gate-level implementations of *pmux*, based on the NAND/NOR gates, is shown in Fig. 3. Its implementation cost is c_{pmux} = 5 gates. We will use this implementation for comparisons which will be performed in this article. However, it is expected that a more compact and efficient transistor-level solution of *pmux* will be available in the future.

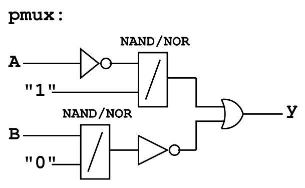

FIGURE 3
Polymorphic multiplexer at the gate-level

The polymorphic multiplexing works as follows: Consider that a target polymorphic circuit has to implement f_1 and f_2. A conventional approach is used to synthesize a circuit (module M_1) implementing f_1 and another circuit (module M_2) implementing f_2 independently. The outputs of the circuits are then multiplexed using polymorphic multiplexers as shown in Fig. 4a. This approach will be denoted as Independent Modules (IM). Note that especially for smaller circuits it is also possible to use evolutionary circuit design instead of conventional methods to create the modules. Larger modules are usually designed by conventional design methods.

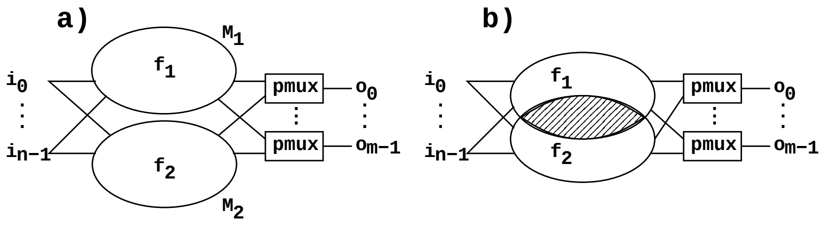

FIGURE 4
Multiplexing conventional circuits by polymorphic multiplexers: a) independent modules, b) sharing some gates between modules

In order to reduce the number of gates, the goal of synthesis can be to maximize the number of gates that are shared by both circuits (see the intersection in Figure 4b). Espresso [2] and ABC [1] are conventional circuit synthesis methods that we chose to synthesize particular modules. We applied them with the aim of minimizing the number of gates in both modules, sharing as much gates as possible between the modules and minimizing the number of outputs that have to be equipped with polymorphic multiplexers.

5.3 Polymorphic BDD-based Synthesis

We propose *polymorphic binary decision diagrams* (PolyBDD) to extend standard decision diagrams. We will show how to construct PolyBDDs and transform them to corresponding polymorphic circuits.

Decision Diagrams

Decision diagram (DD) [4] over a set of Boolean variables $X_n = \{x_1, ..., x_n\}$ and a non-empty terminal set T is defined as a directed acyclic graph $G = (V, E)$ with exactly one root node and the following properties:

- A node in V is a non-terminal or terminal node.
- A non-terminal node is labeled by variable x_i and has two successors $low(x_i)$ and $high(x_i)$ in V.
- A terminal node is labeled with a value from T.

The size of DD is given by the number of its nodes. The level x_i is the set of nodes labeled by x_i. A DD is ordered, if each variable is encountered at most once on each path from the root to a terminal node and the variables are encountered in the same order on each path.

Binary decision diagram (BDD) [4] is defined as DD over X_n; however, its terminal set is $T = \{0,1\}$. If the BDD has root node v, then BDD represents a Boolean function f_v defined as follows: If v is a terminal node of value 0 (1) then $f_v = 0$ ($f_v = 1$); If v is a non-terminal node labeled with index x_i then f_v is the function $f_v(x_1,...,x_n) = \bar{x_i}.f_{low(v)}(x_1,...,x_n) + x_i.f_{high(v)}(x_1,...,x_n)$, where $f_{low(v)}$ ($f_{high(v)}$) denotes the function represented by $low(v)$ ($high(v)$). We can also call the non-terminal nodes as if-then-else nodes: if x_i then $f_v = f_{high}$ else $f_v = f_{low}$.

Multi-Terminal BDD (MTBDD) [5] is an extension of BDDs which allows integers to be placed in terminal nodes. Decision variables are still Boolean.

We define a PolyBDD as a MTBDD in which terminal integers determine elementary polymorphic functions.

Design of PolyBDD

Let f_1 denote a target function in mode $j = 1$ and f_2 denote a target function in mode $j = 2$ (according to Section 4; $f_j : \{0,1\}^n \rightarrow \{0,1\}$). Let R_1 and R_2 be truth tables for f_1 and f_2. Assume that R_1 and R_2 are fully defined and ordered. The design of PolyBDD which represents f_1 and f_2 is a three-step procedure:

1. Create truth table R of the polymorphic circuit as composition of R_1 and R_2. Truth table R has 2^n lines, n columns with all the input variable assignments and two columns with logic values for f_1 and f_2.

2. Choose a decision variable x_c (where $c = 0...n-1$) and divide R into $2^n/2$ segments (rows) in such a way that the input assignment differs only in x_c in each segment. From each segment, extract a signature $S = 2^3.s_{21} + 2^2.s_{20} + 2^1.s_{11} + 2^0.s_{10} = (s_{21}s_{20}s_{11}s_{10})_2$; the least significant bits come from f_1 and the most significant bits come from f_2; the values of variables $s_{21}, s_{20}, s_{11}, s_{10}$ are determined according to a conversion matrix (see Figure 5b). Let $R^{'}$ denote the resulting truth table.

3. Process the resulting $2^n/2$-row truth table R' using a standard algorithm for MTBDD processing to create an optimized if-then-else structure.

Figure 5 shows an example: the original truth table R of the 3-bit Majority/Parity function, transformed truth table (according to input x_0) and example of the signature. Terminal nodes represent simple polymorphic functions, see Table 2.

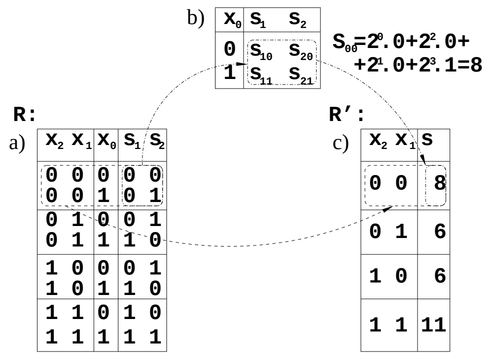

FIGURE 5
Transformation process of the 3-bit Majority/Parity truth table (s_1 denotes the majority; s_2 denotes the parity): a) Truth table R before transformation; b) Conversion matrix of the signature; c) Transformed truth table $R^{'}$

TABLE 2
Signature semantics of the PolyBDD ('neg' denotes the logic inversion; 'id' denotes the identity function)

S	s_1 / s_2	S	s_1 / s_2	S	s_1 / s_2	S	s_1 / s_2
0	0 / 0	4	0 / neg	8	0 / id	12	0 / 1
1	neg / 0	5	neg / neg	9	neg / id	13	neg / 1
2	id / 0	6	id / neg	10	id / id	14	id / 1
3	1 / 0	7	1 / neg	11	1 / id	15	1 / 1

A PolyBDD which has been constructed using the proposed algorithm can be further optimized using standard techniques that are applicable for MTBDDs. The optimization techniques includes:

- Reducing identical (redundant) branches (type I [4]).
- Reducing (redundant) if-then-else node with the same branches (type S [4]).
- Reordering the input variables.

Synthesis of PolyBDD into Polymorphic Circuit

A transformation of the PolyBDD to a polymorphic circuit is straightforward. Firstly, the if-then-else nodes are directly mapped onto 2-input ordinary multiplexers. Then, terminal nodes are implemented using elementary polymorphic circuits according to the conversion table shown in Fig. 6. The elementary polymorphic circuits utilize ordinary gates and the NAND/NOR polymorphic gate. If, for some reasons, another set of gates has to be used, a different conversion table has to be created. Finally, the elementary polymorphic circuits are connected with multiplexers. Figure 7 shows reduced PolyBDD for the 3-bit Majority/Parity benchmark and its implementation using polymorphic gates.

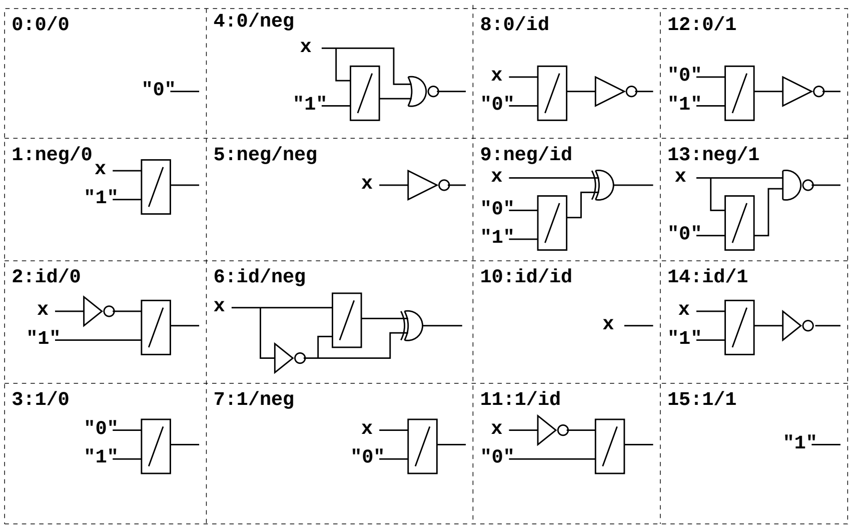

FIGURE 6
Conversion table for the PolyBDD method. Elementary polymorphic circuits composed of NAND/NOR and ordinary gates were constructed for signatures 0...15.

5.4 Optimization of Polymorphic Circuits

Polymorphic circuits created using polymorphic multiplexing or PolyBDDs are large in many cases and thus inefficient. In the case of polymorphic multiplexing, polymorphic gates are solely located around the primary outputs of the circuit. In the case of PolyBDDs, the polymorphic gates are located close

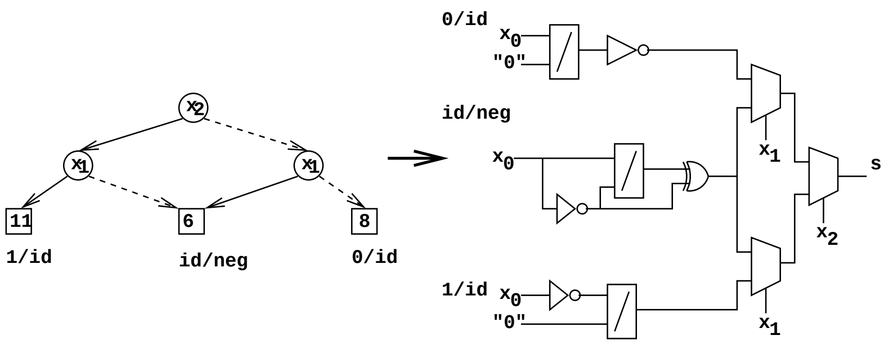

FIGURE 7
Reduced PolyBDD and a corresponding polymorphic circuit for the 3-bit Majority/Parity problem.

to the primary inputs. In order to minimize the total number of gates, it is desirable to integrate polymorphic gates deeply to the circuit structure and also increase the ratio of polymorphic gates to ordinary gates. We will show later that increasing the ratio of polymorphic gates usually leads to decreasing the total number of gates.

For purposes of optimization, polymorphic circuits are converted to the CGP representation. Since CGP uses two-input nodes, all components have to utilize up to two inputs. Hence a 3-input multiplexer is converted into four 2-input gates (in the PolyBDD-based design), a 3-input AND-gate is converted into two 2-input gates (in the Espresso-based designs) etc.

CGP is, in fact, seeded with a fully functional but non-optimal design with respect to the number of gates. The goal of the optimization is the reduction of the number of gates. We will compare the traditional fitness function from Equation 2 which explicitly counts the number of gates with a new fitness function defined as:

$$fit_2 = 1 + (B_1 + B_2) \tag{3}$$

We will show in Section 6.5 that the new fitness function (Eq. 3) significantly improves the results of optimization although it does not take into account the number of gates. This phenomenon has already been studied in domain of evolutionary synthesis of combinational circuits [8].

The fitness evaluation procedure which probes every assignment to the inputs (i.e., $0 \dots 2^n - 1$) is the main time bottleneck of the optimization method.

In order to make the evaluation of a candidate circuit as short as possible, it is only tested whether the candidate circuit is working correctly or incorrectly. In the case that a candidate circuit does not produce a correct output value for the p-th input vector during the evaluation, the remaining $2^n - p - 1$ vectors are not evaluated and the circuit gets the worst possible score. Experimental results have shown that this technique reduces the computational overhead significantly (see Fig. 8). Note that this technique cannot be applied for the evolutionary circuit design conducted in Section 5.1 because we have to know the fitness score as precisely as possible (i.e. the exact number of bits has to be calculated that can be correctly generated by a particular candidate circuit) in order to obtain a relatively smooth fitness landscape.

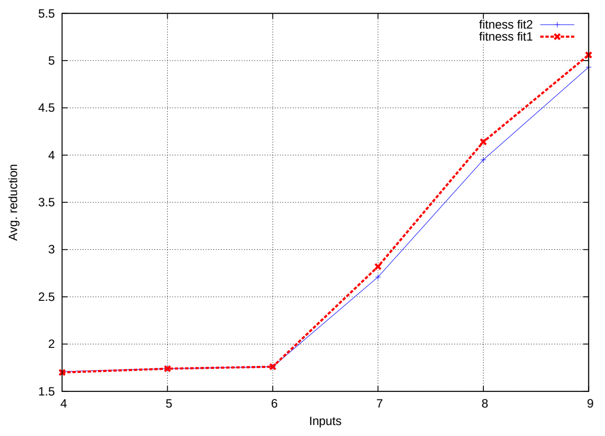

FIGURE 8
Average reduction of the computational overhead in comparison with the complete truth table evaluation for the Multiplier/Sorter benchmark (according to Table 9). Results are shown for a 64 bit architecture with the parallel evaluation.

Parallel simulation is another technique that can be used to accelerate the circuit evaluation [16]. The idea of parallel simulation is to utilize bitwise operators operating on multiple bits in a high-level language (such as C) to perform more than one evaluation of a gate in a single step. For example, when a combinational circuit under simulation has three inputs and it is possible to concurrently perform bitwise operations over $2^3 = 8$ bits in the simulator then the circuit can completely be simulated by applying a single 8-bit test vector at each input (see the encoding in Fig. 9). In contrast, when it is impossible then eight three-bit test vectors must be applied sequentially. In the case of polymorphic circuits the simulation is performed for each mode separately. Practically, current processors allow us to operate with 64 bit operands, i.e. it is possible to evaluate the truth table of a six-input circuit by applying a single 64-bit test vector at each input. Therefore, the obtained speedup is 64 against the sequential simulation. In case that a circuit has more than 6 inputs then the speedup is constant, i.e. 64. This technique has been applied in all evolutionary design and optimization experiments reported in this article.

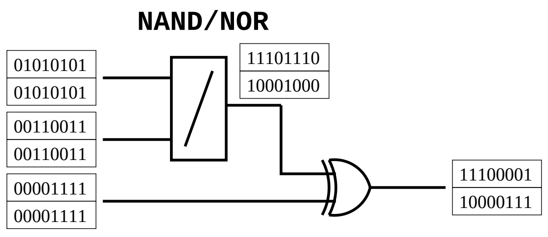

FIGURE 9
Parallel evaluation of a candidate polymorphic circuit

6 RESULTS

This section introduces a new set of benchmark circuits that we have developed and reports the results obtained using the proposed synthesis methods.

6.1 Benchmark Circuits

As no benchmark set is available to evaluate proposed synthesis algorithms, we have to introduce a new one. The proposed benchmark set consists of 14 circuits of 4-13 inputs and 4-20 outputs (see Table 3). This benchmark set includes Multiplier/Sorter circuits of variable sizes, Majority/Parity circuits of variable sizes and four polymorphic constant coefficient multipliers. For example, the $\times 67/\times 127$ circuit multiplies the input value by 67 in the *mode 1* and by 127 in the mode 2 of polymorphic gates. The Multiplier/Sorter circuits have been used as benchmarks in literature; however, only for up to 8 inputs/8 outputs [22, 25, 7]. Note that the area-optimal multipliers as well as sorters have significantly different structures for different numbers of the inputs.

TABLE 3
Proposed benchmark circuits

Circuit	Inp.	Out.	Mode 1	Mode 2
M/S4	4	4	2x2-bit multiplier	4-bit sorter
M/S5	5	5	3x2-bit multiplier	5-bit sorter
M/S6	6	6	3x3-bit multiplier	6-bit sorter
M/S7	7	7	4x3-bit multiplier	7-bit sorter
M/S8	8	8	4x4-bit multiplier	8-bit sorter
M/S9	9	9	5x4-bit multiplier	9-bit sorter
M/P7	7	1	7-bit majority	7-bit parity
M/P9	9	1	9-bit majority	9-bit parity
M/P11	11	1	11-bit majority	11-bit parity
M/P13	13	1	13-bit majority	13-bit parity
$\times$67/$\times$127	7	14	multiply by 67	multiply by 127
$\times$131/$\times$251	8	16	multiply by 131	multiply by 251
$\times$257/$\times$509	9	18	multiply by 257	multiply by 509
$\times$521/$\times$1021	10	20	multiply by 521	multiply by 1021

6.2 Direct Evolutionary Design Using CGP

For all problems, 10 runs were executed in each experiment, the population size was 15, 3 genes were mutated in the search phase ($B_1 + B_2 > 0$), 7 genes were mutated in average in the minimization phase ($B_1 + B_2 = 0$) and up to 100 million generations were produced in each run. Results are reported for the best setting of CGP parameters that we have found.

Note that the results for the Multiplier/Sorter problem are included just for comparison (they were published in detail elsewhere [25, 7]). As shown in Table 4 the 7-input Multiplier/Sorter circuit is the most complex polymorphic circuit evolved directly.

Table 5 summarizes the results for the Majority/Parity benchmark. In this case, CGP can evolve Majority/Parity benchmark circuits with up to 13 inputs. Table 6 shows the results for the polymorphic constant coefficient multipliers.

6.3 Results of Polymorphic Multiplexing

We have used Espresso and ABC to synthesize the modules of benchmark circuits with the aim of minimizing the number of gates and sharing as much

TABLE 4
Parameters and results of CGP for the Multiplier/Sorter problem. Gates in ‘Gate set’ are numbered as: (1) NAND/NOR, (2) AND, (3) OR, (4) XOR, (5) NAND, (6) NOR, (7) NOT A, (8) NOT B, (9) MOV A and (10) MOV B, where MOV denotes the identity operation.

Multiplier/Sorter	**2×2b/4b**	**3×2b/5b**	**3×3b/6b**	**4×3b/7b**
$u \times v$	10×12	100×1	120×1	16×16
L-back	1	100	120	16
Mutation (genes)	1	2	4	4
Gate set	1, 2, 9, 10	1–4, 9, 10	1–10	1, 2, 9, 10
Successful runs	100%	100%	90%	30%
Generations (avg.)	52,580	854,900	26,972,648	62,617,151
Min. # of gates	23	30	52	113

TABLE 5
Parameters and results of CGP for Majority/Parity problem. The gate set includes NAND/NOR, AND, OR, XOR, NAND, NOR, NOT and MOV.

Majority/Parity	**7b**	**9b**	**11b**	**13b**
$u \times v$	80×1	120×1	120×1	160×1
L-back	80	120	120	160
Successful runs	100%	90%	50%	10%
Generations (avg.)	766,362	4,762,745	8,145,890	9,712,501
Min. # of gates	25	42	61	80

gates as possible between the modules. Table 12 shows the results of Espresso and ABC synthesis method for the benchmark problems (see the columns labeled ’Espresso’ and ’ABC’).

We have also used the best-known optimized implementations of *independent* modules that were interconnected by polymorphic gates (i.e., no sharing of gates between the modules allowed). The implementations of multipliers were taken from [30], sorting networks and majority circuits were derived according to [12] and constant coefficient multipliers were composed on the basis of reduced ripple-carry multipliers. Results (denoted as ’IM’) are given in Table 12. It can be seen that the strategy of independent modules leads to fewer gates; however, the circuits synthesized by the IM method are still large and have to be further optimized.

TABLE 6
Parameters and results of CGP for ×Constant/×Constant problem. The gate set includes NAND/NOR, AND, OR, XOR, NAND, NOR, NOT and MOV.

×Constant/×Constant	**×67/×127**	**×131/×251**	**×257/×509**
Inputs	7	8	9
Outputs	14	16	18
$u \times v$	320×1	640×1	320×1
L-back	320	640	320
Successful runs	60%	10%	30%
Generations (avg.)	7,192,359	46,833,855	40,002,719
Min. # of gates	94	239	116

6.4 Results of PolyBDDs

Table 7 shows the results of the polyBDD method for benchmark problems. By ‘Gates’ we mean common 2-input gates, NAND/NORs, 2-input multiplexers and inverters. In PolyBDDs, reduction of branches, reduction of if-then-else nodes with the same branches and reordering the inputs were applied. It can again be seen that synthesized circuits are large and have to be further optimized.

TABLE 7
PolyBDD design results of Multiplier/Sorter, Majority/Parity and ×Constant/×Constant benchmarks

Multiplier/Sorter	**3×2/5b**	**3×3/6b**	**4×3/7b**	**4×4/8b**
Nodes	37	79	135	253
Terminals	10	11	11	12
Gates	50	94	150	269
Majority/Parity	**7b**	**9b**	**11b**	**13b**
Nodes	21	34	49	66
Terminals	5	5	5	5
Gates	31	41	59	73
×Constant/×Constant	**×67/×127**	**×131/×251**	**×257/×509**	**×521/×1021**
Inputs	7	8	9	10
Outputs	14	16	18	20
Nodes	205	407	326	882
Terminals	16	16	15	16
Gates	228	430	348	905

6.5 Optimization of Polymorphic Circuits using CGP

The optimization of the number of gates in the polymorphic circuits constructed by aforementioned methods is performed by CGP with topology $q \times 1$ and $L = q$ where q is total number of elements in the starting circuit (i.e., in the seed for CGP). CGP operates with the population size of 15 individuals. The mutation operator modifies from 1 to 14 integers in the chromosome. The function set contains AND, OR, NOT, NAND, NOR, NAND/NOR and XOR gates. Each experiment is performed 10 times. The number of generations is given in particular tables. Two fitness functions fit_1 and fit_2 have been compared for this task.

Table 8 shows the results obtained by applying CGP on the 4×4-bit Multiplier/8-bit Sorter circuits created by PolyBDD, Espresso, ABC and IM synthesis method. For both fitness functions we can compare the number of gates (minimum, maximum and average), the average number of polymorphic gates and overall average improvement (reduction of gates) in resulting designs. Last part of Table 8 gives the design time for each method and the optimization time consumed by CGP. We can observe that the best solutions are significantly different (see Min. gates). As the IM and ABC methods provide the lowest number of gates for the 4×4-bit Multiplier/8-bit Sorter, we further analyzed their results and compared their performance on other instances of this benchmark problem. Table 10 (Table 11, respectively) summarizes the results for the ABC method (the IM method, respectively).

Moreover, Table 9 analyzes in greater detail the computational effort for the ABC method followed by CGP optimization for some instances of the Multiplier/Sorter problem. Last part of Table 9 shows the reduction of the computational overhead which is achievable when candidate circuits are evaluated incompletely (see Section 5.4). Presented results indicate that fitness function fit_2 clearly outperforms fitness function fit_1. For example, see Figure 10 which demonstrates a typical run of CGP and Figure 11 which demonstrates the average behavior of CGP on a particular circuit.

6.6 Computational Effort

In comparison with conventional synthesis, proposed methods require significantly more computational time. The reason is that the synthesis problem is more difficult than the conventional one and thus the main role plays the evolutionary search which is time consuming.

The computation time of direct CGP evolution can be expressed as follows. It takes 80 seconds to generate 1 million generations for a 4-input polymorphic circuit at Athlon64 3200+ processor. For a 7-input polymorphic circuit, 207 seconds are needed at the same processor.

Table 8 (see Avg. design time) gives the average time required to construct the 4x4-bit multiplier/8-bit sorter by proposed methods at Athlon64 X2 4800+ processor. Except the PolyBDD, the 'design time' is quite reasonable. It is almost zero if independent modules are available in advance. The main bottleneck is the 'optimization time' imposed by CGP (see the average optimization time at last line of Table 8) which is in order of hours for this circuit. However, this time can be reduced by decreasing the number of generations if a slightly larger solution is acceptable. Figures 10 and 11 demonstrate the relation between the number of gates and the generation (i.e., the time spent by optimization).

TABLE 8
Results of the CGP optimization of 4×4-bit Multiplier/8-bit Sorter 'seeded' by proposed methods

Method	fitness	**BDD**	**Espresso**	**ABC**	**IM**
Elements (q)		1043	2330	375	161
Initial gates		1028	2309	359	145
Generations		100M	100M	100M	100M
Max. gates	fit_1	407	697	232	112
Max. gates	fit_2	289	404	213	108
Min. gates	fit_1	355	616	192	109
Min. gates	fit_2	**256**	**318**	**166**	**105**
Avg. gates	fit_1	385.7	646.3	216.1	110.8
Avg. gates	fit_2	274.0	375.3	181.7	106.9
Avg. reduction	fit_1	38%	28%	60%	76%
Avg. reduction	fit_2	25%	14%	51%	74%
Avg. polymorphic gates	fit_1	8%	3%	22%	23%
Avg. polymorphic gates	fit_2	13%	14%	22%	21%
Avg. design time [s]		345	1	3	0.001
Avg. optimization time [s]		154,015	361,394	21,176	9,261

7 DISCUSSION

The best implementations (in terms of the number of gates) obtained by proposed methods and subsequent CGP optimization are summarized for all benchmark circuits in Table 12. The best-achieved results are typed in bold.

Results of Espresso were calculated only for some of the benchmark circuits because resulting circuits are too large and thus not competitive with

TABLE 9
Computational effort of the ABC method followed by CGP optimization for various instances of the Multiplier/Sorter circuit

Mult./Sorter		**2x2/4**	**3x2/5**	**3x3/6**	**4x3/7**	**4x4/8**	**5x4/9**
Elements (q)		45	71	131	212	375	697
Generations		1M	10M	10M	100M	100M	100M
Total eval.		30,000,000	300,000,000	300,000,000	6,000,000,000	12,000,000,000	24,000,000,000
Max. eval.	fit_1	17,718,334	174,887,434	183,569,142	2,150,454,947	2,929,056,984	4,964,315,994
Max. eval.	fit_2	17,610,320	172,659,116	171,835,708	2,245,511,189	3,112,567,474	5,031,752,725
Min. eval.	fit_1	17,522,867	171,223,566	167,859,366	2,108,612,177	2,830,925,681	4,653,366,773
Min. eval.	fit_2	17,351,240	171,691,345	169,738,375	2,190,750,297	2,977,493,490	4,698,975,306
Avg. eval.	fit_1	17,644,628	172,676,084	170,529,349	2,128,827,305	2,896,038,520	4,740,465,595
Avg. eval.	fit_2	17,497,586	172,126,928	170,825,843	2,216,669,961	3,040,183,950	4,872,021,002
Avg. reduc.	fit_1	1.70	**1.74**	**1.76**	**2.82**	**4.14**	**5.06**
Avg. reduc.	fit_2	**1.71**	**1.74**	**1.76**	2.71	3.95	4.93

TABLE 10
Results of the ABC followed by CGP optimization for various instances of the Multiplier/Sorter circuit

Multiplier/Sorter	**fitness**	**2x2/4**	**3x2/5**	**3x3/6**	**4x3/7**	**4x4/8**	**5x4/9**
Elements (q)		45	71	131	212	375	697
Initial gates		37	61	119	198	359	679
Generations		1M	10M	10M	100M	100M	100M
Max. gates	fit_1	22	43	85	135	232	411
Max. gates	fit_2	21	37	72	107	213	378
Min. gates	fit_1	19	36	71	110	192	366
Min. gates	fit_2	**18**	**32**	**59**	**88**	**166**	**323**
Avg. gates	fit_1	20.2	39.0	77.2	121.4	216.1	384.6
Avg. gates	fit_2	19.3	34.8	66.2	97.1	181.7	348.3
Avg. reduction	fit_1	55%	64%	65%	61%	60%	57%
Avg. reduction	fit_2	52%	57%	56%	49%	51%	51%
Avg. polymorph. gates	fit_1	37%	27%	21%	25%	22%	23%
Avg. polymorph. gates	fit_2	36%	28%	24%	24%	22%	21%

other methods. The highest number of gates produced by Espresso is mainly due the fact that many-input gates have to be transformed to circuits composed of 2-input gates. Hence Espresso is considered as the weakest method for the polymorphic synthesis problem.

Although the direct polymorphic circuit evolution using CGP is not scalable it can be considered as a very successful method for small problem in-

TABLE 11
Results of the IM method followed by CGP optimization for various instances of the Multiplier/Sorter circuit

Multiplier/Sorter	**fitness**	**2x2/4**	**3x2/5**	**3x3/6**	**4x3/7**	**4x4/8**	**5x4/9**
Elements (q)		47	67	91	125	161	198
Initial gates		39	57	79	111	145	180
Generations		1M	10M	10M	100M	100M	100M
Max. gates	fit_1	25	41	62	87	112	153
Max. gates	fit_2	23	38	60	83	108	152
Min. gates	fit_1	22	39	59	83	109	150
Min. gates	fit_2	**19**	**34**	**58**	**80**	**105**	**148**
Avg. gates	fit_1	23.0	39.3	61.1	84.6	110.8	151.6
Avg. gates	fit_2	20.6	35.9	59.3	81.4	106.9	149.8
Avg. reduction	fit_1	59%	69%	77%	76%	76%	84%
Avg. reduction	fit_2	53%	63%	75%	73%	74%	83%
Avg. polymorph. gates	fit_1	29%	26%	23%	21%	23%	17%
Avg. polymorph. gates	fit_2	27%	29%	25%	26%	21%	17%

TABLE 12
Summary of the number of gates obtained by all methods for the benchmark problems. Last two columns give the best known results from literature

Circuit	**Inp.**	**Out.**	**CGP**	**BDD**	**Esp**	**ABC**	**IM**	**BDD+ CGP**	**Esp+ CGP**	**ABC+ CGP**	**IM+ CGP**	**Best Lit.**	**Ref.**
M/S4	4	4	23	31	77	37	39	20	20	**18**	19	23	[25]
M/S5	5	5	**30**	50	168	61	57	35	30	32	34	30	[25]
M/S6	6	6	**52**	94	419	119	79	81	75	59	58	52	[25]
M/S7	7	7	113	150	960	198	111	112	116	88	**80**	110	[7]
M/S8	8	8	—	269	2,309	359	145	256	318	166	**105**	205	[7]
M/S9	9	9	—	428	—	679	180	471	—	323	**148**	—	—
M/P7	7	1	25	31	752	39	33	27	44	**21**	27	29	[7]
M/P9	9	1	42	**41**	—	58	49	44	—	42	**41**	45	[7]
M/P11	11	1	61	59	—	79	63	**49**	—	62	51	69	[7]
M/P13	13	1	80	73	—	112	83	70	—	80	**65**	90	[7]
x67/x127	7	14	94	228	—	244	308	141	—	**85**	114	—	—
x131/x251	8	16	239	430	—	513	352	535	—	**187**	259	—	—
x257/x509	9	18	116	348	—	364	396	194	—	**112**	337	—	—
x521/x1021	10	20	—	905	—	983	540	1,289	—	**326**	484	—	—

stances. Polymorphic multiplexing (either of independent modules (IM) or circuits synthesized by ABC) followed by CGP optimization provides best results for larger circuits. PolyBDDs do not perform as well as polymorphic multiplexing. Table 12 also shows that no single method outperforms other

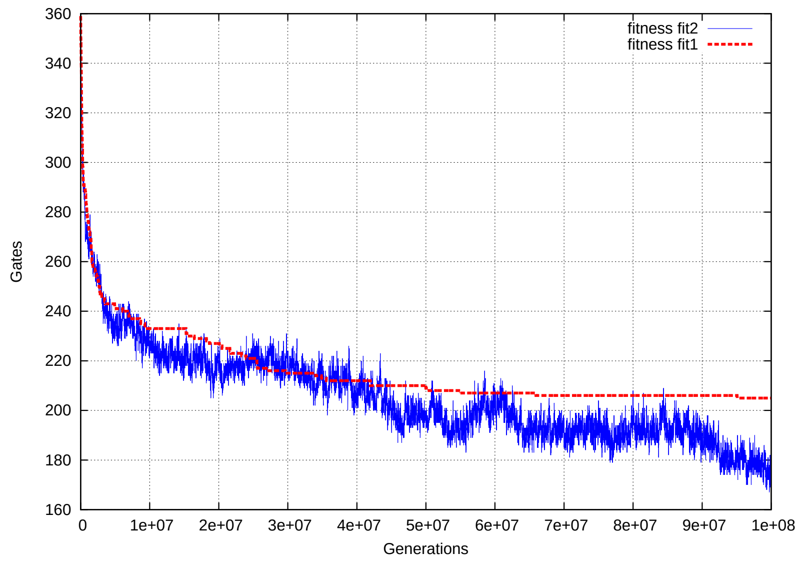

FIGURE 10
A typical progress of optimization for a single 4×4 Multiplier/8-bit Sorter (the best solution is shown)

methods in all problem instances. By using the proposed methods, we were able to significantly improve the implementation cost of almost all benchmark circuits in comparison with the results existing in literature.

Another important property of the CGP optimization applied on circuits created by polymorphic multiplexing is that the number of polymorphic gates utilized in resulting circuits is relatively high. Table 8 shows that the resulting circuits seeded by ABC method contain 22% polymorphic gates in average (23% was achieved for the IM method) while PolyBDDs led only to 8% and Espresso to 3% in average. In most cases combining the PolyBDD with evolutionary optimization of the gate count reduces the implementation cost. However, in some cases the CGP optimization can even increase the number of gates. For example, see the M/P9 benchmark in which the use of PolyBDD leads to 41 gates while the subsequent optimization requires 44 gates. The reason is that the PolyBDD representation uses multiplexers and a single multiplexer is transformed on four two-input gates for CGP.

Although we have considered only the NAND/NOR gate and polymor-

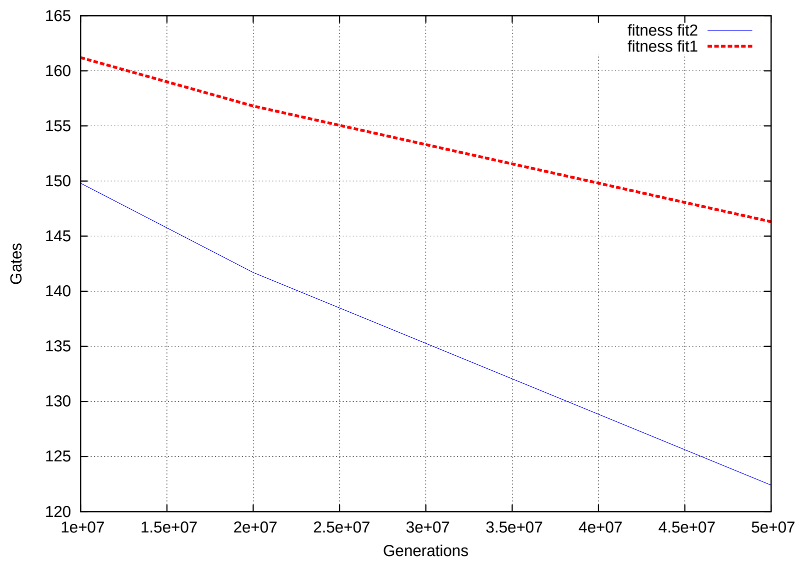

FIGURE 11
Average progress of optimization for the $\times 257/\times 509$ benchmark circuit.

phic circuits with two modes, the proposed methods can easily be extended to utilize other polymorphic gates and support more operational modes. We have applied only a basic set of operations to optimize PolyBDDs. We expect that some improvements will be obtained by applying more advanced optimization techniques over BDDs. The proposed fitness function fit_2 significantly improves the search efficiency although the requirement for reducing the number of gates is not stated explicitly. This very interesting phenomenon has been investigated for conventional circuit evolution in [8].

The proposed synthesis methods have already been utilized to create real-world circuits. We physically demonstrated the behavior of some small evolved circuits in REPOMO32 chip [24]. The evolved constant coefficient multipliers were utilized in polymorphic FIR filter [23]. Self-checking polymorphic adders were evolved using the proposed methods too [20].

8 CONCLUSIONS

We have presented new methods for synthesis of polymorphic circuits. The implementations of polymorphic circuits obtained by proposed methods represent the best known solutions if the number of gates is considered as the decision criterion. We have mentioned in Introduction that there is no reasonable method for synthesis of nontrivial polymorphic circuits. Proposed methods (PolyBDD and polymorphic multiplexing) can generate candidate 'raw' solutions to arbitrary specifications (if it is allowed by a particular SW implementation of Espresso, ABC etc.) in a reasonable time. In order to reduce the number of gates to a reasonable amount, a very time consuming optimization has to be conducted. However, the evolutionary-based optimization procedure seems to be the only way how to obtain reasonable implementations of polymorphic circuits. Our future research will be devoted to reducing the computational overhead of the proposed methods.

9 ACKNOWLEDGMENTS

This work was partially supported by the research project Natural Computing on Unconventional Platforms GP103/10/1517, the BUT FIT grant FIT-S-10-1 and the research plan Security-Oriented Research in Information Technology, MSM 0021630528.

REFERENCES

[1] Berkley Logic Synthesis and Verification Group. (2009). *ABC: A System for Sequential Synthesis and verification*.

[2] Robert K. Brayton, Gary D. Hachtel, Curtis T. McMullen, and Alberto L. Sangiovanni-Vincentelli. (1984). *Logic Minimization Algorithms for VLSI Synthesis*. Kluwer Academic Publishers, Boston, MA, USA.

[3] Rodney Brooks. (2001). The relationship between matter and life. *Nature*, 409(6816):409–411.

[4] Rolf Drechsler and Bernd Becker. (1998). *Binary Decision Diagrams: Theory and Implementation*. Kluwer Academic Publishers, Boston, USA.

[5] Masahiro Fujita, Patrick C. McGeer, and Jerry Chih-Yuan Yang. (2004). Multi-terminal binary decision diagrams: An efficient data structure for matrix representation. *Formal Methods in System Design*, 10(2-3):149–169.

[6] Zbysek Gajda and Lukas Sekanina. (2007). Reducing the number of transistors in digital circuits using gate-level evolutionary design. In *Genetic and Evolutionary Computation Conference*, pages 245–252. Association for Computing Machinery.

[7] Zbysek Gajda and Lukas Sekanina. (2009). Gate-level optimization of polymorphic circuits using cartesian genetic programming. In *Proc. of 2009 IEEE Congress on Evolutionary Computation*, pages 1599–1604. IEEE Computational Intelligence Society.

[8] Zbysek Gajda and Lukas Sekanina. (2010). An efficient selection strategy for digital circuit evolution. In *Evolvable Systems: From Biology to Hardware*, LNCS 6274, pages 13–24. Springer Verlag.

[9] Simon L. Harding, Julian F. Miller, and Wolfgang Banzhaf. (2007). Self-modifying cartesian genetic programming. In *Genetic and Evolutionary Computation Conference, GECCO 2007*, pages 1021–1028. ACM Press.

[10] Tatiana Kalganova and Julian F. Miller. (1999). Evolving more efficient digital circuits by allowing circuit layout evolution and multi-objective fitness. In *The First NASA/DoD Workshop on Evolvable Hardware*, pages 54–63, Pasadena, California. IEEE Computer Society.

[11] Gul Muhammad Khan and Julian F. Miller. (2009). Evolution of cartesian genetic programs capable of learning. In *Genetic and Evolutionary Computation Conference, GECCO 2009*, pages 707–714. ACM.

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

[13] Taras Kowaliw, Wolfgang Banzhaf, Nawwaf Kharma, and Simon L. Harding. (2009). Evolving novel image features using genetic programming-based image transforms. In *IEEE Congress on Evolutionary Computation*, pages 2502–2507.

[14] Houjun Liang, Wenjian Luo, and Xufa Wang. (2007). Designing polymorphic circuits with evolutionary algorithm based on weighted sum method. In *Evolvable Systems: From Biology to Hardware*, volume 4684 of *LNCS*, pages 331–342. Springer.

[15] Wenjian Luo, Z. Zhang, and Xufa Wang. (2007). Designing polymorphic circuits with polymorphic gates: a general design approach. *IET Circuits, Devices & Systems*, 1(6):470–476.

[16] Julian F. Miller, Dominic Job, and Vesselin K. Vassilev. (2000). Principles in the Evolutionary Design of Digital Circuits – Part I. *Genetic Programming and Evolvable Machines*, 1(1):8–35.

[17] Julian F. Miller and Stephen L. Smith. (2006). Redundancy and Computational Efficiency in Cartesian Genetic Programming. *IEEE Transactions on Evolutionary Computation*, 10(2):167–174.

[18] Julian F. Miller and Peter Thomson. (2000). Cartesian Genetic Programming. In *Proc. of the 3rd European Conference on Genetic Programming EuroGP2000*, volume 1802 of *LNCS*, pages 121–132. Springer.

[19] Richard Ruzicka. (2008). On bifunctional polymorphic gates controlled by a special signal. *WSEAS Transactions on Circuits*, 7(3):96–101.

[20] Richard Ruzicka, Lukas Sekanina, and Roman Prokop. (2008). Physical demonstration of polymorphic self-checking circuits. In *Proc. of 14th IEEE International On-Line Testing Symposium*, pages 31–36. IEEE.

[21] Lukas Sekanina. (2002). Image Filter Design with Evolvable Hardware. In *Applications of Evolutionary Computing – Proc. of the 4th Workshop on Evolutionary Computation in Image Analysis and Signal Processing EvoIASP'02*, volume 2279 of *LNCS*, pages 255–266. Springer Verlag.

[22] Lukas Sekanina. (2005). Evolutionary design of gate-level polymorphic digital circuits. In *Applications of Evolutionary Computing*, volume 3449 of *LNCS*, pages 185–194, Lausanne, Switzerland. Springer Verlag.

[23] Lukas Sekanina, Richard Ruzicka, and Zbysek Gajda. (2009). Polymorphic FIR filters with backup mode enabling power savings. In *2009 NASA/ESA Conference on Adaptive Hardware and Systems*, pages 43–50. IEEE.

[24] Lukas Sekanina, Richard Ruzicka, Zdenek Vasicek, Roman Prokop, and Lukas Fujcik. (2009). Repomo32 – new reconfigurable polymorphic integrated circuit for adaptive hardware. In *2009 IEEE Workshop on Evolvable and Adaptive Hardware*, pages 39–46. IEEE Computational Intelligence Society.

[25] Lukas Sekanina, Lukas Starecek, Zdenek Kotasek, and Zbysek Gajda. (2008). Polymorphic gates in design and test of digital circuits. *International Journal of Unconventional Computing*, 4(2):125–142.

[26] Lukas Starecek, Lukas Sekanina, and Zdenek Kotasek. (2008). Reduction of test vectors volume by means of gate-level reconfiguration. In *Proc. of 2008 IEEE Design and Diagnostics of Electronic Circuits and Systems Workshop*, pages 255–258. IEEE Computer Society.

[27] Adrian Stoica, Ricardo Zebulum, Xin Guo, Didier Keymeulen, Ian Ferguson, and Vu Duong. (2004). Taking Evolutionary Circuit Design From Experimentation to Implementation: Some Useful Techniques and a Silicon Demonstration. *IEE Proc.-Comp. Digit. Tech.*, 151(4):295–300.

[28] Adrian Stoica, Ricardo Salem Zebulum, and Didier Keymeulen. (2001). Polymorphic electronics. In *Proc. of Evolvable Systems: From Biology to Hardware Conference*, volume 2210 of *LNCS*, pages 291–302. Springer.

[29] Adrian Stoica, Ricardo Salem Zebulum, Didier Keymeulen, and Jason Lohn. (2002). On polymorphic circuits and their design using evolutionary algorithms. In *Proc. of IASTED International Conference on Applied Informatics AI2002*, Insbruck, Austria.

[30] Vesselin K. Vassilev and Julian F. Miller. (2000). Scalability problems of digital circuit evolution. In *Proc. of the 2000 NASA/DoD Conference on Evolvable Hardware,* pages 55–64, Palo Alta, CA. IEEE Computer Society.

[31] James A. Walker and Julian F. Miller. (2008). The Automatic Acquisition, Evolution and Re-use of Modules in Cartesian Genetic Programming. *IEEE Transactions on Evolutionary Computation*, 12(4):397–417.

[32] Ricardo Salem Zebulum and Adrian Stoica. (2006). Four-Function Logic Gate Controlled by Analog Voltage. *NASA Tech Briefs*, 30(3):8.