Genotype Regulation by Self-modifying Instruction-Based Development on Cellular Automata

Stefano Nichele(✉), Tom Eivind Glover, and Gunnar Tufte

Norwegian University of Science and Technology, Trondheim, Norway
nichele@idi.ntnu.no

Abstract. A novel method for regulation of gene expression for artificial cellular systems is introduced. It is based on an instructon-based representation which allows self-modification of genotype programs, as to be able to control the expression of different genes at different stages of development, e.g., environmental adaptation. Coding and non-coding genome analogies can be drawn in our cellular system, where coding genes are in the form of developmental actions while non-coding genes are represented as modifying instructions that can change other genes. This technique was tested successfully on the morphogenesis of cellular structures from a seed, self-replication of structures, growth and replication combined, as well as reuse of an evolved genotype for development or replication of different structures than initially targeted by evolution.

1 Introduction

In biological systems, the process that produces phenotypes from genotypes, i.e., genotype-to-phenotype mapping, is a complex and intricate combination of interactions between the genotype and the environment. As a result, intermediate phenotypic stages emerge, which themselves influence the decoding/regulation of the genotype for the next phenotypic stage. It may be argued that genotypes possess the ability to self-modify [8]. As such, the development process is influenced not only by the instructions encoded in the genome, but also by the mutual interaction between genotype and phenotype, and with the environment. Biological genomes possess an intrinsic ability to evolve and adapt to novel environments, i.e., evolvability [2]. Modularity [5] is a key factor that contributes to evolvability. In fact, many biological networks are modular [1], e.g., brain networks, gene regulatory networks, metabolic networks. It turns out that it is easier to rewire a modular network with independent substructures than an unstructured network [9]. Kovitz [12] introduced the concept of "cascading design", a form of genotype coordination that allows to preserve the relationship between separate traits. He describes such coordination as *"a good house design, where the plumbing and electrical wiring connect water and electricity to the devices that need them. If you change the architectural plans for a house to move the bathroom from the northwest corner to the middle of the east wall, moving*

J. Handl et al. (Eds.): PPSN XIV 2016, LNCS 9921, pp. 14–25, 2016.
DOI: 10.1007/978-3-319-45823-6_2

Table 1. IBD instruction set. L, C, R, U, D represent Left, Center, Right, Up, and Down neighbors. n represents the number of CA cell states

Instr.	Operation	Description	Instr.	Operation	Description
AND	$N[i_1] = N[i_1] \wedge N[i_2]$	Bitwise AND	INC	$N[i_1] = N[i_1] + 1$	Increment
OR	$N[i_1] = N[i_1] \vee N[i_2]$	Bitwise OR	DEC	$N[i_1] = N[i_1] - 1$	Decrement
XOR	$N[i_1] = N[i_1] \oplus N[i_2]$	Bitwise XOR	SWAP	$N[i_1] \Leftrightarrow N[i_2]$	Swapping
NOT	$N[i_1] = \neg N[i_1]$	Bitwise NOT	ROR	$LCR \Rightarrow RLC$	Rotate right
INV	$N[i_1] = n - N[i_1]$	Inverse	ROL	$LCR \Rightarrow CRL$	Rotate left
MIN	$N[i_1] = min(N[i_1], N[i_2])$	Minimum	ROU	$UCD \Rightarrow CDU$	Rotate up
MAX	$N[i_1] = max(N[i_1], N[i_2])$	Maximum	ROD	$UCD \Rightarrow DUC$	Rotate down
SET	$N[i_1] = N[i_2]$	Replace	NOOP		No Operation

the walls and fixtures is not enough. You must also reroute all the pipes and electrical connections". If an evolutionary algorithm was to design such a house plan, *"then moving the bathroom across the house requires that many mutations occur simultaneously: one mutation for each segment of pipe that needs to move, one mutation for each doorway, etc. As coordination becomes more complex, the probability of making all the needed mutations simultaneously gets lower and lower"*. Biological evolution seems to possess some kind on intrinsic coordination. Lee Altenberg describes genes that affect the probability of mutation of other genes and subroutines in genetic programming [2].

In this paper, cellular automata (CA) are used as a test-bed model of development. Traditional CA transition tables are replaced by an algorithmic representation, i.e. program, which includes instructions that can modify the code itself. Such representation may allow the emergence of genotype coordination mechanisms that may encode different sub-processes. As such, different parts of the genome may be active at different stages of phenotypic development. The problems targeted include the morphogenesis of structures from a zygote, the replication of given shapes, both development and replication achieved by the same genotype, and the reuse of evolved genotypes for development and replication of different structures than those initially targeted by evolution.

2 Background

In nature, phenotypes are not determined only by their genotypes. Genes "behave" in relation to each other and in relation to the environment. Genes may even suppress other genes, i.e., methylation [18]. In fact, not all genes are active at all times and methylation is one of the factors that control gene expression. In the context of artificial evolutionary and developmental (evo-devo) systems, different models of gene regulation mechanism exist. Some models aim to be true to biology [15] whether other are more abstract models of development [10,11,13]. In this paper, the used model can be placed within abstract computational models of development based on CA [27]. Therefore, we do not

aim at a truly biological model of development. The used two-dimensional CA evo-devo model includes the interplay between genotype, developing phenotype, and evolutionary developmental effects. Traditionally, CA rules are in the form of transition tables. One of the implications of such representation is that rules grow exponentially with the increase in cells neighborhood and number of cell states. As such, transition tables do not scale well.

2.1 Instruction Based Development

The idea of evolving instruction-based representations is a rather old approach [14]. Cartesian Genetic Programming (CGP) has been introduced by Miller and Thompson [17] for the evolution of programs as a directed graph. Sipper [22] proposed the evolution of non-uniform cellular automata with cellular programming. Bidlo and Skarvada [3] introduced the Instruction-Based Development (IBD) for the evolutionary design of digital circuits. Bidlo and Vasicek [4] exploited IBD for the development and replication of cellular automata structures. Even though their approach has been shown to improve the overall success rate, the number of available instructions was experimentally chosen. IBD allows representing CA transition functions by means of a sequence of instructions executed on local neighborhoods in parallel and deterministically update the state of each cell. Evolutionary growth of genomes has been used to evolve local transition functions starting from a single neighborhood configuration [19]. Using an instruction-based approach removes the problem of specifying all combinations in the transition table. The initial ruleset of IBD is presented in Table 1.

2.2 Scalability and Modularity

Biological organisms are the best example of scalability, ranging from simple unicellular organisms to multicellular organisms, where trillions of cells develop from a single cell holding the genome. The genomes of different species have variable lengths, as result of biological complexification mechanisms [16] through gene duplications [25] and continuous elaborations. In artificial evo-devo systems, complexification mechanisms have been used to allow variable length genomes. Stanley and Miikkulainen [23,24] introduced NeuroEvolution of Augmenting Topologies (NEAT), a method for the incremental evolution of neural networks. Nichele and Tufte [19] presented a framework for the evolutionary growth of genomes using indirect encodings. Trefzer et al. [26] investigated the advantages of variable length gene regulatory networks in artificial delelopmental systems.

One of the characteristics that allow variable length genomes to evolve and adapt to new environments is their modularity. Clune et al. [5] showed that a key driver for evolvability is the modularity of biological networks. Modular artificial evo-devo systems have been shown to have increased evolvability. Kovitz [12] has investigated the evolution of coordination mechanisms using a modular cascading design based on graph representation inspired by CGP [17]. Harding et al. introduced Self-modifying Cartesian Genetic Programming (SMCGP) [7], a form of CGP where genotype programs are allowed to modify themselves.

Taking inspiration from IBD and SMCGP, we propose a cellular automata developmental system based on self-modifying cellular programs.

2.3 Self-modifying Instrucion Based Developement

We propose Self-modifying Instruction Based Development (SMIBD) for cellular automata, where the instruction set in Table 1 is extended with the instructions in Table 2. As such, genomes have the ability to perform different functions during different developmental stages. Each cell executes the same program in parallel on local neighborhoods in order to update their state. Since the program can change, genomes have the ability to duplicate or even to destroy themselves.

3 Experimental Setup

A genetic algorithm (GA) is designed to evolve solutions to 4 different problems:

✓ Development of given structure from a seed;
✓ Replication of given structures (minimum 3 replicas);
✓ Development of given structure from a seed followed by replication of developed structure (minimum 3 replicas);
✓ Re-evolution of solutions found for a given structure for the development and replication of a different structure.

The used morphologies are shown in Fig. 1, where different colors identify different cellular states. Structures of different sizes, complexities and number

Table 2. Instructions added to IBA instruction-set in order to allow Self Modification (SM).

Instruction	Parameters	Description
SKIP	$N[i_1] = Nskips$	Skip next $N[i_1]$ instructions. Not a SM instruction
MOVE	$N[i_3] = Start$ $N[i_4] = Insert$	Move instruction at line $N[i_3]$ just before $N[i_4]$
DUPE	$N[i_3] = Start$ $N[i_4] = Insert$	Copy instruction at line $N[i_3]$ just before $N[i_4]$
DEL	$N[i_3] = Start$	Delete instruction at line $N[i_3]$
CHF	$N[i_3] = Start$ $N[i_4] = Instr$	Change instruction at $N[i_3]$ to instruction at $N[i_4]$
CHP	$N[i_1] = Param$ $N[i_3] = Start$ $N[i_{2\|4}] = Value$	Change $N[i_1]$ parameter at $N[i_3]$ with value in $N[i_2]$ or $N[i_4]$ depending on $N[i_1]$

Fig. 1. Left to right; 0: simple square (2 states), 1: four squares (2 states), 2: three stripes flag (3 states), 3: generic four colors flag (4 states), 4: small Norwegian flag (3 states), 5: big Norwegian flag (4 states), 6: creeper (3 states). Note that **two versions** of each structure are used: **(a) structure with the necessary number states**, **(b) structure with an additional support state that is not required in the final structure**, but evolution is free to explore it. **The different structures are referred in the text as structure 1a, structure 1b, etc.** (Color figure online)

of states are used. The four problems are tested with two different CA genotype representations: traditional CA transition tables and CA using SMIBD. All the experiments are performed on 2-dimensional CA with von Neumann neighborhood (5 neighbors) with cyclic boundary conditions. The GA used a population of size 50, single point mutation with 2 % probability per genotype symbol, multi-point crossover with 10 % probability per genotype symbol, and fitness proportionate selection. Genotypes in the form of transition tables have a size of N^K, where N is the number of cell states and K is the neighborhood size (5 with von Neumann neighborhood). In case of SMIBD, the genotypes are composed by 10 instructions in the form $rule, op1, op2, op3, op4$, where $rule$ identifies the rule number, $op1$ and $op2$ represent two neighbors, $op3$ and $op4$ are in the range *[0, number of instructions in program]*. CA development is executed for 40 steps. The fitness function for the development problem and for the replication problem is searching for matching structures, one in the case of development and three in the case of replication. For more information on similar fitness functions see [4,21]. For the developement and replication problem, the individual fitness functions are used, the one for development in the first 20 development steps and the one for replication in the 20 steps following the best developmental stage. Note that whether for the development problem the lattice size is the same as the wanted structure size, for the development and replication the lattice size is bigger as to guarantee enough space for the wanted replicas to emerge. As such, the initial development is considered to be a much harder problem because structures cannot rely on border conditions. For the re-evolution problem, the fitness function is modified by the different target structure. The same population is used as for the evolution of the first wanted structure.

4 Results and Analysis

4.1 Development

Results obtained for the morphogenesis of structures are summarized in Table 3. In particular, it is possible to notice that for structures 2a and 3a, the genomes that allow self-modification produce higher success rate in fewer generations on

Table 3. General results on the development problem using TT and SMIBD. Avg. over 100 runs.

	Transition table				Self modifying IBD			
Problem	2a	3a	4a	4b	2a	3a	4a	4b
Success rate %	86	34	0	0	100	96	5	5
Average (numGen)	13139	30676	x	x	1912	13309	57224	47042
StDev. (numGen)	18835	27660	x	x	9416	22687	42635	12950

average. For structures 4a and 4b, working solutions are found which were not achievable by transition table genomes. A general observation that emerged while inspecting the evolved solutions is that transition table representation tends to develop structures that quickly degenerate after few development steps and never recover. Self-modifying genomes often produce more stable solutions that retain the final structure, i.e., point attractors, or cycle a few steps before reaching the wanted structure again, i.e., short cyclic attractors. Another observation (see [6] for detailed results, not included here due to space limitations) is that a genotype representation that allows the program to modify itself, may allow a degree of control in developmental speed. As such, it may be able to evolve solutions that grow in different developmental times. This is of particular interest if one aims at developing given structures at specific points of the developmental time.

4.2 Replication

Table 4 presents the results for the replication problem on the tested structures. It is clearly visible that the proposed method outperforms the traditional CA transition table, both in terms of success rate and average number of generations needed to evolve a solution.

Table 4. Results on the replication problem using TT and SMIBD. Avg. over 100 runs.

	Transition table							Self modifying IBD						
Pattern	1a	2a	3a	4b	5a	6a	6b	1a	2a	3a	4b	5a	6a	6b
Success %	62	5	0	0	0	0	0	100	100	100	100	100	22	100
Avg. Gen	2116	4909	x	x	x	x	x	38	279	54	37	94	4737	54
StDev. Gen	2533	2009	x	x	x	x	x	24	344	53	25	72	2745	42

Continued Replication. It was observed that self-modifying genomes allowed solution to continue replication after the wanted number of replicas was achieved, whether transition tables often degenerated their behavior into a randomized pattern. As side experiment, solutions obtained with self-modification were re-developed in a bigger lattice of size 75×75 cells for a longer developmental time of 120 steps. In some cases, not only the replication process continued, but

it produced a "massive" replication effect. Two examples are shown in Fig. 2, for structures 3a and 1a, respectively. Note: the original GA fitness required a perfect solution to produce at least 3 replicas (no additional award beyond 3).

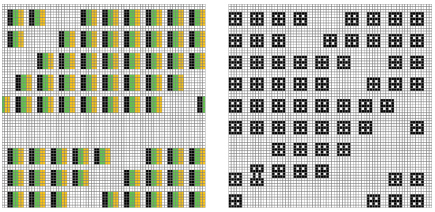

Fig. 2. Two examples of mass replications.

General Replicators. One of the aspects that we investigated was the "universality" of the evolved replicators, or in other words, whether the obtained solutions were able to generalize to other structures. As such, the evolved solutions obtained with self-modifying genomes were reused, i.e., once the solution was found for the replication of a structure, the structure in the lattice was replaced and the CA executed again. Quite surprisingly, in many cases the evolved programs could replicate any of the structures. Examples are shown in Figs. 3(a), (b), (c), and (d). In Fig. 3(a), a solution for structure 3a (4 states) is used on structure 2a (3 states). In this case, the supposedly unused state (yellow) is actually used as a support state. The two available support states are not used in Fig. 3(b), where structure 1a (2 states) is replicated with a solution for structure 3a (4 states). Figure 3(c) replicates structure 4a (4 states) using a solution for 3a (4 states), where the number of necessary states is the same. In Fig. 3(d), the spatial topology is preserved, whether the actual states (colors) are incorrect. Finally, Fig. 3(e) shows the replication of the Norwegian flag (4 states) using a solution for the French flag (4 states). Note that both solutions use a support state as in the final structures only 3 states are actually required. A transition table representation, on the other hand, would require an exponential scaling in the table size to encode for the neighborhood configurations resulting from the additional state. Also, solution evolved for a specific number of states are not practically usable with a different number of states, i.e., scale up or down. Self-modifying programs scale well in this regard.

4.3 Development and Replication

In this section, the described experiments target the development of a given structure first, followed by the replication of the grown structures. As such, the same genotype must encode both processes. This is a very interesting property that might be present in systems that target the development of self-replicating machines. In Table 5 numerical results are given for the tested problems, with a comparison of genotypes using transition tables and self-modification. For a simple structure as 0a (3×3 cells and 2 states), the results are fairly similar. More complex structures, as 2a and 3a did not produce any valid solution using transition tables. Self-modification allowed to produce some working solutions, able to both develop and replicate further. Solutions found by the two methods are inspected in Figs. 4 and 5. It is possible to notice that even if both examples

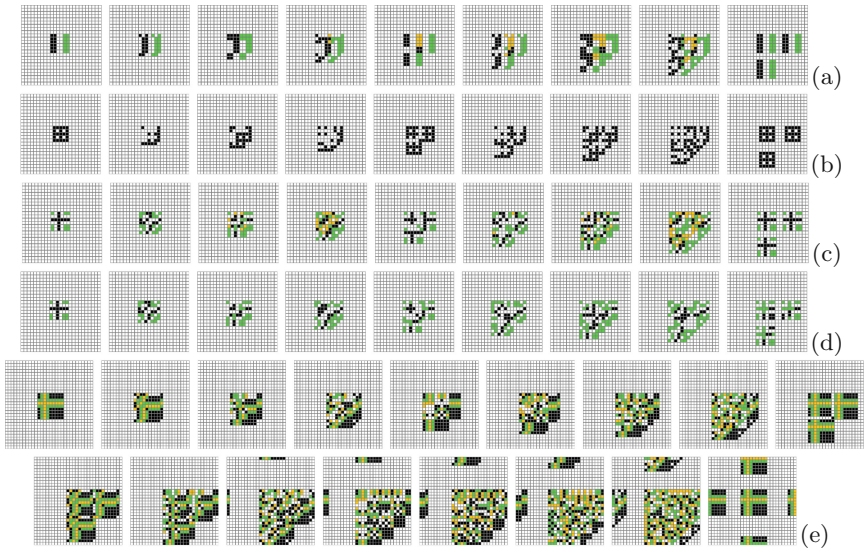

Fig. 3. Example of solution to the replication problem for (a) 3a used on 2a, (b) 3a used on 1a, (c) 3a used on 4a (4 state), (d) 3a used on 4a (3 state, it can no longer replicate the image, but the structure is perfectly replicated), (e) 3a used on the larger structure 5a (note that the lattice size had to be increased to accomodate the larger structure). (Color figure online)

Table 5. Results on development and replication problem using TT and SMIBD. Avg. over 100 runs.

	Transition table			Self modifying IBD		
Pattern	0a	2a	3a	0a	2a	3a
Success rate %	96	0	0	94	2	8
Avg. (NumGen)	877	x	x	1913	7652	5818
StDev. (NumGen)	1228	x	x	1944	2047	2973

produce valid solutions, there are clear differences. Using transition tables, the structure is replicated at step 3 and at step 7 four replicas of the given shape emerge. After, the genotype is not able to keep up the replication process and patterns soon disappear. The self-modifying genome example shows a different scenario (note that in Fig. 5, a bigger lattice is used to demonstrate the replication abilities, whether the original solution was evolved on a smaller lattice). At step 3 the wanted structure emerges and at step 7 five replicas have appeared. This process of replication never stops. In fact, at step 11, the 5 replicas are still present but with some additional space to allow new replicas to emerge, which finally appear at step 15. This process of making space and replicating continues indefinitely (at least for the observed developmental time).

4.4 Re-evolution

Taking inspiration from [12], we want to investigate the ability of the self-modifying representation to evolve solutions to a problem and then re-evolve to a different problem, i.e., adaptation to new fitness requirements/environmental change. We first evolve solutions for the morphogenesis of a given structure and then we use the same evolved population targeting a different morphology. This task is particularly difficult if transition tables are used, as evolution would require a totally different strategy, i.e., moving in a totally different area of the fitness landscape. On the other hand, we expect that a certain structure and modularity will be retained in the self-modifying program solutions. In addition, since both targeted structures are initialized from the same initial seed, i.e. zygote, developmental trajectories [20] may be visualized and shared developmental paths may be identified. Note that re-evolution of solutions for the replication task is considered a much easier task, as shown in the previous section where the evolved replicators presented a certain degree of generalization.

In Fig. 6 (Left) the structure 3a is evolved first, then structure 2b (2a with one additional available state) is used as a new target structure. A new solution is found in only 9 generations. Note that 2b is used as it has the same number of states but different arrangement of colors in the stripes pattern. The first five states in the trajectory are shared, then the paths split but retain the same "algorithmic" structure to reach the different solutions. In general, 100 evolutionary runs were performed. The GA using transition tables evolved the first solution 27 times. Out of those 27 times, 24 were re-evolved successfully to the second solution. On the other hand, with the usage of self-modification, 89 solutions were found to the first structure, which ended up in 84 successful re-evolution of the second structure. Note that with self-modifying genomes, loops (attractors) can be escaped as the regulation mechanism allows different parts of genomes to change or be active at different phenotypic stages.

In Fig. 6 (Right) the structure 4b (Norwegian flag with 4 states) was evolved first and then the structure 2b (French flag with 4 states) is targeted. The selected morphologies have different properties, e.g. horizontal symmetry vs. shifted symmetry. Using transition tables did not produce any result for the re-evolved structure. Self-modifying genomes allowed the 5 solutions found for the first structure to be successfully re-evolved to the second structure. In the shown example, some degree of trajectory modularity is present, indicating that the underlying "algorithmic" structure is retained.

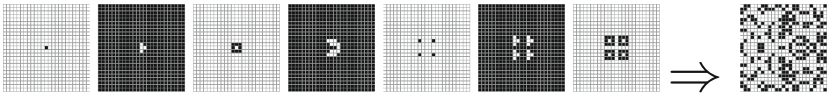

Fig. 4. TT solution to development and replication 0a. States 1 to 7 and 30.

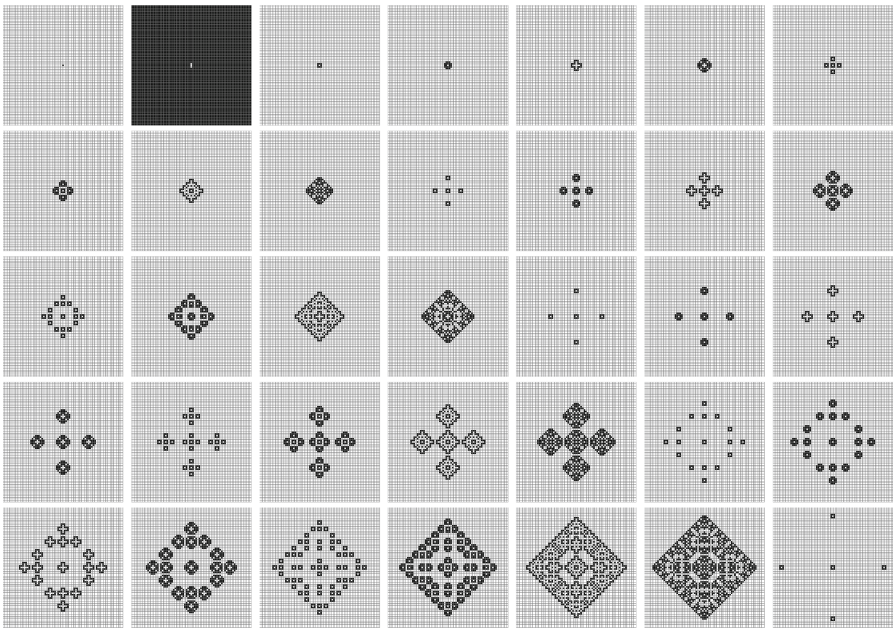

Fig. 5. SMIBD solution to the development and replication problem 0a, using a 75×75 lattice. At step nr. 31 a total of 61 replicas are present.

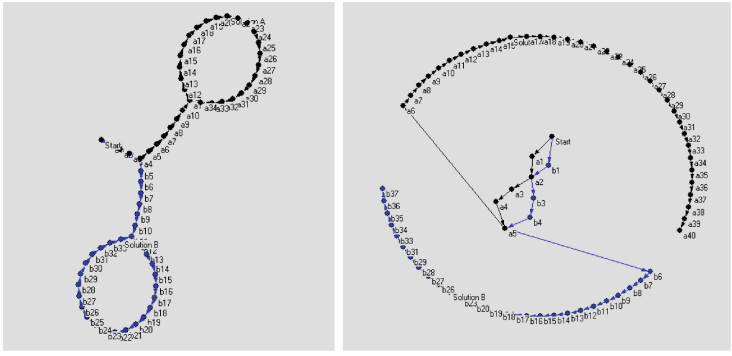

Fig. 6. Left: Developmental trajectories of a solution to problem 3a (black) and a re-evolved solution to 2b (blue), successfully found after 9 generations. The first 5 states are identical, then trajectories split but maintain similar topology. **Right**: Developmental trajectories of a solution to problem 4b (black) and a re-evolved solution to 2b (blue), with intertwined developmental trajectory. (Color figure online)

5 Conclusions and Future Work

In this paper we presented a novel method for regulation of gene expression for artificial evo-devo systems, namely Self-modifying Instruction Based Development.

Cellular automata have been used as experimental platform. Traditional CA transition tables have been compared to CA genotypes in the format of programs with self-modifying instructions, which allowed the emergence of a genome coordination mechanism. In fact, genomes have the ability to self-modify and activate different genes at different stages of the developmental process. Several problems have been solved successfully, as the development of structures, replication, development and replication combined, and reuse of an evolved genotype for development or replication of different structures than those initially targeted by evolution. SMIBD outperformed traditional transition tables, providing the possibility of evolving regulation mechanisms that are more modular. In the future we would like to analyze the scalability aspects of the proposed technique and, in particular, measure the evolvability of solutions with self-modifying genomes. In addition, we want to investigate reuse and re-evolution of solutions towards adaptivity to changing environments of lower and higher complexity.

References

1. Alon, U.: An Introduction to Systems Biology: Design Principles of Biological Circuits. CRC Press, Boca Raton (2006)
2. Altenberg, L., et al.: The evolution of evolvability in genetic programming. In: Advances in Genetic Programming, pp. 47–74 (1994)
3. Bidlo, M., Škarvada, J.: Instruction-based development: from evolution to generic structures of digital circuits. Int. J. Knowl.-Based Intell. Eng. Syst. **12**(3), 221–236 (2008)
4. Bidlo, M., Vasicek, Z.: Evolution of cellular automata using instruction-based approach. In: Congress on Evolutionary Computation, pp. 1–8. IEEE (2012)
5. Clune, J., Mouret, J.B., Lipson, H.: The evolutionary origins of modularity. Proc. Royal Soc. Lond. B: Biol. Sci. **280**(1755), 20122863 (2013)
6. Glover, T.: An investigation into cellular automata: the self-modifying instruction-based approach. Master thesis (2015)
7. Harding, S.L., Miller, J.F., Banzhaf, W.: Self-modifying cartesian genetic programming. In: Miller, J.F. (ed.) Cartesian Genetic Programming, pp. 101–124. Springer, Heidelberg (2011)
8. Kampis, G.: Self-modifying Systems in Biology and Cognitive Science: A New Framework for Dynamics, Information and Complexity, vol. 6. Elsevier, Amsterdam (2013)
9. Kashtan, N., Alon, U.: Spontaneous evolution of modularity and network motifs. Proc. Nati. Acad. Sci. U. S. A. **102**(39), 13773–13778 (2005)
10. Kitano, H.: Designing neural networks using genetic algorithms with graph generation system. Complex Syst. J. **4**, 461–476 (1990)
11. Kitano, H.: Building complex systems using developmental process: an engineering approach. In: Sipper, M., Mange, D., Pérez-Uribe, A. (eds.) ICES 1998. LNCS, vol. 1478, pp. 218–229. Springer, Heidelberg (1998)
12. Kovitz, B.: Experiments with cascading design. In: Proceedings of the 13th European Conference on Artificial Life (ECAL 2015), Workshop EvoEvo (2015)
13. Kowaliw, T., Banzhaf, W.: Augmenting artificial development with local fitness. In: Congress on Evolutionary Computation, pp. 316–323. IEEE (2009)

14. Koza, J.R.: Genetic Programming: On the Programming of Computers by Means of Natural Selection, vol. 1. MIT press, Cambridge (1992)
15. Kumar, S., Bentley, P.J.: Biologically inspired evolutionary development. In: Tyrrell, A.M., Haddow, P.C., Torresen, J. (eds.) ICES 2003. LNCS, vol. 2606, pp. 57–68. Springer, Heidelberg (2003)
16. Martin, A.P.: Increasing genomic complexity by gene duplication and the origin of vertebrates. Am. Nat. **154**(2), 111–128 (1999)
17. Miller, J.F., Thomson, P.: Cartesian genetic programming. In: Poli, R., Banzhaf, W., Langdon, W.B., Miller, J., Nordin, P., Fogarty, T.C. (eds.) EuroGP 2000. LNCS, vol. 1802, pp. 121–132. Springer, Heidelberg (2000)
18. Mitchell, M.: Complexity: A Guided Tour. Oxford University Press, Oxford (2009)
19. Nichele, S., Giskeødegård, A., Tufte, G.: Evolutionary growth of genome representations on artificial cellular organisms with indirect encodings. Artif. Life **22**(1), 76–111 (2016)
20. Nichele, S., Tufte, G.: Trajectories and attractors as specification for the evolution of behaviour in cellular automata. In: 2010 IEEE Congress on Evolutionary Computation (CEC), pp. 1–8. IEEE (2010)
21. Nichele, S., Tufte, G.: Evolutionary growth of genomes for the development and replication of multicellular organisms with indirect encoding. In: 2014 IEEE International Conference on Evolvable Systems (ICES), pp. 141–148. IEEE (2014)
22. Sipper, M.: Evolution of Parallel Cellular Machines. Springer, Heidelberg (1997)
23. Stanley, K.O., Miikkulainen, R.: Achieving high-level functionality through complexification. In: Proceedings of the AAAI-2003 Spring Symposium on Computational Synthesis, pp. 226–232 (2003)
24. Stanley, K.O., Miikkulainen, R.: Competitive coevolution through evolutionary complexification. J. Artif. Intell. Res. (JAIR) **21**, 63–100 (2004)
25. Taylor, J.S., Raes, J.: Duplication and divergence: the evolution of new genes and old ideas. Annu. Rev. Genet. **38**, 615–643 (2004)
26. Trefzer, M.A., Kuyucu, T., Miller, J.F., Tyrrell, A.M.: On the advantages of variable length grns for the evolution of multicellular developmental systems. IEEE Trans. Evol. Comput. **17**(1), 100–121 (2013)
27. Von Neumann, J., Burks, A.W., et al.: Theory of self-reproducing automata. IEEE Trans. Neural Netw. **5**(1), 3–14 (1966)

http://www.springer.com/978-3-319-45822-9

Parallel Problem Solving from Nature – PPSN XIV
14th International Conference, Edinburgh, UK, September 17-21, 2016, Proceedings
Handl, J.; Hart, E.; Lewis, P.R.; López-Ibáñez, M.; Ochoa, G.; Paechter, B. (Eds.)
2016, XXI, 1026 p. 273 illus., Softcover
ISBN: 978-3-319-45822-9