Sorting Network Development Using Cellular Automata

Michal Bidlo, Zdenek Vasicek, and Karel Slany

Brno University of Technology,
Faculty of Information Technology
Božetěchova 2, 61266 Brno, Czech republic
{bidlom,vasicek,slany}@fit.vutbr.cz

Abstract. The sorting network design represents a task that has often been considered as a benchmark for various applications of evolutionary design and optimization techniques. Although the specific structure of this class of circuits allows to use a simple encoding in combination with additional mechanisms for optimizing the area- and delay-efficiency of designed sorting networks, the design of large sorting networks represents a difficult task. This paper proposes a novel cellular automaton-based approach for the development of specific instances of sorting networks. In order to explore the area of generative cellular automata applied on this specific circuit structures, two different encodings are introduced: (1) an absolute encoding and (2) a relative encoding. The abilities of the both techniques are investigated and a comparative study is provided considering a variety of experimental settings.

Keywords: cellular automata, sorting networks, evolutionary design

1 Introduction

In recent years, many approaches were introduced for the evolutionary design of digital circuits. Probably the most popular approach is Miller's cartesian genetic programming [13]. His approach represents typical direct mapping between genotypes and phenotypes in the genetic algorithm for the evolution of digital circuits. Developmental systems represent an other class of systems that may be utilized for the circuit design. For example, Miller's developmental cartesian genetic programming [14], Tufte's FPGA-based approach for evolving functionality in cellular systems [18] or Gordon's developmental approach in evolvable hardware [6] represent instances of evolutionary developmental systems.

1.1 Cellular automata

Cellular automata (CA), originally invented by Ulam and von Neumann in 1966 [15], represent a mathematical model originally intended as a formal framework to study the behavior of complex systems, especially the questions of whether computers can self-replicate. Cellular automata may also be considered as a biologically inspired technique to model and simulate the cellular development.

A cellular automaton consists of a regular structure of cells, each of which can occur in one state from a finite set of states. The states are updated synchronously in parallel according to a local transition function. The synchronous update of all the cells of the CA is called a developmental step. The next state of a cell depends on the combination of states in the cellular neighborhood. In this paper we consider the cellular neighborhood consisting of the cell and its two immediate neighbors. Moreover, cyclic boundary conditions will be considered, i.e. the first and the last cell of the CA are considered to be neighbors and the 1D CA can be then viewed as a circle. The local transition function defines a next state of a cell for all the possible combinations of states in the cellular neighborhood. Let us denote $s_1 s_2 s_3 \rightarrow s_n$ a rule of the local transition function, where $s_1 s_2 s_3$ represents the combination of states of the cells in the cellular neighborhood and s_n denotes the next state of the particular middle cell.

Cellular automata have been applied to solve many complex problems in different areas. A detailed survey of the principles and analysis of various types of cellular automata and their applications is summarized in [19]. Sipper [17] investigated the computational properties of CA and proposed an original evolutionary design method for the "programming" of the cellular automata called cellular programming. He demonstrated the success rate of this approach to solve some typical problems related to the cellular automata, e.g. synchronization, ordering or the random number generation. In the recent years, scientists have been interested in the design of cellular automata for solving different tasks using the evolutionary algorithms. Miller investigated the problem of evolving a developmental program inside a cell to create multicellular organism of an arbitrary size and characteristic [12]. Tufte and Haddow utilized a FPGA-based platform of Sblocks [7] for the online evolution of digital circuits. The system actually implements a cellular automaton whose development determines the functions and interconnection of the Sblock cells in order to realize a function [18].

The cellular automata-based developmental approach has successfully been applied to the evolutionary design of combinational circuits [1]. This paper represents a continuation of this kind of research considering the development of sorting networks. Two different sets of experiments will be presented utilizing various encodings of the sorting networks in the developmental process of the cellular automaton. An absolute encoding and a relative encoding will be proposed in order to determine how the positional information, represented by an index of a cell in a CA, may influence the evolutionary design process and the properties of the sorting networks generated by the cellular automaton by means of those encodings. Several sets of experiments will be presented considering various setups of the developmental system. Statistical results of the evolutionary process and the properties of resulting sorting networks are investigated in the dependence on the experimental setup.

1.2 Sorting Networks and Their Design

The concept of sorting networks (SN) was introduced in 1954; Knuth traced the history of this problem in his book [11]. A sorting network is defined as a sequence of compare–swap operations (comparators) that depends only on the number of elements to be sorted, not on the values of the elements. A compare–swap of two elements (a, b) compares and exchanges a and b so that we obtain $a \leq b$ after the operation.

The main advantage of any sorting network is that the sequence of comparisons is fixed. Thus it is suitable for parallel processing and hardware implementation, especially if the number of sorted elements is small. Figure 1 shows an example of a 3-input sorting network.

The number of compare–swap components and the circuit delay are two crucial parameters of any sorting network. By delay we mean the minimal number of groups of compare–swap components that can be executed sequentially. Designers try to minimize the number of comparators, delay or both parameters. Some of the best currently known sorting networks were designed (or optimized) using evolutionary techniques [3, 5, 4, 8, 10, 9]. In most cases the evolutionary approach was based on the direct encoding given in Fig. 1 (in which comparator connections are encoded by using a pair of integers).

In order to find out whether an N-input sorting network operates correctly we should test $N!$ input combinations. Thanks to the *zero–one* principle this number can be reduced. This principle states that if an N-input sorting network sorts all 2^N input sequences of 0's and 1's into the non-decreasing sequence, it will sort any arbitrary sequence of N numbers into the non-decreasing sequence [11].

Sorting networks are usually designed for a fixed number of inputs. This approach also was applied in the mentioned evolutionary techniques. However, the evolutionary approach is not usually scalable. Conventional approaches already exist for generic design of the sorting networks with some examples of this approach (e.g. straight-insertion sort or select sort) described in [11]. These generic approaches were improved by evolution using a generative encoding called instruction-based development [16], [2]. However, the sorting networks created using these generic principles are not usually efficient in comparison

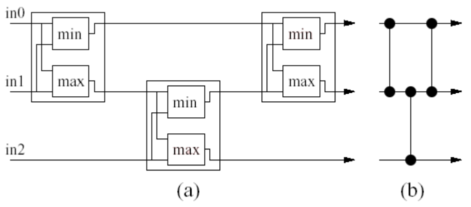

Fig. 1. (a) A three-input sorting network consists of three comparators. (b) Alternative symbol. This network can be described using the string (0,1)(1,2)(0,1).

with the appropriate instances designed and optimized for a fixed number of inputs.

2 Development of Sorting Networks Using Cellular Automata

In this section, two different encodings will be introduced for the development of sorting networks by means of cellular automata. Each encoding of the sorting network is based on a suitable enhancement of the local transition function of the CA. The fundamental principle of this enhancement is based on including an additional information to the local transition function (next to the new cell state) that represents a prescription for generating a compare–swap component. The meaning of this additional information and the way of generating the compare–swap components is described in the next paragraphs. In this paper, two different encodings of the sorting networks inside the local transition function are investigated: (1) an absolute encoding and (2) a relative encoding. Both the encodings are assumed to generate a comparator by each cell during every developmental step of the CA.

In both cases the number of cells (N) of the CA corresponds to the number of inputs of the sorting network to be developed. In general, a comparator is generated by each cell during the development of the CA. The comparator to be generated is specified by the rule of the local transition function that is applied to determine the next state of the cell depending on the combination of states in the cellular neighborhood. Therefore, up to N comparators can be generated in one developmental step of the CA. The conditions for including the generated comparator into the sorting network being developed are specified separately in each encoding that we have used. In order to ensure that the process of generating a sorting network is deterministic, a unique ordering of cells in the CA is introduced. The series of comparators generated by the cells is then specified by the ordering of the cells. The following ordering will be applied in all the experiments presented in this paper. Consider a CA that consists of four cells ordered as $c_0c_1c_2c_3$ and that performs three developmental steps. Then a series of comparators $C_{0,0}C_{1,0}C_{2,0}C_{3,0}C_{0,1}C_{1,1}C_{2,1}C_{3,1}C_{0,2}C_{1,2}C_{2,2}C_{3,2}$ is generated during the development of the CA, where $C_{i,j}$ represents a comparator generated by the cell c_i in the j-th developmental step.

The initial state of the CA together with the enhanced local transition function is the subject of the evolutionary design process (see Section 3 for details).

For the purposes of the experiments presented in this paper, let us denote S the number of possible cell states of the CA and T the number of steps of the CA after which the generated comparator sequence is evaluated.

2.1 Absolute Encoding

In fact, the absolute encoding represents a direct comparator-generating technique using a cellular automaton. In order to accomplish this process, a pair of non-negative digits (w_1, w_2) satisfying a relation $w_1 < w_2$ is associated with each rule of the local transition function. Therefore, a general form of a rule is

$$s_1 s_2 s_3 \rightarrow s_n : w_1\ w_2,$$

where the part on the right of the colon describes a comparator which is generated by a cell of the CA if this cell determines its next state according the given rule. These digits represent indices of inputs of the comparator to be generated; the range of both of them is from 0 to $N-1$, where N corresponds to the number of cells and the number of inputs of the sorting network to be developed.

For example, consider a 3-cell CA whose behavior is specified by its local transition function containing rules

$$(1)\ 010 \rightarrow 0 : 0\ 1,\ \ (2)\ 100 \rightarrow 1 : 1\ 2,\ \ (3)\ 001 \rightarrow 1 : 0\ 1.$$

The initial state of the CA is 100. Let the CA perform a developmental step, i.e. its state is in the form 011 after that. The first cell has determined its next state according to the rule (1), therefore it has generated a comparator (0, 1). The state of the second cell has been calculated using the rule (2) and a comparator (1, 2) has been created in a series with the previous one. Finally, the last comparator, (0, 1), has been generated by the third cell according to the rule (3). In summary, one developmental step of this CA has produced a sequence of comparators (0, 1)(1, 2)(0, 1) which correspond to the sorting network shown in Figure 1.

2.2 Relative Encoding

The aim of the relative encoding is to utilize the positions of the cells in a cellular automaton for generating the compare–swap elements. The enhanced local transition function consists of rules, each of which in the form

$$s_1 s_2 s_3 \rightarrow s_n : r\ d,$$

where the part on the right of the colon has the following meaning. The value of r specifies the index of the first comparator input w_1 relatively to the position (cell index) c of a cell that generates the comparator, i.e. $w_1 = c + r$. The range of r is considered from $-R$ to R, where R is a positive integer specified as a parameter for a given set of experiments. The value of d represents a "width" of a comparator, i.e. the difference between the indices of inputs of the comparator. Therefore, the index of the second input w_2 is calculated as $w_2 = w_1 + d$. The maximal value of d (let us denote it D) represents the second parameter of the design system. The value of D was determined experimentally as $D = 2R$ for a given set of experiments. If w_1 or w_2 exceeds the index range of the inputs of the target sorting network, then the comparator is not generated (i.e. it is not included in the comparator sequence generated by the CA).

An example of a sorting network development using the relative encoding is illustrated by Figure 2 (initial state of CA is 0100). The cells of the CA and the inputs (wires) of the target sorting network are indexed by integer values in range from 0 to 3. The first cell at the position $c = 0$ generates a comparator using the relative value $r = 0$ and the width of the comparator $d = 1$ (see the pair $0, 1$ as specified in the rule on the right of the first cell). Therefore, the first input of the comparator is calculated as $w_1 = c + r = 0 + 0 = 0$. the second input is calculated as $w_2 = w_1 + d = 0 + 1 = 1$ and the comparator (0, 1) is generated (see the comparator denoted as 1 at the right part of Figure 2). The same principle is used for generating comparator 2 (2, 3), 3 (0, 2), and 4 (1, 3). After the first step, CA possesses the state 1110. During the second step the first cell (at $c = 0$) generates a comparator using the relative value -1 and width 1. However, after calculating the comparator inputs a pair (-1, 0) is obtained. This is not a valid comparator for a 4-input network and therefore it is not included in the developed comparator sequence (illustrated by a dashed comparator 5 of the sorting network in Figure 2). Similarly, comparator 7 (3, 4) generated by the cell at $c = 2$ is also invalid and hence meaningless for the target sorting network. The sorting network shown in the right part of Figure 2 has been created in two developmental steps of the CA. Note that the comparator 8 (0, 3) is redundant in this network because it does not swap any values during the complete test of the sorting network and therefore it can be removed from the comparator sequence without loss of the network functionality.

3 Evolutionary System Setup

The simple genetic algorithm was utilized for the evolutionary design of the cellular automaton that generates a target sorting network. Two sets of experiments will be presented regarding the development of this kind of circuits using the absolute and relative encoding. In both sets of experiments the initial state of the CA is evolved together with its local transition function. The initial state is encoded in the chromosome as a finite sequence of integers. A rule of the enhanced local transition function consists of the next state and two integer values whose range and meaning differs for the absolute and relative encoding (see Section 2.1 and 2.2). The general structure of a chromosome is in the form

$$s_0\ s_1 \ldots s_{N-1}\ ns_0\ x_0\ y_0\ ns_1\ x_1\ y_1 \ldots ns_{|Q|^3-1}\ x_{|Q|^3-1}\ y_{|Q|^3-1},$$

where s_i is the initial state of i-th cell ($i = 0, 1, \ldots N-1$), ns_j is the new state for the appropriate combination of states in the cellular neighborhood expressed

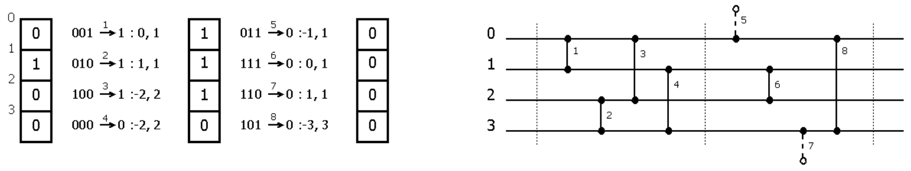
0
1
2
3
0
1
0
0
$001 \xrightarrow{1} 1 : 0, 1$
$010 \xrightarrow{2} 1 : 1, 1$
$100 \xrightarrow{3} 1 : -2, 2$
$000 \xrightarrow{4} 0 : -2, 2$
1
1
1
0
$011 \xrightarrow{5} 0 : -1, 1$
$111 \xrightarrow{6} 0 : 0, 1$
$110 \xrightarrow{7} 0 : 1, 1$
$101 \xrightarrow{8} 0 : -3, 3$
0
0
0
0
0
1
2
3
1
2
3
4
5
6
7
8

Fig. 2. An example of development of a 4-input sorting network (4-cell CA is used).

by its index $j = 0, 1, \ldots |Q|^3 - 1$ ($|Q|$ is the number of possible cell states), x_j and y_j is the additional information according to which the comparators are generated.

The index (position in the genome) is specified implicitly by means of the value expressed by the number representing the combination of states in the cellular neighborhood. Therefore, if we consider the general form of the rule $s_1\ s_2\ s_3 \rightarrow s_n : x\ y$, only the part on the right of the arrow is encoded in the genome. For example, if a cellular automaton with 2 different states and the cellular neighborhood consisting of 3 cells ought to be evolved, there are 2^3 rules of the local transition function. Consider the rule $0\ 1\ 1 \rightarrow 0 : 2\ 3$. Since the combination of states 0 1 1 corresponds to the binary representation of value 3, this rule will be placed in the chromosome at the position 3 of the local transition function.

In all the experiments, the population consists of 20 chromosomes which are initialized randomly (with respect to the correct range of each gene) at the beginning of evolution. The chromosomes are selected by means of the tournament operator with the base 4. Only mutation operator is utilized. In each chromosome selected by the tournament operator, 5 genes are chosen randomly and each of them is mutated with the probability 0.95.

The fitness function is calculated as the number of correct output bits of the sorting network using all the binary input test vectors. For example, there are 2^4 test vectors in case of 4-input SN. Therefore, the fitness value of a perfect solution is $F_{max} = 4 \cdot 2^4 = 64$. If no solution is evolved in 100,000 generations the evolutionary run is terminated.

4 Experimental Results and Discussion

The experiments were focused of the evolutionary development of 16-input sorting networks by means of one-dimensional uniform cellular automata. 16-input networks were chosen as a benchmark problem for the proposed developmental encodings.

In general, sorting networks exhibit a specific structure in which a comparator represents a basic building block. The comparator approach to the design of sorting networks actually represents a higher level of abstraction of this kind of circuits in comparison with the basic gate-level representation. A specific feature of a comparator is that the function of a SN does not go wrong if an arbitrary valid comparator is appended to the existing comparator sequence which, in fact, may simplify the design process. However, unsuitable arrangement of the comparators may cause both area- and delay-inefficient sorting networks. These properties are hence the subject of investigation with respect to different setup of the developmental system. Note that we do not deal with the optimization of the sorting networks during the evolutionary process in this stage of research.

Table 1. Statistical results of the evolutionary process using the absolute encoding.

	Success rate in %					**Average number of generations**				
	CA steps					**CA steps**				
states	**5**	**6**	**7**	**8**	**9**	**5**	**6**	**7**	**8**	**9**
3	-	-	-	1	33	-	-	-	68,9k	58,6k
4	-	45	95	98	100	-	58,5k	23,1k	10,5k	5,4k
5	16	98	100	100	100	49,0k	8,7k	3,3k	2,9k	1,3k
6	65	100	100	100	100	32,8k	5,5k	1,7k	0,8k	0,8k

4.1 Results from the Absolute Encoding

The absolute encoding may be considered as the simplest representation of the comparators inside the developmental process of a cellular automaton. The crucial part of the design process is that the evolution searches for an enhanced local transition function of the CA containing suitable set of comparators that are encoded directly (by the indices of their inputs). The experimental setup of the developmental system includes the setting of the number of cell states and the number of steps of the CA. These values are specified at the beginning of the evolutionary process.

Table 1 shows the success rate and the average number of generations for the experiments using the absolute encoding. A hundred independent experiments were performed for each combination of the number of states and the number of steps of the CA. As evident, it is easy to evolve a fully functional solution if the number of states is greater than 4 and the number of steps greater than 6. As there are many combinations of inputs possible for a comparator in a 16-input sorting networks, three states of the CA showed to be a minimum to evolve a working solution. For two states, no CA was found within 100,000 generations. If the number of states is sufficient, then there is a higher probability that a working solution is found in the given number of generations (see Table 1) for 5 steps. Moreover, the number of generations needed to evolve a working solution is substantially lower in those cases, although the search space is more complex due to the number of CA states. This shows that there are many correct solutions in such search space which is probably heavily influenced by the specific structure of sorting networks. On the other hand, more developmental steps are needed if the number of states is low (see Table 1).

Table 2 contains the average properties (the number of comparators and delay) of the resulting sorting networks for different experimental setup. These results show (as one would usually expect) that if more steps are performed to create *s* working SN, then this SN exhibit worse parameters (more comparators are needed, the delay is higher). However, if the results are compared for a given number of steps, the sorting networks developed with a higher number of states exhibit slightly better properties in most cases (especially from the point of view of the delay). This may be caused by the possibility of higher number of different comparators that can be generated in one step thanks to the higher number of different combinations of states in the cellular neighborhood.

Table 2. Properties of the resulting SNs obtained from the absolute encoding.

	Average number of comparators					**Average delay**				
	CA steps					**CA steps**				
states	**5**	**6**	**7**	**8**	**9**	**5**	**6**	**7**	**8**	**9**
3	-	-	-	93,0	98,8	-	-	-	28,0	30,8
4	-	85,4	90,0	92,4	95,7	-	27,3	29,3	30,1	31,8
5	78,4	85,5	90,1	92,1	94,8	23,1	26,9	28,6	28,9	30,9
6	78,0	86,1	90,0	92,5	94,3	22,2	26,4	28,6	29,3	30,6

4.2 Results from the Relative Encoding

The goal of this set of experiments is to investigate the sorting networks development that involves positional information of cells inside the CA to determine the inputs of the comparators being generated. Since the enhanced transition function of the CA includes information for calculating the indices of comparator inputs (i.e. the relative position value and the comparator width), the experimental setup includes, besides the number of states and the number of steps of the CA, limit value of the relative position R according to which the maximal comparator width is also calculated. These values are specified at the beginning of the evolutionary process.

Table 3 summarizes the success rate for the evolutionary experiments utilizing the relative encoding. It can be observed that the dependence of the success rate on the increasing number of states and developmental steps is for the given value of R very similar to the results obtained from the absolute encoding. Interestingly, the success rate decreases with increasing the maximal comparator width if the number of states is small (2 and 3 states). This dependency is inverse for 4 states. Although no correct solution was evolved in 100,000 generations for less than 6 steps, the evolution succeeded with utilizing only 2 states of the CA. This result shows that the generation of comparators relatively with respect to the cell position has a significant influence on the ease of building a sorting network. However, the increasing number of states does not reduce the computational effort (expressed by the average number of generations needed to find a working solution) in comparison with the absolute encoding - see Table 4. Although the average number of comparators of the resulting SNs decreases with increasing R for a given number of states and developmental steps (see Table 5), it is difficult to observe a significant dependence of average delay on varying R (Table 6). However, it is possible to say that the properties of the resulting networks are better for the relative encoding (in comparison with the absolute encoding) especially for a higher number of developmental steps.

5 Conclusions

In this paper a developmental method based on uniform 1D cellular automaton was presented for the design of sorting networks. Two different encodings of the

Table 3. Success rate in % for the evolution using the relative encoding.

	CA steps / relative limit											
	6			**7**			**8**			**9**		
states	**2**	**3**	**4**	**2**	**3**	**4**	**2**	**3**	**4**	**2**	**3**	**4**
2	-	-	-	41	30	30	99	98	90	100	100	100
3	-	-	3	45	43	33	99	98	97	100	100	100
4	1	10	15	76	92	81	99	100	100	100	100	100

Table 4. Average number of generations of the evolutionary process using the relative encoding. The values are measured in thousands of generations.

	CA steps / relative limit											
	6			**7**			**8**			**9**		
states	**2**	**3**	**4**	**2**	**3**	**4**	**2**	**3**	**4**	**2**	**3**	**4**
2	-	-	-	26,5	26,5	21,3	3,21	4,94	7,13	0,19	0,35	0,57
3	-	-	58,4	22,2	26,8	26,9	5,94	4,84	7,51	0,48	0,56	1,37
4	39,1	36,2	42,2	21,0	21,1	27,2	5,12	5,35	8,86	1,23	1,67	3,89

Table 5. Average number of comparators of SNs developed using the relative encoding.

	CA steps / relative limit											
	6			**7**			**8**			**9**		
states	**2**	**3**	**4**	**2**	**3**	**4**	**2**	**3**	**4**	**2**	**3**	**4**
2	-	-	-	91,2	89,9	88,5	97,5	94,4	93,9	98,0	96,9	97,7
3	-	-	83,7	91,8	89,5	88,3	95,6	93,1	92,7	98,6	96,9	95,5
4	83,0	84,1	83,1	90,3	88,3	87,1	93,5	91,7	90,7	96,1	94,5	92,6

Table 6. Average delay of SNs developed using the relative encoding.

	CA steps / relative limit											
	6			**7**			**8**			**9**		
states	**2**	**3**	**4**	**2**	**3**	**4**	**2**	**3**	**4**	**2**	**3**	**4**
2	-	-	-	20,9	20,3	20,0	22,7	24,6	24,7	27,8	26,9	28,8
3	-	-	30,0	27,1	28,8	27,8	29,2	28,9	29,8	30,4	30,1	29,9
4	24,0	26,0	27,8	28,6	28,8	28,0	30,0	29,5	29,6	31,0	31,3	30,4

sorting networks in the process of development of the CA were proposed: (1) an absolute encoding and (2) a relative encoding. The goal was to investigate the influence of utilization of relative positional information on the evolutionary design process and the properties of resulting sorting networks.

The results showed that the number of states and the number of steps of the CA has a significant influence on the ability of the CA to develop successfully a working sorting network. The relative encoding was shown as more suitable for the development of SNs using a lower number of states. Moreover, the sorting networks designed by means of this encoding exhibit better properties in average in comparison with the absolute encoding.

As evident, the resulting networks are neither area-efficient nor delay-efficient. The currently best-known 16-input SN consists of 60 comparators working with the delay 10. The best result obtained from the absolute encoding contains 75 comparators and its delay is 16. The relative encoding produced the best SN with 92 comparators and the delay 14. The significant difference in the proposed approach is that we have used a developmental encoding whilst the best result was obtained using a direct representation with an explicit area/delay optimization mechanism (e.g. see [3]). Another example represents a generic developmental approach proposed in [16] by means of which a 92-comparator network was created working with the delay 21. The findings from the results presented herein are interesting especially for the future research considering the application of cellular automata in which we are going to focus on advanced encodings able to reduce of the resulting properties during the developmental process. Moreover, the possibilities of designing regular structures will be investigated with the utilization of these encodings which may lead to the research of generic design using cellular automata.

Acknowledgement

This work was partially supported by the Grant Agency of the Czech Republic under contract No. GP103/10/1517 *Natural Computing on Unconventional Platforms*, No. GD102/09/H042 *Mathematical and Engineering Approaches to Developing Reliable and Secure Concurrent and Distributed Computer Systems*, the Grant Fund (GRAFO) of Brno University of Technology (BUT), the internal BUT research project No. FIT-S-10-1 and the Research Plan No. MSM 0021630528 *Security-Oriented Research in Information Technology.*

References

1. Bidlo, M., Vasicek, Z.: Gate-level evolutionary development using cellular automata. In: Proc. of The 3nd NASA/ESA Conference on Adaptive Hardware and Systems, AHS 2008. pp. 11–18. IEEE Computer Society (2008)
2. Bidlo, M., Škarvada, J.: Instruction-based development: From evolution to generic structures of digital circuits. International Journal of Knowledge-Based and Intelligent Engineering Systems 12(3), 221–236 (2008)

3. Choi, S.S., Moon, B.R.: A hybrid genetic search for the sorting network problem with evolving parallel layers. In: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001). pp. 258–265. Morgan Kaufmann, San Francisco, California, USA (2001)
4. Choi, S.S., Moon, B.R.: Isomorphism, normalization, and a genetic algorithm for sorting network optimization. In: GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference. pp. 327–334. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (2002)
5. Choi, S.S., Moon, B.R.: More effective genetic search for the sorting network problem. In: Proc. of the Genetic and Evolutionary Computation Conference GECCO 2002. pp. 335–342. Morgan Kaufmann, New York, US (2002)
6. Gordon, T.G.W., Bentley, P.J.: Towards development in evolvable hardware. In: Proc. of the 2002 NASA/DoD Conference on Evolvable Hardware. pp. 241–250. IEEE Press, Washington D.C., US (2002)
7. Haddow, P.C., Tufte, G.: Bridging the genotype–phenotype mapping for digital FPGAs. In: Proc. of the 3rd NASA/DoD Workshop on Evolvable Hardware. pp. 109–115. IEEE Computer Society, Los Alamitos, CA, US (2001)
8. Hillis, W.D.: Co-evolving parasites improve simulated evolution as an optimization procedure. Physica D 42(1–3), 228–234 (June 1990)
9. J. R. Koza and F.H. Bennett and D. Andre and M. A. Keane: Genetic Programming III: Darwinian Invention and Problem Solving. Morgan Kaufmann (1999)
10. Juillé, H.: Evolution of non-deterministic incremental algorithms as a new approach for search in state spaces. In: Proc. of 6th Int. Conference on Genetic Algorithms. pp. 351–358. Morgan Kaufmann (1995)
11. Knuth, D.E.: The Art of Computer Programming: Sorting and Searching (2nd ed.). Addison Wesley (1998)
12. Miller, J.F.: Evolving developmental programs for adaptation, morphogenesis and self-repair. In: Advances in Artificial Life. 7th European Conference on Artificial Life, Lecture Notes in Artificial Intelligence, volume 2801. pp. 256–265. Springer, Dortmund DE (2003)
13. Miller, J.F., Thomson, P.: Cartesian genetic programming. In: Proc. of the 3rd European Conference on Genetic Porgramming, Lecture Notes in Computer Science, vol 1802. pp. 121–132. Springer, Berlin Heidelberg New York (2000)
14. Miller, J.F., Thomson, P.: A developmental method for growing graphs and circuits. In: Proc. of the 5th Conf. on Evolvable Systems: From Biology to Hardware (ICES 2003), Lecture Notes in Computer Science, vol. 2606. pp. 93–104. Springer-Verlag, Berlin DE (2003)
15. von Neumann, J.: The Theory of Self-Reproducing Automata. A. W. Burks (ed.), University of Illinois Press (1966)
16. Sekanina, L., Bidlo, M.: Evolutionary design of arbitrarily large sorting networks using development. Genetic Programming and Evolvable Machines 6(3), 319–347 (2005)
17. Sipper, M.: Evolution of Parallel Cellular Machines – The Cellular Programming Approach, Lecture Notes in Computer Science, volume 1194. Springer-Verlag, Berlin (1997)
18. Tufte, G., Haddow, P.C.: Towards development on a silicon-based cellular computing machine. Natural Computing 4(4), 387–416 (2005)
19. Wolfram, S.: A New Kind of Science. Wolfram Media, Champaign IL (2002)