Common Developmental Genomes Revisited – Evolution Through Adaptation

Konstantinos Antonakopoulos(✉)

Department of Computer and Information Science,
Norwegian University of Science and Technology, Sem Sælandsvei 7-9,
NO-7491 Trondheim, Norway
kostas@idi.ntnu.no

Abstract. Artificial development has been widely used for designing complex structures and as a means to increase the complexity of an artifact. One central challenge in artificial development is to understand how a mapping process could work on a class of architectures in a more general way by exploiting the most favorable properties from each computational architecture or by combining efficiently more than one computational architectures (i.e., a true multicellular approach). Computational architectures in this context comprise structures with connected computational elements, namely, cellular automata and boolean networks. The ability to develop and co-evolve different computational architectures has previously been investigated using common developmental genomes. In this paper, we extend a previous work that studied their evolvability. Here, we focus on their ability to evolve when the goal changes over evolutionary time (i.e., adaptation), utilizing a more fair fitness assignment scheme. In addition, we try to investigate how common developmental genomes exploit the underlying architecture in order to build the phenotypes. The results show that they are able to find very good solutions with rather simplified solutions than anticipated.

Keywords: Common developmental genomes · Evolvability · Cellular automata · Boolean network · L-systems

1 Introduction

In artificial systems, a species can be linked to a certain computational architecture, such as, a cellular automata (CA) [1] or a boolean network (BN) [2]. Here, computational architectures are considered as structures comprising connected computational elements. A computational element may represent a cell (part of a cellular automaton) or a node (part of a boolean network). Most such systems include a specific genetic representation (genotype), a mapping process (genotype-to-phenotype) and have a specific structure as a target (phenotype).

A big challenge in developmental systems is how a genotype-phenotype mapping can work on a class of computational architectures (species), towards scalable systems for complex computation. So, it is important to investigate whether

A.I. Esparcia-Alcázar et al. (Eds.): EvoApplications 2014, LNCS 8602, pp. 101–112, 2014.
DOI: 10.1007/978-3-662-45523-4_9

it is possible to exploit the most favorable properties from each species or to combine more than one species in a more efficient way (i.e., a true multicellular approach). To study this concept, an experimental approach was undertaken [3] and [4], giving rise to common developmental genomes.

Common developmental genomes are genomes constructed in a modular way (chromosomes), making it possible to develop and evolve more than one species, towards a *common* goal [3],[5]. In [3], it was investigated whether common developmental genomes can favor the evolvability of different species. The species studied therein were cellular automata and boolean networks. Evaluation of the fitness was done by averaging the partial fitnesses of the species involved. Even though common genomes exhibited superior ability to evolve and adapt to the environment than genomes evolved separately for each species, the fitness evaluation scheme in [3] needs some reconsideration. For example, a CA with a fitness 0.1 and a BN with a fitness 0.9, would have an average fitness of 0.5. On a different case, with the CA having a fitness 0.5 and the BN having a fitness 0.5, we will also get an average fitness 0.5. As such, there is no way to discriminate better from worse individuals in a population. Even still, they are all assigned the same fitness score.

In this paper, we continue the study of [3]. The goal herein is to test the ability of common developmental genomes to adapt when the goal changes over evolutionary time (i.e., adaptation), facilitating a more fair fitness assignment scheme. Through this new fitness evaluation scheme we aim at assigning a more fair fitness to the evolving species but also, and perhaps more importantly, since the genetic information (genotype) is common for all species, the scheme may act as a means to indirectly apply evolutionary pressure towards the inferiorly evolving species. In addition, we analyze the structures of the best phenotypes by visual inspection and investigate how common developmental genomes exploit the underlying architectures in order to build their solutions.

The rest of the article is laid out as follows. The developmental model is given at Section 2. Section 3 give a brief description of the emergent dynamics in artificial systems. Section 4 present the experimental setup. Results are given in Section 5, with the conclusion at Section 6.

2 The Developmental Model

In this section, the genetic representation and the developmental model is given in brief. For a detailed description, see [5]. Figure 1, shows the genome constructed by two parts or *chromosomes*. The first chromosome creates the cells / nodes of the species whereas the second chromosome generates the connections. Each chromosome is governed by rules. The rules for node / cell creation are different from those for connectivity.

The rules of the first chromosome describe cell processes like growth, differentiation and apoptosis and are used during the development process (ontogeny). The rules of the second chromosome express the connectivity and are used for developing the connections of the boolean network. To express the rules in the chromosomes, an L-system is used as a developmental model.

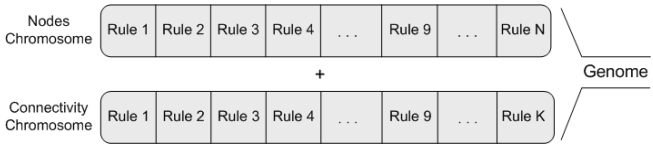


Fig. 1. The genome is split into two chromosomes: Node- and Connectivity-chromosomes

L-systems are rewriting grammars, able to describe developmental or generative systems and have successfully been used to simulate biological processes [7], [8]. Two separate L-systems are used in the representation. The first L-system processes the first chromosome rules where a second L-system deals with the connectivity rules of the second chromosome.

2.1 The L-system for the First Chromosome

The L-system used here is context-sensitive. As such, development is using the strict predecessor/ancestor to determine the applicable production rule. The rules are able to incorporate all the cell processes of a species. Table 1(a), shows the type of symbols used by the L-system of the first chromosome.

Table 1. (a) Symbol table for nodes/cell creation, (b) Symbol table for creating connectivity

(a)

Symbol	Description
a	Add (growth)
b	Add (growth)
c	Add (growth)
d	Delete (apoptosis)
X	Substitute (differentiation)
Y	Substitute (differentiation)
→	Production

(b)

Symbol	Description
x	Node (different from y)
y	Node (different from x)
+	Connect forward
–	Connect backwards
→	Production

Symbol a is the *axiom*. Apart from the symbols a, b, and c, which perform *growth* of the phenotype, symbol d performs *apoptosis*, aiming at the deletion of the current rule (i.e., cell/node). Symbols `X` and `Y`, represent the *differentiation* process, replacing the predecessor cell/node. For example, for `X→Y` the outcome will be `Y`. The length of each rule is 4 symbols (i.e., `4x8bits=32bits`). For node/cell generation the L-system runs for n timesteps and then stops. As such, the intermediate phenotypes generated by development are of variable size. Figure 2a, gives an example of a first chromosome L-system.

Detailed example with step-by-step development of a 2D-CA architecture can be found at [5].

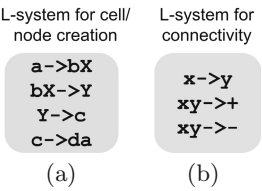


Fig. 2. (a) L-system rule set for node/cell generation, (b) L-system rule set for connectivity

2.2 The L-system for the Second Chromosome

The rules are able to generate the connections necessary for the wiring of the nodes. They contain symbols which when executed by the L-system, result in creating a connection forward or backwards from the current node. Each node in the network has unique numbering; current node holds number zero and any nodes starting from the current node forward have positive numbering. Nodes existing from the current node backwards, have negative numbering. As such, there is a need to differentiate between the current and the next node, using different symbols but also to describe when a connection will be created forward or backwards from the current node.

The length of connectivity rules is also four. The L-system uses a D0L (i.e., with zero-sided interactions). The second chromosome L-system is shown at Figure 2b. Symbols are explained in Table 1(b).

The *axiom* rule for the second chromosome is `x`→`y`. Then, development continues looking for rules of type `xy`→`+value`, or `xy`→`-value`. In short, these two rules imply that if two different (distinct) nodes are found (`x`≠`y`), it creates a connection forward (if the rule includes a '`+`'), or similarly a connection backwards (if the rule includes a '`-`'). The field `value` is encoded in the genotype and denotes the node number of the newly created connection. If `value=0`, a self-connection is created to the current node. Detailed example with step-by-step development of a boolean network architecture is presented at [5].

2.3 The Genetic Algorithm for Common Genetic Representation

A genetic algorithm (GA) is utilized to create and evolve the chromosome rules. Since there are two separate L-systems involved in development, the evolutionary process will be consisted of two phases: a. the creation of nodes and b. the creation of the connections. Mutation and single-point crossover are used as genetic operators. Mutation may occur anywhere inside the 4-symbol rule, such as the production symbol (→) remains undistorted after mutation. Single-point crossover between two parents always takes place at the position of the production symbol in the rule. The evolutionary cycle ends after a predetermined number of generations.

3 Emergent Dynamics in Artificial Systems

In biology, development is a process starting from a zygote and develops into a multicellular organism. Similarly, in the artificial domain, development simulates this biological process; from an given initial condition, the zygote, through an iterative developmental process, it can develop into a final structure (phenotype). Assuming the developmental process is deterministic, i.e,. the outcome of development is defined by the initial zygote (genome), some initial condition and a developmental mapping, then an initial configuration (or a set of configurations) exists and is sufficiently defined by the developmental genome and the initial conditions [6].

Any sparsely connected computational architecture (i.e., CA, BN, etc.) can be represented in the space time domain. Phenotypic structures can be shown as nodes and their transitions in time can be shown as developmental paths from the zygote to the final organism. Development of a structure comprise developmental steps (DS). Each DS may include one or more developmental processes proposed by the model (Section 2). Development starts with the zygote (initial genome).

Fig. 3. Developmental path of a structure shown as a trajectory

Figure 3 shows the path of development of a non-uniform 2D-CA. White cells are considered empty whereas colored cells represent the CA rule of the particular cell. Solid lines represent consecutive developmental steps (DS 10-11 and DS 99-100). Non-consecutive developmental steps are represented by dashed lines (zygote-DS 10 and DS 11-99). The path from the zygote until DS 10 has gone through 10 different intermediate phenotypic structures. Similarly, the path from DS 11 until DS 99 has produced 88 different intermediate phenotypic structures. DS 100 has a loop back to DS 99; this type of behavior is a *cycle attractor* which indicates whether the structure is stable or not. The path until DS 99 represents a *transient period* or phase. The structure at DS 100 is the final phenotype.

The behavior of the system is described by the initial state and the trajectory of all 100 developmental steps of the example. Each developmental step is further analyzed into state steps (SS). A state includes cell/node information giving a snapshot of instantaneous behavior. As such, state steps provide information about the emergent behavior of intermediate and final phenotypes in the space/time domain.

The descriptions on emergent dynamics explained above, are useful to better understand the definitions of the computational goals for the common developmental genomes (Sections 4.3 and 4.4).

4 Experimental Setup

For the experiments, a `6x6` 2D-CA and a `N=36` BN is used. The size chosen for the CA is the minimum possible. By choosing a smaller lattice size, there will be too many dependencies in the cell states of the CA. Also, the maximum number of nodes/cells in the species should allow for easy, visual explanation of the final phenotypic structures. The larger the size of the species, the harder it is to visually interpret their structure.

For the two species to be comparable, they must have the same state space or the same amount of possible states. Since the size of each architecture is 36 and each cell/node can take 2 different distinct values (boolean), the total state space for each species is 2^{36}. The number of outgoing connections per node is $K = 5$. When the number of outgoing connections exceeds five, a self-connection to the originating node is created instead. The number of incoming connections per node is limited only by the total number of nodes found in the network $(N - 1)$.

For each individual, a random initial state is created and fed into the architecture. We use a total number of 36 rules for node generation and connectivity (i.e., `32x36=1152bits`). Each rule can be reused during L-system development. The GA program drives a single population of 20 individuals. Development runs for 100 timesteps (DS) for each individual. In each DS, behavior is defined by 1000 state steps (SS). *Generational mixing* is used as global selection mechanism and *fitness proportionate* for parental selection. Mutation rate is set to .0009 and crossover rate to .001. We run a total of 20 experiments of 10000 generations each. Evaluation of phenotypes is given by the cell types and functionality of Table 2.

Table 2. Cell types and functionality

Cell Type	Function name
a	NAND
b	OR
c	AND
d	IDENTITY CELL
X	XOR
Y	NOT

4.1 Fitness Assignment Scheme

The new fitness evaluation scheme used is described in four steps:

- Run the first 20% of evolutionary time using normal fitness evaluation (final fitness is the average of the fitnesses of CA and BN), e.g., $fitness_{total} = (fitness_{CA} + fitness_{BN})/2$
- In the next 20% – 50% of time and if the partial fitnesses differ more than 30%, there is an extra 10% of fitness credit assigned to the species with the higher fitness. For example if CA has a 30% higher fitness than BN, then $fitness_{total} = [(fitness_{CA} + (fitness_{CA} * 0.1)) + fitness_{BN}]/2$

- In the next 50% – 70% of time, species are evolved using normal fitness evaluation, e.g., $fitness_{total} = (fitness_{CA} + fitness_{BN})/2$
- In the final 70% – 100% of time and if the partial fitnesses differ more than 30%, there is an extra 10% of fitness credit assigned to the species with the higher fitness. For example, if BN has a 30% higher fitness than CA, then $fitness_{total} = [(fitness_{BN} + (fitness_{BN} * 0.1)) + fitness_{CA}]/2$

The highest assigned fitness score is 100 and the lowest is 2 with a worst-case of 0.1. The final fitness for the common developmental genome is the average of the fitness of the species involved. If, for example CA's fitness is 50 and BN's fitness is 20, the final fitness of the common developmental genome will be 35.

4.2 Studying the Dynamic Behavior

To study the evolvability of computational properties, the system must be able to target different behavior on the architectures chosen (CA and BN). Their behavior can be evolved through the study of various dynamic problems i.e., stable point attractor, short attractors or long repetitive/chaotic behavior.

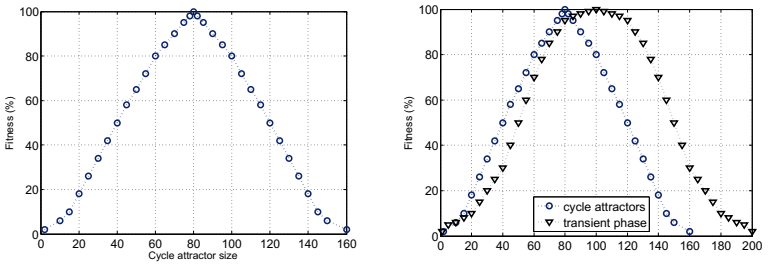
The computational problems chosen here describe some basic dynamic behavior for CA and BN and the goal is generally expected to be reached. Though, the problems as such are of minor importance since we are mainly after the ability of common developmental genomes to adapt during evolution.

4.3 First Problem Definition

Evolution searches for a cycle attractor of size 2-160, at generations 1 - 5000. A minimal cycle attractor can be found as early as in SS 2, that is, behavior is stabilized and the final structures are phenotypes obtained at SS 1 and SS 2. On the other extreme, a maximally big cycle attractor may be found as late as in SS 1000-160=840. Best fitness score is assigned for cycle attractors of size 80. Here, no fitness credit is assigned for cycle attractors found at an earlier or later stage i.e., a cycle attractor can occur after any transient phase. Fitnesss distribution is given at Figure 4(a).

4.4 Second Problem Definition

After generation 5000, the evolutionary goal change. From generation 5000 - 10000, evolution searches for a transient phase of size 1-200, followed by a cycle attractor of 2-160 steps. Best fitness score is assigned for transient phase 100 and cycle attractor 80. This is a harder problem than the previous one, considering that the total number of states / developmental step is 1000. No credit is given for point attractors following a transient phase. Here, separate fitnesses are assigned for the transient phase and the cycle attractor. The final fitness is estimated by averaging their respective fitnesses, e.g., for the CA will be $fitness_{CA} = (fitness_{transient} + fitness_{cycleattractor})/2$. The fitness distribution for this problem is shown at Figure 4(b).



(a) (b)

Fig. 4. Fitness distributions plots: (a) Cycle attractors, (b) Transient phase & cycle attractor

5 Results

Figure 5 shows the average fitness evaluation of common developmental genomes over all runs. The 'AVG' line shows the average fitness of both species (CA and BN). The 'CA' line shows the average fitness of the cellular automata only and the 'BN' line gives the average fitness of the boolean network.

The first problem (search for cycle attractor) is studied at generations 1-5000. During this period, both CA and BN are able to find fairly good solutions. After generation 2000, the effect of the new fitness assignment scheme can be observed. BN is constantly being credited with an extra 10% of fitness due to its fitness difference to the CA. This credit assignment in one of the species in common developmental genomes, can indirectly act as a means of evolutionary pressure for the other species, since they share the same genetic information. Though, the performance of the CA remains constant until the very end. It is not until generation 4600, where an improvement in performance for both species occurs.

The second problem (search for transient period & cycle attractor) is examined at generations 5001-10000. At generation 5001, the genome still contains genetic information optimized for the previous problem (generation 5000). So, the same genetic information acts as a basis for the second problem, which initially gives only average solutions. After generation 7000 the new assignment scheme gets into effect. This is evident from a sharp fitness increase for both species. Here, the performance of BN has an impact in the performance of the CA (generation 7350).

Figure 6 shows some evolutionary steps of one of the best CA runs over time. Solid line shows consecutive generations where dashed lines delineate more than one generation steps. The figure, shows some of the best evolved phenotypes for the first problem (gen.2-5000). From generation 5001, the target changes and the genome tries to adapt to the newly set goal, with a clear impact in the fitness. Some of the phenotypes for the second problem are shown for generations 5001, 8500 and 10000.

The model managed to find several perfect solutions for the first problem, but also, many good solutions for the second problem. The solutions achieved by

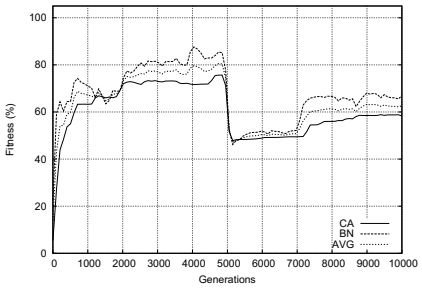


Fig. 5. Fitness evaluation of common developmental genomes (averaged)

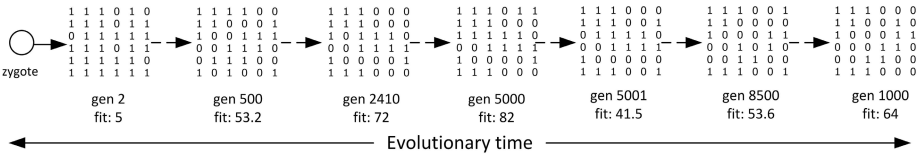


Fig. 6. Some of the best intermediate and final phenotypes of a CA evolution over time

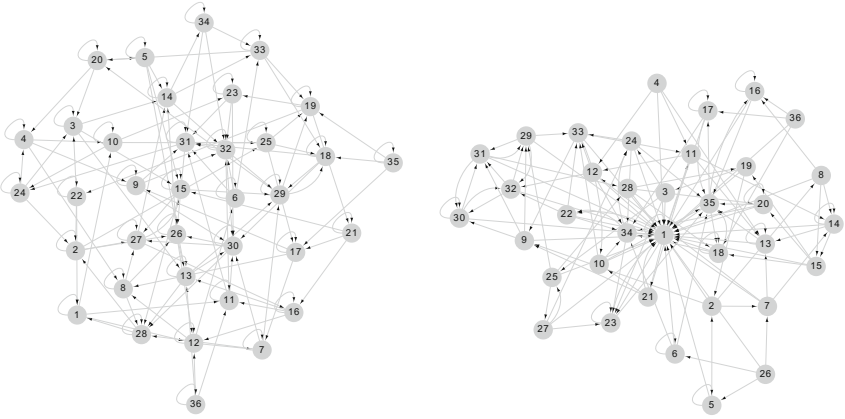
the developmental model with the CA, extended out exploiting the complete CA lattice for both the problems investigated. In addition, development produced maximally big genomes at the very beginning of the process (not shown). As we will see in the next paragraph, this is not the case for the evolved BN phenotypes.

Figure 7 shows two of the best evolved BN solutions for the first problem at generation 5000. Both solutions solved this problem perfectly (fitness 100), but with a quite different structure. The solution at Figure 7(a), shows a network where each node has at least two connections to other nodes and at least one self-connection.

The numbers at the nodes indicate the node number and the connections are shown in black solid lines. Since there is no explicit positional information for the nodes of the BN, the node numbers indicate their sequential position (next, previous node). The arrow at the end of each connection, indicates the flow of information between the originating and destination nodes.

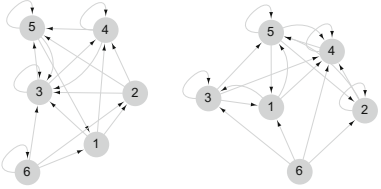
On the other hand, the solution at Figure 7(b), shows a network where one node is rather influential (node nr.1), since the outcome of the majority of the nodes in the network, is dependent on the outcome of node nr.1. Self-connections are rare since most of the connections point to a different node than the originating node.

Some of the near-perfect solutions given by evolution (fitness > 80), include networks with a rather small number of nodes (not shown). All perfect solutions (fitness 100), involved networks having the maximum number of nodes allowed by the model (N=36). This suggests that development initially tries to seek solutions



(a) (b)

Fig. 7. Two of the best evolved boolean networks for the first problem (generation 5000, fitness 100)



(a) (b)

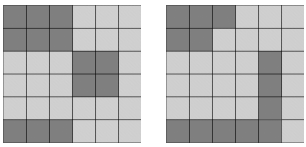
Fig. 8. The best evolved BNs for the second problem (generation 10000, fitness 76)

using less number of nodes and then extends the networks by introducing more nodes in the network. This shows an unexpected emergent behavior of the system since the developmental model was not designed as such.

Figure 8 shows the best evolved BN solutions for the second problem at generation 10000. Both solutions have a rather small number of nodes (N=6) and most of the nodes have at least one self-connection. Other, less than perfect solutions provided networks having the max number of nodes (N=36).

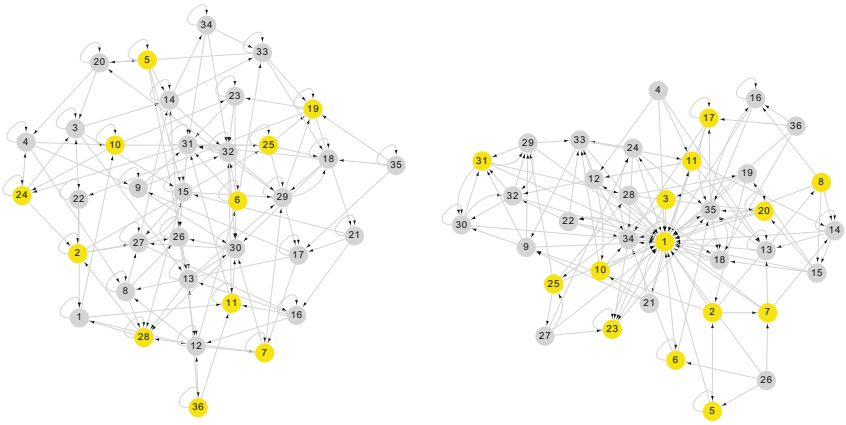
At generation 5001, the goal changes and evolution finds near perfect solutions with networks of similar size as before. At the end of evolution, the solutions included networks with a rather simplified structure. The latter shows that the developmental model is able to give both complex and more simplified solutions, depending on the goal sought.

Next, we investigate how common developmental genomes exploit the underlying architectures, in order to build the final solutions. To achieve this, we focus on the variation of the nodes/cells during evolution. Here, we are interested only in the change of the value of the cell/node, not if the change has a



(a) (b)

Fig. 9. Amount of CA structures that is computing (light gray) versus their static parts (dark gray). (a) First problem, (b) Second problem.



(a) (b)

Fig. 10. Computing parts of BN phenotypes for the two of the best evolved networks for the first problem

positive (i.e., fitness increase), or a neutral (i.e., equal fitness) impact to the fitness. Cells/nodes performing rarely any computation ($\prec$30% of the evolutionary time) are considered static, where cells/nodes computing more than 30% of the time is considered that they are actively contribute to the final solution.

Figure 9 shows two 2D-CA of size 6x6. The light-gray colored cells indicate cells that compute. As such, a total of 70% approximately of the CA structure is actually computing during evolution. Similarly, the dark-gray colored cells indicate cells that are static, constituting a total of 30% of the structure.

Next, Figure 10 shows the two best evolved networks for the first problem (as in Figure 7). The nodes of the networks that are computing are shown in dark gray color. Figure 10(a) indicates that approximately 55.6% of the network is computing with the rest 44.4% of the network being static. Similarly, Figure 10(b), shows that a total of approximately 70% of the network is actually active. The BN solutions found, give quite different statistics; the first network solution involve more self-connections/node than the network solutions for the second

problem. Self-connections contribute to the network's neutrality and this can partially have an impact on the amount of the network that is actually active. Regarding the second problem (network solutions of Figure 8), all the nodes in the networks found to be computing and no static nodes are observed.

6 Conclusion

In this work, we extended a previous study by looking at how common developmental genomes can evolve computational architectures when the goal changes over time (evolution through adaptation). The focus here was to evolve CA and BN computational architectures with simple cycle attractor with transient phase problems as a computational goal and a more fair fitness assignment scheme. Also, it was investigated how common genetic representation is being exploited during development, sometimes exhibiting emergent behavior during phenotype construction. Common developmental genomes where able to adapt fairly well to each problem, considering the number of available state steps during development. In addition, they were able to exploit a large part of the underlying architectures having on average more than 55% of the total number of cells/nodes actively computing, for both problems studied.

References

1. Bidlo, M., Vasicek, M.: Evolution of cellular automata with conditionally matching rules. In: Congress on Evolutionary Computation (CEC 2013), pp. 1178–1185 (2013)
2. Bull, L.: Artificial symbiogenesis and differing reproduction rates. Artificial Life **16**(1), 65–72 (2010)
3. Antonakopoulos, K., Tufte, G.: On the Evolvability of Different Computational Architectures using a Common Developmental Genome. In: Rosa, A., Dourado, A., Madani, K., Filipe, J., Kacprzyk J. (eds.) IJCCI 2012, pp. 122–129. SciTePress Publishing (2012)
4. Antonakopoulos, K., Tufte, G.: Is Common Developmental Genome a Panacea Towards More Complex Problems? In: 13th IEEE International Symposium on Computational Intelligence and Informatics (CINTI 2012), pp. 55–61 (2012)
5. Antonakopoulos, K., Tufte, G.: A Common Genetic Representation Capable of Developing Distinct Computational Architectures. In: IEEE Congress on Evolutionary Computation (CEC 2011), pp. 1264–1271 (2011)
6. Tufte, G.: The discrete dynamics of developmental systems. In: IEEE Congress on Evolutionary Computation (CEC 2009), pp. 2209–2216 (2009)
7. Lindenmayer, A.: Developmental Systems without Cellular Interactions, their Languages and Grammars. Journal of Theoretical Biology **30**(3), 455–484 (1971)
8. Lindenmayer, A., Prusinkiewicz, P.: Developmental Models of Multicellular Organisms: A Computer Graphics Perspective. In: Langton, C.G. (ed.) Proceedings of ALife, pp. 221–249. Addison-Wesley Publishing (1989)