Introducing New Fundamental Classification of Development for Evolutionary Design: Theory and Applications

Michal Bidlo

Faculty of Information Technology, Brno University of Technology
Božetěchova 2, 612 66 Brno, Czech Republic
bidlom@fit.vutbr.cz

Abstract. In this paper two fundamental approaches to the computational development are introduced in connection with the evolutionary design: the infinite development and the finite development. Both the principles are defined formally and, moreover, two practical applications are presented demonstrating the abilities of these non-traditional techniques to design digital circuits. Finally, some key features of the proposed approaches are highlighted.

1 Introduction

Nowadays the evolutionary algorithms are widely used to solve many complex problems of the current science and engineering covering various fields (e.g. computer science, electronics, mechanical engineering, physics etc.). Two basic applications of the evolutionary algorithms can be distinguished: (1) the evolutionary optimization and (2) the evolutionary design. While the first case usually considers a working system for which a set of parameters are identified and subsequently optimized using an evolutionary algorithm, the second case uses evolutionary algorithms to explore the state space of *designs* related to a given problem, i.e. the solution is constructed "from scratch" using the components (building blocks) specified by the designer. Considering the diversity of the target structures (for example, electronic circuits, computer programs, buildings etc.), the representation of the candidate solutions plays a big part.

In general, the problem is that as the task to be solved by means of an evolutionary algorithm gets more complex, the amount of information increases that is needed to encode the candidate solutions in the chromosome. Therefore, the length of the chromosome increases, the search space becames enoumously large and the evolutionary algorithm is not able to explore it effectively. This issue is referred to as the *problem of scale*. This is particularly the difficulty in the traditional evolutionary algorithms as typically there is a one-to-one relationship between the genotype and the corresponding solution description. The standard answer to such issues has been the addition of problem-specific knowledge into the genetic representations. For example, we can do this by telling the computer

that subroutines, loops, symmetry, regularity or subdesigns are necessary. However, for very complex solutions we may not have any knowledge about the best way how to solve the problem. And even if we do, by adding our own ideas about how solutions should be constructed, we constrain evolution and prevent it from finding alternatives [10].

Three major approaches have been developed so far to overcome the problem of scale: functional level evolution [7], incremental evolution [12, 13] and the embryonic approach (usually referred to as the *development*).

This paper deals with the development. Although it does not deal directly with the techniques of the evolutionary algorithms themselves, the topic is important for both theory and practice of the evolutionary computation since it enables us to investigate the features of the evolutionary algorithms in connection with a non-traditional mechanisms — the developmental models — in the applications of the evolutionary design. The objective is to introduce a new classification of the developmental systems in the area of the evolutionary design: the infinite development and the finite development. Both the approaches are defined formally considering a general evolutionary algorithm in which a form of development is applied. Considering the existing applications of the traditional evolutionary design (without using the development), many innovative solutions have been discovered so far in comparison with the conventional approaches (for instance, see [4]). According to this fact, we could expect similar abilities in the applications of the evolutionary design utilizing the development. Therefore, the following hypothesis can be formulated.

Hypothesis *The evolutionary design utilizing a form of development is able to discover solutions which are unreachable using the traditional evolutionary design, possibly solutions which have not been known yet.*

The paper is organized as follows. Section 2 briefly summarizes the basic principles of the development. Section 3 introduces two fundamental approaches to the computational development in connection with the evolutionary design: the infinite development and the finite development. These techniques are described in terms of theoretical computer science. The applications demonstrating the abilities of both the approaches are presented in Section 4. Finally, the discussion of the features of the proposed principles and concluding remarks are stated in Section 5.

2 Computational Development

In nature, the process of development is influenced by the genetic information of the organism and the environment in which the process is carried out. Cells use the mechanism of transcription and translation to read each gene and produce the string of amino acids that makes up a protein. Proteins activate or suppress synthesis of other genes, work as signals among cells, influence internal functions of the cells and perform many other important roles. Therefore, they control the growth, position and behavior of all cells [2].

In evolutionary algorithms (EA), the process of development (more precisely, computational development [5]) is usually considered as a non-trivial *genotype–phenotype mapping*. While genetic operators work with genotypes, the fitness calculation is applied on phenotypes created by means of a developmental system.

The utilization of the computational development is primarily motivated by the fact that natural development is one of the phenomena which is primarily responsible for the extraordinary diversity and sophistication of living creatures. It is assumed that the computational development (inspired by natural development) in connection with an evolutionary algorithm might be utilized to achieve the evolution of complex artificial objects and other objectives desired by evolutionary design systems, including adaptation, compacting genotypes, reduction of search space, regulation, regeneration, repetition, robustness, evolvability, parallel construction, emergent behavior and decentralized control (as discussed in [5]).

Computer science offers a number of suitable methods for modeling the development. Probably the most popular are cellular automata [15], Lindenmayer systems (L systems) [6], general rewriting systems (grammars) or application-specific computer programs. These techniques can be considered as a basis for the construction of more complicated developmental models.

For instance, Sekanina and Bidlo have created an instruction-based developmental mapping for the evolutionary design of arbitrarily large sorting networks using an iterative process – repeated application of an evolved program (a constructor) [11]. Haddow et al have applied the principles of L systems to the development of digital circuits and presented an implementation platform enabling on-chip evaluation of grown solutions [3]. Tufte and Haddow have presented the evolutionary design of cellular computing machines implemented inside a FPGA for the investigation of structural and functional properties generated by the development of a cellular automaton [14].

3 Evolutionary Design Using Development

In classical evolutionary design the chromosomes (genomes) of the evolutionary algorithm directly encode the candidate solutions of a given problem. For the purposes of the fitness calculation the chromosome (genotype) is decoded in order to obtain the candidate solution (phenotype) which is evaluated using the fitness function. The process of decodig the genomes usually represents a one-to-one mapping between the genotype (representation) space and the phenotype (solution) space. For example, Miller's Cartessian Genetic Programming directly encodes the candidate solutions into the genome representation [9].

The development has been introduced into the artificial evolutionary process in order to overcome the problem of scale. It usually represents a *prescription* encoded in the chromosome (genotype) for the construction of a target object (phenotype) from a trivial instance of the problem – an embryo. This type of representation is referred to as the *developmental encoding*. Note that the embryo

can but need not be included in the chromosome, i.e. it can be either evolved or it can be created by the designer as a constant part of the developmental system. It is evident that this approach puts emphasis on the process of the construction of the target object rather than the structure of the target object itself which represents the main difference in comparison with the classical evolutionary design.

3.1 Basic Approaches to the Development

During the investigation of existing models of the development, the following objectives have been identified:

1. The development is utilized in order to overcome the problem of scale.
2. Various properties of a specific developmental model are investigated in solving a particular problem.

In the first case, we can require the target system to "grow" (develop) for a (theoretically infinite) number of iterations and to be fully functional at each moment of the development. It means that the system can develop continually and infinitely. The second case includes the systems in which we are usually interested in the "final object" which emerges by applying the developmental rules, or the systems in which we do not require endless development.

According to this features, two basic approaches to the computational development can be introduced: (1) the *infinite development* and (2) the *finite development*. Since this is a new concept presented in the area of the evolutionary design, more research is required to investigate it in more detail and determine particular relations when applied in a specific developmental model. In this paper the crucial definition will be introduced and, moreover, some initial features will be summarized which emerged from the experimental results obtained so far.

Before introducing the formal definition of the concept, some basic elements of an evolutionary algorithm and development need to be stated. Let $\mathcal{C}$ be a *representation space* containing the chromosomes (genotypes) utilizing a form of developmental encoding. In order to apply the embryonic approach described in the second paragraph of this section, we will consider the representation space $\mathcal{C} \subseteq \mathcal{S} \times \mathcal{D}$, where $\mathcal{S}$ denotes a *solution space* containing the candidate solutions (phenotypes) of a given problem and $\mathcal{D}$ represents a *space of developmental algorithms* related to the selected developental encoding and the problem to be solved. Note that the concept of the chromosome structure enables the embryo to be evolved together with the developmental algorithm or it can be fixed (designed a priori by the designer) depending on the genetic operators of the utilized evolutionary algorithm. A chromosome actually comprises a pair (e, Δ), where $e \in \mathcal{S}$ denotes an embryo and $\Delta \in \mathcal{D}$ represents a *developmental algorithm* which we will define formally as $\Delta : \mathcal{S} \to \mathcal{S}$ and for every $s_i \in \mathcal{S}$: $s_{i+1} = \Delta(s_i)$, where $s_{i+1} \in \mathcal{S}$ and $\Delta(s_i)$ denotes a *developmental step*. If $s_0 \in \mathcal{S}, s_1 = \Delta(s_0), s_2 = \Delta(s_1), \ldots, s_i = \Delta(s_{i-1})$ for arbitrary $i > 0$, then we define i-th iteration of Δ:

$s_i = \Delta^i(s_0)$ and call it the i-th developmental step, where $s_0 = e$ is an embryo and s_i is called the *i-th stage* of the development of e. Finally, let $\mathcal{F}$ be the space of fitness values. Then $\Phi : \mathcal{S} \rightarrow \mathcal{F}$ denote a fitness function performing the evaluation of the candidate solutions. The proposed concept is illustrated in Figure 1.

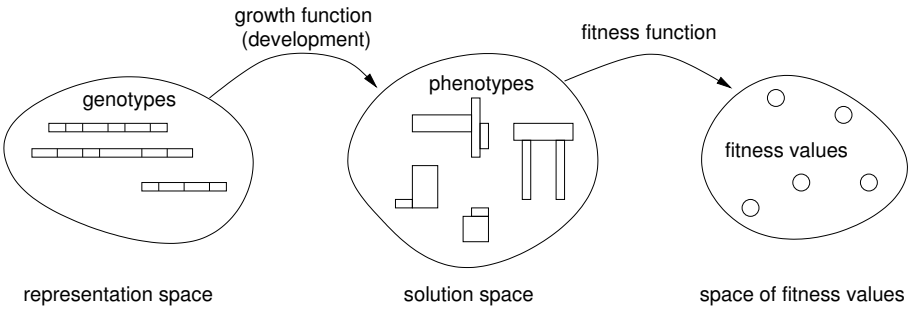

Fig. 1. The concept of genotype–phenotype mapping using the development

The description of the infinite and finite development is based on the fitness function determining how the object under development fulfils the requirements specified by the designer and utilizes the terms introduced in the previous paragraph.

Definition 1 (The infinite development, the finite development). *Let $(e, \Delta) \in \mathcal{C}$ be a chromosome involving a developmental encoding. Δ is said to have been performing the **infinite development**, if and only if there is NO finite integer $k > 0$, such that $\Phi(\Delta^d(e)) < F_d$ for all $d > k$, where F_d represents a fitness value by which the d-th stage of e (i.e. the candidate solution $\Delta^d(e)$) fulfils the requirements specified by the designer. Otherwise, Δ is said to have been performing the **finite development**.*

The definition can be explained as follows:

1. If the target object stops working after a finite number of developmental steps (it can continue the development but its functionality is not satisfactory anymore in order to solve a given problem), then the development is considered to be finite. In this case we are usually interested in the latest working solution such that $\Phi(\Delta^k(c)) \geq F_k$.
2. In case of the infinite development the target object can grow for unlimited number of developmental steps preserving the ability to solve a given problem in defined states of its development. It is evident that the target system need not necessarily work properly after each developmental step. For example, the developmental system can produce correct solutions after each second developmental step, gradually in the steps $d = 1, 3, 6, 10, 15, \ldots$ or even in each tenth stage after the hundredth stage depending on the developmental

encoding, the problem to be solved and the requirements specified by the designer.

Note that the particular developmental models can be adapted to perform either the infinite or the finite development. For instance, cellular automata can develop endlessly or can perform only a given finite number of developmental steps, computer programs can be executed repeatedly and iteratively etc. Therefore, the terms infinite development and finite development relates to the functionality of the objects being developed rather than to the utilized developmental encoding.

4 Applications of the Infinite and Finite Development

In this section two applications of the basic approaches to the development will be presented from the area of the evolutionary design of digital circuits. First, an application of the evolutionary design of arbitrarily large sorting networks will be proposed (the infinite development). The second application deals with the evolutionary design of combinational circuits with a given number of inputs (the finite development).

4.1 Evolutionary Design of Arbitrary Even-Input Sorting Networks Using the Infinite Development

A genetic algorithm has been used to design a *constructor* (a program) that is able to create a larger sorting network from a smaller sorting network (the smallest one is called the *embryo*). The constructor — directly represented by the chromosome — is a finite sequence of instructions, each of which is encoded as three integers: operation code, arg. 1 and arg. 2. The instruction set is described in Tab. 1.

Table 1. Instruction set utilized in the development. Let $[c; d]$ be a comparator. By applying an instruction to it, a new comparator $[c'; d']$ is created. Experiments have shown to be useful to put $[c'; d']$ into the newly created sorting network only if $c' < d' \wedge c' < N \wedge d' \leq N$. N denotes the number of inputs of the emerging sorting network, ip, ep and np represent instruction pointer, embryo pointer and next-position pointer respectively.

Instruction	Arg1	Arg2	Description
0: ModS	a	b	$c' = c + a, d' = d + b, ip = ip + 1, np = np + 1$
1: ModM	a	b	$c' = c + a, d' = d + b, ip = ip + 1, np = np + 1, ep = ep + 1$
2: CpS	k	–	copy $N - k$ comparators, $ip = ip + 1, np = np + N - k$
3: CpM	k	–	copy $N - k$ comparators, $ip = ip + 1, np = np + N - k$, $ep = ep + N - k$

Table 2. The number of comparators and delay of the sorting networks created by means of the evolved constructor in comparison with the conventional algorithm

#inputs	6	8	10	12	14	16	18	20	22	24	26	28
Evolved #comp.	12	22	35	51	70	92	117	145	176	210	247	287
Conv. #comp.	15	28	45	66	91	120	153	190	231	276	325	378
Evolved delay	6	9	12	15	18	21	24	27	30	33	36	39
Conv. delay	9	13	17	21	25	29	33	37	41	45	49	53

First, the constructor is applied to the embryo in order to build a larger sorting network. Then the same constructor is applied to this sorting network and larger circuit emerges again. This approach can be repeated so that the sorting network "grows" continually and infinitely (see Fig. 2). The goal is to obtain a valid sorting network after each constructor application. The constructor possessing this ability is said to be a *general constructor*. A single application of the constructor represents a developmental step. The difference between the number of inputs of two neighbouring sorting networks during the development denotes the *size of the developmental step*. Note that the sorting network resulted from the application of the constructor contains all the comparators of its predecessor. Fintess value is computed as the sum of the number of test vectors sorted correctly using the resulting SN after each developmental step. Since it is not possible to verify all the sorting networks, only three developmental steps are considered for the fitness calculation. For instance, using a 4-input embryo and the developmental step of size 2, $Fitness = F(6) + F(8) + F(10)$, where $F(i)$ is the number of test vectors sorted correctly by the developed i-input sorting network. The generality of the evolved constructors is verified for the construction of up to 28-input sorting networks.

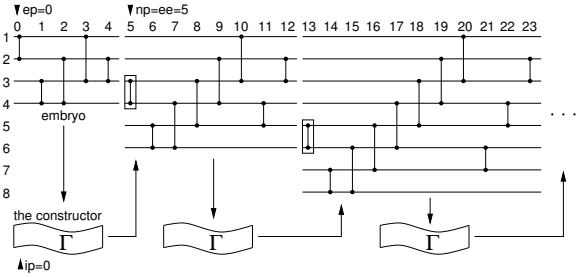

Fig. 2. The principle of designing larger sorting networks from smaller sorting networks by means of a constructor. The growth of the SN is illustrated using the best evolved constructor Γ=(ModS 2 2)(ModS 4 4)(ModS 3 4)(ModM 2 3)(ModM 2 0)(CpM 4 2)(ModS 2 2)(CpM 4 4). The initial configuration of pointers is shown.

Figure 2 shows the principle of the developmental method and the growth of the sorting network using the best evolved constructor. A 4-input embryo was utilized. The embryo was designed a priory by the designer and is not the subject of evolution. The instructions are exetuted sequentially; the instruction pointed by instruction pointer (ip) processes the comparator pointed by embryo pointer (ep) and the resulting comparator is placed to the first free position pointed by next-position pointer (np). After executing an instruction, the pointers are updated according to Tab. 1. A developmental step is finished if the last instruction of the constructor is executed or the end of embryo is reached ($ep = ee$). At the beginning of the developmental process, ep and ip are initialized to 0, ee and np are set to the first free position. After each developmental step, ip is initialized to 0, ep and np keep their values resulted from the last developental step and ee is set to the first free position.

The constructors were evolved as variable-length chromosomes. The following setting of the evolutionary system was determined experimentally to solve the given task. A simple genetic algorithm was utilized with population size 60, crossover 0.7 and mutation 0.023. Using this setting 100 independent experiments were conducted from which 40 general constructors were evolved.

The sorting networks produced by the best evolved constructor contain redundant comparators (in Fig. 2, the marked comparators at positions 5 and 13). However, these comparators can be removed from the SN without any loss of functionality. After their removal, the delay of sorting networks is reduced substantially and, moreover, they require less comparators in comparison with conventional SNs. Table 2 survey the properties of the evolved sorting networks in comparison with the conventional insertion/selection principle. Note that this constructor has been proved to be general, i.e. able to create theoretically infinitely large sorting network [1].

4.2 Evolutionary Design of Combinational Circuits Using the Finite Development

As an example of developmental encoding realizing the finite development, a *generative cellular automaton* will be introduced. Generative cellular automaton (GCA) has been devised in order to generate various structures by means of a cellular automaton. GCA comprises an ordinary cellular automaton (CA) whose each rule of the local transition function is associated with *symbol* to be generated by a cell which applys the rule in order to determine its next state. Since the cells of the automaton operate synchronously (in parallel), one symbol is generated by each cell at each developmental step of the CA. The symbols can be interpreted in an arbitrary manner depending on a particular application.

For the purposes of generating digital circuits, the following settings of the developmental system was chosen. The generative one-dimensional cellular automaton consisting of a linear structure of z cells ($z \geq 1$) is intended to generate a combinational circuit of z inputs. The next state of each cell depends on the state of itself and the states of its two immediate neighbors. The symbols of the form ($F\ i_1\ i_2$) generated by the cells represent logic gates, where F denotes

the logic function, $1 \leq i_1 \leq z$ and $1 \leq i_2 \leq z$ denote the indices determining the connection of inputs of the gate. Only basic one-input and two-input logic gates and wires are considered, each of them contains just one output. Every cell generates just one gate in each developmental step of the GCA, i.e. z independent components (one level of the circuit) is created in each developmental step. Therefore, the number of developmental steps determines delay of the circuit. In case of the first developmental step ($d = 1$) the inputs of the gates being generated are connected to the primary inputs of the circuit, otherwise (for $d = i, i > 1$) they are connected to the outputs of the gates generated in the previous developmental step ($d = i - 1$). Feedback is not allowed. Note that this developmental model does not explicitly use any embryo. The target circuit is created "from scratch" according to the development of the cellular automaton whose cells generate the logic gates. Figure 3 shows the principle of the GCA on an example of the development of 2-bit multiplier.

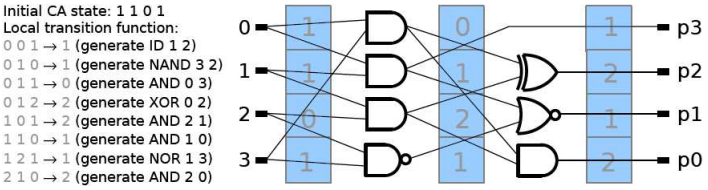

Fig. 3. The development of the best evolved 2-bit multiplier using a generative cellular automaton and the finite development. Note that zero boundary conditions are applied, i.e. the upper neighbor of the top cell and the lower neighbor of the bottom cell possess the states 0. For example, the NAND gate in the first level of the circuit is generated using the rule 010 of the local transition function determining the next state of the bottom cell to be 1, the XOR gate in the second level of the circuit is generated using the rule 012 determining the next state of the second cell to be 2 and so on.

An evolutionary algorithm was utilized for searching the initial configuration of the generative cellular automaton and the rules of the local transition function together with the gates to be generated. Since uniform cellular automaton is considered, the local transition function is identical for all the cells. The number of states of the automaton and the number of developmental steps for a circuit to be developed were determined experimentally. The chromosome consists of z state fields representing the initial configuration of the GCA followed by $|Q|^3$ fields of the local transition function of the form (q_{t+1}, F, i_1, i_2) successively for every combination of states in the 3-cell neighborhood. $|Q|$ denotes the number of states, q_{t+1}, F, i_1 and i_2 denote the next cell state, logic function of the gate and indices of its two inputs, respectively. The setting of the evolutionary agorithm was determined experimentally as follows. A simple genetic algorithm was utilized working over a population of 100 chromosomes using the tournament selection operator of base 4. Each chromosome was mutated with the probability

0.98. The crossover operator was not applied since it showed to slow down finding a solution. This is probably due to the complicated structure of the chromosomes containing both the initial state of the cellular automaton and the local transition function.

This developmental method succeeded in the design of many classes of digital circuits. While the best known conventional and evolved 2-bit multipliers possess delay 3 (see [8]), the best solution developed by means of the evolved GCA possesses delay 2 thereby the best solution known so far was improved (see Figure 3). Other circuits developed successfully using this method include 4-bit full adders, 6-bit sorters, 7-bit median networks or 14-bit parity circuits. Properties of these circuits are comparable with the respective best conventional solutions.

Unfortunately, this approach is not scalable since the time needed for the circuit evaluation increases exponentially with its number of inputs. Moreover, circuits with complex internal structure were more difficult to develop by means of this method. Although cellular automata, in general, are able to perform the infinite development, there is a question of suitable circuit representation in order to be able to grow infinitely.

5 Conclusions

In this paper an ongoing research has been presented related to the utilization of developmental mapping in the evolutionary algorithms. Two basic approaches to the development in the area of the evolutionary design have been identified: the infinite development and the finite development. This concept has been defined formally and, moreover, two sample applications for the evolutionary design of digital circuits have been presented.

Since the development has been intended as a possible technique for improving scalability in the evolutionary design, the new classification presented in this paper can outline some of the abilities and limits of this method. These features can be summarized as follows:

1. The applications of both the infinite and finite development have shown the ability to discover novel solutions which have not been known yet and improve some conventional solutions in chosen area (in this case the evolutionary design of digital circuits).
2. The infinite development is able to solve types of problems which are out of the scope of the traditional evolutionary design (without the development). The application of the evolutionary design of arbitrary even-input sorting networks has demonstrated this feature. The evolution has been able to find the solution not only for one instance but for all the instances of the problem. This is unreachable by means of the traditional evolutionary design because of the infinite hierarchy of the sorting networks which is impossible to encode in a finite-length chromosome.
3. In general, the solutions discovered using the infinite development exhibit a high degree of regularity (see Figure 2 and [11]), which is a significant

feature enabling the target object to develop continually and theoretically infinitely according to a *finite* prescription. However, it usually constrains the evolutionary process and prevents it from finding effective solutions.
4. There have not been any real-world results yet demonstrating the ability of the finite development to overcome the problem of scale in a significant way. The existing experiments have shown the quality of the developed solutions comparable with the traditional evolutionary design. However, the evolutionary process utilizing a form of finite development still exhibit a substantial impact of the problem of scale.

Despite the problems and restrictions highlighted in the previous paragraph, the utilized techniques of development have shown a potential to solve various engineering problems. Considering the ability of discovering novel solutions (see item 1) and the solutions which are impossible to find without the infinite development (see item 2), we can state the confirmation of the hypothesis formulated in the Introduction which poses a significant contribution to the theory and practice of this field. In particular, the applications of the evolutionary computation for solving all the instances of a given problem is still a rare case in contemporary science. Therefore, the next research will be focused on the developmental encodings suitable for the infinite development, applications of the developmental techniques in other fields (e.g. computer graphics, iterative computations, mathematics and so on) and investigation of the theoretical aspects of the proposed concept.

Acknowledgment

The research was performed with the support of the Grant Agency of the Czech Republic in the projects under No. 102/04/0737 *Modern Methods of Digital System Synthesis*, No. 102/05/H050 *Integrated Approach to Education of PhD Students in the Area of Parallel and Distributed Systems* and in the FRVŠ project under No. FR825/2006/G1 *Biology-Inspired Hardware Teaching Support*.

References

1. M. Bidlo and R. Bidlo. An evolved general construction method for the sorting networks. Technical report, Department of Computer Systems, Faculty of Information Technology, Brno University of Technology, 2005.
2. Bruce Alberts et al. *Essential Cell Biology, 2nd edition.* Garland Science/Taylor & Francis Group, 2003.
3. P. C. Haddow, G. Tufte, and P. van Remortel. Shrinking the genotype: L-systems for ehw? In *Proc. of the 4th International Conference on Evolvable Systems: From Biology to Hardware, Lecture Notes in Computer Science, vol. 2210*, pages 128–139. Springer-Verlag, 2001.
4. J. R. Koza et al. *Genetic Programming IV: Routine Human-Competitive Machine Intelligence.* Kluwer Academic Publishers, 2003.

5. S. Kumar. Investigating computational models of development for the construction of shape and form, PhD thesis. Technical report, Department of Computer Science, University College London, 2004.
6. A. Lindenmayer. Mathematical models for cellular interaction in development, parts I and II. *Journal of Theoretical Biology*, 18:280–315, 1968.
7. M. Murakawa et al. Evolvable hardware at function level. In *Proc. of the Parallel Problem Solving from Nature IV, PPSN 1996, Lecture Notes in Computer Science, vol. 1141*, pages 62–71, Berlin Heidelberg New York, 1996. Springer-Verlag.
8. J. F. Miller and D. Job. Principles in the evolutionary design of digital circuits – part I. *Genetic Programming and Evolvable Machines*, 1(1):8–35, April 2000.
9. J. F. Miller and P. Thomson. Cartesian genetic programming. In *Proc. of the 3rd European Conference on Genetic Porgramming, Lecture Notes in Computer Science, vol 1802*, pages 121–132, Berlin Heidelberg New York, 2002. Springer.
10. S. Kumar (ed.) and P. J. Bentley (ed.). *On Growth, Form and Computers*. Elsevier Academic Press, 2003.
11. L. Sekanina and M. Bidlo. Evolutionary design of arbitrarily large sorting networks using development. *Genetic Programming and Evolvable Machines*, 6(3):319–347, 2005.
12. J. Torresen. Divide-and-conquer approach to evolvable hardware. In *Proc. of the 2nd International Conference on Evolvable Systems: From Biology to Hardware, Lecture Notes in Computer Science, vol. 147*, pages 57–65, Berlin Heidelberg New York, 1998. Springer-Verlag.
13. J. Torresen. A scalable approach to evolvable hardware. *Genetic Programming and Evolvable Machines*, 3(3):259–282, September 2002.
14. G. Tufte and P. C. Haddow. Towards development on a silicon-based cellular computing machine. *Natural Computing*, 4(4):387–416, 2005.
15. J. von Neumann. *The Theory of Self-Reproducing Automata*. A. W. Burks (ed.), University of Illinois Press, 1966.