ISBN 978-82-326-2574-1 (printed ver.)
ISBN 978-82-326-2575-8 (electronic ver.)
ISSN 1503-8181

Doctoral thesis

Konstantinos Antonakopoulos

Doctoral theses at NTNU, 2017:251

NTNU
Norwegian University of Science and Technology
Thesis for the Degree of Philosophiae Doctor
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

NTNU
Norwegian University of Science and Technology

NTNU

Doctoral theses at NTNU, 2017:251

Konstantinos Antonakopoulos

Artificial Development and Evolution using Common Developmental Genomes

NTNU
Norwegian University of Science and Technology

Konstantinos Antonakopoulos

Artificial Development and Evolution using Common Developmental Genomes

Thesis for the Degree of Philosophiae Doctor

Trondheim, September 2017

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

NTNU
Norwegian University of
Science and Technology

NTNU
Norwegian University of Science and Technology

Thesis for the Degree of Philosophiae Doctor

Faculty of Information Technology and Electrical Engineering
Department of Computer Science

© Konstantinos Antonakopoulos

ISBN 978-82-326-2574-1 (printed ver.)
ISBN 978-82-326-2575-8 (electronic ver.)
ISSN 1503-8181

Doctoral theses at NTNU, 2017:251

Printed by NTNU Grafisk senter

To my son Michail-Alexandros...

Abstract

Developmental biology seeks to understand how organisms are constructed. Development is a set of very complex processes involving a coded program as the organisms' genome. This genome describes how to build the organism, but not how the organism will look like. The *zygote* (the initial single cell), will eventually develop into a trillion cell organism. This extraordinary phenomenon has been an inspiration to the world of Computer Science and Artificial Life and has led to the creation of Artificial Embryogeny (AE). Artificial Embryogeny is a sub-discipline of evolutionary computation (EC) in which a phenotype undergoes a developmental phase. The number of AE systems currently being developed investigate mainly how principal biological processes and mechanisms can be exploited in the artificial world.

One approach that utilize the phase of biological development in artificial systems is called Artificial Development (AD) where the genotype (genetic representation) contain a similar set of instructions - as in the biological organisms case - called *generative program* or *developmental encoding*. Therefore, the process of development comprise to actually execute those instructions and deal with the highly parallel interactions between them and the structure they create.

On the other hand, nature uses the same fundamental machinery and almost the same genetic information to create vastly different creatures. A study reveals that about 99% of mouse genomes have direct counterparts in humans with cats having 90% of their homologous genes identical to humans. How is it possible for nature to use a vast majority of the same genetic representation in the DNA but still be able to develop such distant species? It was found that a common regulator gene can control the formation of many of the internal organs in both nematodes and vertebrates. Therefore, the very same gene can initiate the process of formation and define its outcome, for example, an intestine or a muscle cell.

This thesis investigates how to design an Artificial Embryogeny by using the same genetic information to develop a class of computational architectures or different computational architectures. The result of this investigation has given rise to the Common Developmental Genomes (CDG). The computational architectures targeted, have a common characteristic of being sparsely-connected networks, with each node acting as a simple computational unit. Such computational architectures are cellular automata and boolean networks, artificial neural networks and cellular neural networks.

The approach followed includes the following steps: a. investigate which architectures are suitable for development with such a model, b. describe a common developmental approach that can handle the targeted architectures, c. define how genetic information can be exploited by the developmental process, so as to develop these architectures and d. identify a suitable genome representation to ensure that different structures can be developed and achieved. The target architectures chosen throughout this thesis were cellular

automata and random boolean networks. The reason for choosing those particular architectures is that they have similar structural and functional properties; in addition, random boolean networks are considered a generalization of cellular automata.

Core work and design principles are given in Paper I. Though this paper it was able to show how genetic information can be represented and how targeted architectures can be integrated in the genotype. It is also shown how target architectures can be evolved and different structures achieved. Paper II studies the ability of CDG to evolve a simple financial market model in problems of varying complexity. CDG was shown to evolve better for some architecture sizes. In addition, CDGs evolvability were studied in case of limited resources (Paper III) with very promising results in certain cases. Paper IV focuses on how genetic operators affect evolution of CDG and studies their developmental dynamics under more complex and random environments. Paper V studies the ability of CDG to adapt when the target goal changes over evolutionary time. CDG were able to find very good solutions with rather simplified structure than anticipated. Paper VI focuses on how CDG exploit the underlying architectures during development and build final structure (network morphology). It was shown that during evolution, CDG exploit a larger number of nodes/cells and manage to maintain only a few neutral and static cells/nodes of the final structure.

Preface

This thesis is submitted to the Norwegian University of Science and Technology (NTNU) for partial fulfillment of the requirements for the degree of philosophiae doctor (PhD). This doctoral work has been performed at the Department of Computer and Information Science, NTNU, Trondheim, with Associate Professor Gunnar Tufte as supervisor.

This PhD thesis has been performed as an integrated PhD study financed by an internal scholarship from the Department of Computer and Information Science and the Faculty of Information Technology, Mathematics and Electrical Engineering at NTNU.

Acknowledgements

First of all, I would like to thank my supervisor Associate Professor Gunnar Tufte for his patience, guidance, encouragement, advice and most importantly, his friendship during my years as a phd student. I have been extremely lucky to have a supervisor who wandered me in the field of artificial development and cellular systems through long brainstorming discussions.

Also, I would like Professor Lasse Natvig for his support and his unique sense of humor, during my first years. Also, the Computer Architecture Group, for a professional and friendly environment.

Last, I would like to thank my family for their moral support and the time I have taken away from them. Without them, it would had been impossible to finish this dissertation.

Konstantinos Antonakopoulos
May 2017

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Everything changes and nothing remains still ... and ... you cannot step twice into the same stream

Heraclitus in Plato's *Cratylus*, p.408a

The chapter introduces the thesis, with an introduction to classical and global optimization methods in Section 1.1. Advantages of evolutionary algorithms are shown in Section 1.2 with an introduction to Artificial Developmental Systems in Section 1.3. Motivation of the thesis is presented in Section 1.4. Section 1.5 presents the research questions that have been identified and explored. An outline is given in Section 1.6.

1.1 Introduction

Nature has been an inspiration to human kind for several thousands years. Human kind has shown remarkable advancements in many areas inspired by nature [17, 19, 46, 94, 97, 99, 124]. Many biological organisms are vastly complex and show sophisticated behaviors and properties that can not be achieved in human engineering. For example, Preissl et al. [92] built a novel cognitive computer architecture simulating 2.084 billion neuro-synaptic cores containing 53×10^{10} neurons and 1.37×10^{14} synapses running 1542 times slower than the human brain in real time. Although computer architecture was inspired by the number of synapses (memory) in a human brain, the design itself has an average of 6 times more than the total number of neurons [47].

In other examples evidence exists that evolution has worked over a very long time to design how sharks look today (fossil findings indicate that the ancestors of today's sharks appeared even before the dinosaurs [123]). The skin of a fast-swimming shark exhibits riblet structures aligned in the direction of a flow that are known to reduce skin friction drag in the turbulent-flow regime [23]. As a result, engineers have tested the effects and have applied similar micro-structures to the surfaces of ships and aircrafts as well as to the swimsuits of Olympic swimmers [13]. So, it is not by luck that humans have turned to nature seeking inspiration, trying to get answers to real-world problems. Although man-made creations may seem advanced and complex, still, in many ways they are inferior to the designs of nature.

1.1.1 Limitations of classical optimization

In general, real-world problems are concerned with some form of optimization. In other words, optimization refers to the process of manipulating a computational structure or system in order to achieve some predefined goal [123]. Many problems are generally formulated as a task of maximizing (or minimizing) a response function for the target problem and involves two types of problem difficulties: i. multiple, conflicting objectives, ii. a highly complex search space [134]. The solutions (methods) provided to these problems tackle usually a small range or a subset of the problem; otherwise are considered rather inefficient. Classical optimization methods can only be used efficiently in certain types of response functions (e.g., smooth, convex, etc). Many practical problems are non-convex and may have many local optima. No classical optimization method guarantees finding this global solution in finite time [125]. Examples of the above are the traveling salesman [90], the knapsack problem [50] but also more specialized ones in branches like finance [29], quantum control [104], medical image analysis [69], engineering optimization [120] and data mining [111].

1.1.2 Global optimization methods

In order to overcome these phenomena, classical optimization methods were replaced by global optimization methods. Evolutionary Algorithms (EA) mimics the processes of Darwinian evolution [21] and were used as a potential global optimization method. EAs have been extensively used and became popular in optimizatioon [25, 83], design problems [24, 49, 101] and robot control [78, 113] and other types of applications [18]. There are many different variants of evolutionary algorithms. The underlying idea behind all these techniques is the same: given a population of individuals within an environment that has limited resources, competition for those resources causes natural selection (survival of the fittest) [31].

In a typical EA, fixed-length (bit) strings form the individuals of a population. Individuals encode single possible solutions and compete continually with each other to discover optimal areas of the search space [52]. In a function optimization, each candidate solution could be represented in the initial population either, as a random real number or as a random binary encoding.

Individuals are being decoded and applied as candidate solutions to the problem. A fitness score is assigned to each individual, based on the fitness function evaluation. Genetic operators such as, mutation (flipping individual bits) and crossover (exchanging substrings of two parents to obtain two offsprings) are applied to the ones with the highest fitness score, producing the children of the next population. Individuals with the highest fitness score have a much better probability of being selected for the next iteration. So, selection is applied either, when choosing individuals to parent children or when choosing individuals to form a new population [52]. The cycle is repeated until some predefined criteria is satisfied.

1.2 Evolutionary Algorithms mechanics

Evolutionary algorithms use the concept of direct genotype-to-phenotype mapping whereby each unit of the phenotype (adult organism) is represented by a single gene in the genotype. This direct mapping is particularly troublesome as problems become increasingly complex [64]. Novel problem domains, such as the evolution of complex neural networks and large commercial buildings require the order of thousands or even millions of structural units for a single phenotype [109]. Using direct genotype-to-phenotype mapping, evolution requires an enormously large search space.

Therefore, in order for evolution to proceed with such problems, the number of genes required to specify a phenotype must be considerably smaller than the number of structural units composing the phenotype. Nature has found a way to achieve this; even with 100 trillion neural connections in the human brain, there are only about 30 thousand active genes in the human genome [26]. This is achieved by reusing genes. In biological organisms, the genome contains a set of instructions on how to construct the phenotype. The same genes can be used at different points during natural development for different purposes, and the order in which activation of genes takes place determines when and where a particular gene is expressed [95]. In other words, by replicating the process of natural development and through indirect genetic encoding, it may be possible to create extremely compact genotypes that represent complex phenotypes.

1.3 Artificial Developmental Systems

Embryogenesis is the first of three complex processes that take place in natural development. It starts with the zygote (fertilized ovum) and from that point on, the zygote undergoes a process called cleavage division, where the cell is split into two (or more) identical cells. Cells at this stage may involve pattern formation as the cells get organized in spatial and temporal patterns [131, 132]. The egg cells are asymmetric with the first cleavage to occur along one axis and the second cleavage along an axis perpendicular to it. The developing embryo receives a substantial amount of information in the starting configuration from its zygote, a process called *molecular prepatterning* [88]. Extraordinary cell movements and conformational changes occur in the developing embryo as part of the morphogenesis process. In the second process of cell differentiation, cells acquire distinct structure and functionality, for example, nerve or skin cells. Growth is the third and last of the complex processes where cells increase in size and multiply, but not further differentiate.

One approach that utilizes the development phase in artificial evolutionary systems, is Artificial Development (AD). Different names have been used in the literature to express the same concept, including Artificial Ontogeny [14], Computational Embryogeny [10], Cellular Encoding [40] and Morphogenesis [51]. In AD, the genotype (genetic representation) contains a similar set of instructions, as in the biological organisms case, called *generative program* or *developmental encoding* [64]. The process of development actually executes those set of instructions, dealing with the highly parallel interactions between them and the structure they generate.

Artificial development systems are mainly divided into those that are based on Turing's cell chemistry approach or reaction-diffusion systems [119] and those based on grammatical approaches [71]. Models based on cell chemical processes are inspired by the natural mechanisms seen in development such as gene expression, cell signaling, gene/protein interaction, gene mutation and recombination, chemical gradients, cell differentiation and cell death. On the other hand, grammatical approaches are based on the evolution of a set of rewriting rules where the grammar may be context-free or context-sensitive and can utilize parameters.

Figure 1.1 shows a single artificial cell (zygote) developing into a 63-cell organism that look like a French flag, just like in nature [84]. The task is considered very difficult as the cell program must replicate to grow to the desired size and hold the desired cell state (colour based) according to local cell interactions only. In Figure 1.1, the cell program holds the complete building plan (DNA) for the artificial organism, including functions like division, specialization and growth. Again, note that this plan is generative. It describes how to build the system and not what the system will look like. The information carried in the evolving genome is the information that passes from generation to generation and the genetic information contained in a cell serves as the constructor of the phenotype [115].

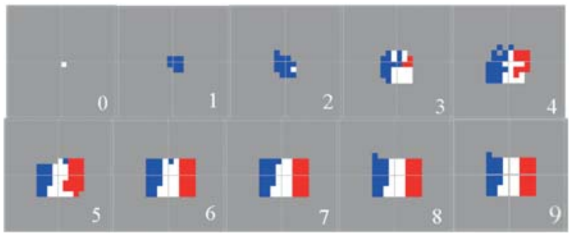


Figure 1.1: Miller's French Flag Problem

1.3.1 Concepts of artificial developmental systems

Scalability is a feature of AD and refers to the possibility of growing artificial designs to tackle harder problems [105]. Kitano [58] proposed the use of a developmental process to create large neural networks, i.e., a structure of neural nodes and connections. By evolving L-system rules as the developmental approach, he managed to increase scalability and scale complexity, i.e., target more complex problems or scale the solution with the problem size. Kitano developed a method for evolving the architecture of an artificial neural network using a matrix re-writing system that manipulated adjacency matrices. Gruau revised a graph re-writing method called *Cellular Encoding* [40]. With this method, he managed to control the division of cells that eventually grow into artificial neural networks. His method was shown to be effective at optimizing both the neural network architecture and its weights at the same time and it scaled better than the direct encoding method [41]. On the other hand, Federici [33] successfully evolved spiking neural networks using a developmental system that showed better performance than the direct encoding method.

Naturally, the developing organism grows and operates within an environment. Food, water, sun and other resources play an important role in defining the growth of an animal as well as external signals do [105]. Environment is an important source of information to the organism where behavior, function and the development process is laid out. Since the information contained in the genome is exploitable by the developmental process in order to "build" the organism, environmental information can be included as an information source to enable adaptation. This implies that natural organisms include developmental plasticity, i.e., phenotypic plasticity [126]. Plasticity in this context is the ability of a species responding to environmental conditions [115]. In other words, the ability of organisms to grow to sizes that fit the environment, maintaining their full functionality from infancy to adulthood. In Lindenmayer systems (L-systems) [73, 74], the models of plants can be run to any size and still retain the same morphology or shape.

Environment may influence the developmental outcome of an organism in a disruptive way. In such cases, the organism must be robust enough so as not to be affected by fluctuations of the environment. Environment includes enforced disturbances or changes to the organism itself [115]. Miller [84] showed that even if his system was externally disturbed by changing parts of the flag structure, development found a way to regenerate it.

1.3.2 Developmental models

An artificial organism can be seen as an Evolutionary Developmental (EvoDevo) system capable of developing some sort of functionality. This implies a developmental system that takes into account information from the environment, intermediate structures and behavior during evolution, together with the genetic information carried along in the genome. The target functionality may include a structure design or a computational function. The functionality is generally given by the computational function of the nodes and their connections. A problem, is usually mapped to a computational architecture and the architecture represents the developing phenotype. Kitano's work on development of Artificial Neural Network structures [59] is an example of a computational architecture, consisting of nodes and an interconnected network. Tufte [114], used a different computational architecture in his developmental model, namely, the von Neumann's cellular automata [121], where cells have a finite number of states that self-replicate, i.e., expand their cellular structure. A non-uniform cellular automata is developed, in other words its structure/form emerges out of a set of development rules capable of cellular growth, differentiation and cell death.

Computational architectures have been extensively used by simple adaptive or EvoDevo design approaches and have shown to be quite beneficial building certain structures or functionalities [11, 39, 84, 107, 114]. However, all studies so far have been focused on designing a developmental mapping towards a specific architecture. When target architecture is a cellular automata, the design and encoding of representational information in the genotype is done under the limitations and restrictions dictated by the architecture itself. In addition, the impediment of target computational architecture further limits the variety of designs or solutions that can possibly be achieved in other cases.

1.4 Motivation of research work

Nature uses the same fundamental machinery and almost the same genetic information to create vastly different creatures. A not so recent study reveals that about 99% of mouse genomes have direct counterparts in humans [42]. Other study show that cats have 90% of their homologous genes identical to humans [91]. How is it possible for nature to use a vast majority of the same genetic representation in the DNA but still be able to develop so distant species? Similarly, Maduro *et al.* found a common regulator gene that controls the formation of many of the internal organs in both nematodes and vertebrates [76]. The same gene determines early on what cells shall become an intestine or a muscle cell. The embryos of most animals divide into three different layers: ectoderm, mesoderm and endoderm. For vertebrate embryos, the mesoderm produces heart, blood and muscles while the endoderm becomes the organs; liver, pancreas and lungs. In the much simpler nematode, just one cell (which is analogous to the mesoderm layer of vertebrates) produces the mesoderm and endoderm organs. As a result, the same genes operate in early stages to control the organ selection in both the worm and vertebrate animals.

These aspects reveal the benefit of having the same genetic representation (genome) to develop and evolve several computational architectures (species). A developmental mapping that can be exploited and be used by several computational architectures or a class of structures is not new [45]. The developmental process must be capable of expressing developmental actions and must be able to express a large variety of network topologies within each architecture, so that it can obtain different structural and computational goals.

Designing an unconstrained EvoDevo system, will enable development and evolution to exploit computational or structural properties from each architecture towards the goal sought.

The current thesis investigates the potentiality of using the same genetic information to develop a class of computational architectures or different types of computational architectures. The analysis and design of a new developmental mapping has led to the proposal of a new approach, called *Common Developmental Genomes* or *CDG*. Cellular Automata (CA) and Random Boolean Networks (RBN) were selected as target computational architectures. RBNs are considered a generalization of CAs, sharing several computational and structural properties. The developmental model is based on a string rewriting grammar L-system [71]. L-systems have shown to be able to describe developmental or generative systems [72, 108]. A Genetic Algorithm (GA), a form of EA, was used in CDG to search through possible solutions maintaining a population of gradually improving candidates until a solution that satisfies the requirements is found. L-systems are responsible to generate the developmental mapping between the genotype and phenotype in CDG. Certain indirect rules are applied to develop both CA and RBN in the developmental system. The GA evaluates the solutions based on a common fitness function.

The goal of this thesis is to investigate and show that CDG are able to exhibit better evolvability than systems developed and evolved by standard genomes (genomes targeting a specific computational architecture). Evolvability, usually describes how well the system adapts to its environment therefore finding good solutions. Evolvability of CDG is studied through the prism of scalability and complexity. Both scalability and complexity are terms that have been interpreted in many different ways in computer science and biological disciplines. Here, the term scalability refers to the ability of CDG to evolve when problem size increases. The term complexity is used as an abstract way to characterize the capacity of information necessary to describe the problem at hand. More detailed analysis is given in Chapter 2.

1.5 Research Questions

The main research question identified and explored by this thesis is:

Is it possible to design an EvoDevo system that can achieve evolvability targeting a class of architectures?

In order to answer the main research question, several smaller and more specific research questions had to be addressed:

- *To what extent can a common developmental mapping be used to evolve classes of architectures?*
- *How and to what extent does the environment affect the development and evolution of common developmental genomes?*

- *What kind of phenotypes are obtained and what are their structural characteristics?*

1.6 Thesis Outline

This thesis is a collection of papers. The main part of this thesis and all relevant research results are found in the papers. The papers were written with this thesis in mind and built naturally on each other, leading to the main research question. Chapter 2 presents the necessary background material and describes the proposed approach, called *Common Developmental Genomes*. Chapter 3 provides an overview of the research process and the papers. Chapter 4 concludes the thesis, summarizes research contributions and outlines future research pathways.

Chapter 2

Background

Thus, from the war of nature, from famine and death, the most exalted object which we are capable of conceiving, namely, the production of the higher animals, directly follows. There is grandeur in this view of life, with its several powers, having been originally breathed into a few forms or into one; and that, whilst this planet has gone cycling on according to the fixed law of gravity, from so simple a beginning endless forms most beautiful and most wonderful have been, and are being, evolved.

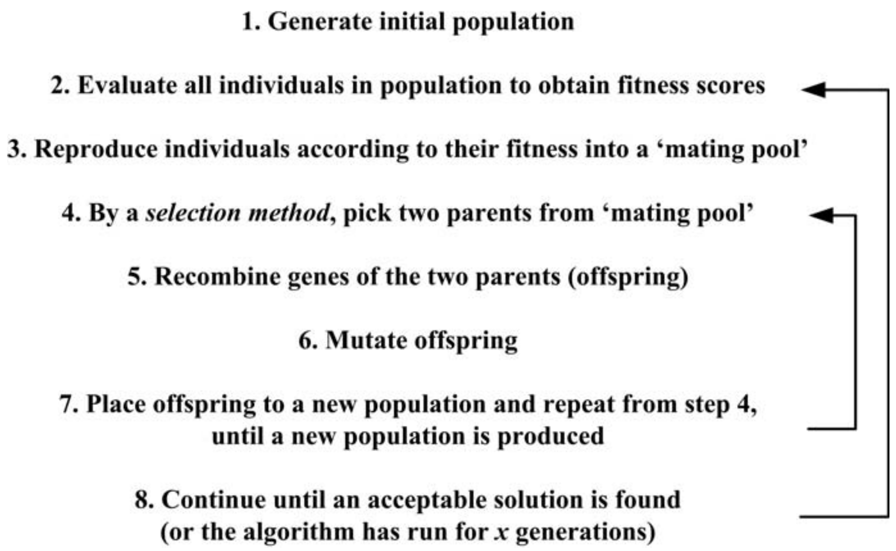
Charles Darwin *The Origin of Species*

This chapter gives an overview of relevant scientific fields and a detailed analysis towards the proposed approach for common development and evolution of classes of computational architectures; the Common Developmental Genomes (CDG) approach. Section 2.1 introduces the concept of evolution and explains basic evolutionary algorithm. Artificial development with specific notions from dynamic systems are presented in Section 2.2. Section 2.3 discusses the concept of evolvability setting it under the right perspective. Section 2.4 explains what scalability. Complexity concept is defined in Section 2.5. Environment is explained in Section 2.6. Introduction to computational architectures is briefly presented in Section 2.7 with a short introduction to the computational models used in this thesis, cellular automata (Section 2.7.1) and random boolean network (Section 2.7.2).

Finally, the proposed approach is introduced in Section 2.8 with reference to Lindenmayer systems. The chapter closes with an illustrative example using Common Developmental Genomes.

2.1 Evolutionary Algorithms

Evolutionary algorithms (EA), as Section 1.2 mentions, are population-based global optimization methods whereby environmental pressure causes natural selection (survival of the fittest) resulting in a rise of the population fitness. Four categories of EA can be found in the literature: the Genetic Algorithm (GA) [48], Genetic Programming (GP) [63], Evolutionary Programming (EP) [35] and Evolutionary Strategies (ES) [96].



1. Generate initial population

2. Evaluate all individuals in population to obtain fitness scores

3. Reproduce individuals according to their fitness into a ‘mating pool’

4. By a *selection method*, pick two parents from ‘mating pool’

5. Recombine genes of the two parents (offspring)

6. Mutate offspring

7. Place offspring to a new population and repeat from step 4, until a new population is produced

8. Continue until an acceptable solution is found (or the algorithm has run for *x* generations)

Figure 2.1: Flow of a simple genetic algorithm (GA). Adapted from [64].

The population includes *individuals* that “explore” the fitness landscape, namely, the space of all possible genotypes. Individuals of the population are evaluated and their fitness score is obtained. A “mating pool” is produced out of individuals with the best fitness score where a higher fitness indicates that more copies of that individual are added to the population. Two parents are randomly picked from the “mating pool” for crossover creating an offspring. The offspring is the randomly mutated and put back into the population. This process is repeated until an acceptable solution is found, or a certain number of generations is produced. The result of each generation is, by definition, a more fit population. The problem solving process that involves the application of EAs, is called *Artificial Evolution* (AE). Figure 2.1 shows the flow of a simple genetic algorithm.

2.2 Artificial Development

Artificial Development (AD) is inspired by biological development and involves processes that use “instructions” encoded into the genome (DNA) to “build” a multi-cellular organism. It typically uses indirect encoding between the genotype and phenotype [109], usually through the expression of rules. Rules can be anything from simple grammar rules [40, 58] to complex gene regulatory systems, designed to closely mimic biological development [64, 28]. AD alows the possibility of shrinking the genome size (genetic representation) of an organism drastically compared to the total number of cells of the developed organism. The example of figure 1.1, shows a cell program i.e., the zygote, that is placed at the center cell of the 16×16 grid (time 0). The cell is a square in a non-toroidal two-dimensional cellular automaton. The inputs to each live cell program are bits defining the cell states and the chemicals in the Moore neighborhood (Section 2.7.1 presents the main neighborhoods for cellular automata). This information is being exploited by the cell’s program in order to decide the amount of each chemical that will produce (binary bits), whether it will live, die or change into a different cell type at next time step and eventually how it will grow into the Moore neighborhood. In such a case, each cell program runs in parallel and there is no knowledge of the overall system whatsoever. Computation is being done locally based purely on each cell’s state in Moore’s neighborhood.

Artificial developmental systems are considered discrete dynamical systems where each developmental state is represented by a point in time. The series of all developmental states from the zygote to the adult organism form the developmental *trajectory* [55]. Figure 2.2 shows an example of a developmental trajectory of a system with states A, B, C, D representing respective timesteps during development.

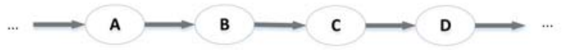


Figure 2.2: An example of a developmental trajectory

Developmental systems have a finite number of states and the system will eventually re-enter a state previously visited. Since these systems are deterministic and the developmental trajectory must always pass from a state to the same successor state, the system will cycle repeatedly among those states. This cyclic behavior is called *state cycle* and state cycles are a form of *dynamical attractors*. Figure 2.3 shows an example of an attractor with state cycle 3.

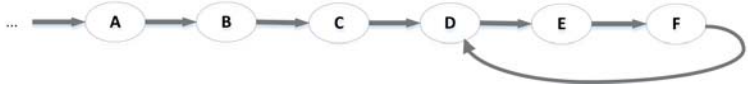


Figure 2.3: Attractor with state cycle 3

In the literature, one can find examples of possible models that can be used in artificial development. Most important example models include *reaction-diffusion systems*

[119], *activation-inhibition* [81, 82], *random boolean networks* [55], *Lindenmayer systems* [71, 72], *cellular encoding* [40], *evolutionary neurogenesis* [58, 59]. In this thesis, a Lindenmayer (L-system) grammar is used to describe the developmental mapping. The proposed developmental approach employs the processes of cell differentiation, cell growth and cell apoptosis. Further details on their implementation are given in Section 2.8.

2.3 Evolvability

Evolvability, is the capacity of a system to evolve [56] or its ability to reach "good" solutions via evolution [87]. In essence, is the amount of phenotypic variation on which selection can act with a given amount or a minimum amount of genetic non-lethal variation. Biological systems are able in a sense, to accumulate random variation allowing the system to function, but also to adapt during its lifetime [56].

Kirschner and Gerhart [57] explain evolvability as the capacity to generate heritable, selectable phenotypic variation, with two basic components: (i) to reduce potential lethality of mutations, and (ii), to reduce the number of mutations needed to produce phenotypically novel traits (properties). Evolvability is considered still at a maturing stage and definitely is a field of active research within: (i) the characteristics of evolvable systems and the requirements of a system to show evolvability, (ii) actual sources of evolvability within biological and non-biological systems, and (iii) the evolution of evolvability.

The main focus of this thesis, is to study how a common developmental process is capable of evolving different systems and what the characteristics of such systems are. Papers I-IV take an experimental approach to investigate evolution in artificial systems and look at the possibilities of evolving certain phenotypic structures. More specifically, paper I focuses on the characteristics of the developmental mapping in order to evolve classes of computational architectures, (see Section 2.7) towards a structural goal. Papers II-IV have the same focus, using basic dynamics as target behavior. Paper III investigates whether the proposed developmental approach (see Section 2.8) favors the evolvability of phenotypes in dynamic behavior problems. Paper IV focuses on the influence that mutation has over the lifetime of an organism and investigates how developmental processes influences evolution over different external environments (see section 2.6).

2.4 Scalability

As explained in Section 1.2, EAs typically use one-to-one mapping from genotype to phenotype, where by each property and characteristic of the phenotype needs to be encoded in the genotype. Figure 2.4, shows a genotype represented by a binary string, where each part indicates the respective status of the cell (active/inactive) in the phenotype, namely, a

4×4 cellular automata. In this case, the genotype size is predefined and is strictly specified by the size of phenotype. In other words, the more complex the phenotype, the more complex the genotype.

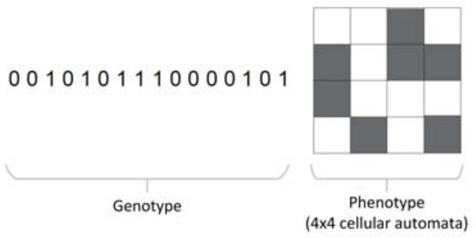


Figure 2.4: An example of one-to-one mapping from genotype to phenotype. Adapted from [10].

Another option is to have redundant genotypic representations with a higher cardinality than that of the phenotype [100], i.e., many-to-one mapping. This type of mapping allows for neutral genotypic mutations that don't change the properties of the phenotype [102] to reduce the chance of evolutionary search trapped in local optima [103].

Nature has found a way of producing incredibly complex organisms from compact genetic representations. Inspired and motivated by natural development, indirect genotype-to-phenotype mappings have enabled researchers to improve their designs and solutions on a much larger scale [43, 84, 39, 38]. This type of mapping reduces the complexity of the genotype and the size of the search space, but transfers the complexity to the mapping itself to the underlying developmental process. When using indirect genotype-to-phenotype mapping, the challenge is to find an efficient developmental process and an indirect representation able to exploit it. Eggenberger [30], suggests that the complex genotype-phenotype mappings typically employed in developmental models allow the reduction of genetic information without losing the complex behaviour and that developmental approaches will scale better on complex problems. However, reducing complexity of genotype and shifting this complexity into a developmental mapping has turned out to be more difficult than anticipated [85]. It is true, that using indirect developmental mapping has not always been as efficient but still considered as a promising approach in some problem areas, such as, evolutionary design optimization[58, 40, 30].

In general, a design or solution is said to be scalable if it is able to provide results in a computational time that is acceptable for increasing instance size. Scaling for any system, biological or artificial is a question of available resources [116]. So, it is important to have an understanding of the domains the resources are fitting in. Tufte [116], proposed two different resource domains for scaling. The first domain regards scaling of mapping external factors (generations, population size, genome size and other factors related to the EA). The second resource domain concerns scaling properties of developmental processes e.g., phenotype size, cellular actions and information processing.

In this thesis, scalability refers to the size of the problem at hand or the size of evolved phenotypes and are investigated through the proposed developmental mapping and genotypic representation. The size aspect relates to the available structural or functional resources for development of the phenotype, e.g. cells [116]. Papers II and VI study scalability property of the proposed developmental mapping in phenotypic sizes, $N = 144$, $N = 169$ and $N = 196$ for both cellular automata and random boolean networks.

Problem sizes considered as a starting point to our investigations and this property alone is by no means significant to scalability. Other problem sizes could also have been considered. Though, extending the information available to gene regulation can be considered as a way to scale up the developmental resources [116]. In that sense, scalability, besides problem size, is also studied in terms of developmental resources, e.g., growth, differentiation, number of developmental iterations and other mechanisms. Other information is included and available to the development process, such as, inter-cell communication, environmental information, N-state memory problem and conditional developmental processes. Therefore, the amount of information available for gene regulation allows the developmental process to emerge organisms with more complex structure and functionality.

2.5 Complexity

Complexity is a term used with its meaning not being universally accepted [62] and many authors use it implicitly. McShea referred to structural complexity [79] depending purely on the number of different parts and interactions of the system, but not looking into its functionality. In [80], he theoretically and empirically explored the notion of functional complexity by looking into large-scale trends and correlations between functions and other variables, such as, size of organism.

Kolmogorov complexity, acts as an inspiration, since it is a measure that can be used on individual, finite objects [67] and associates the complexity of an object, (phenotype) with the length of the shortest computer program that produces the object as output.

In this thesis, the complexity of a problem or instance size is associated with the number of cell or node required in order to define the state of the current cell/node. Complexity is investigated in papers II and VI, through the proposed developmental mapping. Paper II gives a detailed analysis of “own” definition of complexity used in this context.

2.6 Environment

Living organisms have withstood staggering assaults of harmful influences from their environment over the years and not only has life survived, but also it has thrived to evolve into diverse species. Biological organisms proved to be robust and continue to function in the face of perturbations. Perturbations can be twofold; first it can be genetic in terms of mutations [122] or they can be non-genetic, for example, through environmental change

[117]. The term environment has been used in the literature in different levels of granularity. In [118] it was used to define the intra-cell environment that the DNA resides in; also referred to as *cell's metabolism* [32, 39]. The notion of environment found in most artificial developmental systems, is the neighborhood or inter-cell environment, namely, the environment that enables communication between neighboring cells [15, 32, 84]. Biological organisms are affected by the environment emerging from development. The property of phenotypic plasticity [66] enables the organism to adapt to its environment during development. This adaptation may be interpreted as a change in its phenotypic structure and/or behavior. In other words, the developing organism may adapt its structure and/or functionality based on the information it has received from external stimulus [117]. The "environment" of a biological organism refers to the external environment that affects its development. This environment can also be expressed as a combination of *initial environment* and an *external environment* [118]. The initial environment influences the cell as it grows along with the developmental path of any given cell and eventually the whole developing organism. At the same time, the developing organism interacts with its environment in order to survive and this is the external environment.

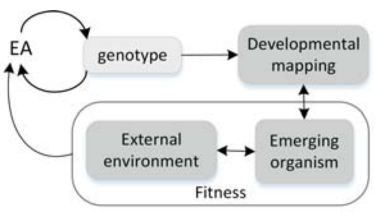


Figure 2.5: Indirect environmental influence through evolution. Adapted from [114].

In this work, we consider both initial and external environments. The initial environment is set only in the beginning of the development process and then the developing organism is affected by external environment only indirectly i.e., through evolution. This is shown in figure 2.5 where the organism emerges from the interplay between the genotype and the emerging organism and through the developmental mapping. What is important about this setting is that at any point in time, information about the genotype and the organism (at that point) is available to the mapping process. As such, fitness evaluation is being performed using the emerging organism (until that point) with its external environment during development. The accumulated fitness is fed back to the evolutionary algorithm. Under this setting, the external environment does not have a direct impact to the developmental process itself. Paper IV investigates how the developing organism adapts its behavior under different external environments, namely, single-cell, random and multiple random environments.

2.7 Computational Architectures

Artificial developmental systems often target problems with some sort of functionality, aiming to solve e.g., a structural problem [110] or a computational function [44] and are usually mapped into a computational model. These models involve nodes or cells representing elements of computation and connections through which nodes or cells interact. As such, the functionality of a developmental system is given by the function of the nodes/cells and their connections. In this thesis, the computational models are typically architectures that consist of simple computational elements that are sparsely connected. Cellular automata and random boolean networks are such target architectures, developed and evolved through the proposed developmental approach.

2.7.1 Cellular Automata (CA)

Several biological systems in nature have been found to involve many simple components. Their overall behavior arises from the cooperative effect of a very large number of parts that each follow rather simple rules [130]. A cellular automaton (CA) is a computational architecture (lattice) that exploits this exact principle and has been extensively used to study biological systems, evolution, ecology and games [128]. A CA includes a regular 2D lattice of N sites with each site taking k possible values. The lattice is updated synchronously in discrete time steps according to a local rule ϕ that depends on the value (state) of sites in some neighborhood structure. The most commonly studied neighborhood structures for two-dimensional cellular automata include the von Neumann neighborhood, consisting of the four horizontal and vertical sites, adjacent to the center site of interest (Figure 2.6(a)), and the Moore neighborhood consisting of all eight sites, immediately adjacent to the center site (Figure 2.6(b)). Several other possible lattices and neighborhood structures are possible for two-dimensional cellular automata as the triangular and hexagonal lattices [130]. The triangular and hexagonal cellular automaton may be considered as special cases of the general nine-neighbor square CA.

Figure 2.6: Neighborhood structures for two dimensional cellular automata: (a) von Neumann neighborhood, (b) Moore neighborhood.

Another important parameter in the definition of cellular automata is the applicable "range" of the rule r: the value of a given site depends on the values of the neighborhood, $2r + 1$ sites at previous time step $t-1$. In other words, the features of the cellular automaton may travel at most r sites per time step. As an example, an "elementary" two-dimensional CA with a von Neumann neighborhood $(s = 4)$ with two possible values $(0, 1)$ per site $k = 2$ and $r = 1$, the total number of rules will contain $2^{2^4} = 65536$ possible states. The value α of the center site at position i, j at time step $t + 1$ is the outcome of the rule that,based on the von Neumann neighborhood, depends only on five-nearest neighbors according to

$$\alpha_{i,j}^{(t+1)} = \phi[\alpha_{i,j}^{(t)}, \alpha_{i,j+1}^{(t)}, \alpha_{i+1,j}^{(t)}, \alpha_{i,j-1}^{(t)}, \alpha_{i-1,j}^{(t)}] \tag{2.1}$$

John von Neumann was the first to introduce the notion of cellular automaton in the mid-1940s as he developed an abstract model for studying self-reproduction inspired by biology [128]. Stanislaw Ulam working independently in 1951 considered the problem and managed to simplify Neumann's model with a two-dimensional cellular automaton. That particular automaton was constructed in 1952-3 and had 29 possible colors for each cell and rather complicated rules. They were able to emulate the operation of the components of a digital computer and various mechanical devices. Later on, and based on Ulam's work, von Neumann was able to mathematically provide the principles for self-reproduction and eventually investigate systems exhibiting such sophisticated capabilities. Cellular automata have been worked out as parallel computers since the 1950s and later their formal computational capabilities have been theoretically proved and shown as an analog to Turing machines. Other specific types of 2D CAs started to be used in the 50s and 60s, as a way to optimize circuits for arithmetic operations, simulate idealized neural networks or other body parts, such as, heart or muscles as well as reaction-diffusion processes [128].

Cellular automata can be viewed either as computers themselves or as logical universes within which computers may be embedded [65]. CAs are systems exhibiting interesting dynamics and have been used to investigate their global properties. Wolfram [129, 89] has proposed four qualitative classes, according to the results of evolving the system from a "disordered" initial configuration:

1. Class I evolves to a homogeneous state, that is, a unique state from all possible state-space for any random initial configuration. This class of automaton are equivalent to dynamical systems that reach a fixed-point attractor and this type of automata cannot be reversible since the initial information is lost.

2. Class II evolves to simple structures which can be stable or periodic. For any random initial configuration resembles dynamical systems with periodic behavior. Automata of this class may act as filters in the sense that certain data sequences can be maintained in the system while others are made void. In addition, Class II systems are of finite size with finite number of cells and that exact property leads to periodic behavior. The system eventually does find itself in a previously visited state and thus it is inevitable to repeat itself. During evolution, different parts of the system may evolve separately from each other without any long-range information transmission.

3. Class III evolves to a chaotic pattern with long duration at any random initial configuration. This class is a equivalent to a chaotic dynamical system and play an

important role in studying randomness.

4. Class IV yields complex patterns of localized structures, sometimes long-lived. In this class, non-trivial structures emerge and may include structures from the uniform, periodic or chaotic regimes. Although Classes I, II and III demonstrate a basic type of behavior already found in dynamical systems, automata of Class IV have no dynamical system equivalent. According to Wolfram [130], at the border between order and chaos is where universal computation could be possible [129] and Langton called this area "edge-of-chaos" [65].

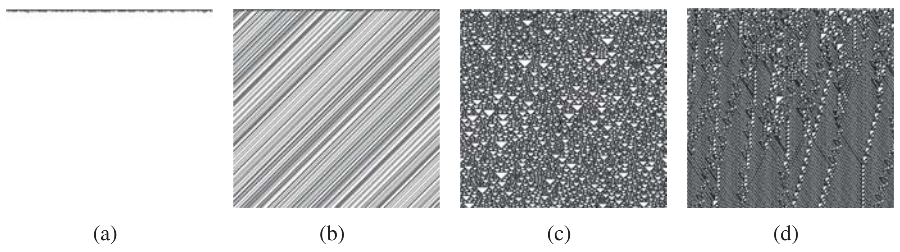


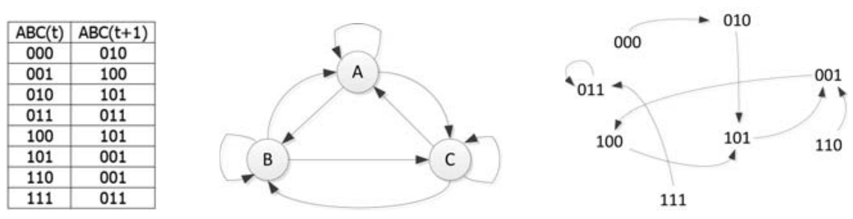
Figure 2.7: Wolfram's computational classes for the evolution of rule 32 in elementary cellular automata. Adapted from [77].

Figure 2.7 illustrates Wolfram's classes with a focus on evolving rule 32 for an elementary cellular automaton. All evolutions were run with the same random initial configuration, 50% density for state 0 (white dots) and state 1 (black dots). The evolution space begins with a CA of 358 cells and evolved for 344 generations. Evolutionary time runs from top to bottom. Figure 2.7(a) shows rule 32 converging quickly into a uniform state. Figure 2.7(b) shows a periodical evolution of blocks of state one cells in a leftward shift fashion. A typical chaotic evolution with aperiodic patterns or unstable points is shown in figure 2.7(c). Small changes in initial configuration leads to an increasing change in the final structure, usually circular or at least rounded. Finally, figure 2.7(d)) shows the emergence of non-trivial patterns typically from the uniform, periodic and chaotic regimes.

2.7.2 Random Boolean Networks

Random Boolean Networks (RBN) are discrete dynamical systems and were originally developed by Kauffmann, towards modeling and understanding gene regulatory mechanisms (also known as $N-K$ models) [54, 55]. RBNs are generic models since their nodes assume no function nor do they imply any connectivity among the nodes. Kauffmann first proposed the hypothesis that living organisms could be constructed from random elements without the need of precisely programming them and he has shown great analogies to biological organisms [54]. An RBN consists of N nodes which can take values of zero or one (Boolean), called *state*. The state is determined by K connections coming from other (or the same) nodes. Connections are randomly assigned but remain fixed throughout network's dynamics.

Each node holds a logic function which is generated randomly, using a node lookup table, that takes the state of each incoming node as an input and the state of the "current" node as an output. So, each node affects each other by the connectivity pattern and by its output state. RBNs are considered a generalization of Boolean Cellular Automata (CA) [121, 133, 127], where each node may not be affected only by its neighborhood but potentially by any node in the network. Another property of RBNs is the rate by which nodes updates their status in the network, called *updating scheme*. Kauffmann studied the standard RBN models where nodes' state is updated synchronously: the state of the node at time $t + 1$ depends on the states of nodes at time t.



ABC(t)	ABC(t+1)
000	010
001	100
010	101
011	011
100	101
101	001
110	001
111	011

(a) (b) (c)

Figure 2.8: (a) Lookup table for state transitions, (b) RBN with node elements and their connections, (c) State space diagram

The dynamics of RBNs are usually determined by their initial state (i.e., random state), the updating functions and the scheme. Since the state space is finite (2^N), a state will eventually be repeated and since the dynamics are deterministic, the network will definitely reach an attractor (either point or cycle attractor) [37]. An example of an RBN with $N = 3$ nodes and $K = 3$ connections is shown at figure 2.8. Figure 2.8(a) shows the lookup table for the state transitions of the RBN at time t and $t + 1$ respectively. Figure 2.8(b) shows the wiring diagram of the RBN indicating the nodes affected. In this example, all nodes affect all nodes. Each node can be active or inactive (zero or one), with the number of combinations of the states of $K = 3$ inputs is $2^3 = 8$. The boolean function specifies whether the regulated element is active or inactive for each of these combinations. The total number of possible boolean functions F of K inputs for each node in all potential networks we can obtain, is $F = 2^{2^K}$. The dynamic behavior of the network is the set of states in the total state space that are visited. Figure 2.8(c) shows the state space diagram of an RBN. There is one point attractor (011) with one state flowing into it (111), and one cycle attractor of period three (001, 100, 101), with two states flowing into it (110, 010). The set of states flowing into an attractor is called attractor *basin*.

One observes how much the network changes and therefore distinguishes different dynamic regimes: *ordered*, *chaotic* and *critical* (figure 2.9), by plotting the states of an RBN network, where the state of a node depends on its neighbors. In the ordered regime, after the initial random state, the states of the nodes in the network changes but the dynamics of the network quickly stabilize and most of the nodes become static. Networks in this regime are very robust to perturbations. In the chaotic regime, networks take time to reach an attractor as most of the states change constantly with only a few "stable islands".

Networks in this regime are sensitive to initial conditions in a sense that small perturbations can have large consequences to the network (butterfly effect). RBNs with uniform connectivity have been found to have a phase transition from ordered to chaotic for a critical value $K = 2$ [55]. This critical value can be influenced by the bias existing in the functions (not an equal opportunity for zeros and ones) and the topology of the networks [3, 8].

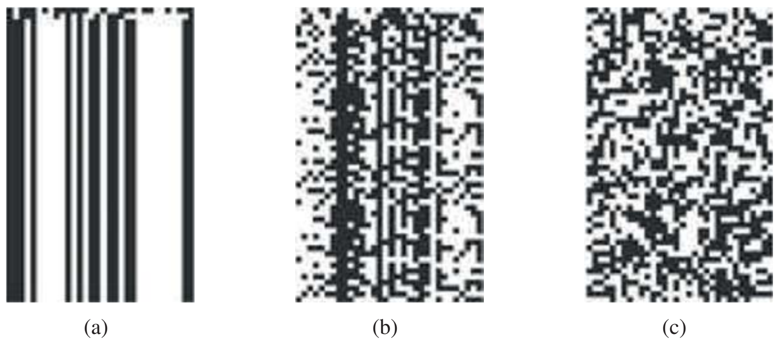


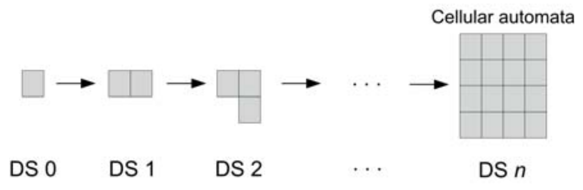
Figure 2.9: A RBN with its dynamic regimes ($N = 32$). Square represent the state of a node. Initial conditions shown at the top with time flowing downwards (a) ordered $K = 1$, (b) critical $K = 2$, (c) chaotic regime, $K = 5$. Adapted from [37].

The structure of the nodes is very important for the dynamics of the RBNs. Parameters like N, K, and p (the probability of a node being one), have been extensively used to study RBNs both statistically and analytically in terms of the number and length of attractors [34, 27, 9], sizes and distributions of their basins and how those interact with each other [12]. Good overviews with more in-depth analysis of RBNs can be found in [36, 55, 53].

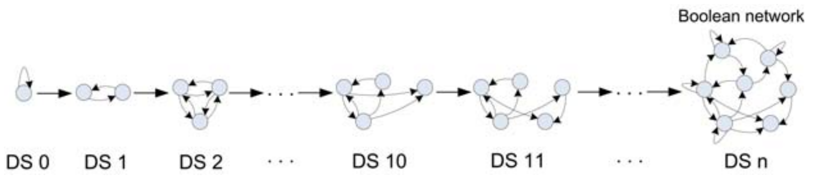
2.8 Common Developmental Genomes (CDG)

The aim of this thesis is to be able to generate not a specific but different classes of structures using the same approach. Such developmental approach requires sufficient knowledge of the targeted computational architectures along with their governing properties. The computational architectures targeted are 2-dimensional non-uniform cellular automata and random boolean networks. The properties of these architectures in a developmental setting are identified and described in [4]. Three major requirements were identified and worked out via the proposed approach:

- Developmental model. The model should be able to develop classes of structures, taking into account the special properties governing each computational architecture (dimension, neighborhood, connectivity) and have the same genome as input.
- Genome. The genome should contain information about the cell/node type and its wiring at each developmental step (figure 2.10)



(a)



(b)

Figure 2.10: The genome contains information about cell/node type and connectivity. Genome gives input to the developmental model and the model is able to develop both (a) a Cellular Automata, and (b) a Random Boolean Network.

- Developmental processes. The concept of *chromosomes* was incorporated to provide a logical grouping and separate functional (processes-related) from structural (connectivity-related) genomes. Figure 2.11, shows what the genome looks like; the first chromosome concerns the developmental processes and the second chromosome holds information about connectivity.

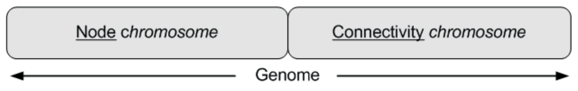


Figure 2.11: The complete genome with the Node and Connectivity chromosomes.

Each chromosome is built out of rules. The first chromosome includes rules on how to process cells/nodes. The rules of the first chromosome include processes like growth, differentiation and apoptosis. The second chromosome has rules supporting the wiring of nodes in a random boolean network (since wiring of cellular automata is given). Chromosomes may have configurable size. Figure 2.12 shows how the genome is made out of rules.

To express the rules of the chromosomes, Lindenmayer grammar has been employed. Separate grammars are devoted for node/cell generation and connectivity. In that way, the wiring of the system is not designed into the system but rather developed by grammar. The model has the capacity to map genetic and environmental information to phenotypic properties, e.g., growth and differentiation.

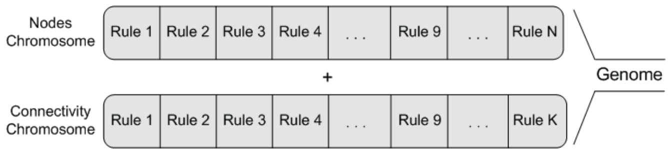


Figure 2.12: Chromosome are made of rules. Each chromosome may have different size.

Table 2.1: (a) Symbols used in the first L-system, (b) Symbols used in the second L-system

(a)

Symbol	Description
a (AXIOM)	Add (growth)
b	Add (growth)
c	Add (growth)
d	Delete (apoptosis)
X	Substitute (differentiation)
Y	Substitute (differentiation)
$\rightarrow$	Production

(b)

Symbol	Description
x	Node (different from y)
y	Node (different from x)
+	Connect forward
−	Connect backwards
$\rightarrow$	Production

2.8.1 Lindenmayer grammar

Lindenmayer systems (L-systems) are rewriting grammars, able to describe developmental or generative systems and have successfully been used to simulate biological processes [70] and describe computational machines [108].

The developmental model includes two L-system grammars. The first L-system is context-sensitive and development uses the strict predecessor/ancestor to determine the applicable production rule. This L-system is responsible for generating the phenotype with the final set of cells/nodes. Some cells/nodes perform special cell processes and influence the intermediate and final phenotypes. The cell processes are represented as symbols in the L-system. Symbol *a* is the axiom and each phenotype starts with this node. Symbols *a*, *b* and *c* are responsible for phenotypic growth. Symbol *d* performs apoptosis of the current cell/node (cell death). Symbols *X* and *Y* are responsible for cell differentiation leading to the replacement of the predecessor cell/node. Table 2.1(a) shows the symbols used in the first L-system. It is obvious that several combinations of rules when deployed may lead to the same phenotype (symbol sequence).

The second L-system is D0L (zero-sided interactions) and responsible for building the connectivity of boolean networks. Each node in the network has a unique numbering separating them from other nodes. The current node has always the number 0 and any nodes starting from the current node onward have positive numbering, where nodes bounce backwards have negative numbering. This is an easy way to discriminate and realize the relative location of the nodes in the network. The L-system includes the plus (+) symbol to represent connections that are created onwards from the current node and the minus (-) symbol for connections that bounce back from the current node.

The length of each rule in both chromosomes is 4 symbols. That is, the length of each rule takes up $4 \times 8 = 32$ bits. Table 2.1(b) shows the symbols used in the second L-system. The *axiom* rule for the second chromosome is x→y. This means that development initially searches in the chromosome whether the axiom actually exists. If so, development continues and looks for rules of type xy→+<*value*> or xy→-<*value*>. These two rules imply that if two different (distinct) nodes are found (x≠y), then it creates a connection forward (assuming the rule includes '+') or bounces backward (if the rule includes '-'). The field <*value*> denotes the node number for the onwards/bounce backwards connection. For example, rule xy→+3 denotes that a connection will be created from the current node (node 0), to the one that stands three nodes onwards (figure 2.13(a)). Similarly, rule xy→-3 denotes that a connection will be created starting from the current node (node 0) to the one that stands three nodes backwards (figure 2.13(b)). If the destination node does not exists (i.e., has not been created yet), a self-connection to the current node is created instead. Selected examples of how the developmental model works is shown in Section 2.8.2.

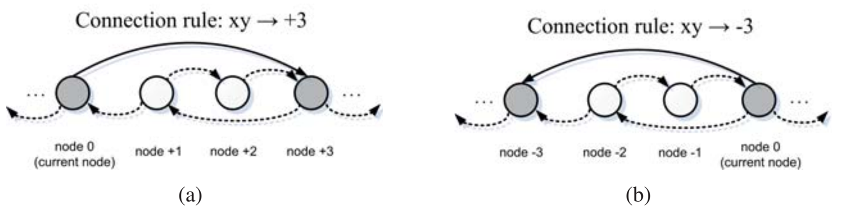


Figure 2.13: Example of (a) forward connection and (b) backward connection. New connection is shown in solid line while existing connections in dashed line.

2.8.2 Example of proposed developmental system

In this section we present an example with step-by-step development of a Cellular Automaton and a Random Boolean Network, based on L-system rules that show how phenotypes are generated and mapped to the computational architecture.

Suppose the chromosome for node generation has the rules shown in figure 2.14 (Node Chromosome). Each rule can be triggered multiple times during development. The construction of the phenotype always starts from the axiom (symbol *a*), at developmental step 0. The axiom triggers the first rule of the chromosome a→bX at dev. step 1, resulting in phenotypic growth (*bX*). In the next development step, two rules are triggered; rules b→Xa and X→Y. The first rule adds growth to the phenotype where the second rule differentiates cells X and Y. The phenotype in dev. step 2 becomes *XaY*. Next, rules X→Y, a→bX and Y→a are triggered. The phenotype in dev.step 3 becomes *YbXa*. In dev.step 4, four rules are triggered and the phenotype becomes *aXaYbX*. Development continues for a predefined number of steps and the cell processes act upon the phenotype which is directly mapped at the cellular automata lattice, as shown in the bottom of Figure 2.14.

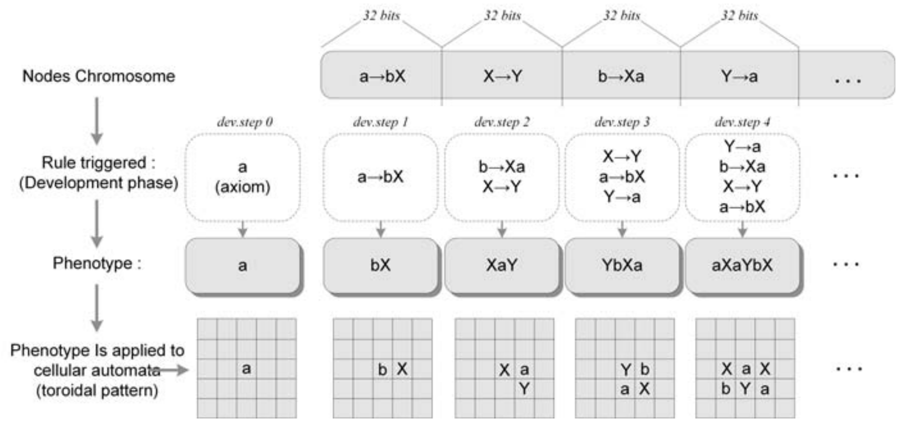


Figure 2.14: The example shows how the rules of the first chromosome are triggered during development. The sequence of the symbols represent the phenotype. The generation of the phenotype is shown separately here only for illustration purposes. Cell processes act upon the phenotype which is directly mapped at the cellular automata lattice.

The proposed developmental approach uses the rules of the node chromosome (figure 2.14), to develop the nodes for all targeted architectures. Therefore, for random boolean network development, same nodes shall be used for structuring the network. Figure 2.15 shows an example of how rules of a second chromosome may look like.

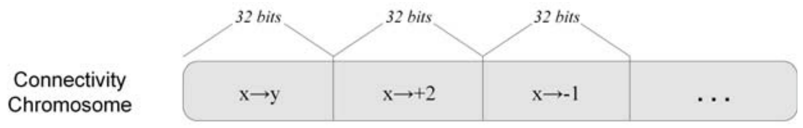


Figure 2.15: Example of second chromosome including rules for generating the connections in a random boolean network.

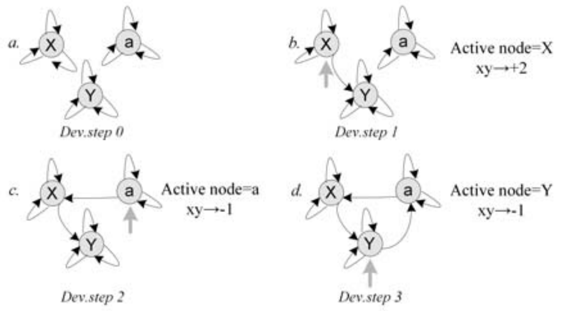


Figure 2.16: Connectivity rules are being applied to the nodes of the random boolean network. Development of connections occurs after phenotype is generated.

Figure 2.16 shows an example of how connectivity rules are applied to the nodes of a random boolean network. An intermediate phenotype `XaY` of dev.step 2 of figure 2.14 is considered.

Initially, nodes are generated with a default number of 3 self-connections (dev.step 0). Next, `X` becomes the "active" node and rule `xy`$\rightarrow$`+2` is triggered creating a forward connection between nodes `X` and `Y` (dev.step 1). Node `a` become "active" at dev.step 2, with the valid rule `xy`$\rightarrow$`-1` being applied. Hence, a backward connection is created between nodes `a` and `X`. In dev.step 3, node `Y` becomes the "active" node. In this step, the same connectivity rule applies as before and a backward connection is created between nodes `Y` and `a`. Finally, it it important to remember that for a rule to be valid, the destination node referenced in the connectivity rule must exist in the network, otherwise the rule does not apply.

Chapter 3

Research Summary

A computer would deserve to be
called intelligent if it could deceive
a human into believing that it was
human

Alan Turing, 1950

This chapter gives a summary of the research that took place during this thesis. The chapter starts in Section 3.1 with a description of the research process defining the problem determining the research questions. Section 3.2 lists all publications by the author. Note that all publications are relevant to the topic and are built on top of each other. In other words, they should be read in a proper sequence. The papers listed represent the research path taken towards finding answers to the research questions of this work. Section 3.3 gives the abstracts and retrospective views of each paper.

3.1 Research Process

This section presents the research process that led to this thesis. The major choices and motivation behind them are presented together with possible research pathways. An illustration of how the published papers relate to each other is shown in figure 3.1.

3.1.1 Background

The initial PhD project proposal was connected to the research group CRAB lab (Complex, Reconfigurable, Adaptive, Bio-inspired Computing hardware) with the main focus in using biological inspiration from evolution and development towards hardware capable of unconventional computation. So, a potential approach for this project would be

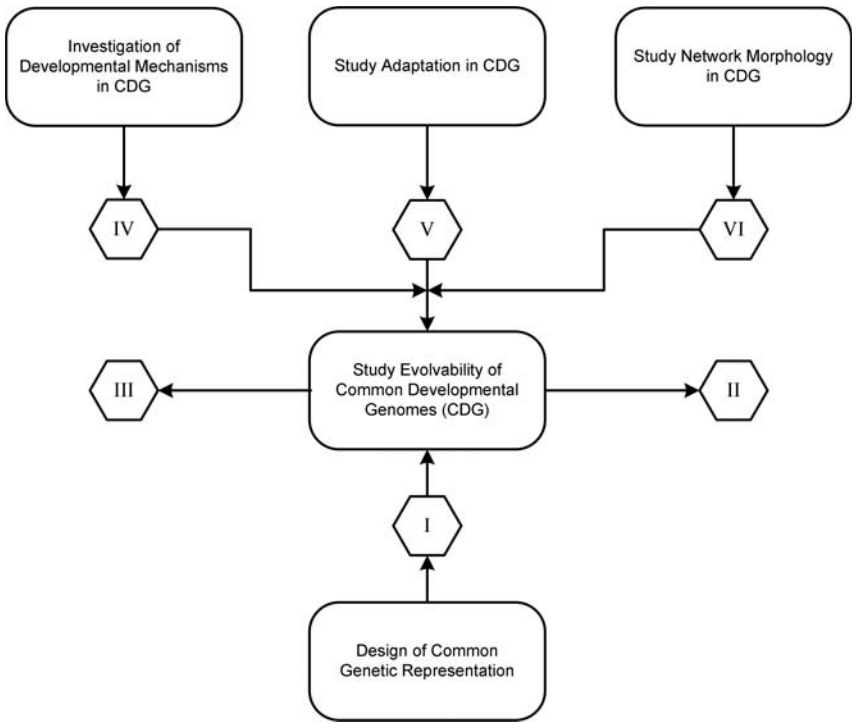

Investigation of Developmental Mechanisms in CDG
Study Adaptation in CDG
Study Network Morphology in CDG
IV
V
VI
III
Study Evolvability of Common Developmental Genomes (CDG)
II
I
Design of Common Genetic Representation

Figure 3.1: Research process and relation of papers

to use cellular computation in combination with artificial development and evolutionary algorithms. An initial goal was to develop computational circuits using the lab's existing hardware platform (FPGA) for experimentation. Although FPGA was set as the target hardware platform, due to my previous academic experience in cellular architectures, it was a reasonable decision to use computer simulations instead. So, the approach of the PhD project turned out to have an initial focus on:

1. Exploit artificial development with regard to the creation of basic computational functions to fit in an unconventional computational architecture (i.e., cellular computing systems [106]), and
2. Investigation of developmental properties and their impact during development and evolution of such computational architectures

This work was initially planned to be a close collaboration with the Intelligent Systems Group, University of York, with Dr. Julian F. Miller.

3.1.2 Initial Investigations

Applying biologically inspired design methods such as evolutionary algorithms and artificial development as a design tool is generally not an easy task. The approach is relatively new and there is little knowledge on how to better design a biologically inspired method to target a problem. As such, studying basic concepts and processes found in biology was the first way to go. It is important to increase own understanding on how biological processes are designed and applied in the artificial world through computer simulations. After this introductory period, a literature study was carried out to get a concrete overview of bio-inspired computing systems with a focus in developmental mappings. It became evident that some sort of computational architecture was necessary to be used as an underlying computational model.

To reach one of the initial goals, to develop an artificial organism and investigate how basic computational functions fit into it, the focus had targeted some sort of developmental mapping able to produce phenotypic structures consisting of connected computational elements. To enable the development of such structures, further knowledge was required concerning the capabilities and constraints of the target computational architectures.

The literature study made evident that some computational architectures comprise sparsely connected networks and their behavior emerges from cell interactions where a cell is a simple and uniform computational element. The computational architectures initially chosen to be studied, were cellular automata (CA), random boolean networks (RBN), artificial neural networks (ANN) and cellular neural networks (CNN).

A research effort was undertaken to investigate how computational structures can be achieved by the aforementioned architectures. Their functionality and behavior was analyzed and compared. The main focus was the type of information required by the computational architectures and how developmental approaches of figure 3.2 could be used in an Evo-devo design approach [4]. The fact that the proposed computational architecture yield common properties, namely, comprise sparsely connected networks with simple computational elements, inspired us to focus on how a developmental mapping could work on a class of architectures, in a more general way.

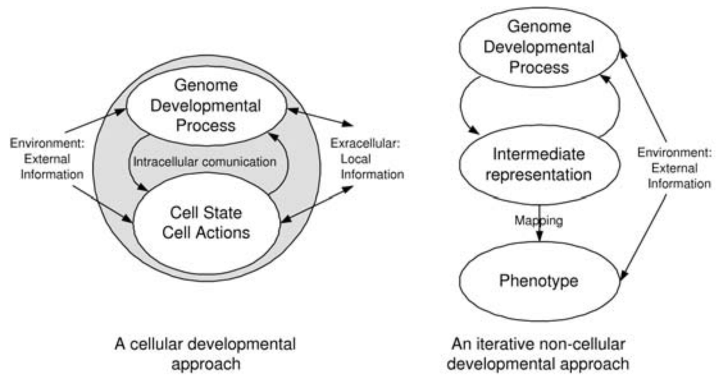


Figure 3.2: Potential developmental mappings to be used in an Evo-devo design approach.

Two possibilities were considered; the first possibility involved defining the developmental mechanisms to express the architecture's inherent properties, namely connectivity and functionality. This option provides a relative freedom to specify relevant cell processes (e.g., growth, division, differentiation, etc.), genome representation and how these cell processes will be carried out. The second possibility includes an evolutionary search of the developmental mapping itself. The goal is to find ways to map the information at hand, namely genome, environment and intermediate phenotypes, to be expressed in the emerging phenotype. In the next section, it will become evident the path chosen and the reasons that lie behind.

3.1.3 Evolvability and Common Genetic Representation

Based on initial investigations, two out of the four proposed computational architectures were considered, that is, cellular automata and boolean networks, since their dynamic behavior can be expressed in discrete time. In addition, the first possibility was chosen as the way forward because, as explained in section 3.1.2, gives the privilege to architect the developmental mapping in a non-restrictive manner.

The design challenges that had to be fulfilled are addressed in paper I. The first challenge was to define a developmental mapping able to sufficiently describe and handle all computational architectures, taking into account architectures' specific properties. The developmental model should be able to receive the same kind of genome as input, depending on some genome properties, to determine whether it will develop a cellular automata or a random boolean network.

For example, figure 2.10(a) shows a step-by-step development of a 2D cellular automata. In DS 0, the first cell of the cellular automata is created. In DS 1, the cellular automata grows in size and a new cell is added. In DS 2, architecture grows again. In DS n, development has stopped and the cellular automata has its final structure (adult organism). Similarly, figure 2.10(b) presents a step-by-step development of a random boolean network using the same developmental model. In DS 0, the first node with its self-connections is created. In DS 1, the random boolean network grows and a new node is added. This causes new connections to be created for the nodes of the network. The algorithm continues until all nodes in the network and their connections have been defined (DS n). We used literature study and several months of experimental simulations to decide that L-system rewriting grammars [71, 72] would be one of the most prominent developmental models, to evolve target architectures. Other approaches were also considered, based on biochemical processes, but L-system grammars are more easily understood and implemented.

The second challenge was to define the information type to be included in the developmental genome. The genome should contain information regarding the cells in each developmental step and in order to place them in the architecture. For example, in the case of a 2-D CA architecture, the properties of dimensionality and neighborhood had to be defined, where the connectivity is pre-determined (figure 2.10(a)). For boolean networks, the connectivity (i.e., connections among the nodes in the network) is not given but has to be worked out (figure 2.10(b)).

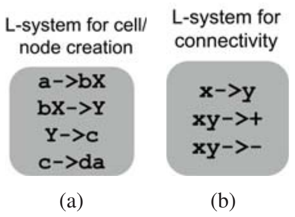


Figure 3.3: Example of L-system grammars: (a) L-system for node/cell creation, (b) L-system for connectivity.

The third challenge was to identify the cell processes to be included in the developmental model. After a couple of months researching the different cellular processes applied in an artificial setting, the conclusion was to take the simplest possible cell processes, that is, *growth*, *differentiation* and *apoptosis*. The decision was driven by the basic requirements of the target computational goal, be able to develop structures. These requirements dictate that structures should be able to grow (growth), differentiate their cell functionality (differentiation) and shrink in size (apoptosis). An example of these cell processes assigned to L-system symbols is shown in table 2.1(a). Similarly, table 2.1(b) shows an L-system describing the connectivity rules for the target structures.

Figure 3.3 gives an example of two L-systems: (a) an L-system for node/cell creation, and (b) an L-system for creating the connectivity of a random boolean network. After identifying the cell processes, considerable work was carried out on how cell processes could be incorporated into the genome. Here, the notion of *chromosomes* was employed, with each chromosome containing the information addressed by the second challenge. So, the genome would comprise a chromosome to withhold the cell/node information of the architecture and a second chromosome to describe the connectivity details among nodes of the structure (figure 2.11).

The notion of chromosomes was chosen because it allows to exploit the genome in a modular way, in a sense that if an additional computational architectures need to be described in the future by the same genome, more chromosomes can be added to it. This new chromosome would incorporate the special properties of the new architecture. In other words, although cellular automata and boolean networks where the chosen computational architectures, the established principles enable us to facilitate any other architecture with reasonable flexibility.

Paper I, deals with the details of the developmental model briefly described here. In addition, it was shown that the genetic representation was able to build and evolve structurally stable solutions. The developmental model is called *Common Developmental Genomes* (CDG), as the genome aiming to develop and evolve more than one architecture. Paper I verified also the functionality of the computational platform being developed over the previous two years. This computational platform was built with a mindset enabling to run all future experiments required by this thesis.

After this work, it was evident that Common Developmental Genomes approach was able to evolve solutions with a structural goal as a target. Though, further experimentation was required having a computational goal as target. Discussion took place around the types of problems to be executed and how. The discussions pointed to a need to experimentally test the evolvability of CDG under two main conditions. First, evolve CDG with limited resources under the same external environment and second, evolve CDG in problems of increasing complexity. Papers II and III, investigate the evolvability of CDG in the first and second conditions respectively and are further detailed in the next section.

3.1.4 Studying the Evolvability of Common Developmental Genomes

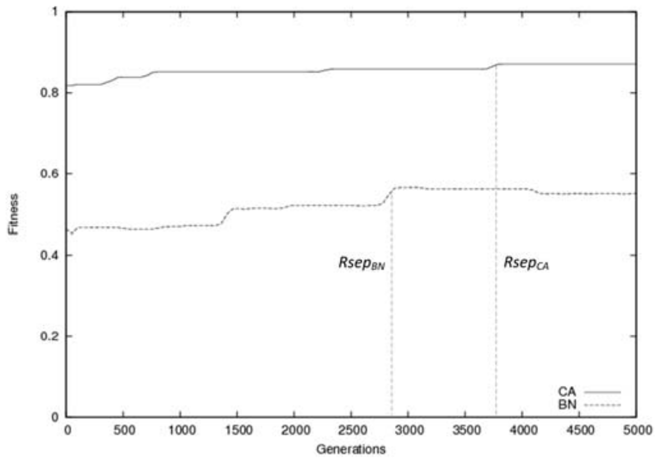
A big part of this thesis investigates the capacity of CDG to evolve genomes that target a specific architecture. Some work has been undertaken towards understanding how to experimentally show this. We concluded that CDG had to be tested against limited resources but also when the target problem increases in complexity. Two different approaches were taken for each case.

For the first case, the approach taken was towards studying the dynamic behavior of the target architectures, cellular automata and boolean networks. The dynamic behavior could involve problems ranging from cycle attractors, point attractors, transient phases and their combinations. Since these dynamic behavior problems would be neutral to the underlying architecture, we had to “challenge” CDG by specifying an additional problem that is best suited for one of the architectures only, i.e., cellular automata. The problem chosen is the *synchronization task* which has been studied to be favorable to cellular automata [22]. By performing this experiment, we could gain a better understanding of CDG and its evolvability regarding the architectures.

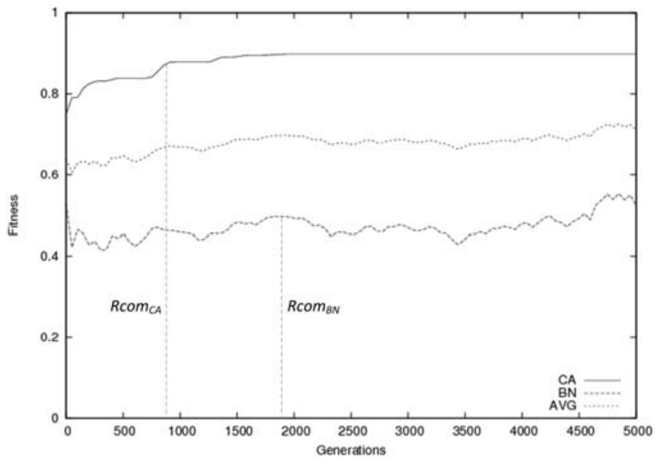
Paper III studies the ability of CDG to evolve computational architectures with limited resource conditions under the same external environment. There are cases where resources are not infinitely available or not available at a given moment. Artificial organisms need to have ways to overcome such problems if they are to continue to evolve at all (much like in nature). For the first time, there were indications that CDG could evolve better, utilizing fewer resources than genomes targeting a specific architecture only.

Figure 3.4 shows the average plot of 10 experiments with a goal of finding a transient phase with a specific cycle attractor. For each individual, a random initial state was created and fed into the architecture. The fitness function gives credit for transients with a maximum size of 10 after which a cycle attractor of size 20 must follow. Point attractors are also taken into account (cycles of 1). Fitness from transient phase and cycle attractors are averaged to give the final fitness score. Figure 3.4(a) shows the average fitness plot over 10 runs for genomes targeting a specific architecture. The CA was able to find a sufficient solution, acquiring a large amount of resources ($Rsep_{CA}$). The RBN was able to find moderate solutions acquiring more than half of the available resources ($Rsep_{BN}$). In the case of CDG in figure 3.4(b), both architectures were able to achieve similar performance as previously, but consumed significantly less resources. The cellular automata reached a max average fitness of 86% at generation 800 ($Rcom_{CA}$), where the random boolean network reached a 55% fitness at generation 1750 ($Rcom_{BN}$). This shows that both genome cases ended up with a similar performance but CDG consumed considerably fewer resources.

Figure 3.5 shows another interesting finding for the synchronization task. The goal here, is to find a CA that given any initial configuration s within M time steps, reaches a final configuration that oscillates between all zeros and all ones, in successive time steps. M, the desired upper bound of synchronization time, is a parameter of the task that depends on the lattice size [22]. For this experiment, we relaxed the rule of all ones and all zeros by introducing a *synchronization threshold*. This means that we may have configurations of zeros or ones up to the threshold limit. In this case, the threshold limit is set to 80%. This means that configurations filled up with 80% zeros or ones make them an acceptable configuration. Figure 3.5(a) shows the average fitness plot over 10 runs for genomes evolved targeting a specific architecture. The CA was able to achieve a maximum average fitness of 40% at generation 2000 ($Rsep_{CA}$), where the random boolean network gave below average solutions (18%) at generation 850 ($Rsep_{BN}$). In the case of CDG (figure 3.5(b)), both architectures required fewer resources achieving the same results compared to the previous case. Specifically, the CA reached the same fitness in generation 750 ($Rcom_{CA}$), where the RBN reached a fitness of 18% in generation 90 ($Rcom_{BN}$). Finally, CDG achieved better overall fitness; the CA reached an average of 60% and the RBN an average of 20% (for the total of the available resources). The synchronization task



(a)

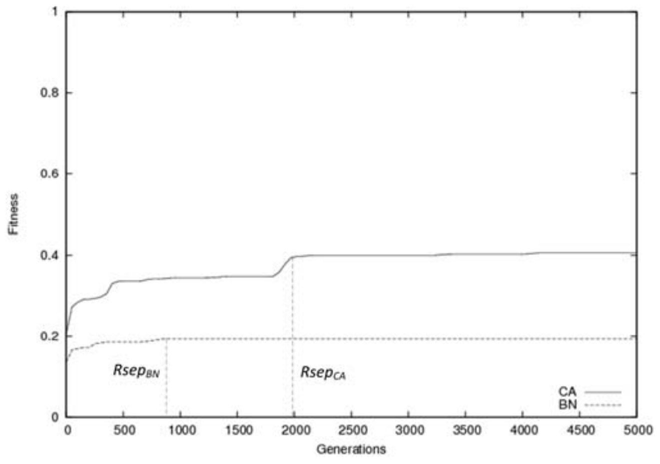


(b)

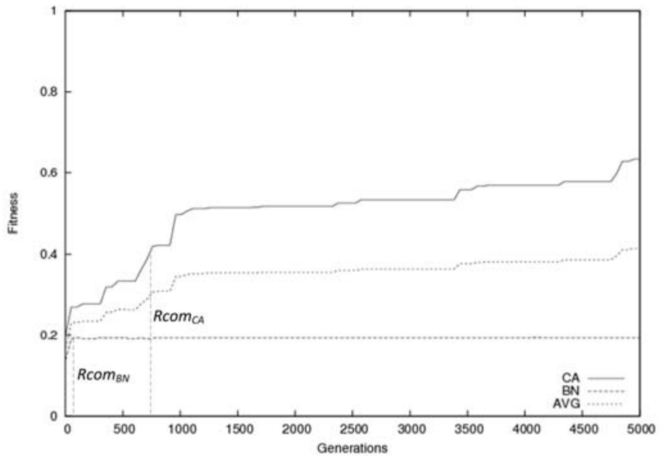
Figure 3.4: Evolvability of CDG under limited resources conditions from paper III. The experiment has a computational goal of finding a transient phase with a cycle attractor. Average fitness plots: (a) Standard genomes, (b) Common Developmental Genomes.

was proven to be particularly challenging regarding the random boolean network finding solutions in both cases.

For the second case, the approach taken was towards identifying a simple complexity metric, enabling us to associate it with a problem instance. A problem instance in this case is defined as a problem with a specific configuration. Considerable work was done studying complexity measures [1, 2, 20, 61, 80, 75], to identify the complexity measure fitting our purpose. We were inspired by *Kolmogorov* complexity definition, adapting it



(a)



(b)

Figure 3.5: Synchronization task: Averaged fitness plot for the different architectures with (a) standard genomes evolved separately, (b) CDG.

in paper II for our purpose.

The next challenge was to define the target problem and incorporate our own definition of complexity, as to be able to associate a problem instance in terms of its complexity, i.e., problem instance X is more complex than Y. As explained in section 2.5, we associated the complexity of a problem instance with the number of cells/nodes required to define the state of the current cell/node. As an example, we mention the no-state memory problem instance where previous states are not taken into consideration, deciding upon the state of the current cell. A slightly more complex instance example, is the 1-state memory prob-

Table 3.1: Rules table for the Stock Market model

Left neighbor	Right Neighbor	Next state
buy	buy	buy
buy	sell	buy
sell	buy	buy
sell	sell	sell

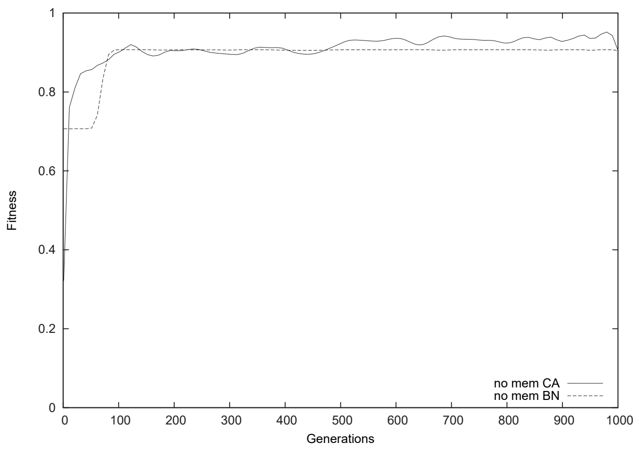
lem. Development rules take into consideration the state of the current cell at previous timestep (state memory 1), in order to decide upon the state of the current cell. Similarly, 2-state memory problem instance takes into account the state of the current cell during the previous two timesteps (state memory 2) and so forth. Several problem instances of the same problem were considered, each having a higher level of complexity, i.e., an instance with no-state memory and instances having previous memory of 1-, 2-, 5- and 10-state. Paper II, studied the ability of CDG to evolve target architectures for problem instances where the level of complexity is increasing along with the size (N) of the underlying architecture (i.e., with lattice sizes $N = 144$, $N = 169$, and $N = 196$).

In this paper, we found that CDG performed as well with genomes that targeted specific architectures. A simple stock market model was the target problem with the status of each cell being determined by looking at the behavior of its two neighbors and the state of the cell in previous timesteps, based on some rules. This problem was selected as the computational task since it provides interesting dynamics phenomena [93]. Two dimensional CA and RBN are used as target architectures. Each cell/node corresponds to a trader that either *buys* or *sells*, in each timestep. The model is based on local interactions and involves simple rules that represent the behavior of the traders. The behavior of a trader X in timestep t is determined by the behavior of its two neighboring traders in timestep $t-1$. The governing trading rules are given in table 3.1. If, for example, the left neighbor buys and the right neighbor sells, the state of the current trader in timestep t is *buy*. The states of the stock market model are translated into binary values for assessing the fitness, that is, $sell = 0$ and $buy = 1$. Fitness is assessed employing the total number of *buy* states in the architecture.

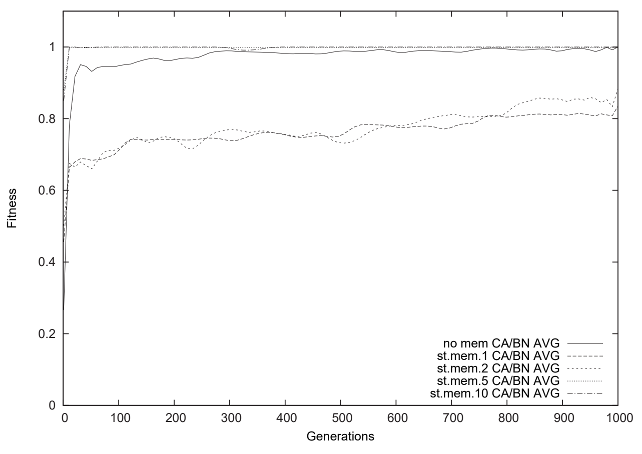
Figure 3.6 shows that better solutions were found on average using CDG regarding $N = 144$ no-state memory problem. For $N = 169$, slightly worse solutions were obtained for the no-state memory problem (figure 3.7). In addition, CDG showed better performance on average, for the $N = 196$ no-state memory problem (figure 3.8). Generally, CDG showed signs of robustness during evolution towards disruptive genetic operations for the 5- and 10-state memory instances. This is evident from the plots of paper II.

After the results obtained in papers II and III, there were indications that CDG could show better evolvability, but only in certain cases. Still, it was not very clear why. The question we focused on, was *why could CDG show better evolvability in some cases?*

Many ideas were set on the table and discussed not only with the supervisor of this thesis but also with other researchers in research forums. Some of the ideas were partially ex-



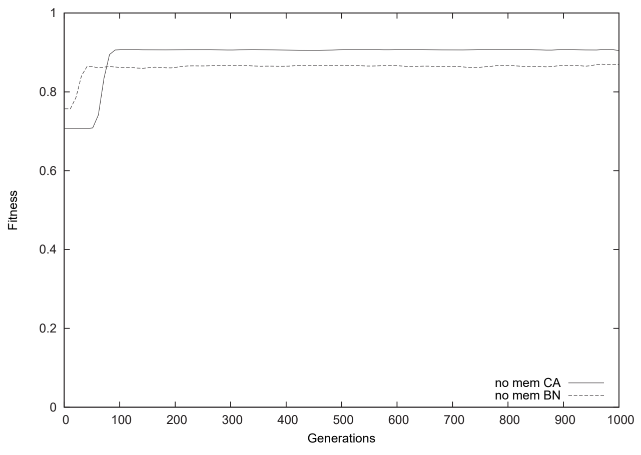
(a)



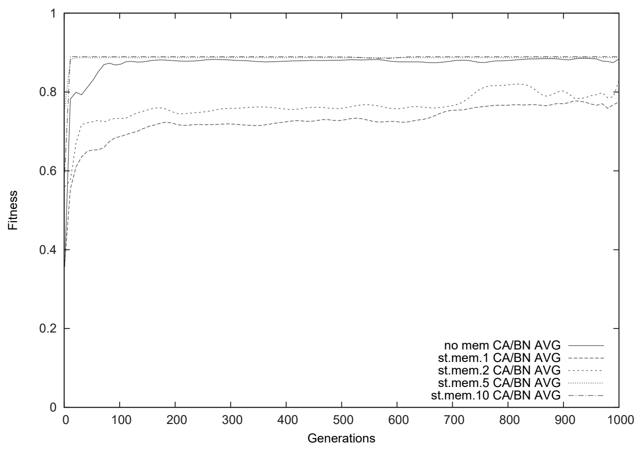
(b)

Figure 3.6: Average fitness plots for a simplified financial market model for $N = 144$ cell architecture: (a) Standard genomes (b) CDG.

plained by the findings but other ideas would need additional experimentation. We started considering questions like: *Is better evolvability a result of the way CDG is constructed?*, *are there other reasons explaining CDG's ability to evolve?*



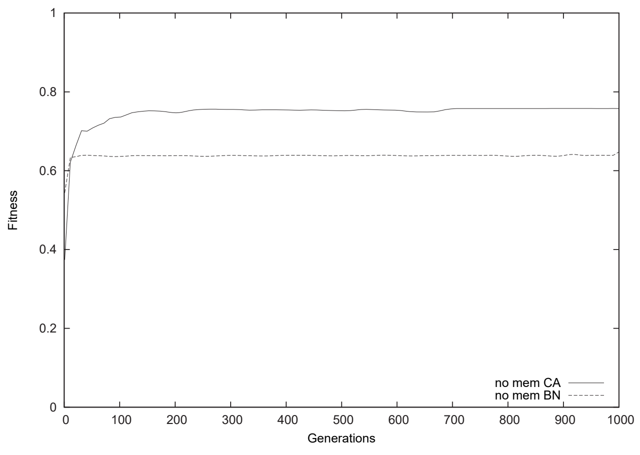
(a)



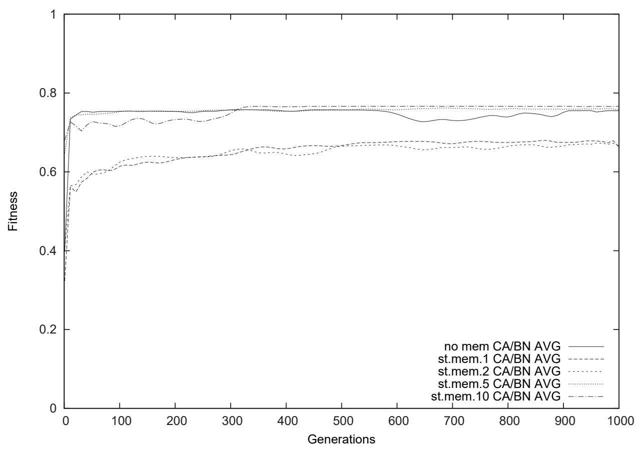
(b)

Figure 3.7: Average fitness plots for the stock market model for N=169-cell architecture: (a) Standard genomes (b) CDG.

To be able to have a better understanding of our findings, we started looking into developmental and evolutionary topics from the literature and tried to identify developmental factors that could enable phenotypic evolution. Arthur [6, 7], inspired us to look closer



(a)



(b)

Figure 3.8: Average fitness plots for the stock market model for N=196-cell architecture: (a) Standard genomes (b) CDG.

into how genetic operators, mutation and selection, affect evolution in CDG as the genome is potentially part of an "orientation-mechanism" of both short- and long term evolution. Additionally, we could investigate how environment affects development. This task was explored in paper IV. We considered only external environment since the behavior of the organism may be distinctly different for the same organism when developed in different environments [118]. So, we introduced different external environments and gave the possibility to the developing organism to adapt its behavior into them.

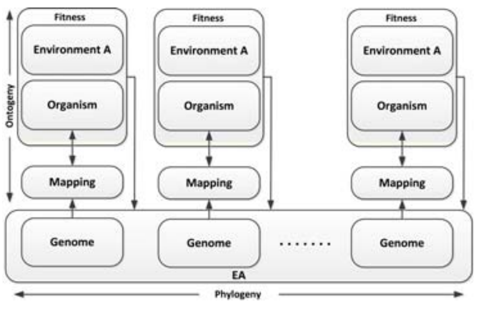
The three different environmental setups are shown in figure 3.9. In the first subfigure, individuals are evaluated based on environment A. In the second subfigure, individuals are still being evaluated in a single environment, but the environment can be different for each evaluation (i.e., environment A, environment B, etc.). In the third subfigure, each individual is being assessed on a set of different environments. The environment in figure 3.9(a), is a single-cell environment where everywhere except in the first node/cell is zero. The first cell/node has initially the value of one. The environments in figures 3.9(b) and 3.9(c) are random. Individuals are being evaluated in dynamic environment, that is, an environment that is changing at each fitness evaluation. Feeding evolution with different external information is expected to affect the genome, intermediate phenotypes and the final phenotype, not only in terms of cell types, but also in terms of connectivity (for the case of random boolean networks).

Paper IV investigates the influence genetic operators (i.e., mutations) may have to the final phenotype under different environmental conditions. That is, we try to identify specific patterns by counting positive, neutral or negative influence a mutation has, over the phenotype in each generation. The fitness value of the new phenotype after the mutation is compared to the fitness of the phenotype from previous generation. If the fitness of the new phenotype is better, then mutation is considered positive. Neutral mutation is the case when phenotypes from current and previous generations have the same fitness. In other words, mutation does not have any effect on individual's fitness. Accordingly, negative mutation has a destructive influence to the phenotype and hence, its fitness value is worse than that of the previous generation.

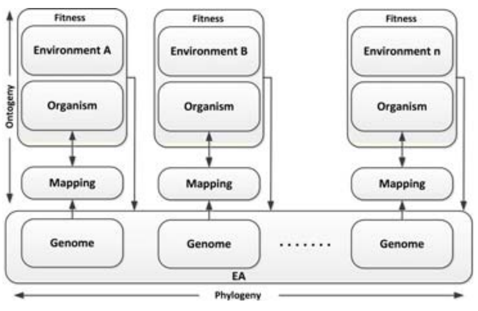
In addition, we aimed to get a better understanding of how development works in CDG by investigating the influence of developmental processes i.e., growth, apoptosis and differentiation, during evolution. We focused on how these processes are deployed under different environmental conditions. The study of the influence had a two-fold target: a. First study the appearance rate of growth, apoptosis and differentiation per individual, evaluated per generation and b. capture the conditional appearance for each process, given a certain process has already appeared in a previous developmental step.

In developmental biology, conditional speciation exists as a way to determine how a particular cell develops into the final cell type (or organism). This type of speciation is a cell-extrinsic process that relies of cues and interactions between cells, or from concentration gradients of morphogens. In this type of speciation, one or more cells from a group of cells with the same developmental potential are exposed to a signal (morphogen) coming outside of the group. Only cells exposed to that signal are induced to follow a different developmental pathway, leaving the rest of the group unchanged.

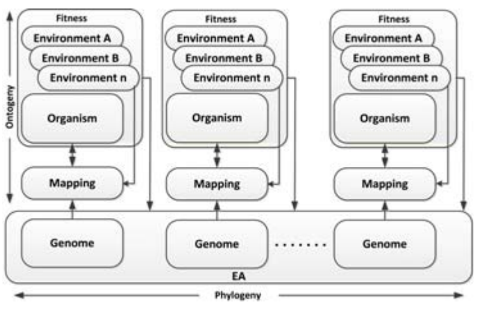
This functionality inspired us to apply it into our developmental model and gave rise to



(a)



(b)



(c)

Figure 3.9: The three different environmental setups from paper IV. (a) Single-cell environment/genome evaluation, (b) Random environment/genome evaluation, (c) Multiple-random environments/genome evaluation.

conditional developmental processes concept. In our aim to get a better insight of how CDG genotypes work, we studied what developmental processes are "exposed to that signal" given a certain developmental process (morphogen) is triggered. For example, we measure the number of a growth process after apoptosis has occurred ($growth|apoptosis$), or after differentiation ($growth|differentiation$). This second exercise will hopefully give us a better indication of the relation between processes during the development phase. Given that we have three different developmental processes and each process can be in

Table 3.2: Conditional Appearance of the Developmental Processes

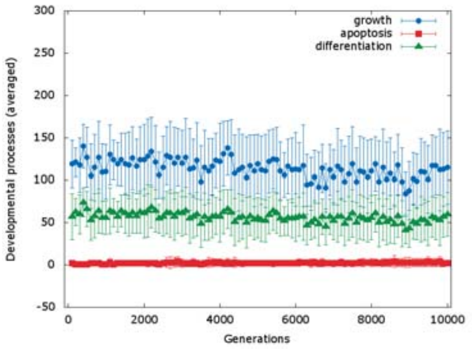
Cond. case 1	Cond. case 2	Cond. case 3
growth|*growth*	*growth*|*apoptosis*	*growth*|*differentiation*
apoptosis|*growth*	*apoptosis*|*apoptosis*	*apoptosis*|*differentiation*
differentiation|*growth*	*differentiation*|*apoptosis*	*differentiation*|*differentiation*

one of the three different conditional cases, we conclude on a total of 9 conditional cases for evaluation (Table 3.2).

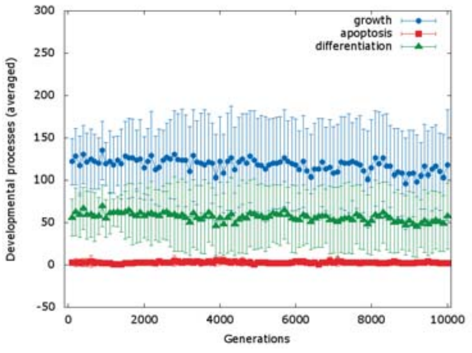
The design and run of experiments took around six months to complete and their analysis an additional two months. The analysis of the influence of mutation over the phenotypes in various external environments was unfortunately not conclusive. Specially in the random and multiple-random environments the number of positive, neutral and negative mutation showed a rather random behavior and unfortunately no conclusions can be drawn. When it comes to the second part of our experiments, the findings obtained with the multiple-random environment were slightly more interesting. Figure 3.10 shows the developmental process plots for multiple-random environment. In figure 3.10(c), common developmental genomes show a higher growth and differentiation ratios compared to development of standard genomes for CA (figure 3.10(a)) and RBN (figure 3.10(b)). Growth and differentiation are important properties of a genome towards evolvability [5], but also indicates that CDG are able to generate phenotypes of quite different structure and function.

The last set of experiments for the second part of the investigation for the multiple-random environment, contribute to the findings above. For the common genome case, there is a higher exploitation of *growth*|*apoptosis* conditional process (figure 3.11(c)). A better exploitation of conditional growth (*growth*|*growth*) was shown by CDG comparing to genomes targeting a specific architecture. This is evident in figure 3.11(c), where conditional growth has clearly increased, compared to the other two cases (figures 3.11(a) and 3.11(b)). In addition, CDG shows a higher conditional growth after apoptosis and a slightly increasing trend. Finally, multiple-random environment was shown to be rather challenging for the standard genomes to cope with and CDG was able to show a better behavior in such demanding environment.

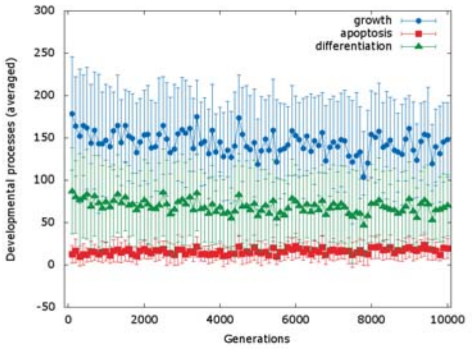
The next step is to look closer into how CDG manage to evolve when the goal changes (adaptation). The next two papers show this. Paper V, explores CDG's capacity to evolve through an adaptation problem. Paper VI extends paper II, by focusing on how CDG exploit the underlying architecture during development and build structure (network morphology) in the final phenotypes.



(a)

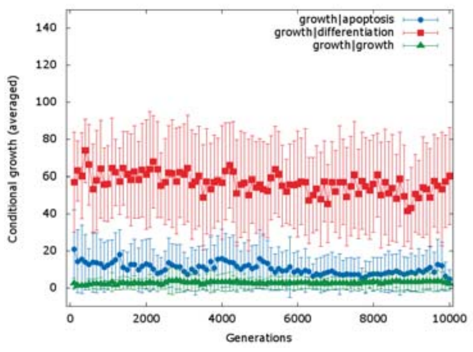


(b)

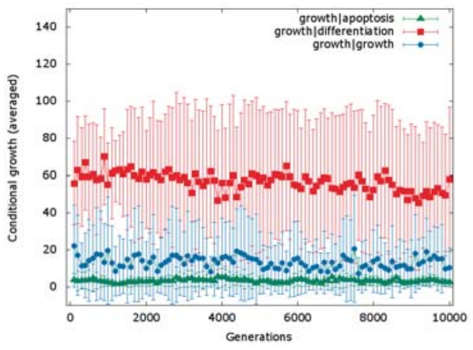


(c)

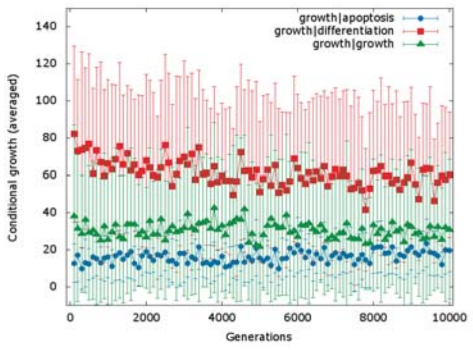
Figure 3.10: Developmental processes for multiple-random environment from paper IV: (a) CA, (b) RBN, (c) Common developmental genome.



(a)



(b)



(c)

Figure 3.11: Conditional developmental processes for multiple-random environment from paper IV: (a) CA, (b) RBN, (c) Common developmental genome.

3.1.5 Common Developmental Genomes - Evolution through Adaptation

After studying evolvability in CDG, it was time to look closer into the capacity of CDG to adapt. In paper V, a more fair fitness evaluation scheme is defined where the computational architecture that performs better gets a fitness incentive of 10%. The paper investigated the ability of CDG to evolve when the goal changes over time (adaptation). CDG was studied under some basic dynamic problems including cycle attractor and a transient phase. The experiment run for 10000 generations. Evolution searches for a cycle attractor of 80 in the first 5000 generations. Then, the target goal changed to a different problem, reaching a transient phase of 100 followed by a cycle attractor of 80. For the experiments, a 6×6 2D CA and a $N = 36$ RBN is used. The size of the lattice is such that there would not 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 structure. The larger the size of the species, the harder it is to visually interpret a structure. Figure 3.12 shows the average fitness evaluation of CDG over all runs. The AVG line shows the average fitness of both species, CA and RBN. The CA line shows the average fitness of the cellular automata only and the RBN line gives the average fitness of the random boolean network.

The first adaptation problem (cycle attractor) is studied in generations 1-5000. During this period, both CA and RBN are able to find fairly good solutions. Round about generation 2000, the effect of the new fitness assignment scheme can be observed. RBN is being credited with an extra 10% incentive due to the fact that shows better evolvability than the CA. This credit assignment in one of the species in CDG, 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 for the first problem. It is not until generation 4600, where an improvement in performance for both species occurs. The second adaptation problem (transient period & cycle attractor) is examined after generation 5000. In generation 5001, the genome still contains genetic information optimized for the previous adaptation problem. 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 at a round about generation 7350, where the performance of the RBN has an impact on the performance of the CA. Though, the solutions provided overall were close to average.

The model managed to find several perfect solutions for the first problem, but also many good solutions for the second. The solutions achieved by the developmental model with the CA, exploited the full CA lattice for both problems studied. Next, we visually analyzed the best phenotypic structures and studied how well CDG exploited the architectures during evolution to build solutions.

Interestingly enough, it was discovered that CDG were able to find very good solutions with rather simple network structures, depending on the problem. Figure 3.13 shows two of the best phenotypes for the random boolean network for the first adaptation problem (fitness=100). 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 RBN, the node numbers indicate their sequential position (next, previous node). The arrow at the end of each connection, indicates the flow of information be-

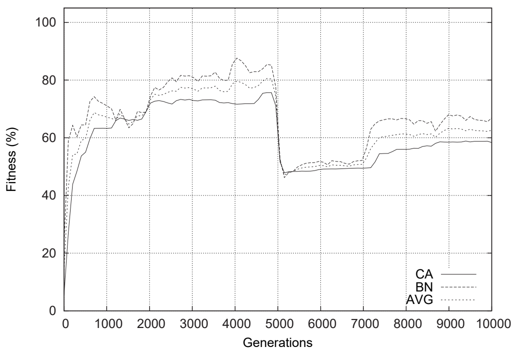


Figure 3.12: Fitness evaluation of common developmental genomes (averaged).

tween the originating and destination nodes. The two random boolean network solutions found, have obviously quite different structure. The solution found in figure 3.13(a), is a network where each node has at least two connections to other nodes and at least one self-connection. On the other hand, the solution of figure 3.13(b), shows a network where one node is rather central to the network (node 1), since the outcome of the majority of the nodes in the network, is dependent on the outcome of node 1. Self-connections in this network are rare since most of the connections point to a different node than the originating one.

Some of the near-perfect solutions given by evolution (fitness > 80), include networks with a rather small number of nodes. Solutions with a fitness score $>= 80$, included networks with a total number of 6-10 nodes. All perfect solutions (fitness 100), included networks having the maximum number of nodes allowed by the architecture ($N = 36$). In other words, the smallest "perfect" solution found were networks with $N = 36$ nodes. This suggests that in the proposed developmental model, development tries initially to search for solutions that involve fewer number of nodes. Later, the model tries to find solutions incorporating additional nodes in the network.

Similarly, figure 3.14 shows two of the best RBN solutions for the second adaptation problem. Both structures include a rather small number of nodes ($N = 6$) with some nodes having one self-connection. The final phenotypes obtained by both problems (figures 3.13 and 3.14), indicate an ability of CDG to create different structures and to find solutions well adapted to the problem targeted. The reason for choosing cellular automata and random boolean networks as target computational architectures was due to the fact that they have similar functional and structural properties. Also, the "multi-chromosome" concept allows the two L-system grammars to develop both cells/nodes and connectivity using randomly-generated grammar rules.

One can argue that this concept allows too much "design in the solution". This argument could be relevant in case where based on the problem, the proposed approach could decide which computational architecture would be best suited and therefore evolve. In our case this is not true; the approach will evolve both computational architectures in any case. Still, this option can be considered as extension of this research.

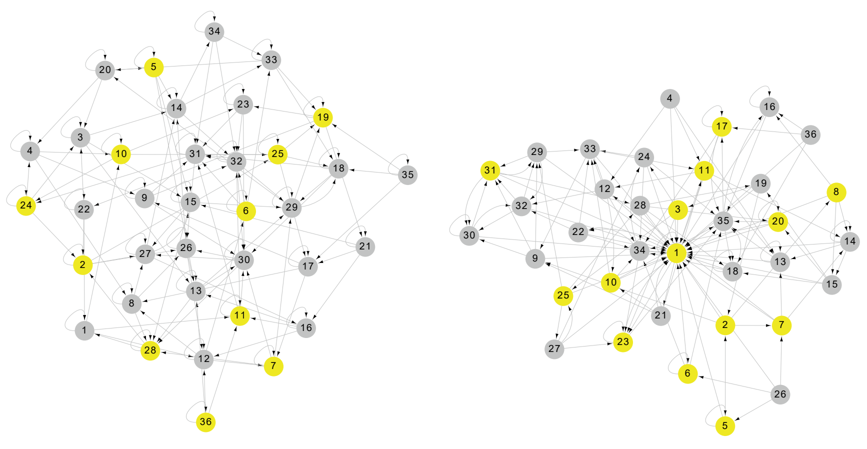

(a) (b)

Figure 3.13: Exploitation of RBN architecture from paper V. Two of the best evolved RBN structures for the first adaptation problem showing also the nodes that are computing.

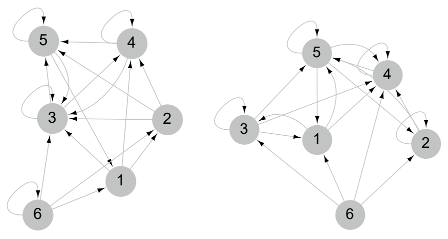

(a) (b)

Figure 3.14: The best evolved RBN structures for the second adaptation problem from paper V.

Next, we studied how CDG managed to exploit the underlying architectures, in order to build the final solutions. To achieve this, we focused 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 positive (i.e., fitness increase), or a neutral (i.e., equal fitness) impact to the fitness. Cells/nodes performing rarely any computation ($< 30\%$ of evolutionary time) are considered static (*quasi-static* seems to be a better definition, since those cells are performing but their state changes at a slower rate enough to keep the network functioning). Quasi-static cells were not tested in terms of their significance of their contribution to the total computation. Cells/nodes that compute more than 30% of the time are considered as active and that contributes to the final solution.

Figure 3.13 show the two best evolved networks for the first problem. The nodes of the networks that are computing are shown in dark gray color. Figure 3.13(a) indicates that approximately 55% of the network is computing with the rest 45% of the network being static. Similarly, figure 3.13(b), shows that a total of approximately 70% of the network includes nodes that compute more than 30% of the time. The RBN solutions found, give a quite different picture; the first network solution involves more self-connections/node than the solutions found for the second problem. Self-connections generally contribute to a network's neutrality and may partially have an effect on the amount of network nodes that are actually contributing with computation. Regarding the second adaptation problem, all nodes in the network were found to be computing and no static nodes were observed (figure 3.14).

Similarly, figure 3.15 shows two 2D-CA of size 6×6. The light-gray colored cells indicate cells that compute more than 30% of the time. As such, a total of 70% approximately of the CA cell structure, is actually computing during evolution. Similarly, the dark-gray colored cells indicate cells that are quasi-static, constituting a total of 30% of the structure.

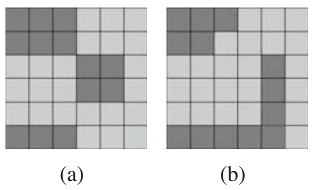


Figure 3.15: Exploitation of CA architecture from paper V. The number of CA cells that compute (light gray) versus quasi-static cells (dark grey). (a) First adaptation problem, (b) Second adaptation problem.

3.1.6 Studying Network Morphology in Common Developmental Genomes

Based on the interesting results of paper V, it was time to focus again on the capacity of CDG to develop and evolve network structures in more scalable conditions, targeting problems of increasing complexity. We linked the problems analyzed in paper III with paper VI. The definitions of problem instances and complexity are defined in section 3.1.4 of paper III.

The motivation here is to understand how CDG manage to develop and evolve network structures. This is supported by a need to understand what dynamic conditions apply during evolution and whether architecture size affects the ability of CDG to evolve. One potential possibility could be that the model builds different network structures for "less complex" problem instances for smaller architecture sizes. Another potential possibility could be that CDG exploit architectures similarly given any problem instance, or perhaps the size of architecture does not really affect the capacity of CDG to evolve. Therefore, the goals of the experiments of paper VI are: (i) to study the capacity of the model to evolve for problem instances with varying architecture size and complexity, and (ii) to study how the developmental model builds network structure (morphology) for best phenotypic solutions.

A set of 10 experiments of 1000 generations for each problem instance and architecture size was initially run. The problem target was a simple stock market model (section 3.1.4). For each problem, instances of different configurations are considered; an instance with no-state memory and instances of 2-, and 5-state memory for $N = 36$, $N = 64$, and $N = 100$ architecture size. A random initial environment is fed into the architectures. This preliminary experiment resulted in ten final phenotypes. For each architecture size, we chose the two best and two worst solutions, based on their fitness. Since the focus of this work is to investigate how the model builds network structure, phenotypes yielding a distant fitness one should anticipate diverse genetic information in genotypes. Evolving further these genotypes will presumably result in very different network structure morphologies.

After a preliminary run selecting the best and worst phenotypes for each problem instance, we studied how common developmental genomes build the network morphology for the solutions. *Network morphology*, studies network structures that have certain characteristics. These characteristics are based on i. the developmental dynamics and ii. changes in phenotypic structure. Since any of these factors may have an impact to the "local" fitness of the individual, we record and analyze the best only individuals, by identifying the following conditions:

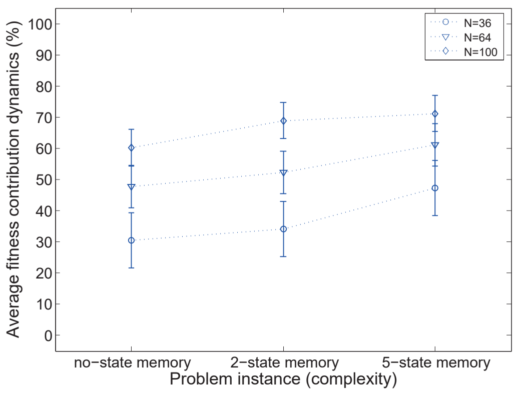
1. *Positive* impact to the fitness. For example, a new cell/node added or a different connection pattern may assign a greater fitness score than previously.
2. *Neutral* impact to the fitness (neither positive nor negative). For example, a cell/node with differentiated functionality or a newly deleted connection between two nodes has no impact to the fitness.
3. *Static* impact to the fitness. Cell/node functionality or connectivity remain static during evolution, i.e., no changes occured.

Figure 3.16(a) shows that the number of cells/nodes contributing positively to fitness is generally increasing as the problem instance does. Similarly, as the size of architecture increases, the number of cells/nodes contributing to the fitness increase as well. The results regarding $N = 100$, follow the other two trends, for $N = 36$ and $N = 64$, reaching a maximum for the 2-state memory problem. Unfortunately, the trend flattens out for the 5-state memory problem. Figure 3.16(b) shows that the amount of neutrality for $N = 36$ and $N = 64$ is almost constant regarding all problem instances. For $N = 100$, the average neutrality decreases as the problem instance becomes more complex. Generally speaking, the number of nodes that are neutral is low, for all problem instances. Figure 3.16(c) shows that the smaller the architecture size, the larger number of static nodes are involved during development, regarding all problem instances. For $N = 100$, the number of static nodes remains almost constant across all problem instances.

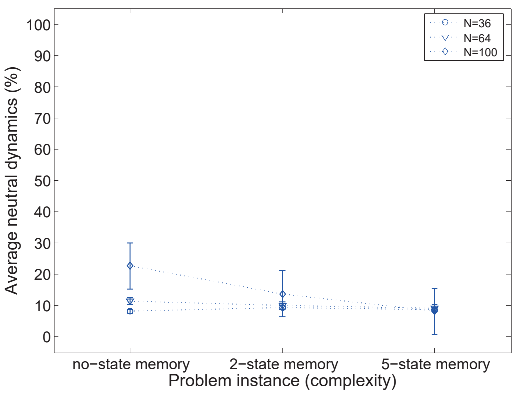
Although the target problems of paper VI were very different in nature, CDG showed an ability to explore the solution space by finding phenotypes whose network structure ranged from simplified to complex. Simple solution is considered a network structure with only a few nodes where complex solution are structure including as many nodes as the network allows.

Figure 3.17 is an example of emergent behavior for one of the best solutions for the no-state memory problem (for definition of -state memory problems, see 3.1.4), a $N = 64$ random boolean network. All networks shown in this figure represent perfect solutions, that is, individuals with the best fitness score. In generation 1, the phenotypes (network structures) generated by CDG initially employs a large number of nodes. As early as in generation 2, CDG managed to achieve solutions with a considerably smaller number of nodes ($N = 52$). Smaller network solutions were obtained by CDG until generation 423. From that point on, CDG started to incorporate again more nodes, reaching the maximum available number of nodes for the network, $N = 64$. Networks of figure 3.17 are plotted in Cytoscape [98] (Dynnetwork plugin), using the force-based graph layout algorithm [60].

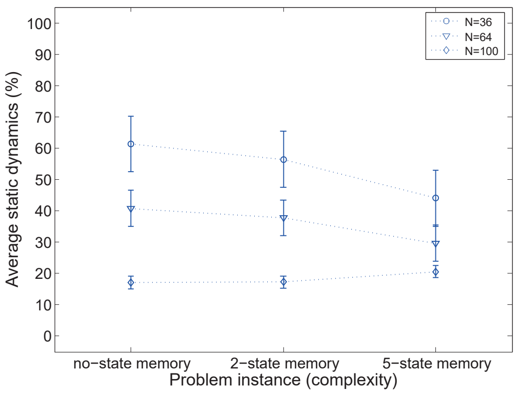
In papers V and VI, the solutions provided by CDG showed emergence under different conditions (problem complexity and architecture size). That gives an indication of the ability of CDG to explore the solution space, providing results whose structures range from simple to complex. The example of figure 3.17 indicate that CDG may exhibit emergence since the representation was not designed to demonstrate such behavior.



(a)



(b)



(c)

Figure 3.16: Results of the three dynamic conditions for the problem instances during evolution from paper VI.

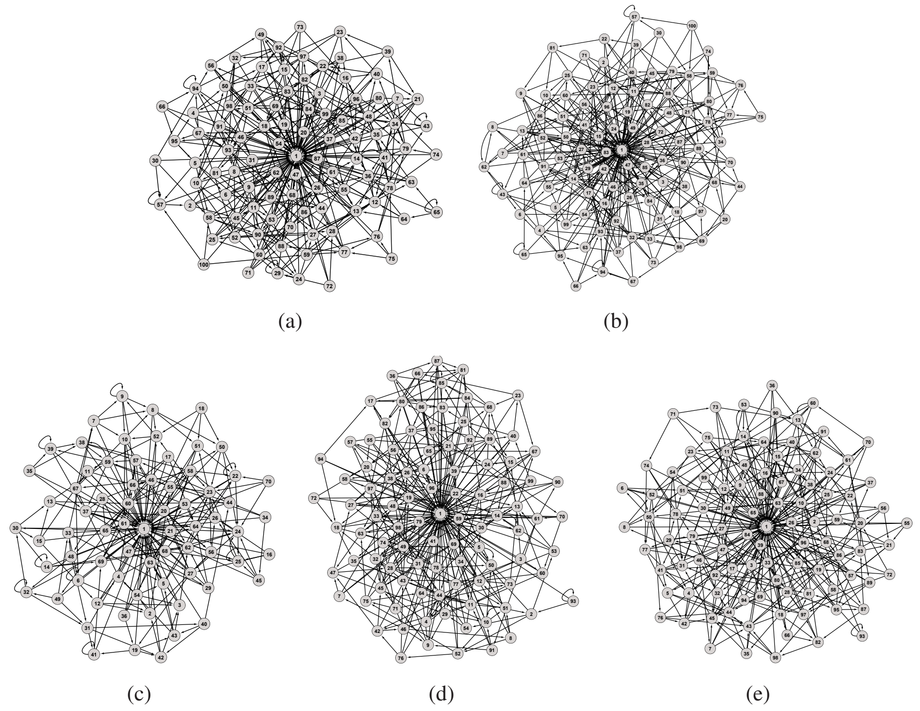


Figure 3.17: Evolution stages for best no-state memory, $N = 64$ random boolean network from paper VI. (a) Initial solutions involve phenotypes that incorporate more nodes into the final solution (gen. 1). (b) More simple solutions are found between generations 2-423 ($N = 52$), (c)-(e) At later stages, CDG found solutions including once again the maximum number of available nodes. Sample phenotypes are shown from generations 500, 999 and 1000 ($N = 64$).

3.2 List of Publications

Papers Included in Thesis

I - K. Antonakopoulos and G. Tufte. A Common Genetic Representation Capable of Developing Distinct Computational Architectures. In *IEEE Congress on Evolutionary Computation*, 2011.

II - K. Antonakopoulos and G. Tufte. Is Common Developmental Genome a Panacea Towards More Complex Problems? In *13th IEEE International Symposium on Computational Intelligence and Informatics*, 2012.

III - K. Antonakopoulos and G. Tufte. On The Evolvability of Different Computational Architectures Using a Common Developmental Genome. In *4th International Conference on Evolutionary Computation Theory and Applications*, 2012.

IV - K. Antonakopoulos and G. Tufte. Investigation of Developmental Mechanisms in Common Developmental Genomes. In *7th International Conference on Bio-Inspired Models of Network, Information, and Computing Systems*, 2012.

V - K. Antonakopoulos. Common Developmental Genomes - Evolution through Adaptation. In *16th European Conference on the Applications of Evolutionary Computation (EvoApps)*, 2014.

VI - K.Antonakopoulos. Studying Network Morphology in Common Developmental Genomes. In *IEEE Systems, Man and Cybernetics*, 2014.

3.2.1 Other Papers

- K.Antonakopoulos and G.Tufte. Possibilities and Constraints of Basic Computational Units in Developmental Systems. In *Norsk Informatikk Konferanse (NIK)*, 2009.
- K. Antonakopoulos. Studying Common Developmental Genomes in Hybrid and Symbiotic Formations. In *7th International Conference on Genetic and Evolutionary Computing*, 2013.

3.3 Paper Abstracts

This section presents abstract for each paper included in this thesis. In addition, retrospective comments are given for each paper, where necessary.

3.3.1 Paper I

A Common Genetic Representation Capable of Developing Distinct Computational Architectures

Abstract
A big challenge in the area of developmental and generative systems, is the design of a method for building complex systems with specific structural and/or functional properties. Most developmental models target specific computational architectures or structures of strictly defined building blocks, in both cases developmental models have strong connection to the target computational architecture/phenotype structure. In this work we seek a common developmental model that can target different architectures but also to find a common genetic representation that can include information that enables such a developmental model. The computational architectures with sparsely connected computational elements considered herein are cellular automata and boolean networks. The experiments study the evolvability of the genetic representation and prove that it is able to build stable structures for distinct computational architectures.

Retrospective View
This paper includes the main work and the design principles for common developmental genomes. It actually manages to answer some of the initial challenges for this thesis. It describes a common developmental approach that can handle the target architectures, cellular automata and random boolean networks. It defines a common way of representing genetic information that can be exploited by the developmental process, so that it enables to develop both architectures. Finally, it manages to integrate target architectures and genome representation in a way that enables us to evolve target architectures; structure is the target here, ensuring that different structures of different architectures can be achieved.

3.3.2 Paper II

Is Common Developmental Genome a Panacea Towards More Complex Problems?

Abstract
The potentiality of using a common developmental mapping to develop not a specific, but different classes of architectures (i.e., species), holding different structural and/or computational phenotypic properties is an active area of research in the field of bio-inspired systems. To be able to develop such species, there is a need to understand the governing properties and the constraints involved for their development. In this work we investigate the ability of common developmental genomes to evolve more than one specie (i.e., computational architecture), towards problems with increasing complexity. The architectures considered as different species were cellular automata and boolean networks and the problem studied was a simple financial market model over various architecture sizes. We considered problem instances of the same problem, each having a higher level of complexity, i.e., an instance with no state memory and with a previous memory of 1-, 2-, 5-

and 10-state. The results show that the common developmental genome was able to find better results for certain cell architectures sizes.

Retrospective View
In this paper, we defined a rudimentary metric to specify complexity in the problems targeted. Our definition was based on *Kolmogorov* complexity. Many may argue about our choice, but *Kolmogorov* complexity is a measure that can be used on individual, finite objects [67] and one can associate the complexity of an object (phenotype) with the length of the shortest description for the object. For an unambiguous measure of an object's description length, the description length was defined as the length of the shortest program that generates the object on a fixed, universal Turing machine [68]. This essentially implies that an object (cell/node) is as complex as its description length. The shortest description is defined as the amount of environmental information necessary to specify the cell/node.

Errata

- At Section VI.C, 2nd paragraph, it should read "(...figures 1(a) and 1(f))" instead of "(...figures 1(a) and 2(f))".
- At reference [19], the author should read "W. Arthur".

3.3.3 Paper III

On The Evolvability of Different Computational Architectures Using a Common Developmental Genome

Abstract
Artificial organisms comprise a method that enables the construction of complex systems with structural and/or computational properties. In this work we investigate whether a common developmental genome can favor the evolvability of different computational architectures. This is rather interesting, especially when limited computational resources is the case. The commonly evolved genome showed ability to boost the evolvability of the different computational architectures requiring fewer resources and in some cases, finding better solutions.

Retrospective View
In section 5 of this paper (Conclusion and Future Work), it is stated that "...CDG may have a positive influence in directing evolution and pushing the developmental system in phenotypic directions where it would have been impossible to achieve with ordinary genomes". Although it may sound impetuous, it nevertheless projects our overall understanding of the encouraging results we had obtained up to that point. Soon after the encouraging results, the fact that CDG was not able to evolve in every case, it became a pool of discussion and further exploration; something that eventually limited our expectations. Concluding, we point out ways to further our research, that is (a) potential relations between mutation and selection in the underlying genetic process, and (b) understand the

ontogenetic directionality (i.e., developmental dynamics) of CDG. This becomes the goal for paper IV.

Errata

- At references section, author "Wallace, A." should read "Arthur, W.".

3.3.4 Paper IV

Investigation of Developmental Mechanisms in Common Developmental Genomes

Abstract
The potentiality of using a common developmental mapping to develop not a specific, but different classes of architectures (i.e., species), holding different structural and/or computational phenotypic properties is an active area of research in the field of bio-inspired systems. To be able to develop such species, there is a need to understand the governing properties and the constraints involved for their development. In this work, we investigate how common developmental genomes influence evolution and how they push the developmental process in directions where it would have been impossible to achieve with ordinary genomes. Relations between mutation and evolution along with a comprehensive study of developmental mechanisms involved in development are worked out. The results are promising as they unveil that common developmental genomes perform better in more complex and random environments.

Retrospective View
In this work, we explore how genetic operators (i.e., mutation) affect evolution (i.e., selection) in common developmental genomes. Second, we study whether development and dynamics of common developmental genomes prescribe a certain pathway for evolution. Third, we focus on the external environment to be able to assess the capacity of CDG in more complex environments. The first and third tasks were sufficiently studied in this paper since the results obtained leave little room for argumentation. The second task was a harder one and was more difficult to answer. The findings of this paper, were investigated but not documented in a satisfactory way.

3.3.5 Paper V

Common Developmental Genomes - Evolution through Adaptation

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.

3.3.6 Paper VI

Studying Network Morphology in Common Developmental Genomes

Abstract

In most Evo-devo (Evolutionary Developmental) systems, genotypes are developed and evolved towards a structural or computational goal utilizing some kind of computational architecture (i.e., structures made of connected elements that may compute). Exploiting a common genotype to develop and evolve different classes of computational architectures towards a common goal has previously been successfully implemented, through common developmental genomes. In this work, we focus at how common genomes exploit the underlying architectures during development and build structure (network morphology) in phenotypes for different problem instances and architecture sizes. Common developmental genomes showed an ability to exploit the size of the architecture by actively involving a larger number of nodes/cells while managed to maintain a small number of neutral and static parts in the evolved structures.

Retrospective View

Papers V and VI gave us the opportunity to focus on evolved network structures. As cellular automata's structure is known beforehand, it made more sense to bring random boolean network under the spotlight and a need to investigate further. In paper VI, CDG showed evidence of emergence since the solutions achieved included a smaller number of nodes than the total number of nodes allowed by the architecture. This behavior was neither "programmed" in the design nor induced in the fitness function. CDG also provided solutions (network structures) with different structural properties, for the same problem.

Chapter 4

Concluding Remarks

Only two things are infinite, the universe and human stupidity, and I'm not sure about the former.

Albert Einstein

4.1 Conclusion

The work in this thesis has addressed the challenge of designing a model, called Common Develomental Genomes, that is capable of developing and evolving classes of computational architectures. The approach was first to study suitable architectures for development with such a model. Next step, was trying to describe a common developmental approach that could handle the targeted architecture and define further how genetic information could be exploited by the developmental process, resulting in developing these architectures. Last step, was to identify a suitable genome representation ensuring that different structures can be achieved.

The architectures chosen and worked out throughout this thesis were cellular automata and random boolean networks. The reason for choosing those particular architectures was that they have similar structural and functional properties since random boolean networks are considered a generalization of cellular automata. The characteristic of our proposed approach is that genome representation and genetic information is common for all targeted architectures. Therefore, the same information is being used for developing both CA and RBN.

Section 1.5 formulated the following main research question for this thesis:

Is it possible to design an EvoDevo system that can achieve evolvability targeting a class of architectures?

The current thesis can only answer partially in a positive way to the question, since compromises have been made. Papers I, II, III and IV are focusing on this question. First, CDG involve rather simplified developmental processes. Second, CDG had to be experimentally tested further to cover a larger range of problems. In addition, the genetic representation needs to be adapted and tested to incorporate other computational architectures as well i.e., artificial neural networks or other biologically plausible models like *gene regulatory networks*. In this case, we can forsee a better picture of the result, after these pointer have been fulfilled.

Section 1.5 also formulated several more specific questions.

1. *To what extent can a common developmental mapping be used to evolve classes of architectures?*

   In this thesis, we were able to show a possible design and use of a common developmental mapping, capable of developing and evolving cellular automata and random boolean networks. Several types of problems have been studied: a. basic dynamic behavior problems (transient phase/cycle and point attractors with variations in papers I, III, IV and V), b. problems targeting a structural goal (paper I), c. simulation of a simplified stock market model (papers II and VI) and d. a task synchronization problem (paper III). In addition, CDG have been experimentally tested against problems with increasing complexity (using our own definition of complexity), with varying architecture sizes (papers II and VI). We can conclude, that common developmental genomes approach can be used to evolve classes of architectures in limited cases. It was not possible to draw certain generalized conclusions since more experimentations were required. Since CDG is considered an approach or a developmental model, it would be interesting to apply it on real-world problems, such as, robotic control, manufacturing, hardware design, adaptive control, fault detection, decision support and pattern recognition.

2. *How and to what extent does the environment affect the development and evolution of common developmental genomes?*

   In this work, we considered only external environments, in terms of initial conditions for the computational architectures. All papers presented in this thesis, employed an external environment serving as input (to the genome, intermediate phenotypes and final phenotype), not only in terms of cell types but also in terms of connectivity (for the case of random boolean networks). Paper IV, introduced different external environments (single, random and multiple random) and studied processes of development. CDG showed a capability to cope with more complex environments but no general conclusions can be drawn from the current findings.

3. *What kind of phenotypes are obtained and what are their structural characteristics?*

   Final phenotypes and their structure were studied in papers I, V and VI. In paper I, a first attempt to verify that the proposed developmental mapping is capable of evolving phenotypes with certain cell processes is outlined but also it was shown to be able of building stable structures (i.e., reach a point attractor). Paper V, studied network structures of evolved phenotypes in an adaptation problem. It was found that CDG produced quite different phenotypes (structures) for the same problem. Specifically, it obtained optimal solutions that involved varying number of nodes during evolution. Solutions with emergent behavior were evident in paper VI. Generalized conclusions would again be perilous to draw from current findings.

4.2 Research Contributions

The following are the main contributions of this thesis:

1. A model able to develop and evolve a class of sparsely-connected network architectures using a common genetic representation.

2. The research has contributed to an understanding of how such a model can be applied through an Evo-devo approach for the target architectures.

3. Through the CDG model, a step closer towards adaptive, scalable systems has been taken.

The following are other contributions:

- Identify what cellular processes can be included in such a model, so as to enable it to develop different computational architectures.

- Identify a way to represent genetic information that can be exploited by a common developmental process, so as to develop classes of computational architectures

4.3 Limitations

The genetic information is represented in the genome through the chromosome approach. Each chromosome represents the different information required by architectures to be built. The first chromosome, node/cell information, is common for all architectures evolved by CDG. This poses an inherent limitation to the types / functionality of the nodes / cells that are represented in the genome for each architecture which may not always be beneficial.

In terms of computational time, CDG experiments took approximately twice the time required to evolve a cellular automata or a random boolean network. This is a important limiting factor when designing complex experiments and high performance is anticipated.

4.4 Future Work

This section suggests ways of extending research in this thesis.

- The main work of paper I can be extended to include artificial neural networks (ANN) as an additional architecture. This can easily be done by employing an extra chromosome in the genetic representation, to map the weights of the ANN edges.
- More research should be carried out to understand if neutrality is a property of CDG; evident in problems having both structural and/or functional targets. Also, it would be of great interest to study whether static/neutral parts of a network structure would directly impact the robustness of the evolved system.
- More research needs to be done to improve the limitation of the first chromosome described in section 4.3.
- A first step has been taken towards merging two computational architectures into one, integrated biological organism (section 3.2.1), to understand how different species can be evolved in close-association forming a *hybrid* architecture. This hybrid architecture was developed and evolved with the same genetic information.
- Study to what extent it can be guaranteed that the RBN mapping is not actually closely similar to an equivalent CA mapping.
- In the same paper (section 3.2.1), a symbiotic model was studied (based on CDG), with two types of associations and reproduction rates. This work can be easily extended to understand how can CDG evolve hybrid architectures, with different symbiotic relationships and reproduction rates. References [16, 112, 86], may act as an inspiration to futher the work.
- Real-world problems need to be worked out and applied using CDG to understand if there are any benefits by this approach. See 1.5 for a list of potential real-world problems.

Bibliography

[1] Adami, C. What is complexity ? *Bioessays*, 12(24):1085–94, 2002.

[2] Adami, C., and Cerf, N.J. Physical complexity of symbolic sequences. *Physica D*, 137:62–69, 2000.

[3] M. Aldana. Boolean dynamics of networks with scale-free topology. *Physica D: Nonlinear Phenomena*, 185(1):45 – 66, 2003.

[4] Antonakopoulos, K., and Tufte, G. Possibilities and constraints of basic computational units in developmental systems. In *Norsk informatikkonferanse, NIK*, pages 73–84. Tapir akademisk forlag, 2009.

[5] Antonakopoulos, K., and Tufte, G. On the evolvability of different computational architectures using a common developmental genome. In *4th International Joint Conference on Computational Intelligence (ECTA 2012)*, pages 122–129. SciTePress, 2012.

[6] Arthur, W. The concept of developmental reprogramming and the quest for an inclusive theory of evolutionary mechanisms. *Evolution & Development*, 2(4):243–251, 2000.

[7] Arthur, W. Developmental drive: an important determinant of the direction of phenotypic evolution. *Evolution & Development*, 3(4):271–278, 2001.

[8] Fernando J Ballesteros and Bartolo Luque. Random boolean networks response to external periodic signals. *Physica A: Statistical Mechanics and its Applications*, 313(3ï£¡4):289 – 300, 2002.

[9] Bastolla, U., and Parisi, G. Relevant elements, magnetization and dynamical properties in kauffmann networks: A numerical study. *Physica D*, 115(3-4):203–218, 1998.

[10] Bentley, P. J., and Kumar, S. Three ways to grow designs: A comparison of embryogenies for an evolutionary design problem. In *Genetic and Evolutionary Computation Conference (GECCO '99)*, pages 35–43, 1999.

[11] Bidlo, M., and Vaï£¡ï£¡cek, Z. Investigating gate-level evolutionary development of combinational multipliers using enhanced cellular automata-based model.

In *Congress on Evolutionary Computation(CEC2009)*, pages 2241–2248. IEEE, 2009.

[12] Bilke, S., and Sjunnensson, F. Stability of the kauffmann model. *Physica Review E*, 65:016129, 2002.

[13] Bixler, G.D. and Bhushan, B. Fluid drag reduction with shark-skin riblet inspired microstructured surfaces. *Advanced Functional Materials*, 23(36):4507–4528, 2013.

[14] J. C. Bongard and R. Pfeifer. Repeated Structure and Dissociation of Genotypic and Phenotypic Complexity in Artificial Ontogeny. In L. Spector, editor, *GECCO 2001: Proceedings of The Genetic and Evolutionary Computation Conference*, pages 829–836. Morgan Kaufmann publishers, 2001.

[15] Bongard, J.C., and Pfeifer, R. chapter evolving complete agents using artificial ontogeny. In *Morpho-functional Machines: The New Species (Designing Embodied Intelligence),*, pages 237–258, 2003.

[16] Bull, L. Artificial symbiogenesis and differing reproduction rates. *Artificial Life*, 16(1):65–72, 2010.

[17] Cedeno, W. and Agrafiotis, D.K. Combining particle swarms and k-nearest neighbors for the development of quantitative sturcture-activity relationships. In Phillip A. Laplante, editor, *Biocomputing*, pages 43–53. Nova Science Publishers, Inc., Commack, NY, USA, 2003.

[18] Chiong, R. and Weise, T. and Michalewicz, Z., editor. *Variants of Evolutionary Algorithms for Real-World Applications*. Berlin/Heidelberg: Springer-Verlag, 2012.

[19] M. Cortesi, G. Sangiovanni, and F. B. Zazzera. Embodied and evolved dynamical neural networks for robust planetary navigation. In *2007 IEEE/ASME INTERNATIONAL CONFERENCE ON ADVANCED INTELLIGENT MECHATRONICS, VOLS 1-3*, IEEE ASME International Conference on Advanced Intelligent Mechatronics, pages 907–912. IEEE Robot & Automat Soc; ASME, Dynam Syst & Control Div; IEEE Ind Elect Soc, 2007. IEEE/ASME International Conference on Advanced Intelligent Mechatronics, Swiss Fed Inst Technol, Zurich, SWITZERLAND, SEP 04-07, 2007.

[20] Crutchfield, J., and Young, K. Inferring statistical complexity. *Physica Review Letters*, 2(63):105–108, 1989.

[21] Darwin, C. *The Origin of Species*. Dent Gordon, London, 1973.

[22] Das, R. and Crutchfield, J.P. and Mitchell, M. and Hanson, J.E. Evolving globally synchronized cellular automata. In L. J. Eshelman, editor, *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 336–343. Morgan Kaufmann Publishers Inc., 1995.

[23] Dean, B. and Bhushan, B. Shark-skin surfaces for fluid-drag reduction in turbulent flow: a review. *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 368(1929):4775–4806, 2010.

[24] Deb, K. Evolutionary algorithms for multi-criterion optimization in engineering design, 1999.

[25] Deb, K. *Multi-Objective Optimization using Evolutionary Algorithms*. Wiley-Interscience Series in Systems and Optimization. John Wiley & Sons, Chichester, 2001.

[26] Deloukas, P. and Schuler, G. and Gyapay, G. and Beasley, E. and Soderlund, C. and Rodriguez-Tome, P. and Hui, L. and Matise, T. and McKusick, K. and Beckmann, J. and Bentolila, S. and Bihoreau, M. and Birren, B. and Browne, J. and Butler, A. and Castle, A. and Chiannilkulchai, N. and Clee, C. and Day, P. and Dehejia, A. and Dibling, T. and Drouot, N. and Duprat, S. and Fizames, C. and and Bentley, D. A physical map of 30,000 human genes. *Science*, 282(5389):744ï£¡746, 1998.

[27] Derrida, B., and Flyvbjerg, H. The random map model: A disordered model with deterministic dynamics. *J. Physique*, 48:971–978, 1987.

[28] Doursat, R., Sayama, H., AND Michel, O. Morphogenetic engineering. In *Toward Programmable Complex Systems*, Understanding Complex Systems, pages IX,517. Springer-Verlag Berlin Heidelberg, 2012.

[29] Drake, A. E. and Marks, R. E. *Genetic Algorithms and Genetic Programming in Computational Finance*, chapter Genetic Algorithms In Economics and Finance: Forecasting Stock Market Prices And Foreign Exchange — A Review, pages 29–54. Springer US, Boston, MA, 2002.

[30] P. Eggenberger. Evolving morphologies of simulated 3d organisms based on differential gene expression. In *4th European Conference on Artificial Life*, pages 205–213. MIT press, 1997.

[31] Eiben, A. E. and Smith, J. E. *Handbook of Memetic Algorithms*, chapter Evolutionary Algorithms, pages 9–27. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[32] D. Federici. Evolving a neurocontroller through a process of embryogeny. In *Simulation of Adaptive Behavior (SAB 2004)*, LNCS, pages 373–384. Springer, 2004.

[33] Federici, D. Evolving developing spiking neural networks. In *Evolutionary Computation, 2005. The 2005 IEEE Congress on*, volume 1, pages 543–550 Vol.1, 2005.

[34] Flyvbjerg, H., and Kjaer, N.J. Exact solution of kauffman's model with connectivity one. *Journal of Physics A: Mathematical and General*, 21(7):1695–1718, 1988.

[35] Fogel, L.J., and Owens, A.J., and Walsh, M.J. *Artificial Intelligence through Simulated Evolution*. Wiley, 1966.

[36] Gershenson, C. Classification of random boolean networks. In *In Standish, R.K., Bedau, M.A., and Abbass, H.A. (eds.) Artificial Life VIII: Proceedings of the Eight International Conference on Artificial Life*, pages 1–8. MIT Press, 2002.

[37] Gershenson, C. Introduction to Random Boolean Networks. In *In Bedau, M., P. Husbands, T. Hutton, S. Kumar, and H. Suzuki (eds.) Workshop and Tutorial Proceedings, Ninth International Conference on the Simulation and Synthesis of Living Systems (ALife IX)*, pages 160–173, 2004.

[38] Gordon, T. G. W., and Bentley, P. J. Bias and scalability in evolutionary development. In *GECCO 2005*, pages 83 – 90. ACM Press, 2005.

[39] Gordon, T. G. W., and Bentley, P. J. Development brings scalability to hardware evolution. In *The 2005 NASA/DOD Conference on Evolvable Hardware*, pages 272–279. IEEE, 2005.

[40] Gruau, F. *Neural Network Synthesis using Cellular Encoding and the Genetic Algorithm*. PhD thesis, PhD Thesis, France, 1994.

[41] Gruau, F. and Whitley, D. and Pyeatt, L. A comparison between cellular encoding and direct encoding for genetic neural networks. In *Genetic Programming 1996: Proceedings of the First Annual Conference*, pages 81–89. MIT Press, 1996.

[42] Gunter, Chr., and Dhand, R. The mouse genome. *Nature*, 420(509):509, 2002.

[43] Haddow, P.C., and Tufte, G., and van Remortel, P. "shrinking the genotype: L-systems for ehw ?". In *4th International Conference on Evolvable Systems (ICES01)*, Lecture Notes in Computer Science, pages 128–139. Springer, 2001.

[44] Harding, S.L., and Miller, J.F., and Banzhaf, W. Self-modifying cartesian genetic programming. In *GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 1021–1028, New York, NY, USA, 2007. ACM.

[45] Hartmann, M. *Evolution of Fault and Noise Tolerant Digital Circuits*. PhD thesis, Dept.of Computer and Information Science, NTNU, Trondheim, Norway, 2005.

[46] Henrey, M. and Ahmed, A. and Boscariol, P. and Shannon, L. and Menon, C. Abigaille-III: A Versatile, Bioinspired Hexapod for Scaling Smooth Vertical Surfaces. *JOURNAL OF BIONIC ENGINEERING*, 11(1):1–17, JAN 2014.

[47] Herculano-Houzel, S. and Lent, R. Isotropic fractionator: A simple, rapid method for the quantification of total cell and neuron numbers in the brain. *The Journal of Neuroscience*, 25(10):2518–2521, 2005.

[48] Holland, J.H. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, 1975.

[49] Hornby, G. and Lohn, J. D. and Linden, D. S. Computer-automated evolution of an x-band antenna for nasa's space technology 5 mission. *Evolutionary Computation*, 19(1):1–23, 2011.

[50] Horowitz, E. and Sahni, S. Computing partitions with applications to the knapsack problem. *J. ACM*, 21(2):277–292, April 1974.

[51] Jakobi, N. Harnessing morphogenesis. In *International Conference on Information Processing in Cells and Tissues*, pages 29–41, 1995.

[52] G. Jones. *Genetic and Evolutionary Algorithms*, chapter 2. John Wiley & Sons, Ltd, 2002.

[53] S.A. Kauffman. *Investigations*. Oxford University Press, 2000.

[54] S. Kauffmann. Metobolic stability and epigenesis in randomly constructed genetic sites. *Journal of Theoretical Biology*, 22:437–467, 1969.

[55] S.A. Kauffmann. *The Origins of Order*. Oxford University Press, 1993.

[56] Kirschner, M. Beyond Darwin: evolvability and the generation of novelty. *BMC Biology*, 11(110), 2013.

[57] Kirschner, M. and Gerhart, J. Perspective: Evolvability. *Proc. Natl. Academy of Sciences*, 95:8420–8427, 1998.

[58] Kitano, H. Designing neural networks using genetic algorithms with graph generation systems. *Complex Systems*, 4(4):461–476, 1990.

[59] Kitano, H. Building complex systems using development process: An engineering approach. In *Evolvable Systems: from Biology to Hardware, Lecture Notes in Computer Science*, pages 218–229. Springer, 1998.

[60] S.G. Kobourov. Spring embedders and force directed graph drawing algorithms. *CoRR*, abs/1201.3011, 2012.

[61] Korb, K.B., and Dorin, A. Evolution unbound: releasing the arrow of complexity. *Biology and Philosophy*, 26:317–338, 2011.

[62] Kowaliw, T. Measures of complexity for artificial embryogeny. In *In Proceedings of the 10th annual conference on Genetic and evolutionary computation*, pages 843–850. ACM, 2008.

[63] Koza, J. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, 1992.

[64] Kumar, S., and Bentley, P. J., editor. *An Introduction to Computational Development*. Elsevier, 2003.

[65] Langton, C.G. Computation at the edge of chaos: Phase transitions and emergent computation. *Physica D*, 42:12–37, 1990.

[66] Larsen, E.W. *Environment, development, and Evolution Toward a Synthesis*, chapter chapter 7 A View of Phenotypic Plasticity from Molecules to Morphogenisis, pages 117–124. MIT Press, 2004.

[67] Lehre, P.C. *Complexity and Geometry in Artificial Development*. PhD thesis, Dept.of Computer and Information Science, NTNU, Trondheim, Norway, 2006.

[68] Li, M., and Vitányi, P.M.B. *An Introduction to Kolmogorov Complexity and Its Applications*. Springer-Verlag, New York, 2nd edition, 1997.

[69] Li, R. and Emmerich, M.T.M. and Eggermont, J. and Bovenkamp, E. G. P. and Bäck, T. and Dijkstra, J. and Reiber, J.H. C. *Evolutionary Image Analysis and Signal Processing*, chapter Optimizing a Medical Image Analysis System Using Mixed-Integer Evolution Strategies, pages 91–112. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[70] A. Lindenmayer and P. Prusinkiewicz. Developmental models of multicellular organisms: A computer graphics perspective. In C. Langton, editor, *Artificial Life: Proceedings of an Interdisciplinary Workshop on the Synthesis and Simulation of Living systems*, pages 221–249, Redwood City, September 1989. Addison-Wesley.

[71] Lindenmayer, A. Mathematical Models for Cellular Interactions in Development. *Journal of Theoretical Biology*, 18:280–299, 1968.

[72] Lindenmayer, A. Developmental systems without cellular interactions, their languages and grammars. *Journal of Theoretical Biology*, 1971.

[73] Lindenmayer, A. Developmental algorithms for multicellular organisms: A survey of l-systems. *Journal of Theoretical Biology*, 1975.

[74] Lindenmayer, A. Theoretical plant morphology. In Sattler, R., editor, *Foundations of Computational, Intelligence Volume 1*, pages 37–81. Leiden University Press, The Hague, 1978.

[75] Lloyd, S., and Pagels, H. Complexity as thermodynamic depth. *Annals of Physics*, 1(188):186–213, 1988.

[76] Maduro, F.M., and Meneghini, M.M., and Bowerman, B., and Broitman-Maduro, G., and Rothman, J.H. Restriction of mesendoderm to a single blastomere by the combined action of skn-1 and a gsk-3? homolog is mediated by med-1 and -2 in c. elegans. *Molecular Cell*, 7(3):475 – 485, 2001.

[77] Martínez, G.J., and Seck-Tuoh-Mora, J.C., and Zenil, H. Computation and universality: Class IV versus class III cellular automata. *J. Cellular Automata*, 7(5-6):393–430, 2012.

[78] Matari?, M. and Cliff, D. Challenges in evolving controllers for physical robots. *Robotics and Autonomous Systems*, 19(1):67 – 83, 1996. Evolutional Robots.

[79] McShea, D.W. Metazoan complexity and evolution: is there a trend? *Evolution*, 2(50):477–492, 1996.

[80] McShea, D.W. Functional complexity in organisms: Parts as proxies. *Biology and Philosophy*, 15:641–668, 2000.

[81] H. Meinhardt. *Models of Biological pattern formation*. Academic Press, 1982.

[82] H. Meinhardt. *The Algorithmic Beauty of Seashells*. Springer-Verlag, 1998.

[83] Michalewicz, Z. and Schoenauer, M. Evolutionary algorithms for constrained parameter optimization problems. *Evolutionary Computation*, 4:1–32, 1996.

[84] Miller, J.F. Evolving a self-repairing, self-regulating, french flag organism. In *Genetic and Evolutionary Computation (GECCO 2004)*, Lecture Notes in Computer Science, pages 129–139. Springer, 2004.

[85] Miller, J.F. and Thompson, P. A developmental method for growing graphs and circuits. In *Evolvable Systems: from Biology to Hardware, Lecture Notes in Computer Science*, pages 93–104. Springer, 2003.

[86] Momeni, B., and Chen, C.C., and Hillesland, K.L., and Waite, A., and Shou, W. Using artificial systems to explore the ecology and evolution of symbioses. *Cellular and Molecular Life Sciences*, 68(8):1353–1368, 2011.

[87] Nehaniv, C.L. Editorial for special issue on evolvability. *BioSystems*, 69:77–81, 2003.

[88] Nusslein, V. *Coming To Life*. Yale University Press, 2006.

[89] Packard, H.P., and Wolfram, S. Two-dimensional cellular automata. *Journal of Statistical Physics*, 38:901–946, 1985.

[90] Papadimitriou, Chr. H. and Steiglitz, K. *Combinatorial Optimization: Algorithms and Complexity*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1982.

[91] Pontius, J.U., and Mullikin, J.C., and Smith, D.R., and Agencourt Sequencing Team, and Lindblad-Toh, K., and Gnerre, S., and Clamp, M., and Chang, J., and Stephens, R., and Neelam, B., and Volfovsky, N., and Schï£¡ffer, A.A., and Agarwala, R., and Narfstrï£¡m, K., and Murphy, W.J., and Giger, U., and Roca, A.L., and Antunes, A., and Menotti-Raymond, M., and Yuhki, N., and Pecon-Slattery, J., and Johnson, W.E., and Bourque, W., and Tesler, G., and NISC Comparative Sequencing Program and Oï£¡ Brien, S.J. Initial sequence and comparative analysis of the cat genome. *Genome Research*, 17(11):1675–1689, 2007.

[92] Preissl, R. and Wong, T.M. and Datta, P. and Flickner, M. and Singh, R. and Esser, S.K. and Risk, W.P. and Simon, H.D. and Modha, D.S. Compass: A scalable simulator for an architecture for cognitive computing. In *High Performance Computing, Networking, Storage and Analysis (SC), 2012 International Conference for*, pages 1–11, Nov 2012.

[93] Qiu, G., and Kandhai, D., and Sloot, P.M.A. Understanding the complex dynamics of stock markets through cellular automata. *Phys. Rev.*, 75(4), 2007.

[94] D. Qu, Ch. Z. Mosher, M.K. Boushell, and H.H. Lu. Engineering Complex Orthopaedic Tissues Via Strategic Biomimicry. *ANNALS OF BIOMEDICAL ENGINEERING*, 43(3):697–717, MAR 2015.

[95] R. A. Raff. *The Shape of Life : Genes, Development, and the Evolution of Animal Form*. University Of Chicago Press, 1996.

[96] Rechenberg, I. *Evolutionsstrategie ï£¡ Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Fromman-Holzboog Verlag, 1973.

[97] D.W. Repperger, C.A. Phillips, A. Neidhard-Doll, D.B. Reynolds, and J. Berlin. Power/energy metrics for controller evaluation of actuators similar to biological systems. *MECHATRONICS*, 15(4):459–469, MAY 2005.

[98] Saito, R., and Smoot, M.E., and Ono, K., and Ruscheinski, J., and Wang, P.L., and Lotia, S., and Pico, A.R., and Bader, G.D., and Ideker, T. A travel guide to cytoscape plugins. *Nature Methods*, 9(11):1069–76, 2012.

[99] Sarkar, P. and Phaneendra, S. and Chakrabarti, A. Developing engineering products using inspiration from nature. *JOURNAL OF COMPUTING AND INFORMATION SCIENCE IN ENGINEERING*, 8(3), SEP 2008.

[100] Shackleton, M. and Shipman, R., and Ebner, M. An investigation of redundant genotype-phenotype mappings and their role in evolutionary search. In *In Evolutionary Computation, 2000. Proceedings of the 2000 Congress on*, pages 493–500. IEEE, 2000.

[101] Shen, H. and Zhu, Y. and Niu, B. and Wu, Q.H. An improved group search optimizer for mechanical design optimization problems. *Progress in Natural Science*, 19(1):91 – 97, 2009.

[102] Shipman, R. Genetic redundancy: desirable or problematic for evolutionary adaptation? In *In Artificial Neural Nets and Genetic Algorithms; Proceedings of the International Conference on*, pages 337–344. Springer-Verlag, 1999.

[103] Shipman, R., Shackleton, M., Ebner, M., and Watson, R. Neutral search spaces for artificial evolution: a lesson from life. In *In Artificial Life VII, Proceedings of the Seventh International Conference on Artificial Life*, pages 162–169. MIT Press, 2000.

[104] Shir, O.M. and Emmerich, M. and Back, T. and Vrakking, M.J.J. The application of evolutionary multi-criteria optimization to dynamic molecular alignment. In *Evolutionary Computation, 2007. CEC 2007. IEEE Congress on*, pages 4108–4115, Sept 2007.

[105] Simon, H. and Wolfgang, B. Organic computing. In Rolf P. Wï£¡rtz, editor, *Understanding Complex Systems*, chapter Artificial Development, pages 201 – 220. Springer Verlag, 2008.

[106] M. Sipper. *Evolution of Parallel Cellular Machines The Cellular Programming Approach*. Springer-Verlag, 1997.

[107] Sipper, M., and Sanchez, E., and Mange, D., and Tomassini, M., and Pérez-Uribe, A., and Stauffer, A. A phylogenetic, ontogenetic, and epigenetic view of bio-inspired hardware systems. *IEEE Transactions on Evolutionary Computation*, 1(1):83–97, April 1997.

[108] Staffer, A., and Sipper, M. Modeling cellular development using l-systems. In *2nd International Conference on Evolvable Systems (ICES98)*, Lecture Notes in Computer Science, pages 196–205. Springer, 1998.

[109] Stanley, K.O., and Miikkulainen, R. A taxonomy for artificial embryogeny. *Artificial Life*, 9(2):93–130, 2003.

[110] T. Steiner, J. Trommler, M. Brenn, Y. Jin, and B. Sendhoff. Global shape with morphogen gradients and motile polarized cells. In *Congress on Evolutionary Computation(CEC2009)*, pages 2225–2232. IEEE, 2009.

[111] Stepaniuk, J. *Rough ï£¡ Granular Computing in Knowledge Discovery and Data Mining*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[112] Thompson, J.N., and Medel, R. The coevolving web of life. *Evolution: Education and Outreach*, 3(1):6, 2009.

[113] Tsuji, T. and Hachino, T. and Oguro, R. and Umeda, N. and Takata, H. A control design of robotics using the genetic algorithm. *Artificial Life and Robotics*, 2(1):24–27, 1998.

[114] G. Tufte. Evolution, development and environment toward adaptation through phenotypic plasticity and exploitation of external information. In S. Bullock, J. Noble, R. Watson, and M. A. Bedau, editors, *Artificial Life XI: Proceedings of the Eleventh International Conference on the Simulation and Synthesis of Living Systems*, pages 624–631. MIT Press, Cambridge, MA, 2008.

[115] Tufte, G. From Evo to EvoDevo: Mapping and Adaptation in Artificial Development. *Development*, 2008.

[116] Tufte, G. Phenotypic, developmental and computational resources: Scaling in artificial development. In *Genetic and Evolutionary Computation (GECCO)*, pages 859–866. ACM, 2008.

[117] Tufte, G., and Haddow, P. C. Extending artificial development: Exploiting environmental information for the achievement of phenotypic plasticity. In *7th International Conference on Evolvable Systems (ICES07)*, Lecture Notes in Computer Science, pages 297–308. Springer, 2007.

[118] Tufte, G., and Haddow, P.C. Achieving environmental tolerance through the initiation and exploitation of external information. In Srinivasan, editor, *IEEE Congress in Evolutionary Computation*, pages 2485–2492. IEEE, 2007.

[119] A. Turing. The chemical basis of morphogenesis. *Philosophical Transactions of the Royal Society B*, 237:37–72, 1952.

[120] Urselmann, M. and Emmerich, M.T.M and Till, J. and Sand, G. and Engell, S. Design of problem-specific evolutionary algorithm/mixed-integer programming hybrids: two-stage stochastic integer programming applied to chemical batch scheduling. *Engineering Optimization*, 39(5):529–549, 2007.

[121] von Neumann, J. *Theory of Self-Reproducing Automata*. University of Illinois Press, Urbana, IL, USA., 1966.

[122] Wagner, A. *Robustness and Evolvability in Living Systems*. Princeton University Press, 2005.

[123] Wahde, M. *Biologically Inspired Optimisation Methods*. WIT Press, 2008.

[124] Wang, B. and Yang, W. and McKittrick, J. and Meyers, M. A. Keratin: Structure, mechanical properties, occurrence in biological organisms, and efforts at bioinspiration. *PROGRESS IN MATERIALS SCIENCE*, 76:229–318, MAR 2016.

[125] Wehrens, R. and Buydens, L.M.C. *Classical and Nonclassical Optimization Methods*. John Wiley & Sons, Ltd, 2006.

[126] West-Eberhard, M.J. *Developmental Plasticity and Evolution*. Oxford University Press, 2003.

[127] S. Wolfram. *Theory and Applications of Cellular Automata*. World Scientific, 1986.

[128] S. Wolfram. *A New Kind of Science*. Wolfram Media Inc., 2002.

[129] Wolfram, S. Universality and complexity in cellular automata. *Physica D*, 10:1–35, 1984.

[130] Wolfram, S. Twenty Problems in the Theory of Cellular Automata. *Physica Scripta*, T9:170–183, 1985.

[131] Wolpert, L. Positional information and pattern formation in development. *Developmental Genetics*, 15(6):485–90, 1994.

[132] Wolpert, L. *Principles of Development*. Oxford University Press, 1998.

[133] Wuensche, A., and Lesser, M. *The Global Dynamics of Cellular Automata; An Atlas of Basin of Attraction Fields of One-Dimensional Cellular Automata*. Santa Fe Institute Studies in the Sciences of Complexity. Addison-Wesley, Reading, MA, 1992.

[134] Zitzler, E. Evolutionary algorithms for multiobjective optimization: Methods and applications, 1999.

A Common Genetic Representation Capable of Developing Distinct Computational Architectures

K. Antonakopoulos and G. Tufte

2011 IEEE Congress on Evolutionary Computation (CEC 2011)
New Orleans, USA, June 5-8, 2011

Abstract

A big challenge in the area of developmental and generative systems, is the design of a method for building complex systems with specific structural and/or functional properties. Most developmental models target specific computational architectures or structures of strictly defined building blocks, in both cases developmental models have strong connection to the target computational architecture/phenotype structure. In this work we seek a common developmental model that can target different architectures but also to find a common genetic representation that can include information that enables such a developmental model. The computational architectures with sparsely connected computational elements considered herein are cellular automata and boolean networks. The experiments study the evolvability of the genetic representation and prove that it is able to build stable structures for distinct computational architectures.

A Common Genetic Representation Capable Of Developing Distinct Computational Architectures

Konstantinos Antonakopoulos* and Gunnar Tufte†

Norwegian University of Science and Technology
Department of Computer and Information Science
Sem Sælandsvei 7-9, NO-7491, Trondheim, Norway
Email: {*kostas, †gunnart}@idi.ntnu.no

***Abstract*—A big challenge in the area of developmental and generative systems, is the design of a method for building complex systems with specific structural and/or functional properties. Most developmental models target specific computational architectures or structures of strictly defined building blocks, in both cases developmental models have strong connection to the target computational architecture/phenotype structure. In this work we seek a common developmental model that can target different architectures but also to find a common genetic representation that can include information that enables such a developmental model. The computational architectures with sparsely connected computational elements considered herein are cellular automata and boolean networks. The experiments study the evolvability of the genetic representation and prove that it is able to build stable structures for distinct computational architectures.**

***Index Terms*—Genetic representation, cellular automata, random boolean network, L-systems**

I. Introduction

Artificial developmental systems often target organisms or systems with some kind of functionality. Target functionality may include systems aiming to solve e.g., a structural problem [1], or a computational function [2]. In the first case, the target is the structure itself. In the second case, the target is a computational function. The functionality is given by the computational function of the nodes and their connections.

Node and node functions are closely coupled to machines and how a computational problem can be mapped (fitted) into a computer architecture. Kitano's work on development of neural network structures [3], is an example of an architecture consisting of nodes and an interconnected network. Kitano proposed the use of a developmental process to create large neural networks, i.e., a structure of neural nodes and connections. Including a developmental approach was in Kitano's work to increase scalability. Even if size was an important factor, scaling of complexity was a more prominent goal, i.e., to be able to target more complex problems or scale the solution with the problem size.

The goal of this work is in many ways similar to Kitano's; to develop artificial organisms (phenotypic structures), consisting of connected computational elements. However, further inspired by multicellularity and organisms ability to exploit different developmental paths based on environmental factors, the target structure is expanded from being of a specific type of computational structure. Instead of devising a developmental system that is for a specific computational structure, i.e., genotype information, developmental actions and phenotypic properties exploited by the developmental process, are extended to represent a class of architectures. To be able to develop such structures, there is a need to further understand the properties of the targeted class of computational architectures. An analysis of the possibilities and constraints involved in the development of such computational architectures, focusing on the form, functionality and the inherent biologically inspired properties, was presented [4]. The architectures studied therein were Boolean Networks (BN) [5], Artificial Neural Networks (ANN) [6], Cellular Automata (CA) [7], and Cellular Neural Networks (CNN) [8]. The common property of all these architectures is that they can be called *sparsely connected networks*. This common property motivates a further investigation on how a mapping process can work on a class of architectures in a more general way. This is elaborated by examining how universal properties and processes can be included in development mapping, through an *EvoDevo* approach [9].

In biology, a specie is often used as a basic unit for biological classification and for taxonomic ranking. As such, an organism with unifying properties and same characteristics can be of the same specie. Organisms with different DNA, morphology or ecological niche, are to be considered as different species [9].

In this work, we move one step further by exploring the potentiality of using the same developmental mapping, in order to develop more than one class of structures at the same time or different types of organisms (i.e., species). The architectures considered herein are CAs with a regular connectivity pattern and boolean networks as NK networks [10].

The notion of species and multicellularity in artificial organisms is in contrast to the rather well defined biological counterpart open for interpretations. Herein different types of computational node functions are defined as different cell types, i.e., defining an organism to be multicellular, and the type of interconnection, i.e., defining an organism to be of a specie. However, this notion is blurred, in other work what is defined as species may be closer to what is treated as multicellularity [11]. However, our point is to extend, or increase, the freedom of how the phenotype can be composed. The extended freedom should open the possibility for evolution to choose what phenotype structure, e.g., ANN, CA or BN, that is best suited for the computational problem at hand.

To devise a common developmental process that enables larger freedom, i.e. develop several architectures, the developmental process must be capable of expressing developmental actions, e.g., growth and differentiation, that can result in the structures sought. Further, the developmental process must be able to express a large variety of topologies within each architecture, as to be able to meet different computational goals. The introduction of a common developmental process assumes that the genetic information given to the mapping process must contain enough information to enable development of all architectures. As the amount of genetic information required varies from architecture to architecture, a notion of chromosomes is introduced.

Towards the actual goal of evolving computational functions, it is required to gain knowledge on how a developmental process capable of expressing different architectures operates in combination with the provided genetic information. An experimental approach is taken to investigate if evolvability and a variety of structural difference can be achieved.

The rest of the article is laid out as follows. Section II presents the developmental goals and challenges that need to be addressed. In section III, the common genetic representation is described in detail. Experimental results come in section IV with the conclusion and future work in section V.

II. Development for sparsely-connected computational structures

The developmental goal in this work, is to be able to generate not a specific, but different classes of structures (i.e., species), using the same developmental model. This should be achieved through the same developmental approach. Such developmental approach would require sufficient knowledge of the targeted computational architectures and of their governing properties. That is, for the 2-dimensional CA architecture, the properties of dimensionality and neighborhood must be defined, where the connectivity is predetermined (i.e., the Euclidean space). For boolean networks, the connectivity (i.e., the connections among the nodes in the network), must be determined. To address the problem described, we have classified three subproblems, that may be expressed as: (a) the *developmental model* challenge, (b) the *genome* challenge, and (c) the *developmental processes* involved in the model.

A. The Developmental Model challenge

The developmental model should be able to develop these kind of structures, taking into account the special properties governing each computational architecture (i.e., CAs and BNs). Figures 1 and 2, illustrate this requirement. The developmental model should receive the same kind of genome as input, regardless of the target architecture. Then, it should be possible – depending on some properties of the genome – to discriminate whether it will develop a cellular automata or a boolean network.

Figure 1, visualizes this by showing step-by-step the development of a cellular automata from the developmental model. At DS 0, the first cell of the cellular automata is created. At DS 1, the cellular automata grows in size and a new cell is added. At DS 2, the architecture grows again by adding one more cell to the cellular automata. At DS *n*, development is finished and the cellular automata holds its final structure (adult organism).

Similarly, figure 2 presents step-by-step the development of a boolean network with the same developmental model. At DS 0, the first node with its self-connections is created. At DS 1, the boolean network grows in size and a new node is added to the network. This will cause new connections to be created for all the nodes existing in the network. At DS 2, the network adds another node and new connections are created for the existing nodes. This algorithm continues until the boolean network has created all the nodes and the connections for the existing nodes (DS *n*).

B. The Genome challenge

The second challenge for the developmental model is the type of information to be included in the genome. Based on the properties of a 2D CA explained earlier, the genome should contain enough information about the cells at each developmental step, in order to place them on a 2D CA structure. The wiring of a cell is given by the CA's neighborhood (Fig.1). At the same time and based on the properties of a boolean network, the genome should contain the wiring of the nodes (Fig. 2).

C. The Developmental Processes challenge

The third challenge is to identify what processes should be incorporated in the developmental model, in order for the resulting structure to be able to grow, to alter the function of a cell/node, and to shrink. These processes can be introduced in the developmental mapping through *growth*, *differentiation*, and *apoptosis* (i.e., the death of the cell/node). To better illustrate how these processes will influence the developing structure, figure 3, shows the three developmental processes as applied to a CA, and figure 4 show the same processes as applied to a boolean network.

Having these properties in mind, our genome can incorporate the notion of *chromosomes*. Each chromosome can contain respective information about the structural and/or functional requirements (see section II). More specifically, one chromosome can contain the information required for the cell/node creation (i.e., for the CAs and BNs), where a second chromosome can contain the information required for wiring the nodes (i.e., for BNs). The notion of chromosomes allows us to exploit the genome in a modular way in the sense that if an additional computational architecture need to be described in the future through the same genome, more chromosomes can be added to it.

III. A common genetic representation

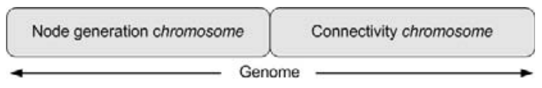


Fig. 5. The common genome with the two chromosomes. The first chromosome maps the cell/nodes of the target architecture. The second chromosome maps the connectivity.

Figure 5, shows how the genome can look like. It is split in two parts (i.e., *chromosomes*). The first chromosome is responsible for creating the cells/nodes. The second chromosome is responsible for creating the connectivity (i.e., when the target architecture is a BN).

Each chromosome should be built out of rules. Each rule must include sufficient information for cell/node creation and for connectivity. Also, the rules should preferably be of certain length. Also, the rules for cell/node creation should be different from the ones for connectivity. Consequently, the chromosome for cell/node creation must have different rules from the ones responsible for connectivity. This is illustrated in figure 6.

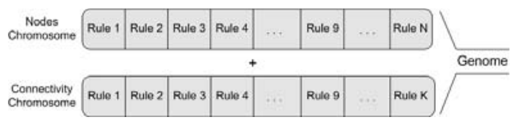


Fig. 6. The first chromosome has rules for cell/node creation. The second chromosome has rules for connectivity. The type of rules included in the chromosomes should be different, since they serve different purposes.

The rules of the first chromosome should be able to express the cell processes like growth, differentiation and apoptosis (fig.3), for the 2D CA. In addition, the rules of the second chromosome should be able to express connectivity, as this was illustrated in figure 4, for boolean network development.

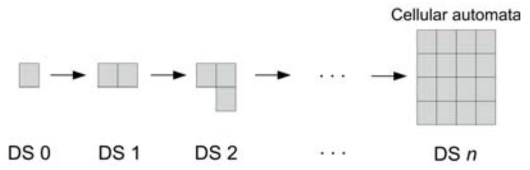


Fig. 1. The developmental model should be able to develop a cellular automata. Here, the development of the automata is shown from developmental step (DS) 0 until the DS *n*.

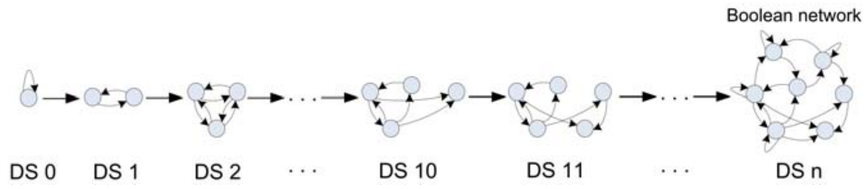


Fig. 2. The developmental model should be able to develop a boolean network. Here, the development stage of a BN is shown from developmental step (DS) 0, until DS *n*.

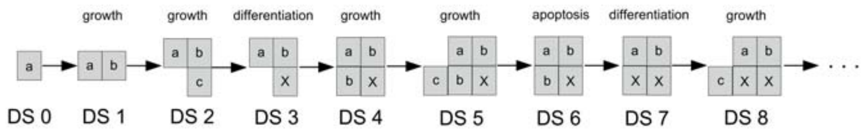


Fig. 3. The developmental model should be able to incorporate the processes of growth, differentiation and apoptosis. Here, each of these processes are illustrated, as the model develops a 2D cellular automata.

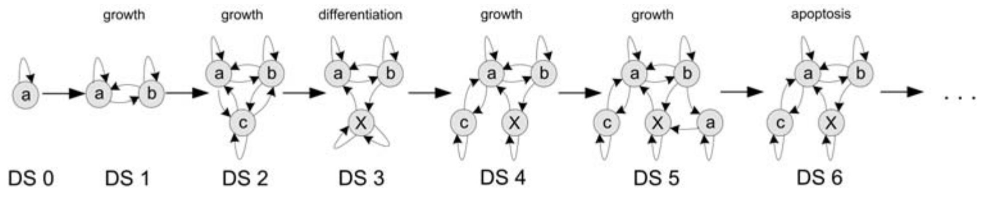


Fig. 4. The processes of growth, differentiation and apoptosis are illustrated, as the model develops a boolean network.

A. An L-system for the genetic representation

To express the rules in the chromosomes, there is a need for a model able to describe developmental systems. The developmental model is the process that maps genetic and environmental information to phenotypic properties, e.g., by growth and differentiation. Gene regulation networks is a possible model used in many systems [12], [13], targeting phenotypic structures and system addressing phenotypic computational properties [6]. Another possible model, is rewriting systems that exploits rules for expanding and changing the structure. Herein a rewriting approach is chosen due to the ease of defining specific rule set, that can target to rewrite specific features of a structure, e.g., connections or node functions this enables a way of splitting genetic information into separate information carrying units (chromosomes), as explained in section III. A prominent candidate model is the L-system. L-systems are rewriting grammars, able to describe developmental or generative systems and have successfully been used to simulate biological processes [14], and describe computational machines [15].

Since there are different types of rules in the two chromosomes, there is a need for two separate L-systems. The first L-system will process the rules found in the first chromosome, with the goal of creating the cells/nodes of the target architecture. The second L-system processes the connectivity rules in the second chromosome.

B. 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.

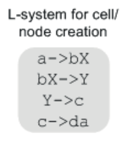
The rules should be able to incorporate the cell processes. As such, there should be symbols which when executed by the L-system, the result would be one of these processes (i.e., growth, differentiation, and apoptosis). It is possible that several symbols of the L-system, can have the same outcome.

Table I, shows the type of symbols used by the L-system of the first chromosome, in order to generate the phenotype with the final cells/nodes. Some cells perform special cell processes and influence the intermediate and final phenotypes. Symbol *a* is the *axiom* of the system. Apart from symbols *a*, *b*, and *c*, which perform *growth* of the phenotype, symbol *d* performs *apoptosis*, leading to the deletion of the current rule (i.e., cell/node), of the intermediate phenotype.

TABLE I
SYMBOLS USED FOR CELL/NODE DEVELOPMENT

Symbol	Description
a (AXIOM)	Add (growth)
b	Add (growth)
c	Add (growth)
d	Delete (apoptosis)
X	Substitute (differentiation)
Y	Substitute (differentiation)
$\rightarrow$	Production

Additionally, symbols `X` and `Y`, are responsible for *differentiation*, leading to the replacement of the predecessor cell/node (i.e., if `X->Y` the outcome will be `Y`, whereas, if `Y->X` the outcome will be `X`). For the shake of simplicity, the length of each rule is 4 symbols (i.e., `4x8bits=32bits`). So, it should be possible for the chromosome to hold a sufficient number of rules, for the L-system to be able to develop.



```
a->bX
bX->Y
Y->c
c->da
```

Fig. 7. An example of L-system rules for the first chromosome

Figure 7, gives an example of a L-system for the first chromosome. Its step-by-step development is shown in figure 8. The target structure is a 2D CA. The symbols of the L-systems and their meanings, are based on Table I.

Development starts with the axiom (`a`) representing a cell at developmental step 0. Since the axiom is found in the L-system rules, development continues and the next rule triggered is the `a->bX`. This rule will create two more cells `b` and `X`, resulting in growth of the CA, at DS 1. The next rule triggered is `bX->Y`. Since `X->Y` denotes differentiation, the symbol `X` is replaced by `Y`, at DS 2. For differentiation to occur, the rules should either be `X->Y`, or `Y->X`. Next, rule `Y->c` triggers causing again growth of the CA, at DS 3. At DS 4, the rule `c->da` is triggered, causing the death of the cell `c` and the growth of the CA with the cell `a`. From DS 5 up to DS 8, the rules are being triggered once more in the same sequence.

C. The L-system for the second chromosome

The rules should be able to generate the connections necessary for the wiring of the nodes. As such, there should be symbols which when executed by the L-system, will result creating a connection forward or backwards from the current node. Each node in the network has a unique number which separates it from the other nodes. Also, the current node has always the number 0 and any nodes starting from the current node forward have positive numbering, where nodes that are from the current node backwards, have negative numbering. So, there is a need to differentiate between the current and the next node, using different symbols and also a need to describe whether the connection will be created forward from the current node, or backwards from the current node.

The rules involved in connectivity are not as complex as the ones found in the first chromosome. The length of the rules here is also 4 symbols / rule. Also, we need to assure that the chromosome will have sufficiently enough information for the L-system developmental processes (i.e., growth, differentiation and apoptosis). The L-system used here is not context-sensitive, but D0L (i.e., with zero-sided interactions). The symbols used in the L-system for the second chromosome are shown at Table II.

TABLE II
SYMBOLS USED FOR CONNECTIVITY DEVELOPMENT

Symbol	Description
x	Node (different from y)
y	Node (different from x)
+	Connect forward
–	Connect backwards
$\rightarrow$	Production

The *axiom* rule for the second chromosome is `x->y`. It means that development initially searches if the axiom exists in the rules. If so, development continues and looks for rules of type `xy->+value`, or `xy->-value`. In short, these two rules imply that if two different (i.e., distinct) nodes are found (`x<>y`), then it creates a connection forward (i.e., if the rule includes a '`+`'), or backwards (i.e., if the rule includes a '`-`'). The field `value`, denotes the node number for the positive/negative connection. For example, rule `xy->+3` denotes that a connection will be created from the current node (node `0`), to the one standing three nodes forward (Fig.9(a)). Similarly, rule `xy->-3`, denotes that a connection will be created starting from the current node (node `0`), to the one that stands three nodes backwards (Fig. 9(b)). If the destination node does not exist (i.e., has not been created yet), a self-connection to the current node is created instead.

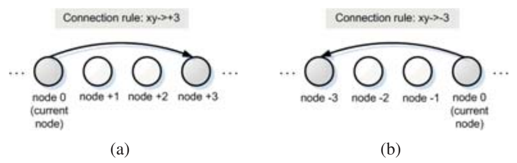
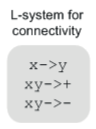


Fig. 9. Example of forward and backward connections for the second chromosome.

The field `value` – although randomly generated – is not allowed to exceed the *cluster size* for connectivity. A cluster denotes a small number of highly-connected nodes and its size can not be more than a predetermined number (usually between 4-6). Cluster size is a parameter to the system and constrains very long positive (or negative) connections in the network, but also reassures that all nodes will have incoming and outgoing connections. In addition, using cluster connectivity is easier to track down the development process. If `value=0`, a self-connection is created to the current node.



```
x->y
xy->+
xy->-
```

Fig. 10. A L-system for the second chromosome responsible for the connectivity of the target architecture.

Figure 11, shows the step-by-step development of a boolean network based on the chromosomes of figures 7 and 10. The symbols of the chromosomes are based on Tables I and II.

The axiom is placed at DS 0 (node `a`) with three self-referencing connections. Each newly generated node comes with three self-

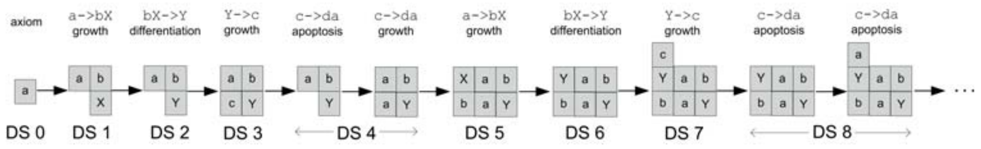


Fig. 8. A step-by-step development of a 2D CA architecture based on the example L-system for the first chromosome

referencing connections. Then at DS 1, the rule `x->y` is triggered, causing the next node *b* to be generated.

For each developmental step, each node in the network has the opportunity to establish three random connections, based on the rules `xy->+` and `xy->-`. As such, at DS 1 the assignment of random connections start with the first node `a`. The grey-colored arrow, shows the current node for which the connections are created. Further on, the grey-colored arrow moves to the next node `b`, which now becomes the current node. For node `b`, three new connections are created.

What is important to notice is that at the beginning of DS 3, the process of differentiation has occurred, causing node `Y` to substitute the previously existed node `X`. At the same step, a new node is added; node `c`.

At the beginning of DS 4, the process of apoptosis has occurred, causing the node `c` – along with all incoming and outgoing connections – to be deleted from the network. At the same step, node `a` is added since is the next symbol following in the L-system (figure 7).

The modularity of the genome, let us enable or disable parts of the genome (i.e., chromosomes), when this is not required by the target architecture. For example, if the target architecture is a 2D CA, the second chromosome (i.e., connectivity) can be disabled, since connectivity is predetermined in CAs. In the same way, if the target architecture is a boolean network, then both chromosomes of the genome need to be enabled (i.e., nodes and connectivity).

D. The genetic algorithm for the common genetic representation

A genetic algorithm (GA), is used to generate and evolve the rules found in the common genetic representation (i.e., in the chromosomes). Since there are two separate L-systems involved in development, the evolutionary process will be consisted of two phases: node and connectivity generation. Mutation and single-point crossover were used as genetic operators. Mutation may happen anywhere inside the 4-symbol rule, ensuring that the production symbol (`->`) is not distorted by mutation. In other words, we want to make sure that after mutation, the production symbol is still in the rule (i.e., the rule is valid). Single-point crossover between two parents is executed at the location of the production symbol. In this way, it is assured that the offspring will also be valid rules. The evolutionary cycle ends after a predetermined number of generations.

IV. Experimental results

In order to be sure the genetic representation can reach the goals set from the beginning, it is of great importance to test the representation in terms of evolvability. This refers to how well the developmental rules can evolve (i.e., adapt to the environment).

First, we should measure the maximum and the average number of rules triggered for a predefined number of developmental steps. During development, we aim to find the highest number of rules triggered. A low number for the maximum number of rules triggered would mean that our mapping is not able to evolve properly and would imply a representation issue. On the other hand, the average number of rules triggered during development is an extra indication for the average evolvability of our developmental mapping. We would wish the average number of rules triggered to be as high as possible. Generally, one should evaluate the average number of rules triggered with respect to the maximum number of rules triggered. As such, a possible low value for the average number of rules triggered should be evaluated with respect to the maximum number of rules triggered.

Second, the proposed genetic representation should be able to build structures which are stable (i.e., reach a point attractor). As such, it is necessary to measure if our mapping can reach point attractors during development, along with the number of steps required before a point attractor is found. For these two factors, we should be able to find the maximum number of point attractors and the maximum number of developmental steps required before a point attractor is found over the whole population for one evolutionary run. Here, it is evident that we wish the highest possible value for the point attractors and the lowest for the developmental steps required, before a point attractor is found.

Additionally, it would be interesting to see how a structure evolves exhibiting certain cell processes, like growth and differentiation, or growth and apoptosis, and perform the same measurements during development. To be able to find point attractors during development, there should be no change in structure.

A. Experimental setup

To be able to run the tests described earlier in section IV, the description of the setup first needs to be clarified. The total number of rules for nodes generation is 20. That means that the first chromosome should have a total size of `20x32bits=648bits`. As far as the second chromosome is concerned, the number of rules included for connectivity is 8. That means that the second chromosome will have a total size of `8x32bits=256bits`.

Generational mixing protocol was used for adult selection, where adults and children compete equally for the total number of adult spots. As a global selection mechanism, *roulette wheel* with fitness proportionate was used for parent selection, in which fitness values are scaled by the average population fitness. Simple symbol mutation and single-point crossover with the production symbol (`->`) as the crossover point, are the GA's genetic operators. The number of total generation is 1000, the population size is 100, the mutation rate is 0.2 and the crossover rate is 0.5.

B. Experiment I - Maximum/Average number of rules triggered

Using the experimental setup described at subsection IV-A, we run a total set of 10 experiments to test the evolvability of the proposed genetic representation. That is, the maximum and the average number of rules triggered during development. The fitness function for the first chromosome consists of the following factors (a) The total number of rules been used `x 0.20`, (b) the total number of different rules used `x 0.80`, (c) an extra 10 points in case an *axiom* node is found in the genotype, and (d) an extra 10 points in case all rules of the genotype were used at least once during development. The fitness function for the second chromosome consists of (a) importance to

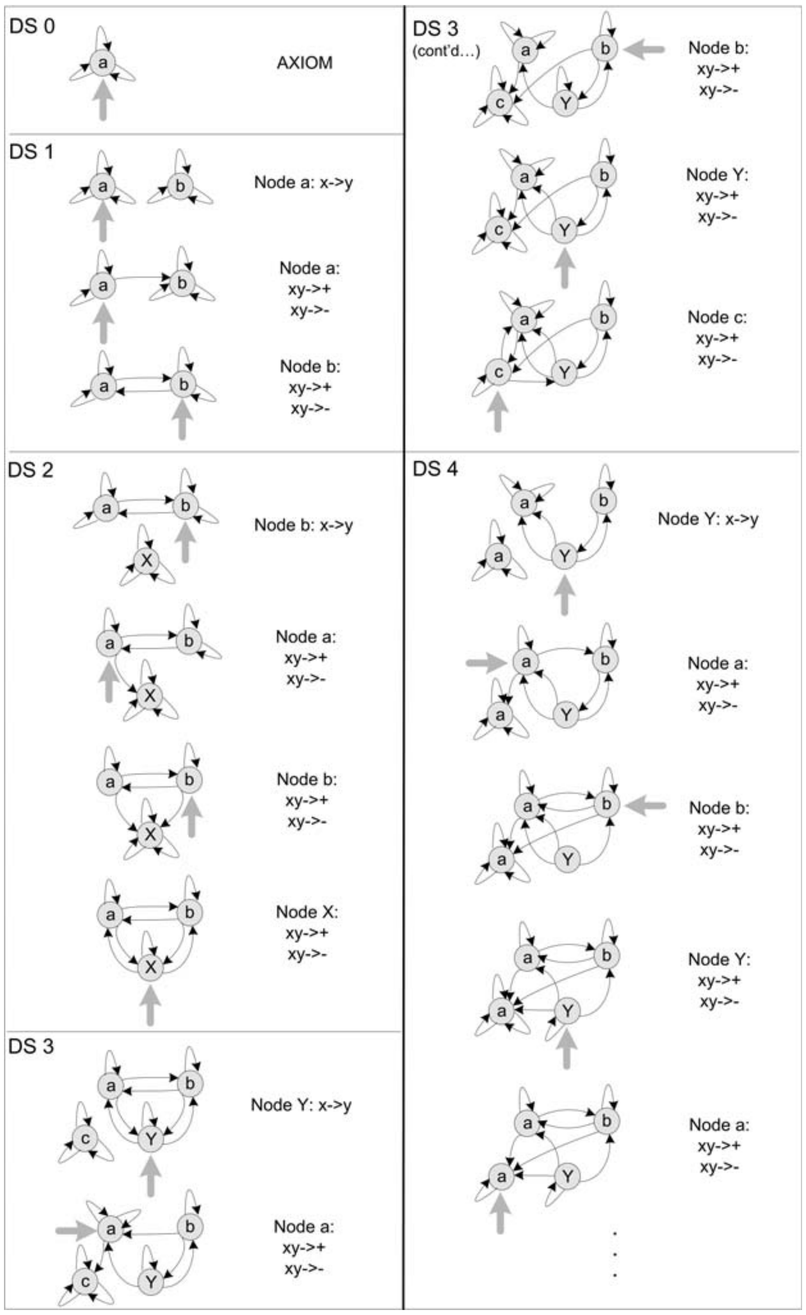


Fig. 11. A L-system for the second chromosome responsible for the connectivity of the target architecture.

growth `x 0.70`, and (b) importance to self-connectivity `x 0.30`. This means that we are more interested in cellular automata that its cells are amenable to growth but also in boolean networks that can grow, and less interested if the nodes contain self-connections.

Figure 12, shows the maximum and the average number of rules triggered during development with standard deviation. The values are averaged over the number of individuals in the population.

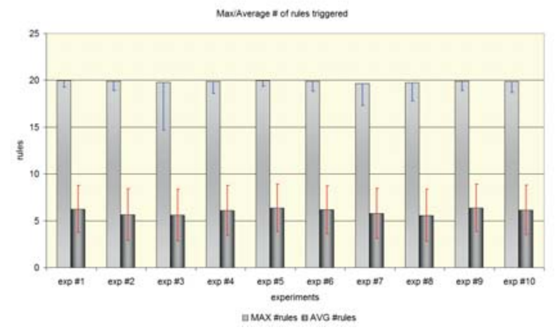


Fig. 12. Maximum/Average number of rules triggered with standard deviation during development

What we can make out of the results is that development could reach the maximum number of rules (20) throughout all the experiments that took place, with experiment #7 sightly deviating from the maximum. Also, the average number of rules triggered is within a range between 6-6.5, meaning that there are no large variations. The average number of rules triggered (if seen separately), is comparatively low, but if seen with respect to the maximum number of rules, we can say that our developmental mapping has reached the experimental goal.

C. Experiment II - Point attractors / Number of developmental steps

With the same experimental setup of subsection IV-A, we run a total set of 10 experiments, to see if the proposed genetic representation is able to build stable structures (i.e., reach a point attractor). So, we measured the total point attractors found during development for the whole population over one evolutionary run, along with the number of developmental steps required before a point attractor is found. We used the same fitness function, mutation and single-point crossover as in the previous experiment (Experiment IV-B).

Figure 13, shows a slight deviation of the maximum number of point attractors over the 10 experiments totally conducted. The maximum number of point attractors ranges from 31-37.5, with an extreme low of 11, at exp. #7. The average number of developmental steps required before a point attractor is found, ranges also between 4 and 18. The low range is found at exp. #7 again. This figure shows that a point attractor is found from the very beginning of development (around DS 4 of experiment #7), up to the very late developmental steps (around DS 18, of experiment #10). It is obvious that the proposed genetic representation can reach several point attractors during development, which satisfies our second experimental goal.

D. Experiment III

In order to verify our findings, it would be interesting to illustrate how a structure evolves, exhibiting certain cell processes. Here, we show an example of one developmental run for the first chromosome with only cell growth and differentiation (figure 14(a)), and another example with only cell growth and apoptosis (figure 14(b)). For this experiment, we used the experimental setup of subsection IV-A, with the same fitness function and genetic operators.

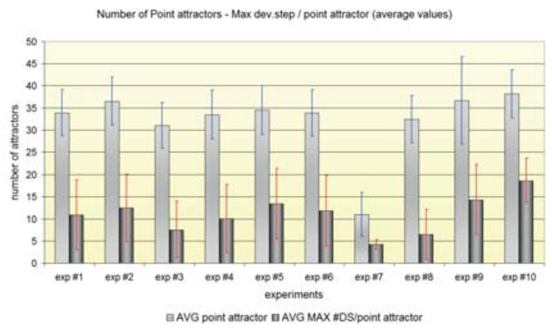
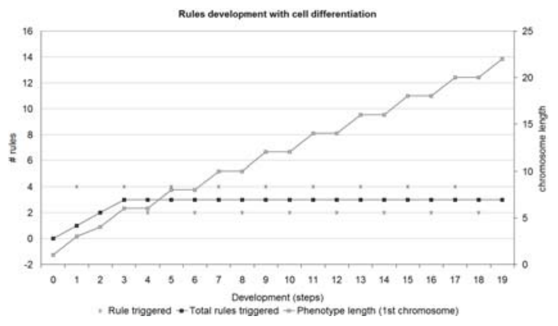
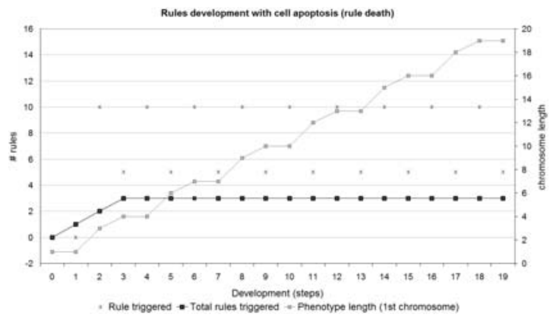


Fig. 13. Total point attractors / Maximum number of steps (averaged) before a point attractor is found with the standard deviation.



(a)



(b)

Fig. 14. Development of rules for 20 steps. The figures show the rules being triggered, the total number of rules triggered and the phenotypic length. (a) development occurs with cell growth and differentiation processes, (b) development occurs with cell growth and apoptosis.

What figure 14(a) shows is that at developmental steps 4, 6, 8, 10, 12, 14, 16 and 18, cell differentiation occurs, causing the length of the phenotype to remain at the same length (secondary Y axis). The rules triggered are 0, 4 and 2. Therefore, the total rules triggered is three after the third developmental step, and remain constant throughout development. The final phenotype of the structure after development using only growth and differentiation processes, has a total length of 22 symbols.

The next figure 14(b), show that at developmental steps 1, 4, 7, 10, 13, 16 and 19, cell apoptosis occurs, causing the death of the current rule. Based on the fourth rule of fig. 7 (`c->da`), after apoptosis, there is growth. These two processes are executed in the same developmental step (also shown in figures 8 and 11). The rules triggered are 0, 10 and 5. The maximum number of rules triggered here is also three, therefore the constant line after the third developmental step. The phenotype of the structure at the end of the development has a total length of 19 symbols. The length of the structure at the last two developmental steps is the same, interpreting that a point attractor has been found. This last observation shows that the genetic representation can reach point attractors, as well as, to grow phenotypes.

Since the goal for our genetic representation is to investigate whether it can evolve phenotypes, but also to be able to reach stable structures, figure 14, proves this point.

V. Conclusion and future work

In this work, a genetic representation for sparsely connected computational architectures was proposed, using a common chromosome and developmental L-systems for nodes generation and connectivity. The computational architectures considered herein were CAs and BNs. The proposed genetic representation takes advantage of the similarities and differences observed in these architectures. The experiments studied the evolvability of the genetic representation and it was shown to be able to build stable structures.

What is important is that the notion of chromosomes in our representation, allows us to exploit the genome in a modular way in a sense that if additional computational architectures need to be incorporated in the future and expressed by the same genome, more chromosomes can easily be attached.

Consequently future work will involve the extension of the representation, so as to incorporate other types of sparsely connected computational architectures (i.e., artificial neural networks), the coupling of different architectures by development and perhaps what is most promising, the exploration of (hybrid) organisms with a fitness-based on function and not gene triggering and structural properties-through a common developmental genome.

References

[1] T. Steiner, J. Trommler, M. Brenn, Y. Jin, and B. Sendhoff, "Global shape with morphogen gradients and motile polarized cells," in *Congress on Evolutionary Computation(CEC2009)*. IEEE, 2009, pp. 2225–2232.

[2] S. L. Harding, J. F. Miller, and W. Banzhaf, "Self-modifying cartesian genetic programming," in *GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*. New York, NY, USA: ACM, 2007, pp. 1021–1028.

[3] H. Kitano, "Building complex systems using development process: An engineering approach," in *Evolvable Systems: from Biology to Hardware, ICES*, ser. Lecture Notes in Computer Science. Springer, 1998, pp. 218–229.

[4] K. Antonakopoulos and G. Tufte, "Possibilities and constraints of basic computational units in developmental systems," in *Norsk Informatikkonferanse, NIK-2009*, 2009, pp. 73–84.

[5] S. A. Kauffman, *The Origins of Order*. Oxford University Press, 1993.

[6] J. C. Astor and C. Adami, "A developmental model for the evolution of artificial neural networks," *Artificial Life*, vol. 6, no. 3, pp. 189–218, 2000.

[7] G. Tufte and P. C. Haddow, "Towards development on a silicon-based cellular computation machine," *Natural Computation*, vol. 4, no. 4, pp. 387–416, 2005.

[8] L. Chua and L. Yang, "Cellular neural networks: theory," *IEEE Transactions on Circuits and Systems*, vol. 35, no. 10, pp. 1257–1272, October 1988.

[9] J. S. Robert, *Embryology, Epigenesis and Evolution: Taking Development Seriously*, ser. Cambridge Studies in Philosophy and Biology. Cambridge University Press, 2004.

[10] C. Gershenson, "Classification of random boolean networks," in *Proceedings of the Eight International Conference on Artificial Life*, 2002, pp. 1–8.

[11] J. F. Knabe, C. L. Nehaniv, and M. J. Schilstra, "Evolution and morphogenesis of differentiated multicellular organisms: Autonomously generated diffusion gradients for positional information," in *Artificial Life XI: Proceedings of the Eleventh International Conference on the Simulation and Synthesis of Living Systems*, S. Bullock, J. Noble, R. Watson, and M. A. Bedau, Eds. MIT Press, 2008, pp. 321–328. [Online]. Available: http://panmental.de/ALifeXIflag

[12] P. Eggenberger, "Evolving morphologies of simulated 3d organisms based on differential gene expression," in *Fourth European Conference on Artificial Life*. MIT press, 1997, pp. 205–213.

[13] M. Joachimczak and B. Wróbel, "Evo-devo in silico: a model of a gene network regulating multicellular development in 3d space with artificial physics," in *Artificial Life XI: Proceedings of the Eleventh International Conference on the Simulation and Synthesis of Living Systems*, S. Bullock, J. Noble, R. Watson, and M. A. Bedau, Eds. MIT Press, Cambridge, MA, 2008, pp. 297–304.

[14] A. Lindenmayer and P. P., "Developmental models of multicellular organisms: A computer graphics perspective," in *Artificial Life: Proceedings of an Interdisciplinary Workshop on the Synthesis and Simulation of Living Systems*, C. G. Langton, Ed. Addison-Wesley Publishing Company, 1989, pp. 221–249.

[15] A. Staffer and M. Sipper, "Modeling cellular development using l-systems," in *Second International Conference on Evolvable Systems (ICES98)*, ser. Lecture Notes in Computer Science. Springer, 1998, pp. 196–205.

Is Common Developmental Genome a Panacea Towards More Complex Problems?

K. Antonakopoulos and G. Tufte

13th IEEE International Symposium on Computational Intelligence and Informatics
Budapest, Hungary, November 20-22, 2012

Abstract

The potentiality of using a common developmental mapping to develop not a specific, but different classes of architectures (i.e., species), holding different structural and/or computational phenotypic properties is an active area of research in the field of bio-inspired systems. To be able to develop such species, there is a need to understand the governing properties and the constraints involved for their development. In this work we investigate the ability of common developmental genomes to evolve more than one specie (i.e., computational architecture), towards problems with increasing complexity. The architectures considered as different species were cellular automata and boolean networks and the problem studied was a simple financial market model over various architecture sizes. We considered problem instances of the same problem, each having a higher level of complexity, i.e., an instance with no state memory and with a previous memory of 1-, 2-, 5- and 10-state. The results show that the common developmental genome was able to find better results for certain cell architectures sizes.

Is Common Developmental Genome a Panacea Towards More Complex Problems?

Konstantinos Antonakopoulos* and Gunnar Tufte†

Norwegian University of Science and Technology
Department of Computer and Information Science, Sem Sælandsvei 7-9, NO-7491, Trondheim, Norway
Email: {*kostas, †gunnart}@idi.ntnu.no

***Abstract*—The potentiality of using a common developmental mapping to develop not a specific, but different classes of architectures (i.e., species), holding different structural and/or computational phenotypic properties is an active area of research in the field of bio-inspired systems. To be able to develop such species, there is a need to understand the governing properties and the constraints involved for their development. In this work we investigate the ability of common developmental genomes to evolve more than one specie (i.e., computational architecture), towards problems with increasing complexity. The architectures considered as different species were cellular automata and boolean networks and the problem studied was a simple financial market model over various architecture sizes. We considered problem instances of the same problem, each having a higher level of complexity, i.e., an instance with no state memory and with a previous memory of 1-, 2-, 5- and 10-state. The results show that the common developmental genome was able to find better results for certain cell architectures sizes.**

***Index Terms*—Genetic representation, cellular automata, boolean network, L-systems, complexity, kolmogorov**

I. Introduction

In man-engineered systems, one can study digital organisms in a "much simplified" artificial EvoDevo world. Most EvoDevo (Evolutionary Developmental) systems consist of a genotype that target special phenotypic structures (i.e., structures that comprise connected computational elements). To be able to develop such structures, there is a need to understand further the underlying properties of such architectures and the constraints involved in their development with a focus on the form, functionality and inherent biological properties [1]. In their classic paper, Gould and Lewontin argued that biological organisms are "so constrained" by a variety of factors, including developmental pathways, that "...the constraints themselves become more important in delimiting pathways of change than the selective force that may mediate change when it occurs." [2]. It is therefore worth investigating how development can work on phenotypes with looser constraints; by investigating how universal properties and processes can be included in a developmental mapping [3]. In biology, species are individuals sharing the same developmental and ecological processes [4]. As such, in [5], it was shown the potentiality of using same developmental mapping to develop not a specific, but different classes of architectures (i.e., species), holding different structural phenotypic properties (i.e., looser phenotypic constraints).

The hypothesis for this work is whether common developmental genomes can prove superior over genomes that are evolved separately for each specie, towards complex computations. To see if the hypothesis holds, two problems should sufficiently be investigated and answered. Firstly, whether the same mapping (i.e., a common developmental genome), can favor the evolvability of different computational architectures under the same environment with limited resources. This problem was studied in [6]. Secondly, and as the motivation for this work, is to study whether the same developmental mapping can favor the evolvability of different computational architectures (i.e., CA and BN), with a focus in problems of increasing complexity. Then, we will have enough evidence and a proof of concept that common developmental genomes are more evolvable or can favor the development of different architectures under a given environment. The notion of complexity is approached from the view of Kolmogorov complexity. Here again, we consider the architectures as different species (since they exhibit different structural properties).

The rest of the article is organized as follows. Section II gives an overview of the various interpretations of complexity so far. The main ideas of Kolmogorov complexity are introduced in Section III. The challenges involved in such a developmental model and the description of the common genetic representation are addressed in Sections IV and V, respectively. Experiments come at section VI with discussion and future work in section VII.

II. Related work on complexity

In this section, we give a brief overview of the ideas that have been delineated in the field of complexity. Complexity can be a very general term and can take various meanings which may depend primarily on the field domain but also under the angle one wishes to infer and explain various phenomena. The major interest groups are physicists interested in dynamical systems and biologists who ponder whether complexity and its evolution is simply a trend [7]. Physicists are mainly interested in the relationship between structure and complexity, namely, *computational complexity*. In this context, there were various endeavors to give a meaningful measurement of complexity (i.e., thermodynamical depth [8], statistical complexity [9]). Computational complexity has also been used by computer scientists as a means to quantify the computational resources needed to solve the problem as a function of the problem instance size [10].

Biologists on the other hand, are trying to capture the form, function or the genetic sequence that comprise an organism, a.k.a., *structural complexity*. McShea, tried to study structural complexity in animals in terms of number of cell types and limb-pair types [11], but also made the case for a measure of *functional complexity* of organisms, counting the different functions an organism can perform [12]. Another measure of complexity pointed out by Adami [7], was *hierarchical complexity*, measuring the number of levels of nestedness during the course of evolution of an organism. The goal was to find a measure able to capture the biological complexity in the functional protein level; looking at correlations among the symbols of a sequence with *features of the environment within which sequence is functional*. In other words, correlations between the symbols in a genome and a description of the environment within which that sequence is functional. He defined *physical complexity* (of a sequence), as the amount of information that is stored in that sequence about a particular environment (i.e., a niche).

Both evolutionary biologists and computer scientists have strived over the past years to create the necessary conditions for biological evolution and measurement of the complexity involved in open-ended A-Life systems. Currently, no A-Life system has demonstrated anything like the creativity of biological evolution [13]. Most of the A-Life literature on the prospects and means for generating open-ended evolution focuses exclusively upon generating more complex organisms (i.e., organismic complexity). As of today, there seems to be no consensus on a formal definition of biological complexity [7]. In many cases though, organismic complexity is essentially realized as physical complexity.

In this work, we try to find a more intuitive approach to characterize the notion of complexity. Towards that direction, physical complexity technically can be defined as the *shared Kolmogorov complexity* between a sequence (phenotype), and a description of the environment in which that sequence is to be interpreted [14]. As such, concepts from Kolmogorov complexity theory can be borrowed and use them abstractively herein.

III. Kolmogorov complexity

In this section we introduce Kolmogorov complexity and how notions of its theory can help us formulate our reasoning for the experiments to follow. We don't intend to give any review nor we try to prove the theory mathematically. Formal definitions and results of the theory can be found in [15]. Kolmogorov complexity is a measure that can be used on individual, finite objects [16]. It associates the complexity of an object (phenotype), with the length of the shortest description of the object. For an unambiguous measure of an object's description length, the description length was defined as the length of the shortest program that generates the object on a fixed, universal Turing machine [15]. The Kolmogorov complexity itself is incomputable. So, for practical experiments, approximations are used instead. It is mostly being used in conjunction with compression algorithms, as an overestimate measure for complexity. The idea is that bitstrings that are easily compressible have low complexity, whereas bitstrings that cannot be compressed are more complex. So, complexity is proportional to the compression ratio.

Since an object can be given as a description of its environment, we can infer that the more information we have about an object, the more complex this object is. Similarly, it can be implied that the longer the object's trajectory, the more complex the object is. To formulate this, let n be the amount of information required to describe an object (for the purpose of this example, think of the object being a finite state automaton). For $n = 4$, we say that the finite state automaton at timestep t is sufficiently described by an amount of information of four. We suppose that information about the environment is included. Similarly, if $n = 5$, the finite state automaton at timestep t, is given by an information amount of five. Therefore the latter automaton is a more complex machine. Although the above explanation is intuitive, it gives a clear, rudimentary view for our experimental basis at Section VI.

IV. Development for Sparsely-Connected Computational Structures

The developmental goal is to be able to generate not a specific, but different classes of structures (i.e., species), using the same developmental model. This should be achieved through the same developmental approach. Such developmental approach requires sufficient knowledge of the targeted computational architectures and of their governing properties. That is, for the 2-dimensional CA architecture, the properties of dimensionality and neighborhood must be defined, where the connectivity is predetermined (i.e., the Euclidean space). For boolean networks, the connectivity (i.e., the node connections of the network), must be determined. The problem just described can be better expressed as *three-challenge* problem: (a) the *genome* challenge, (b) the *developmental processes* involved in the model, and (c) the *developmental model* challenge. The three-challenge problem and the genetic representation (section V), was initially introduced in [5] and is presented here as is.

A. The Genome Challenge

Based on the properties of a 2D-CA, the genome contains information about the cells at each developmental step, in order to place them on a 2D-CA lattice structure. The wiring of the cell is given by the CA's neighborhood. At the same time and based on the properties of a boolean network, the genome contains enough information to feed the developmental model to develop a boolean network, at each developmental step.

B. The Developmental Processes Challenge

The resulting structure is able to grow, alter the functionality of a cell/node, and shrink. These processes are introduced in the developmental mapping through *growth*, *differentiation*, and *apoptosis* (i.e., the death of the cell/node). Having these properties in mind, our genome incorporates the notion of *chromosomes* - inspired by biology. Each chromosome contains respective information about the structural and/or functional requirements. More specifically, a chromosome will contain the information required for the cell/node creation (i.e., for the CAs and BNs), where another chromosome will contain the information required for wiring the nodes (i.e., for BNs). The notion of chromosomes allows us to exploit the genome in a modular way in the sense that if an additional computational architecture need to be described through the same genome, more chromosomes can be added to it.

C. The Developmental Model Challenge

The developmental model is able to develop these structures, taking into account the special properties employed by each architecture. The developmental model receives the same genome as input, regardless of the target architecture. Then, it is possible – depending on some properties of the genome – to discriminate whether it will develop a CA or a BN.

V. THE COMMON GENETIC REPRESENTATION

In biology, a specie is often used as the basic unit for biological classification and for taxonomic ranking [4]. As such, an organism with unifying properties and same characteristics can be of the same specie. The genome is split into two parts (*chromosomes*). The first chromosome is responsible for creating the cells/nodes. The second chromosome is responsible for creating the connectivity (i.e., for the BNs). Each chromosome is built out of rules. Each rule has sufficient information for cell/node creation and connectivity. Also, the rules are of certain length. Those destined for cell/node creation are different from the ones for connectivity. Consequently, chromosomes contain different rules.

A. An L-system for the genetic representation

A rewriting approach was chosen due to the ease of defining specific rule set, that can target to rewrite specific features of a structure, e.g., connections or node functions that enable a way of splitting genetic information into separate information carrying units (i.e., chromosomes).

A prominent model is L-systems. They are rewriting grammars, able to describe developmental systems, simulate biological processes [17], and describe computational machines [18]. Since there are different types of rules in the two chromosomes, there is a need for two separate L-systems. The first L-system processes the rules of the first chromosome, while the second L-system deals with the connectivity rules of the second chromosome.

B. 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. Table I(a), shows the type of symbols used by the L-system of the first chromosome. Some cells perform special cell processes and influence the intermediate and final phenotypes. Symbol *a* is the *axiom*. Apart from the symbols *a*, *b*, and *c*, which perform *growth* of the phenotype, symbol *d* performs *apoptosis*, leading to the deletion of the current rule (i.e., cell/node), of the intermediate phenotype. Additionally, symbols `X` and `Y`, are responsible for *differentiation*, leading to the replacement of the predecessor cell/node (i.e., if `X`$\rightarrow$`Y` the outcome will be `Y`, whereas, if `Y`$\rightarrow$`X` the outcome will be `X`). For the shake of simplicity, the length of each rule is 4 symbols (i.e., `4x8bits=32bits`). For node/cell generation the L-system runs for 100 timesteps and then stops. As such, the intermediate phenotypes generated by development are of variable size.

An example with step-by-step development of a 2D-CA architecture is illustrated in [5].

C. 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; the current node has always the number zero and any nodes starting from the current node forward have positive numbering, where nodes that exist from the current node backwards, have negative numbering. So, there is a need to differentiate between the current and the next node, using different symbols and also a need to describe whether a connection will be created forward or backward from the current node.

The rules involved for connectivity are not as complex as the ones found in the first chromosome. The length of the rules here is also 4 symbols / rule. Also, there is a need to assure that the chromosome will have sufficient information for the developmental processes (i.e., growth, differentiation and apoptosis). The L-system uses is a D0L (i.e., with zero-sided interactions). The symbols used, are explained at Table I(b). The *axiom* rule for the second chromosome is `x`$\rightarrow$`y`. It means that development initially searches if the axiom exists. If so, development continues and looks for rules of type `xy`$\rightarrow$`+value`, or `xy`$\rightarrow$`-value`. In short, these two rules imply that if two different (i.e., distinct) nodes are found (`x`$\neq$`y`), then it creates a connection forward (if the rule includes a '`+`'), or backwards (if the rule includes a '$-$'). The field `value` is encoded in the genotype and denotes the node number for the generated connection. For example, rule `xy`$\rightarrow$`+3` denotes that a connection will be created from the current node (node `0`), to the one being three nodes forward. Similarly, rule `xy`$\rightarrow$`-3`, denotes that a connection will be created starting from the current node (node `0`), to the one that is three nodes backwards. If `value=0`, a self-connection is created to the current node. A step-by-step development of a boolean network based on the chromosomes of Table I(a) and I(b), can be found at [5] and is not shown here due to page limitation. The modularity of the genome, gives the possibility to development itself to enable or disable parts of

TABLE I
(A) SYMBOL TABLE FOR NODE GENERATION, (B) SYMBOL TABLE FOR CONNECTIVITY GENERATION

(a)

Symbol	Description
a (AXIOM)	Add (growth)
b	Add (growth)
c	Add (growth)
d	Delete (apoptosis)
X	Substitute (differentiation)
Y	Substitute (differentiation)
$\rightarrow$	Production

(b)

Symbol	Description
x	Node (different from y)
y	Node (different from x)
+	Connect forward
−	Connect backwards
$\rightarrow$	Production

it (chromosomes), when this is required and driven by the goal set. For example, if the target architecture is a 2D-CA, the second chromosome (i.e., connectivity) is disabled, since connectivity is predetermined. Similarly for BN development, both chromosomes are enabled (i.e., nodes and connectivity).

D. The genetic algorithm for the common genetic representation

A genetic algorithm is used to generate and evolve the rules found in the genome (i.e., in the chromosomes). Since there are two separate L-systems involved in development, the evolutionary process comprise two phases: node and connectivity generation phases. Mutation and single-point crossover were used as genetic operators. Mutation may happen anywhere inside the 4-symbol rule, ensuring that the production symbol ($\rightarrow$) is not distorted by mutation. In short, we want to make sure that after mutation, the production symbol is still in the rule (i.e., the rule is valid). Single-point crossover between two parents is executed at the location of the production symbol, ensuring that a valid rule is created as offspring. The evolutionary cycle ends after a predetermined number of generations.

VI. EXPERIMENTS

In [6], we investigated the ability of our representation to deal with problems using a computational function as fitness. This was achieved by assessing the evolvability of each architecture to find sufficiently good solutions when i. a separate genome is evolved for each architecture, and ii. a common genome is evolved for the architectures altogether. In this work, we study the ability of our representation to deal with problems of increased complexity. We do that by taking an experimental approach for the previous cases (i) and (ii). By increased complexity, we mean targeting problems that can be comparable to each other, have the same underlying specification and is possible to intuitively differentiate them in terms of their complexity, as described in Section III.

TABLE II
RULES TABLE FOR THE FINANCIAL MARKET MODEL

Left neighbor	Right Neighbor	Next state
buy	buy	buy
buy	sell	buy
sell	buy	buy
sell	sell	sell

A. Experimental setup

We develop a number of N rules for node generation and connectivity, depending on the size of the architecture. Here we develop two different architectures (1D-CA and BN), each holding sizes of a. $N = 144$ cells, b. $N = 169$ cells, and c. $N = 196$ cells. The architecture sizes were decided upon preliminary results. As such, three different experimental setups were being evaluated based on the architecture size. The number of rules being created were a. 32x144=4608 bits, b. 32x169=5408 bits, and c. 32x196=6272 bits. Each rule can be reused during development. Development runs for 100 timesteps for each individual.

The number of outgoing connections per node is $K = 5$. For more than 5 inputs/node, a self-connection to the originating node is created instead. *Generational mixing* protocol was used for the GA's global selection mechanism and *Rank selection* for parental selection. Unless otherwise stated, mutation rate was set to .009 and crossover rate to .001.

B. A Simple Financial Market Model

A simple financial market model is undertaken as the target problem. Each cell corresponds to an entity that either *buys* or *sells* on each step. The behavior of a given cell is determined by looking at the behavior of its two neighbors on the step before, according to the rules of Table II. As we can see, the rules correspond to the OR logical function. The computational goal (fitness) here is *all entities buy at the same timestep*.

We considered problem instances of the same problem, each having a higher level of complexity, i.e., an instance with no state memory and with a previous memory of 1-, 2-, 5- and 10-state. Based on Section III, each problem instance that is described next, holds an increased complexity as compared to all the previous ones. Using the experimental setup (Section VI-A), we run a set of 10 experiments of 1000 generations each. For each individual, a random initial environment was set and fed into the architecture (i.e., CA and BN).

1) No-State Memory Instance: For this model, the architectures hold no state memory (i.e., are not interested in the previous states, in order to decide upon the next cell state). Since we use two neighbors to decide upon the state of the cell for the next timestep, we say that this problem has a complexity of $n = 2$. This number by itself may not mean much, but in perspective with the problems of increasing complexity described next, it is clear that this problem holds the minimum complexity of all.

2) A 1-State Memory Instance: For this model (except the two current neighbors), the rules take also into consideration the state of the cell at the previous timestep (state memory 1), in order to decide upon the state of the current cell. The rules are based on the behavior (buy/sell) that dominate within the environment, using a simple majority decision. This problem has a complexity of $n = 3$, since we need information from three different sources before decision is made (two current neighbors & the state of the cell at the previous timestep).

3) A 2-State Memory Instance: For this model (except the two current neighbors), the rules take also into consideration the state of the cell during the two previous timesteps (state memory 2), in order to decide upon the state of the current cell. The rules are based again on the behavior (buy/sell) that dominate within the environment, using simple majority. Similarly, this problem has a complexity of $n = 4$.

4) A 5-State Memory Instance: For this model (except the two current neighbors), the rules take also into consideration the state of the cell during the five previous timesteps (state memory 5), in order to decide upon the state of the current cell. The rules are based again on the behavior (buy/sell) that dominate within the environment, using simple majority. This problem has a complexity of $n = 7$.

5) A 10-State Memory Instance: For this model (except the two current neighbors), the rules take also into consideration the state of the cell during the ten previous timesteps (state memory 10), in order to decide upon the state of the current cell. The rules here are based again on the behavior (buy/sell) that dominate within the environment, using simple majority. This problem has a complexity of $n = 12$.

C. Results

The results for the 1D-CA and BN, for $N = 144$, $N = 169$ and $N = 196$ cell architectures for the separate and commonly evolved genome, are shown at figures 1, 2 and 3, respectively.

For the 144-cell architecture, the common genome finds on average better solutions for the no-memory problem instance (figures 1(a) and 2(f)). In the case of state memory 1 and 2, we observe that although the common genome performs on average slightly worse than in the case of genomes evolved separately for each architecture, it exhibits a better exploratory trend; has a monotonically increasing fitness (figures 1(b), 1(c) and 1(f)). For the 5- and 10-state memory cases, both genome cases find best solutions with the commonly evolved genome case showing higher levels of robustness, i.e., mutations have smaller phenotypic effects (figures 1(d), 1(e), and 1(f)).

For the 169-cell architecture, the common genome demonstrated slightly worse solutions on average, with high robustness in the no-state memory casev(figures 2(a) and 2(f)). For 1- and 2-state memory, the commonly evolved genome obtained slightly worse solutions but shows a monotonically increasing fitness (figures 2(b), 2(c), and 2(f)). 5- and 10-state memory cases showed high levels of robustness for architectures evolved with a common genome where the solutions evolved with separate genomes were equally good (figures 2(d), 2(e), and 2(f)).

In the 196-cell architecture, the common genome showed better on average solutions for the no-state memory problem (figures 3(a) and 3(f)). The commonly evolved genome for the 1- and 2-state memory cases, demonstrated a monotonically increasing fitness with similarly good solutions like in the separately evolved genome case (figures 3(b), 3(c), and 3(f)). 5- and 10-state memory cases, exhibited on average similar solutions in both cases. Though, the commonly evolved genome showed highly robust solutions achieving similar solutions on average (figures 3(d), 3(e), and 3(f)).

VII. Discussion and future work

In this work we investigated the ability of common developmental genomes to evolve more than one specie (i.e., looser phenotypic constraints), towards problems with increasing complexity. The notion of complexity was approached from the viewpoint of Kolmogorov complexity. The architectures considered as different species were CAs and BNs and the problem studied was a simple financial market model over various architecture sizes, that is, 144-, 169- and 196-cell architectures with no-memory, 1-, 2-, 5-, and 10-state memory. The system showed consistent behavior over the architecture sizes, for the same problem instance. Generally, the commonly evolved genome in the no-memory, 5- and 10-state memory problems, showed a much more robust behavior and performed rather well. Even if the 5- and 10-state memory instances are considered more complex than the 1- or 2-state memory problems, they challenged the financial market model and succeeded in finding better solutions. It seems that the 5- and 10-previous cell state taken into account to decide upon the value of the current state, contributed to the robustness of evolution than being disruptive. On the other hand, 1- and 2-state memory instances showed better exploratory trend with monotonically increasing fitness. The financial market problem was sufficiently solved for the 144- and the 196-cell architecture sizes.

Overall, the cell architecture size did not seem to play any significant role in the exploratory capacity for our genetic representation and its ability to tackle problems of greater complexity. The results show that the common genome was able to find on average, equally good solutions but showed better results for the 144- and 196-cell architectures. Figures 1(f), 2(f) and 3(f) show the performance of the common developmental genome on average. If one wishes to look at the individual performance of the architectures (i.e., BN or CA), evolved with the common genome, their performance was highly superior in most of the cases (not shown).

The ability of the common genomes to drive evolution lies in the developmental process and may have a positive influence in directing evolution, i.e., common developmental genomes are the essence for what is called *developmental drive* [19]. Similarly, it may be that genomes evolved for each architecture separately, pose limitations to the developmental system in ways that prevent evolution from going in certain phenotypic directions (developmental bias [20]). Although these concepts are currently an area of active research for developmental

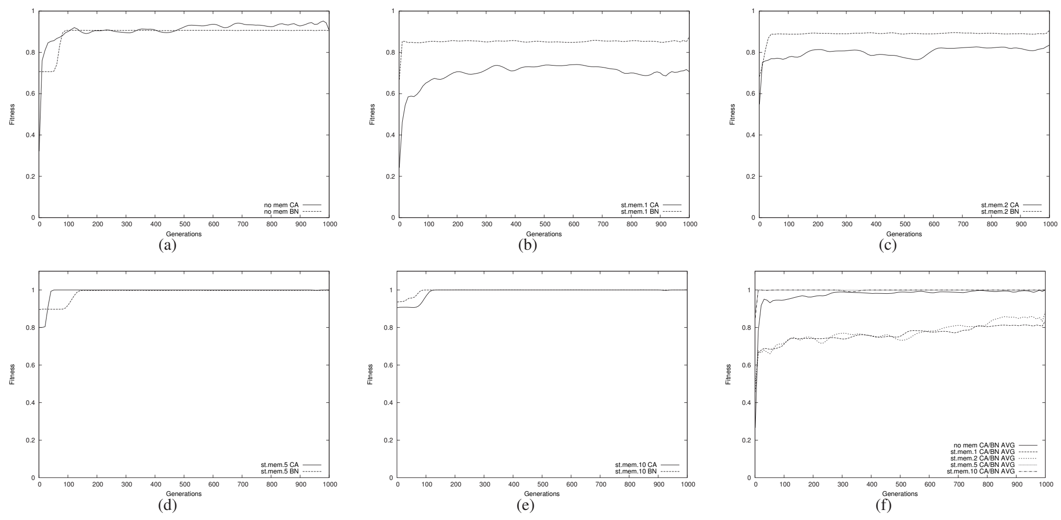


Fig. 1. Experiments for 144-cell architecture for genomes evolved separately, for (a) no state memory, (b) 1-state memory , (c) 2-state memory, (d) 5-state memory, (e) 10-state memory. The average fitness plots for commonly evolved genomes is shown at (f).

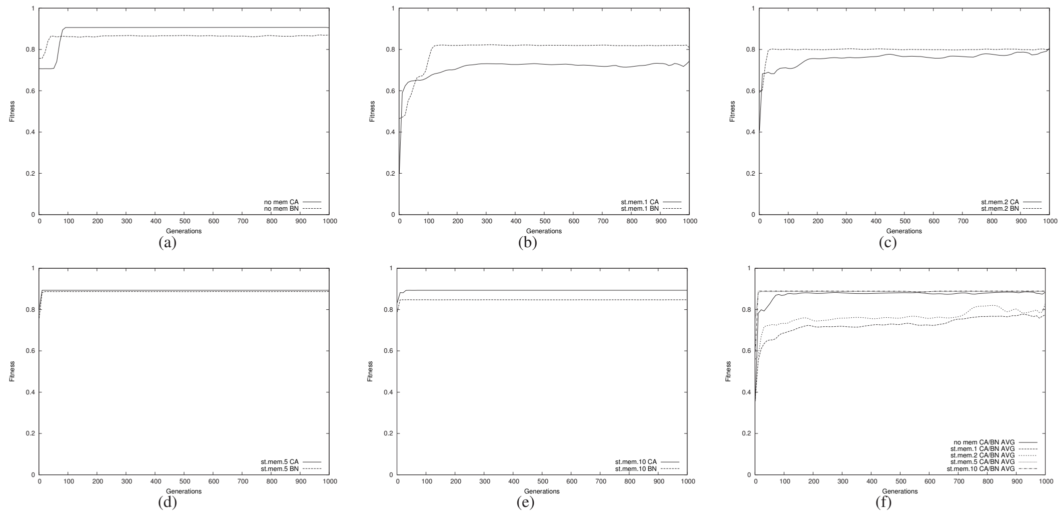


Fig. 2. Experiments for 169-cell architecture for genomes evolved separately, for (a) no-state memory, (b) 1-state memory, (c) 2-state memory, (d) 5-state memory, (e) 10-state memory. The average fitness plots for commonly evolved genomes is shown at (f).

biologists, we have every reason to believe that this might be the case in the artificial domain.

However, the results achieved in [6] and in this work, dictate a need for further investigation. The first step would be to identify relations between mutations and natural selection in the underlying genetic process under different environmental conditions, since environmental factors may affect the ontogenetic pathway. Second, – and indeed a more challenging step – is to discover inherent ontogenetic directionalities (i.e., dynamics) for the common developmental genomes during the stages of evolution. So, to answer the main question *"if common developmental genomes are panacea towards more complex problems?"* and by the results obtained so far, we can only tentatively agree. The final answer can't be put forward

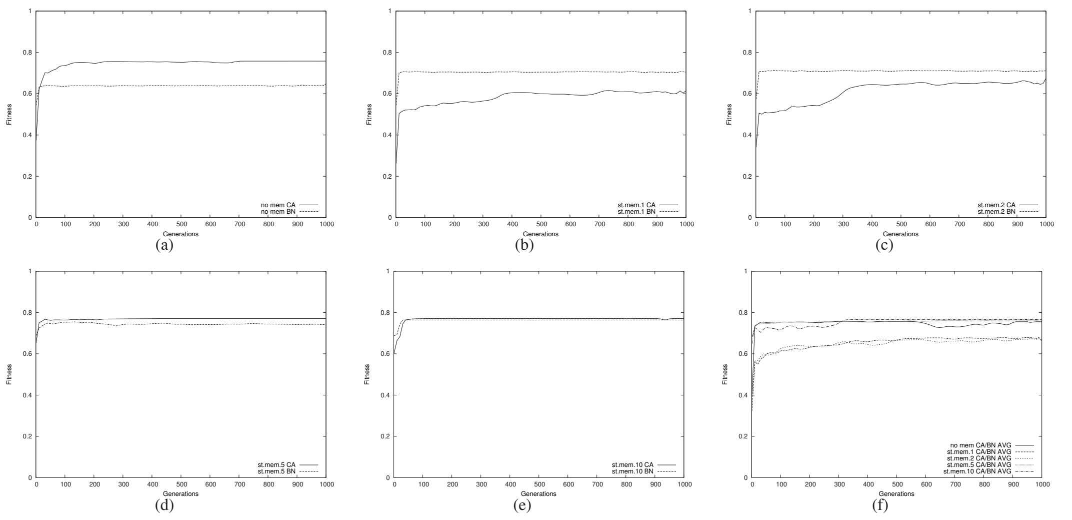


Fig. 3. Experiments for 196-cell architecture for genomes evolved separately, for (a) no-state memory, (b) 1-state memory, (c) 2-state memory, (d) 5-state memory, (e) 10-state memory. The average fitness plots for commonly evolved genomes is shown at (f).

unless these two steps are fully explored.

Additional future work include other ways of looking into the architectures, i.e., instead of looking at them as different species, they could be considered as organs of a common developing biological entity. That brings the case where architectures need to be merged (as is the case of biological organs), towards a. *hybrid architectures* with a common genome, and b. the ability to shape their phenotypes as modules, in order to change their dynamic properties (i.e., *phenotypic shaping*).

References

[1] K. Antonakopoulos and G. Tufte, "Possibilities and constraints of basic computational units in developmental systems," in *Norsk Informatikkonferanse, NIK-2009*, 2009, pp. 73–84.

[2] S. Gould and R. Lewontin, "The spandrels of san marco and the panglossian paradigm: a critique of the adaptionist programme." *Proceedings of the Royal Society of London*, vol. 205, no. 1161, pp. 581–598, Sep. 1979.

[3] J. S. Robert, *Embryology, Epigenesis and Evolution: Taking Development Seriously*, ser. Cambridge Studies in Philosophy and Biology. Cambridge University Press, 2004.

[4] J. Wilkins, "What is a species? essences and generation," *Theory in Biosciences*, vol. 129, no. 2, pp. 141–148, 2010.

[5] K. Antonakopoulos and G.Tufte, "A common genetic representation capable of developing distinct computational architectures," in *IEEE Congress on Evolutionary Computation (CEC)*, 2011, pp. 1264–1271.

[6] ——, "On the evolvability of different computational architectures using a common developmental genome," in *4th Int'l Conf. on Evolutionary Computation Theory and Applications (ECTA 2012)*, 2012, pp. 122–129.

[7] C. Adami, "What is complexity ?" *Bioessays*, vol. 12, no. 24, pp. 1085–94, 2002.

[8] S. Lloyd and H. Pagels, "Complexity as thermodynamic depth," *Annals of Physics*, vol. 1, no. 188, pp. 186–213, 1988.

[9] J. Crutchfield and K. Young, "Inferring statistical complexity," *Physica Review Letters*, vol. 2, no. 63, pp. 105–108, 1989.

[10] C. Papadimitriou, *Computational Complexity*. Addison-Wesley, 1993.

[11] D. McShea, "Metazoan complexity and evolution: is there a trend?" *Evolution*, vol. 50, no. 2, pp. 477–492, 1996.

[12] ——, "Functional complexity in organisms: Parts as proxies," *Biology and Philosophy*, vol. 15, pp. 641–668, 2000.

[13] K. Korb and A. Dorin, "Evolution unbound: releasing the arrow of complexity," *Biology and Philosophy*, vol. 26, pp. 317–338, 2011.

[14] C. Adami and N. Cerf, "Physical complexity of symbolic sequences," *Physica D*, vol. 137, pp. 62–69, 2000.

[15] M. Li and P. Vitányi, *An Introduction to Kolmogorov Complexity and Its Applications*. Springer-Verlag, New York, 2nd edition, 1997.

[16] P. Lehre, "Complexity and geometry in artificial development," Ph.D. dissertation, Dept.of Computer and Information Science, NTNU, Trondheim, Norway, 2006.

[17] A. Lindenmayer and P. Prusinkiewicz, "Developmental models of multicellular organisms: A computer graphics perspective," in *Artificial Life: Proceedings of an Interdisciplinary Workshop on the Synthesis and Simulation of Living Systems*, C. G. Langton, Ed. Addison-Wesley Publishing Company, 1989, pp. 221–249.

[18] A. Staffer and M. Sipper, "Modeling cellular development using l-systems," in *Second International Conference on Evolvable Systems (ICES98)*, ser. Lecture Notes in Computer Science. Springer, 1998, pp. 196–205.

[19] A. Wallace, "Developmental drive: an important determinant of the direction of phenotypic evolution." *Evolution & Development*, vol. 3, no. 4, pp. 271–278, 2001.

[20] R. Raff, "Evo-devo: The evolution of a new discipline." *Nature Reviews in Genetics*, vol. 1, pp. 74–79, Oct. 2000.

On The Evolvability of Different Computational Architectures Using a Common Developmental Genome

K. Antonakopoulos and G. Tufte

4th International Conference on Evolutionary Computation Theory and Applications
Barcelona, Spain, October 5-7, 2012

Abstract

Artificial organisms comprise a method that enables the construction of complex systems with structural and/or computational properties. In this work we investigate whether a common developmental genome can favor the evolvability of different computational architectures. This is rather interesting, especially when limited computational resources is the case. The commonly evolved genome showed ability to boost the evolvability of the different computational architectures requiring fewer resources and in some cases, finding better solutions.

On The Evolvability of Different Computational Architectures Using a Common Developmental Genome

Konstantinos Antonakopoulos and Gunnar Tufte
Department of Computer and Information Science, Norwegian University of Science and Technology, Sem Sælandsvei 7-9, NO-7491, Trondheim, Norway
{kostas, gunnart}@idi.ntnu.no

Keywords: Genetic Representation: Cellular Automata: Boolean Network: L-systems: Evolvability.

Abstract: Artificial organisms comprise a method that enables the construction of complex systems with structural and/or computational properties. In this work we investigate whether a common developmental genome can favor the evolvability of different computational architectures. This is rather interesting, especially when limited computational resources is the case. The commonly evolved genome showed ability to boost the evolvability of the different computational architectures requiring fewer resources and in some cases, finding better solutions.

1 INTRODUCTION

Artificial systems often target organisms or systems with some kind of functionality. Target functionality may include systems aiming to solve problems, be it structural (Steiner et al., 2009), or computational (Harding et al., 2007). In the first case, the target is the structure itself. In the second case, the target is a functionality given by the computational function of the nodes and their connections.

Targeting problems with a structural versus a computational goal is quite different. Most EvoDevo (i.e., Evolutionary Developmental) systems consist of a genotype targeting special phenotypic structures (i.e., structures that comprise connected computational elements) - even when the target is a computational function. The structural property of the phenotype may reduce the solution efficiency or even the computational goal that can be achieved.

To be able to develop such phenotypic structures, there is a need to further understand the properties of the targeted computational architectures. An analysis of the possibilities and constraints involved in the development of various computational architectures, focusing on the form, functionality and the inherent biological properties, was presented in (Antonakopoulos and Tufte, 2009). The architectures studied therein were Boolean Networks (BN) (Kauffman, 1993), Artificial Neural Networks (ANN) (Astor and Adami, 2000), Cellular Automata (CA) (Tufte and Haddow, 2005), and Cellular Neural Networks (CNN) (Chua and Yang, 1988). The common property of these architectures is that they are considered as *sparsely-connected networks*. This common property motivates a further investigation on how a mapping process can work on a class of architectures in a more general way towards complex problems. This is elaborated by examining how universal properties and processes can be included in a development mapping, through an *EvoDevo* approach (Robert, 2004).

In biology, a specie is often used as a basic unit for biological classification and for taxonomic ranking. Species are individuals sharing the same genetic, developmental and ecological processes (Wilkins, 2010). Inspired by multicellularity and the organisms' ability to exploit different developmental paths based on environmental factors, we explored the potentiality of using the same developmental mapping to develop not a specific, but different classes of architectures (i.e., species), using a common genetic representation (Antonakopoulos and Tufte, 2011).

The hypothesis for this work is to see whether common developmental genomes can prove beneficial over developmental genomes evolved for each specie, separately, towards complex computations. To see if the hypothesis holds, two things should be further investigated. First, whether the same mapping (i.e., a common developmental genome), can favor the evolvability of different computational architectures under the same environment when resources are limited. Second, if the same developmental mapping can favor the evolvability of different computational architectures (i.e., CA and BN), with a focus in problems of increasing complexity. Only then, we will have concrete evidence for our hypothesis.

The first step mentioned above, comprise also the motivation of this work. The second step will be examined and published elsewhere. Here, we are studying the same computational architectures (i.e., CA and BN), as different species. The computational architectures in this setup will have limited computational resources to evolve. It is of interest then to investigate whether common developmental genomes can favor the evolvability of these architectures, as opposed to genomes evolved separately for each architecture.

The rest of the article is laid out as follows. Section 2 addresses the challenges involved in such a developmental model. The common genetic representation is given at section 3. Experiments come in section 4 with the conclusion and future work in section 5.

2 DEVELOPMENT FOR SPARSELY-CONNECTED COMPUTATIONAL STRUCTURES

The developmental goal is to be able to generate not a specific, but different classes of structures (i.e., species), using the same developmental model. This should be achieved through the same developmental approach. Such developmental approach requires sufficient knowledge of the targeted computational architectures and of their governing properties. That is, for the 2-dimensional CA, the properties of dimensionality and neighborhood must be defined, where the connectivity is predetermined (i.e., the Euclidean space). For boolean networks, the connectivity (i.e., the node connections of the network), must be determined. The problem just described can be better expressed as *three-challenge* problem: (a) the *genome* challenge, (b) the *developmental processes* involved in the model, and (c) the *developmental model* challenge.

2.1 The Genome Challenge

Based on the properties of a 2D-CA, the genome contains information about the cells at each developmental step, in order to place them on a 2D-CA lattice structure. The wiring of the cell is given by the CA's neighborhood. At the same time and based on the properties of a boolean network, the genome contains enough information to feed the developmental model to develop a boolean network, at each developmental step.

2.2 The Developmental Processes Challenge

The resulting structure is able to grow, alter the functionality of a cell/node, and shrink. These processes are introduced in the developmental mapping through *growth*, *differentiation*, and *apoptosis* (i.e., the death of the cell/node). Having these properties in mind, our genome incorporates the notion of *chromosomes* - inspired by biology. Each chromosome contains respective information about the structural and/or functional requirements. More specifically, a chromosome will contain the information required for the cell/node creation (i.e., for the CAs and BNs), where another chromosome will contain the information required for wiring the nodes (i.e., for BNs). The notion of chromosomes allows us to exploit the genome in a modular way in the sense that if an additional computational architecture need to be described through the same genome, more chromosomes can be added to it.

2.3 The Developmental Model Challenge

The developmental model is able to develop these structures, taking into account the special properties employed by each architecture. The developmental model receives the same genome as input, regardless of the target architecture. Then, it is possible – depending on some properties of the genome – to discriminate whether it will develop a CA or a BN.

3 THE COMMON GENETIC REPRESENTATION

In biology, a specie is often used as the basic unit for biological classification and for taxonomic ranking. As such, an organism with unifying properties and same characteristics can be of the same specie. Figure 1, show how the genome looks like. The genome is split into two parts (*chromosomes*). The first chromosome is responsible for creating the cells/nodes. The second chromosome is responsible for creating the connectivity (i.e., for the BNs). Each chromosome is built out of rules. Each rule has sufficient information for cell/node creation and connectivity. Also, the rules are of certain length. Those destined for cell/node creation are different from the ones for connectivity. Consequently, chromosomes contain different rules.

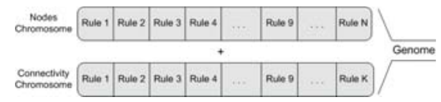


Figure 1: This is how the genome looks like with the genome split into two (*chromosomes*). The first chromosome is responsible for generating the cells/nodes whereas the second chromosome is responsible for generating the connectivity of the network.

3.1 An L-system for the Genetic Representation

A rewriting approach was chosen due to the ease of defining specific rule set, that can target to rewrite specific features of a structure, e.g., connections or node functions that enable a way of splitting genetic information into separate information carrying units (i.e., chromosomes).

A prominent model is L-systems. They are rewriting grammars, able to describe developmental systems, simulate biological processes (Lindenmayer and Prusinkiewicz, 1989), and describe computational machines (Staffer and Sipper, 1998). Since there are different types of rules in the two chromosomes, there is a need for separate L-systems. The first L-system processes the rules of the first chromosome, while the second L-system deals with the connectivity rules of the second chromosome.

3.2 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. Table 1(a), shows the type of symbols used by the L-system of the first chromosome. Some cells perform special cell processes and influence the intermediate and final phenotypes. Symbol *a* is the *axiom*. Apart from the symbols *a*, *b*, and *c*, which perform *growth* of the phenotype, symbol *d* performs *apoptosis*, leading to the deletion of the current rule (i.e., cell/node), of the intermediate phenotype. Additionally, symbols `X` and `Y`, are responsible for *differentiation*, leading to the replacement of the predecessor cell/node (i.e., if `X`$\rightarrow$`Y` the outcome will be `Y`, whereas, if `Y`$\rightarrow$`X` the outcome will be `X`). For the shake of simplicity, the length of each rule is 4 symbols (i.e., `4x8bits=32bits`). For node/cell generation the L-system runs for 100 timesteps and then stops. As such, the intermediate phenotypes generated by development are of variable size.

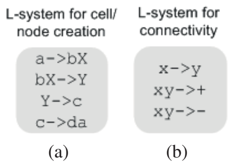


Figure 2: (a) Example of L-system rules for the first chromosome, (b) Example of L-system rules for the second chromosome.

Figure 2(a), gives an example of a L-system for the first chromosome. A simple example with step-by-step development of a 2D-CA architecture is illustrated at figure 3. Development starts with the axiom (`a`) representing a cell at developmental step (DS) 0. Since the axiom is found in the L-system rules, development continues and the next rule triggered is the `a`$\rightarrow$`bX`. This rule will create two more cells `b` and `X`, resulting in growth of the CA, at DS 1. The next rule triggered is `bX`$\rightarrow$`Y`. Since `X`$\rightarrow$`Y` denotes differentiation, the symbol `X` is replaced by `Y`, at DS 2. For differentiation to occur, the rules should either be `X`$\rightarrow$`Y` or `Y`$\rightarrow$`X`. Next, rule `Y`$\rightarrow$`c` triggers causing again growth of the CA, at DS 3. At DS 4, the rule `c`$\rightarrow$`da` is triggered causing the death of the cell `c` and the growth of the CA with the cell `a`. From DS 5 up to DS 8, rules are being triggered once more in the same sequence.

3.3 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; the current node has always the number zero and any nodes starting from the current node forward have positive numbering, where nodes that exist from the current node backwards, have negative numbering. So, there is a need to differentiate between the current and the next node, using different symbols and also whether a connection will be created forward or backward from the current node.

The rules involved in connectivity are not as complex as those of first chromosome. The length of the rules here is also 4 symbols / rule. Also, there is a need to assure that the chromosome will have sufficient information for the developmental processes (i.e., growth, differentiation and apoptosis). The L-system uses is a D0L (i.e., with zero-sided interac-

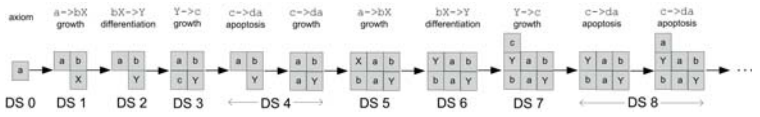


Figure 3: A step-by-step development of a 2D-CA architecture based on the example L-system for the first chromosome.

tions). An example L-system for the second chromosome is shown at figure 2(b), and the symbols used are explained at Table 1(b).

Table 1: (a) Symbol table for Node generation, (b) Symbol table for Connectivity generation.

(a)

Symbol	Description
a (AXIOM)	Add (growth)
b	Add (growth)
c	Add (growth)
d	Delete (apoptosis)
X	Substitute (differentiation)
Y	Substitute (differentiation)
$\rightarrow$	Production

(b)

Symbol	Description
x	Node (different from y)
y	Node (different from x)
+	Connect forward
–	Connect backwards
$\rightarrow$	Production

The *axiom* rule for the second chromosome is `x`$\rightarrow$`y`. It means that development initially searches if the axiom exists. If so, development continues and looks for rules of type `xy`$\rightarrow$`+value` or `xy`$\rightarrow$`-value`. In short, these two rules imply that if two different (i.e., distinct) nodes are found (`x`$\neq$`y`), then it creates a connection forward (if the rule includes a '+'), or backwards (if the rule includes a '-'). The field `value` is encoded in the genotype and denotes the node number for the generated connection. For example, rule `xy`$\rightarrow$`+3` denotes that a connection will be created from the current node (node `0`), to the one being three nodes forward. Similarly, rule `xy`$\rightarrow$`-3` denotes that a connection will be created starting from the current node (node `0`), to the one that is three nodes backwards. If `value=0`, a self-connection is created to the current node. A step-by-step development of a boolean network based on the chromosomes of Table 1(a) and 1(b), can be found at (Antonakopoulos and Tufte, 2011) and is not shown here due to page limitation. The modularity of the genome, gives the possibility to development itself to enable or disable parts of it (chromosomes), when this is required and driven by the goal set. For example, if the target architecture is a 2D-CA, the second chromosome (i.e., connectivity) is disabled, since connectivity is predetermined. Similarly for BN development, both chromosomes are enabled (i.e., nodes and connectivity).

3.4 The Genetic Algorithm for the Common Genetic Representation

A genetic algorithm is used to generate and evolve the rules found in the genome (i.e., in the chromosomes). Since there are two separate L-systems involved in development, the evolutionary process comprise two phases: node and connectivity generation. Mutation and single-point crossover were used as genetic operators. Mutation may happen anywhere inside the 4-symbol rule, ensuring that the production symbol ($\rightarrow$) is not distorted by mutation. That is, we want to make sure that after mutation, the production symbol still exists in the rule (i.e., the rule is valid). Single-point crossover is performed at the location of the production symbol, ensuring that a valid rule is created as offspring. The evolutionary cycle ends after a predetermined number of generations.

4 EXPERIMENTS

In (Antonakopoulos and Tufte, 2011), we investigated the ability of the representation to evolve different computational architectures using a structured-based fitness. Here we study the ability of our representation to deal with problems using a computational fitness; we take an experimental approach using the same genetic representation on both architectures (CA and BN), towards sufficient solutions when: i. a separately genome is evolved for each of the architecture, and ii. a common genome is evolved for architectures altogether.

4.1 Experimental Setup

We use a total number of 36 rules for node generation and for connectivity (i.e., `32x36=1152bits`). It is important to note that a rule can be reused during development. Development runs for 100 timesteps for each individual. The evaluation of the phenotypes for the CA and the BN is given by the cell types of Table 2.

A 6x6 2D-CA and a $N = 36$ BN is used. The reason is that the two architectures must to be compara-

Table 2: Cell types and their functionality.

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

ble; have the same state space (i.e., 2^{36}). The number of outgoing connections per node is $K = 5$. For more than 5 inputs/node, a self-connection to the originating node is created instead. *Generational mixing* was used as global selection mechanism and *Rank selection* for parental selection. Unless otherwise stated, mutation rate was set to .0005 and crossover rate to .001.

4.2 Search for Cycle Attractors

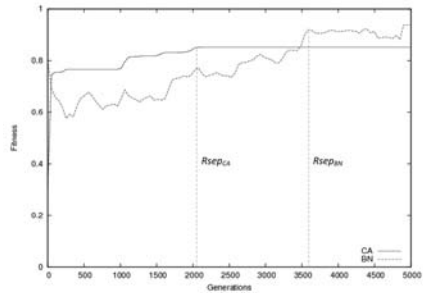
Using the experimental setup described in section 4.1, we run a set of 10 experiments of 5000 generations each. For each individual, a random initial state was created and fed into the architecture. The fitness function gives credit for cycle attractors between 2-21; the best score is assigned for cycle attractors of size 11.

Figure 4(a) shows the average fitness plots over the 10 runs for genomes evolved separately. The BN managed to find sufficiently good solutions using a large amount of available resources ($Rsep_{BN}$). The CA found also rather good solutions, needing less than half of the available resources ($Rsep_{CA}$).

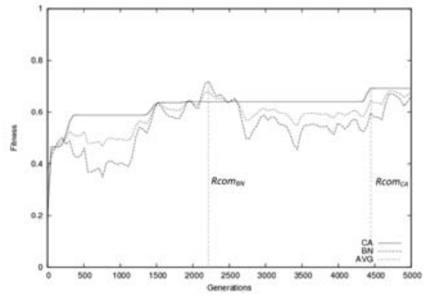
The average fitness plot over the 10 runs for commonly evolved genomes is shown at figure 4(b). In this case, genomes were able to find reasonable solutions. The BN achieved a max average of 70% fitness using half of the available resources ($Rcom_{BN}$), where the CA achieved a fitness of 68%, acquiring almost all of the resources ($Rcom_{CA}$).

4.3 Search for Transient Phase and Attractors

Using the same setup, we also run a set of 10 experiments of 5000 generations each. For each individual, a random initial state was created and fed into the architecture. The fitness function gives credit for transients with a maximum size of 10 after which an attractor of maximum size of 20 must follow. Point attractors are also being taken into account (cycles of 1). The transient phase has a range between 1-10 (best score is assigned for transients of size 5), where attractors have a range between 1-21 (best score is assigned for attractors of size 11). Both fitness param-



(a)



(b)

Figure 4: Cycle Attractor experiment: (a) Averaged fitness plot for the different architectures with genomes evolved separately, (b) Averaged fitness plot for different architectures with a commonly evolved genome.

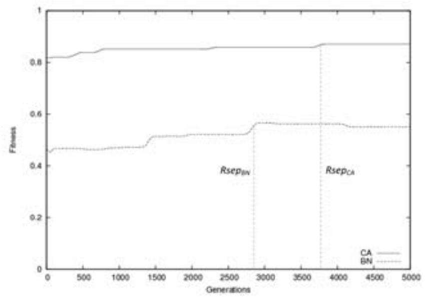
eters (i.e., transient phase and attractors) are normalized into half and their partial scores were summed to give the final score.

Figure 5(a) shows the average fitness plot over 10 runs for genomes evolved separately. The CA was able to find a sufficient solution, requiring a large amount of resources ($Rsep_{CA}$). The BN was able to find only fair solutions acquiring more than half of the resources available ($Rsep_{BN}$).

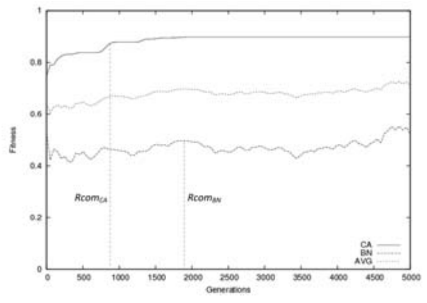
In the case of the commonly evolved genome of figure 5(b), both architectures were able to achieve similar performance as previously, but consumed significantly less resources. The CA reached a max average fitness of 86% at generation 800 ($Rcom_{CA}$), where the BN reached a 55% fitness at generation 1750 ($Rcom_{BN}$).

4.4 Synchronization Task

For this task, the goal is to find a CA that given any initial configuration s within M time steps, reaches a final configuration that oscillates between all zeros and all ones on successive time steps. M, the desired

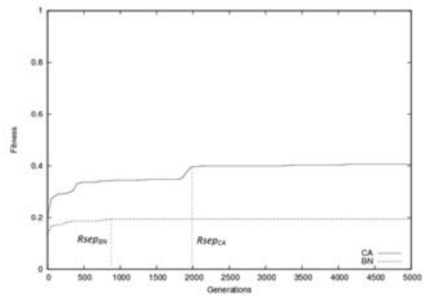


(a)

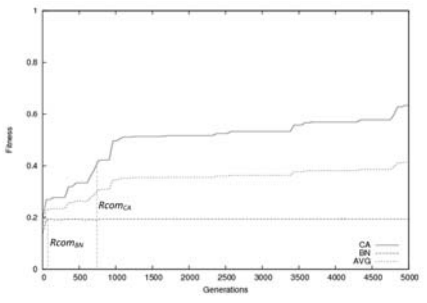


(b)

Figure 5: Transient with attractors experiment: (a) Averaged fitness plot for the different architectures with genomes evolved separately, (b) Averaged fitness plot for different architectures with a commonly evolved genome.



(a)



(b)

Figure 6: Synchronization task experiment: (a) Averaged fitness plot for the different architectures with genomes evolved separately, (b) Averaged fitness plot for different architectures with a commonly evolved genome.

upper bound on the synchronization time, is a parameter of the task that depends on the lattice size (Das et al., 1995). Here, we relaxed the rule of *all ones and all zeros* by introducing a *synchronization threshold*. It means that we may have configurations of zeros or ones up to the *threshold* limit. In this case, this threshold is set to 80%. This implies that configurations filled up with 80% zeros or ones are eligible as target configurations. Here, a 1D-CA and a BN of size 36 is used, with the mutation rate being .002 and the crossover rate .001. Each individual is developed for 1000 timesteps.

Figure 6(a) shows the average fitness plot over 10 runs for the separately evolved genomes case. The CA was able to achieve a max average fitness of 40% at generation 2000 ($Rsep_{CA}$), where the BN gave moderate solutions (18%) at generation 850 ($Rsep_{BN}$).

In the case of commonly evolved genomes at figure 6(b), both architectures needed fewer resources to achieve the same results as in the separately evolved genome case. As such, the CA reached the same fitness at generation 750 ($Rcom_{CA}$), where the BN reached the fitness of 18% at generation 90 ($Rcom_{BN}$). In addition to that, the commonly evolved genome achieved better overall fitness; the CA reached an average of 60% and the BN an average of 20% (for the total of the available resources).

5 CONCLUSION AND FUTURE WORK

In this work, we investigated whether and when commonly evolved genomes favor the evolvability of different computational architectures, as opposed to genomes evolved separately for each architecture. The computational architectures targeted herein were a 6x6 non-uniform 2D-CA and a BN of size $N = 36$ and were considered as different species. In addition, the different genome cases evolved in a setup with only limited computational resources.

The search for a cycle attractor problem showed that the common genome was not able to show favorable results for the development of the architectures.

The separately evolved genome was able to give partially better results; the CA was able to evolve requiring less than half of the resources available.In the case of transient phase and attractor search, both genome cases ended up with a similar fitness performance but the commonly evolved genome consumed considerably fewer resources. In the synchronization task problem, the commonly evolved genome was able to evolve the architectures acquiring again less resources than in the separately evolved genome case.

The construction of a common genome able to develop different computational architectures proved to be beneficial. There are cases where resources are not infinitely available or not available at the given moment. Artificial organisms need to have ways to overcome such problems if they are to continue to evolve at all (much like in the nature). Commonly evolved genomes boosted the evolvability of the architectures. In two of the experiments we studied, it required considerably fewer resources than in the case of the genome evolved separately. The reason behind the superiority of the common developmental genomes is somewhat intuitive. Common genomes in this configuration, involve two fitness functions (i.e., one for node generation and another for connectivity), defining a set of optimal solutions over each evolutionary cycle. So, we can say that the nodes genome stands as an ideal source of information for the connectivity genome and ultimately, the development of the phenotypes.

But there is more to that. It may be that common developmental genomes are more amenable to developmental drive (Wallace, 2001). Or, they may have a positive influence in directing evolution and pushing the developmental system in phenotypic directions where it would have been impossible to achieve with ordinary genomes (i.e., genomes evolved separately for each architecture at hand). The latter is identified as *developmental bias* (Raff, 2000). To conclude, more research needs to be done towards the identification of: *i.* potential relations between mutation and selection in the underlying genetic process, and *ii.* inherent ontogenetic directionalities (i.e., dynamics) for common developmental genomes, during the stages of evolution.

Closing, the notion of chromosomes in our representation, allows us to exploit the genome in a modular way in a sense that if additional computational architectures need to be incorporated in the future and expressed by the same genome, more chromosomes can be attached. Changing the way of looking into the architectures, i.e., instead of looking at them as different species, we could consider them as organs of a common developing biological entity. That brings up a case where architectures need to be merged (as is the case in biological organs). Since the overall goal of this work is to target more adaptive scalable systems able of complex computation, the exploration of these merged computational architectures (i.e., *hybrid architectures*) with the same genome and developmental model but even further, the ability to shape the phenotype of our system (*phenotypic shaping*) as modules in order to change the dynamic properties of the entire system, paves the way for promising future research.

REFERENCES

Antonakopoulos, K. and Tufte, G. (2009). Possibilities and constraints of basic computational units in developmental systems. In *Norsk Informatikkonferanse, NIK-2009*, pages 73–84.

Antonakopoulos, K. and Tufte, G. (2011). A common genetic representation capable of developing distinct computational architectures. In *IEEE Congress on Evolutionary Computation (CEC)*, pages 1264–1271.

Astor, J. C. and Adami, C. (2000). A developmental model for the evolution of artificial neural networks. *Artificial Life*, 6(3):189–218.

Chua, L. and Yang, L. (1988). Cellular neural networks: theory. *IEEE Transactions on Circuits and Systems*, 35(10):1257–1272.

Das, R., Crutchfield, J. P., Mitchell, M., and Hanson, J. E. (1995). Evolving globally synchronized cellular automata. In Eshelman, L. J., editor, *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 336–343. Morgan Kaufmann Publishers Inc.

Harding, S. L., Miller, J. F., and Banzhaf, W. (2007). Self-modifying cartesian genetic programming. In *GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 1021–1028, New York, NY, USA. ACM.

Kauffman, S. A. (1993). *The Origins of Order*. Oxford University Press.

Lindenmayer, A. and Prusinkiewicz, P. (1989). Developmental models of multicellular organisms: A computer graphics perspective. In Langton, C. G., editor, *Artificial Life: Proceedings of an Interdisciplinary Workshop on the Synthesis and Simulation of Living Systems*, pages 221–249. Addison-Wesley Publishing Company.

Raff, R. (2000). Evo-devo: The evolution of a new discipline. *Nature Reviews in Genetics*, 1:74–79.

Robert, J. S. (2004). *Embryology, Epigenesis and Evolution: Taking Development Seriously*. Cambridge Studies in Philosophy and Biology. Cambridge University Press.

Staffer, A. and Sipper, M. (1998). Modeling cellular development using l-systems. In *Second International*

Conference on Evolvable Systems (ICES98), Lecture Notes in Computer Science, pages 196–205. Springer.

Steiner, T., Trommler, J., Brenn, M., Jin, Y., and Sendhoff, B. (2009). Global shape with morphogen gradients and motile polarized cells. In *Congress on Evolutionary Computation(CEC2009)*, pages 2225–2232.

Tufte, G. and Haddow, P. C. (2005). Towards development on a silicon-based cellular computation machine. *Natural Computation*, 4(4):387–416.

Wallace, A. (2001). Developmental drive: an important determinant of the direction of phenotypic evolution. *Evolution & Development*, 3(4):271–278.

Wilkins, J. (2010). What is a species? essences and generation. *Theory in Biosciences*, 129(2):141–148.

Investigation of Developmental Mechanisms in Common Developmental Genomes

K. Antonakopoulos and G. Tufte

7th International Conference on Bio-Inspired Models of Network,
Information, and Computing Systems
Lugano, Switzerland, December 10-11, 2012

Abstract

The potentiality of using a common developmental mapping to develop not a specific, but different classes of architectures (i.e., species), holding different structural and/or computational phenotypic properties is an active area of research in the field of bio-inspired systems. To be able to develop such species, there is a need to understand the governing properties and the constraints involved for their development. In this work, we investigate how common developmental genomes influence evolution and how they push the developmental process in directions where it would have been impossible to achieve with ordinary genomes. Relations between mutation and evolution along with a comprehensive study of developmental mechanisms involved in development are worked out. The results are promising as they unveil that common developmental genomes perform better in more complex and random environments.

Investigation of Developmental Mechanisms in Common Developmental Genomes

Konstantinos Antonakopoulos and Gunnar Tufte

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

Abstract. The potentiality of using a common developmental mapping to develop not a specific, but different classes of architectures (i.e., species), holding different structural and/or computational phenotypic properties is an active area of research in the field of bio-inspired systems. To be able to develop such species, there is a need to understand the governing properties and the constraints involved for their development. In this work, we investigate how common developmental genomes influence evolution and how they push the developmental process in directions where it would have been impossible to achieve with ordinary genomes. Relations between mutation and evolution along with a comprehensive study of developmental mechanisms involved in development are worked out. The results are promising as they unveil that common developmental genomes perform better in more complex and random environments.

Key words: Common developmental genomes, evolvability, cellular automata, boolean network, L-systems

1 Introduction

The importance of development in life is crucial since it enables multicellular organisms to grow in well-defined stages. Besides 'direct-development', which is the simplest form of development, one can also find 'indirect development' through which the organism changes radically. These changes impose a new kind of adult organism over a period of evolutionary generations [1]. In the artificial analog, one can find a "simplified" artifact comprised of a genotype targeting a special phenotypic structure or other computational goal (a.k.a., Evo-Devo systems).

Taking a step further, there is a possibility of creating a genome (i.e., a genetic representation), which can be common for more than one phenotypes. In previous work, we studied whether the same mapping (i.e., common developmental genomes), can favor the evolvability of different computational architectures under the same (single-cell) environment i. with limited resources [2], and ii. in problems with increasing complexity [3].

The ability of common developmental genomes to drive evolution lies in the developmental process and has a positive influence in directing evolution, as it has been seen so far. In other words, common developmental genomes may be the essence for what is called *developmental drive* [4]. Though, it is not yet clear how common developmental genomes influence evolution and how they push the developmental process in directions where it would have been impossible to achieve with ordinary genomes (i.e., genomes evolved separately for each architecture at hand). Also, it is still unclear whether and in what way the environment affects the ontogenetic pathways of development. Therefore, further research is needed towards three aspects. First, how genetic operators (i.e., mutation) affect evolution (i.e., selection) in common developmental genomes, as they are part of the "orienting mechanism" of both short-term and long-term evolution. Second, whether development and the developmental dynamics of common developmental genomes prescribe a certain pathway for evolution. Third, it is interesting to see if the external environment is at all important to the final phenotype.

The motivation for this work is to identify potential relations between mutation and selection in the underlying genetic process, discover inherent ontogenetic directionalities during the stages of evolution, and see whether environmental conditions affect the outcome in some way.

The rest of the article is laid out as follows. Section 2 describes the challenges involved in designing such a developmental model. The common genetic representation is given in section 3. Detailed explanation of the genetic representation and developmental model can be found in [5]. The definition and structure of the environment is given in section 4. Experiments come in section 5 with the conclusion and future work in section 6.

2 Development for Sparsely-Connected Computational Structures

The developmental goal is to be able to generate not a specific, but different classes of structures (i.e., species), using the same developmental model. This should be achieved through the same developmental approach. Such developmental approach requires sufficient knowledge of the targeted computational architectures and of their governing properties. That is, for the 2-dimensional cellular automata (CA) architecture, the properties of dimensionality and neighborhood must be defined, where the connectivity is predetermined (i.e., the Euclidean space). For boolean networks (BN), the connectivity (i.e., the node connections of the network), must be determined. The problem just described can be better expressed as *three-challenge* problem: (a) the *genome* challenge, (b) the *developmental processes* involved in the model, and (c) the *developmental model* challenge.

2.1 The Genome Challenge

Based on the properties of a 2D-CA, the genome contains information about the cells at each developmental step, in order to place them on a 2D-CA lattice structure. The wiring of the cell is given by the CA's neighborhood. At the same time and based on the properties of a boolean network, the genome contains enough information to feed the developmental model to develop a boolean network, at each developmental step.

2.2 The Developmental Processes Challenge

The resulting structure is able to grow, alter the functionality of a cell/node, and shrink. These processes are introduced in the developmental mapping through *growth*, *differentiation*, and *apoptosis* (i.e., the death of the cell/node). Having these properties in mind, our genome incorporates the notion of *chromosomes* - inspired by biology. Each chromosome contains respective information about the structural and/or functional requirements. More specifically, a chromosome will contain the information required for the cell/node creation (i.e., for the CAs and BNs), where another chromosome will contain the information required for wiring the nodes (i.e., for BNs). The notion of chromosomes allows us to exploit the genome in a modular way in the sense that if an additional computational architecture needs to be described through the same genome, more chromosomes can be added to it. To better illustrate how these processes will influence the developing structure, Figure 1 shows the three developmental processes as applied to a developing cellular automata.

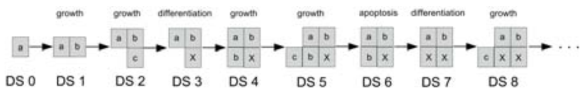


Fig. 1. The developmental model should be able to incorporate the processes of growth, differentiation and apoptosis. Here, each of the processes are illustrated as the model develops step-by-step a 2D cellular automata.

2.3 The Developmental Model Challenge

The developmental model is able to develop these structures, taking into account the special properties employed by each architecture. Figures 2 and 3, illustrate this requirement. The developmental model receives the same genome as input, regardless of the target architecture. Then, it is possible – depending on some properties of the genome – to discriminate whether it will develop a CA or a BN.

Figure 2, visualizes this by showing step-by-step the development of a cellular automata from the developmental model. At DS 0, the first cell of the cellular automata is created. At DS 1, the cellular automata grows in size and a new cell is added. At DS 2, the architecture grows again by adding one more cell to the cellular automata. At DS n, development has stop and the cellular automata has its final structure (adult organism).

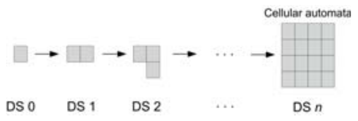


Fig. 2. The developmental model should be able to develop a cellular automata. Here, the development of the automata is shown from developmental step (DS) 0 until the final developmental step (DS) *n*.

Similarly, Figure 3 presents step-by-step the development of a boolean network with the same developmental model. At DS 0, the first node with its self-connections is created. At DS 1, the boolean network grows in size and a new node is added to the network. This will cause new connections to be created for all the nodes existing in the network. At DS 2, the network adds another node and new connections are created for the existing nodes. This algorithm continues until the boolean network has created all the nodes and the connections for the existing nodes at DS *n*).

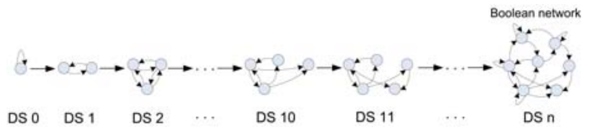


Fig. 3. The developmental model should be able to develop a boolean network. Here, the development stage of a BN is shown from developmental step (DS) 0, until the last developmental step (DS) *n*.

3 The Common Genetic Representation

In biology, a specie is often used as the basic unit for biological classification and for taxonomic ranking [6]. As such, an organism with unifying properties and same characteristics can be of the same specie. Figure 4, shows how the genome looks like. The genome is split into two parts (*chromosomes*). The first chromosome is responsible for creating the cells/nodes. The second chromosome is responsible for creating the connectivity (i.e., for the BNs). Each chromosome is built out of rules. Each rule has sufficient information for cell/node creation and connectivity. Also, the rules are of certain length. Those destined for cell/node creation are different from the ones for connectivity. Consequently, chromosomes contain different rules.

3.1 An L-system for the genetic representation

A rewriting approach was chosen due to the ease of defining a specific rule set, that can target to rewrite specific features of a structure, e.g., connections or

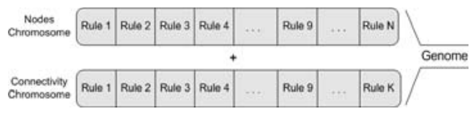


Fig. 4. This is how the genome looks like with the genome split into two (*chromosomes*). The first chromosome is responsible for generating the cells/nodes whereas the second chromosome is responsible for generating the connectivity of the network

node functions that enable a way of splitting genetic information into separate information carrying units (i.e., chromosomes).

A prominent model is L-systems. They are rewriting grammars, able to describe developmental systems, simulate biological processes [7], and describe computational machines [8]. Since there are different types of rules in the two chromosomes, there is a need for two separate L-systems. The first L-system processes the rules of the first chromosome, while the second L-system deals with the connectivity rules of the second chromosome.

3.2 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. Table 1(a), shows the type of symbols used by the L-system of the first chromosome. Some cells perform special cell processes and influence the intermediate and final phenotypes. Symbol *a* is the *axiom*. Apart from the symbols *a*, *b*, and *c*, which perform *growth* of the phenotype, symbol *d* performs *apoptosis*, leading to the deletion of the current rule (i.e., cell/node), of the intermediate phenotype. Additionally, symbols `X` and `Y`, are responsible for *differentiation*, leading to the replacement of the predecessor cell/node (i.e., if `X`→`Y` the outcome will be `Y`, whereas, if `Y`→`X` the outcome will be `X`). The length of each rule is 4 symbols (i.e., `4x8bits=32bits`). For node/cell generation the L-system runs for 100 timesteps and then it stops. As such, the intermediate phenotypes generated by development are of variable size.

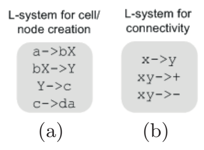


Fig. 5. (a) Example of L-system rules for the first chromosome, (b) Example of L-system rules for the second chromosome

Figure 5(a), gives an example of a L-system for the first chromosome. A simple example with step-by-step development of a 2D-CA architecture is illustrated in Figure 6. Development starts with the axiom (a) representing a cell at developmental step (DS) 0. Since the axiom is found in the L-system rules, development continues and the next rule triggered is the a→bX. This rule will create two more cells b and X, resulting in growth of the CA, at DS 1. The next rule triggered is bX→Y. Since X→Y denotes differentiation, the symbol X is replaced by Y, at DS 2. For differentiation to occur, the rules should either be X→Y, or Y→X. Next, rule Y→c triggers causing again growth of the CA, at DS 3. At DS 4, the rule c→da is triggered, causing the death of the cell c and the growth of the CA with the cell a. From DS 5 up to DS 8, the rules are being triggered once more in the same sequence.

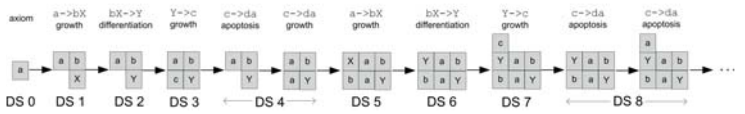


Fig. 6. A step-by-step development of a 2D-CA architecture based on the example L-system for the first chromosome

3.3 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; the current node has always the number zero and any nodes starting from the current node forward have positive numbering, where nodes that exist from the current node backwards, have negative numbering. So, there is a need to differentiate between the current and the next node, using different symbols and also a need to describe whether a connection will be created forward or backward from the current node.

The rules involved for connectivity are not as complex as the ones found in the first chromosome. The length of the rules here is also 4 symbols / rule. Also, there is a need to assure that the chromosome will have sufficient information for the developmental processes (i.e., growth, differentiation and apoptosis). The L-system uses a D0L (i.e., with zero-sided interactions). An example L-system for the second chromosome is shown in Figure 5(b), and the symbols used are explained at Table 1(b). The *axiom* rule for the second chromosome is x→y. It means that development initially searches if the axiom exists. If so, development continues and looks for rules of type xy→+value, or xy→-value. In short, these two rules imply that if two different (i.e., distinct) nodes are found (x≠y), then it creates a connection forward (if the rule includes a '+'), or backwards (if the rule includes a '-'). The field value is encoded in the genotype and denotes the node number for the generated connection. For example, rule xy→+3 denotes

Table 1. (a) Symbol table for Node generation, (b) Symbol table for Connectivity generation

(a)

Symbol	Description
a (AXIOM)	Add (growth)
b	Add (growth)
c	Add (growth)
d	Delete (apoptosis)
X	Substitute (differentiation)
Y	Substitute (differentiation)
$\rightarrow$	Production

(b)

Symbol	Description
x	Node (different from y)
y	Node (different from x)
+	Connect forward
−	Connect backwards
$\rightarrow$	Production

that a connection will be created from the current node (node `0`), to the one being three nodes forward. Similarly, rule `xy`$\rightarrow$`-3`, denotes that a connection will be created starting from the current node (node `0`), to the one that is three nodes backwards. If `value=0`, a self-connection is created to the current node. A step-by-step development of a boolean network based on the chromosomes of Table 1(a) and 1(b), can be found in [5] and is not shown here due to page limitation. The modularity of the genome, gives the possibility to develop itself to enable or disable parts of it (chromosomes), when this is required and driven by the goal set. For example, if the target architecture is a 2D-CA, the second chromosome (i.e., connectivity) is disabled, since connectivity is predetermined. Similarly for BN development, both chromosomes are enabled (i.e., nodes and connectivity).

3.4 The genetic algorithm for the common genetic representation

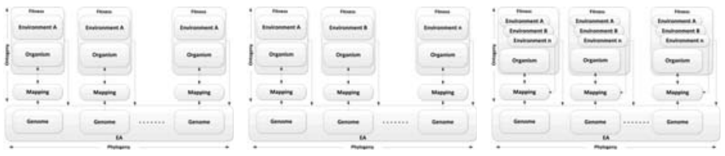
A genetic algorithm is used to generate and evolve the rules found in the genome (i.e., in the chromosomes). Since there are two separate L-systems involved in development, the evolutionary process comprise of two phases: node and connectivity generation phases. Mutation and single-point crossover were used as genetic operators. Mutation may happen anywhere inside the 4-symbol rule, ensuring that the production symbol ($\rightarrow$) is not distorted by mutation. In short, we want to make sure that after mutation, the production symbol is still in the rule (i.e., the rule is valid). Single-point crossover between two parents is executed at the location of the production symbol, ensuring that a valid rule is created as offspring. The evolutionary cycle ends after a predetermined number of generations.

4 Definition of the Environment

Here, we define the 'environment' in a consistent way. In the literature, the environment has been used in various levels, depending on the system. In

[9] for example, the environment is used within the cell itself (i.e., the *cell's metabolism* [9],[10]). In most models, environment refers to inter-cell communication, where cells can communicate their protein levels [11] or chemical levels and cell types [10]. The neighborhood of the underlying architecture (i.e,. cellular automata) can also become the environment, as in [10] with a 2D von Neumann neighborhood. Herein, we consider only an external environment, where the emerging organism needs to survive in it. Also, the behavior of the organism - interpretation of the state plot, may be distinctly different for the same organism when developed in two different environments [9]. As such, we introduce different external environments and give the possibility to the developing organism to adapt its behavior.

Figure 7 shows the various external environmental setups applied to the developing organisms. In the first subfigure, the individuals are evaluated based on environment A. In the second subfigure, the individuals are still being evaluated in a single environment, but the environment can be different (i.e., environment A, environment B, etc.). In the third subfigure, each individual is being assessed on a set of different environments. The environment in Figure 7(a), is a single-cell environment where everywhere except in the first node/cell has a value of zero. The first cell/node holds initially the value of one. The environments in Figures 7(b) and 7(c) are random. Feeding evolution with different information (i.e., environments), is expected to affect the genome, intermediate phenotypes and the final phenotype, not only in terms of cell types (i.e., for CA, BN), but also in terms of connectivity (i.e., for the BN).



(a) (b) (c)

Fig. 7. The three different environmental setups. (a) Single-cell environment/genome evaluation, (b) Random environment/genome evaluation, (c) Multiple-random environments/genome evaluation.

5 Experiments

The motivation in Section 1, dictates a need to identify whether common developmental genomes have an inherent advantage over the separately developed genomes. We take different approaches to be able to draw solid conclusions. First, the influence mutations may have to the final phenotype and in driving evolution in different environmental conditions (Subsection 5.1). Second, we try to discover inherent ontogenetic trends of common developmental genomes, by focusing on the developmental mechanisms and how these are deployed under different environments. This is studied in Subsections 5.2 and 5.3.

5.1 Influence of Mutation Over Evolution

In this experiment, we study whether mutations help to determine the direction of phenotypic evolution. That is, try to identify specific patterns by simply counting the positive, neutral or negative influence a mutation has over the phenotype for each generation. The new phenotype after a mutation is compared to the phenotype from the previous generation. If the fitness of the new phenotype is bigger, then the mutation is considered positive. Neutral mutation is when both phenotypes have the same fitness where negative mutation will have a destructive influence to the phenotype.

5.2 Influence of Developmental Processes Over Evolution

Here, the mechanisms involved in development, i.e., the developmental processes during evolution, are investigated. More specifically, the appearance rate of growth, apoptosis and differentiation per individual are measured for each generation. In this way, we hope to get a better understanding of how development works for the separate and common genomes respectively.

5.3 Influence of Conditional Developmental Processes Over Evolution

Taking one step further, we capture conditional appearance for each process, given a certain process has appeared earlier during development. For example, we measure the number of growths after an apoptosis has occurred (*growth*|*apoptosis*) or after a differentiation has occurred (*growth*|*differentiation*). Given we have three different developmental processes and each process can be in one of the three different conditional cases, we conclude to a total of 9 conditional cases for evaluation (Table 2).

Table 2. Conditional Appearance of the Developmental Processes

Cond. case 1	Cond. case 2	Cond. case 3
growth|*growth*	*growth*|*apoptosis*	*growth*|*differentiation*
apoptosis|*growth*	*apoptosis*|*apoptosis*	*apoptosis*|*differentiation*
differentiation|*growth*	*differentiation*|*apoptosis*	*differentiation*|*differentiation*

5.4 Experimental Setup

The experiments were performed both for separate genomes and the common genome cases. Each experiment is based on the different settings shown in Figure

7. For each setup, 20 runs were performed resulting in 20 different organisms. The developmental process was apportioned of 1000 state steps. The fitness function gives credit for cycle attractors between 2 - 800; the best score 100 is assigned to individuals with a cycle attractor of 400. The fitness scores have a bell-curve "distribution" with the worst score 4 being assigned at limit values. A total number of 70 rules for node generation (i.e., total size `70x32=2248bits`). For the second chromosome, we use the same number of rules (i.e., a size of `70x32=2248bits`). In this setup, each rule can be used more than once during development. For the common developmental genomes case, fitness is the average of CA and BN fitnesses. In the multiple-random environment case, fitness is the average over 10 evaluations (i.e., different external environments). The evaluation of CA and BN phenotypes was based on the cell types of Table 3.

Table 3. Cell types with their functionality

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

The 2D-CA is non-uniform of size `6x6`. The BN network has a maximum size of $N = 36$ nodes. The reason for this choice is that we want the architecture to have the same state space. The number of outgoing connections per node is $K = 5$. For inputs more than 5, a self-connection to the originating node is created instead. *Generational mixing* protocol was used as the GA's global selection mechanism and *Rank selection* for parental selection. For CA development, the mutation rate was set to .005 and crossover rate at .001. The population size is 20. The GA was set to 10000 generation/Run.

5.5 Results

The result figures were generated after sampling the data every 100 values. To present the influence mutation has over evolution, an average number for each mutation type (i.e., positive, neutral and deleterious), is drawn across all runs along with the standard deviation per generation. Common developmental genomes show higher ratio of positive mutation during evolution for the single-cell environment (Figures 8(a), 8(b), and 8(c)). Neutrality levels seems to be less throughout evolution. Deleterious mutations follow positive mutations across all species. In the random environment case, common genome holds a constant level of neutrality which is slightly higher than the separate genomes for CA and BN (Figures 8(d), 8(e), and 8(f)).

Higher neutrality ratio gives genomes the ability to choose amongst larger span of potential trajectories in the fitness landscape, and greater ability to cope

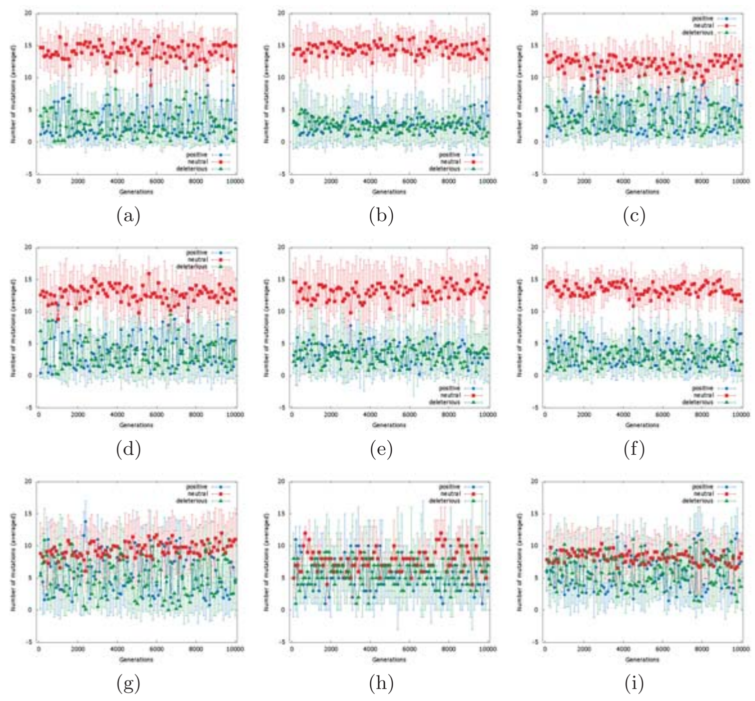


Fig. 8. Mutational analysis. Single-cell environment: (a) CA, (b) BN, (c) Common developmental genome. Random environment; (d) CA, (e) BN, (f) Common developmental genome. Multiple-random environment: (g) CA, (h) BN, (i) Common developmental genome.

with uncertain and random environments. The same applies for the multiple-random environment, where common genome has a higher constant ratio on positive and neutral levels as compared to the other architectures (Figures 8(g), 8(h), and 8(i)). Also, common genomes appear to have lower standard deviation, for the random and multiple-random environment cases.

To present the influence of developmental processes over evolution, an average for each developmental process (i.e., growth, apoptosis and differentiation), is drawn along with the standard deviation per generation. Common developmental genomes hold higher growth and differentiation ratios for the single-cell, random and multiple-random environments (Figures 9(c), 9(f), and 9(i)). Growth and differentiation are crucial components of a genome towards evolvability [2]. Here, it is obvious that as the environment becomes more difficult (i.e., from the single-cell to the multiple-random), common genomes acquire higher ratios of growth and differentiation, if compared to the other architectures. Apoptosis levels are close to zero in all cases. Higher standard deviation is shown for the multiple-random environment case.

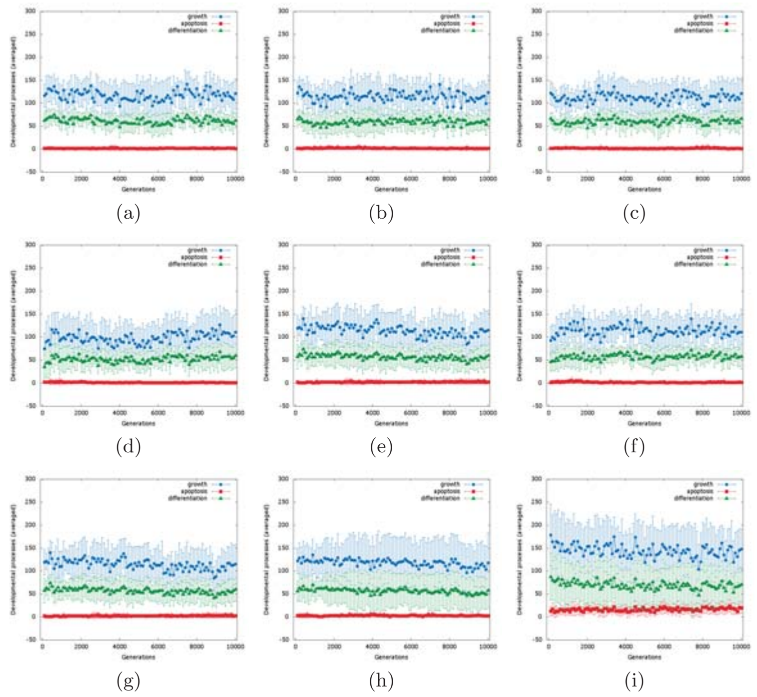


Fig. 9. Developmental processes for the single-cell environment: (a) CA, (b) BN, (c) Common developmental genome. Random environment; (d) CA, (e) BN, (f) Common developmental genome. Multiple-random environment: (g) CA, (h) BN, (i) Common developmental genome.

Each conditional developmental process is averaged across all 10 runs and the standard deviation is shown per generation. The conditional developmental processes ratios for the single-cell environment seem homogeneous in all architectures (not shown). What is worth noting is that the ratio of the *apoptosis*|*growth* conditional process for the common genome looks like a convolution of the other two architectures (Figures 10(a), 10(b), and 10(c)), with values being spread out over a larger range (i.e., high standard deviation). It is not yet clear why and what effect this has for common genomes development. The conditional growth and differentiation processes show similar behavior across all architectures (not shown).

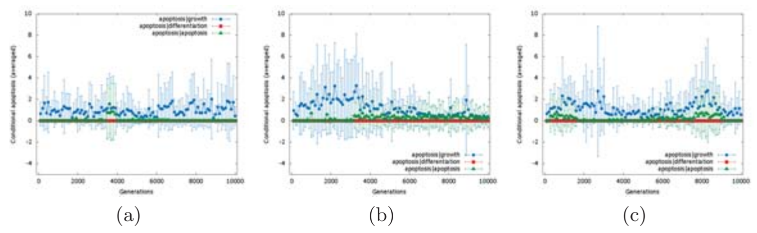


Fig. 10. Conditional apoptosis process for the single-cell environment. CA (a), BN (b), Common developmental genomes (c).

The same result is obtained also for the random environment. The overall pattern is similar across all architectures (not shown). The ratio of the *apoptosis*|*growth* conditional process for the common genome looks (again) like a convolution of the other two architectures (Figures 11(a), 11(b), and 11(c)).

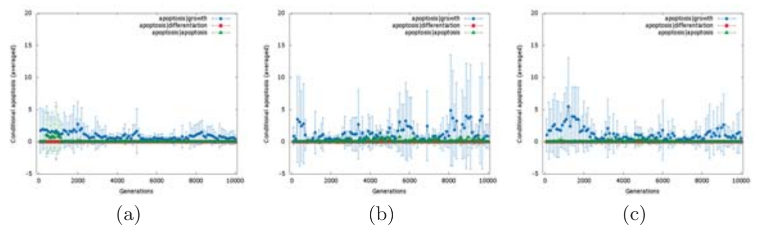


Fig. 11. Conditional apoptosis process on a random environment. CA (a), BN (b), Common developmental genomes (c).

The last set of experiments on the multiple-random environment, contributes clearly to the overall findings. For the common genome case, there is higher exploitation of *growth*|*differentiation* and *growth*|*apoptosis* conditional processes (Figure 12(e)). Also, a higher exploitation ratio of *apoptosis*|*growth* and *apoptosis*|*differentiation* conditional processes (Figure 12(f)), and higher standard deviation is observed. No *apoptosis*|*differentiation* conditional processes were observed for any of the architectures (not shown). Higher ratios of these conditional processes dictates the complexity of the multiple-random environment and the effect it applies on the common genomes as they need to respond and better adapt to the environment.

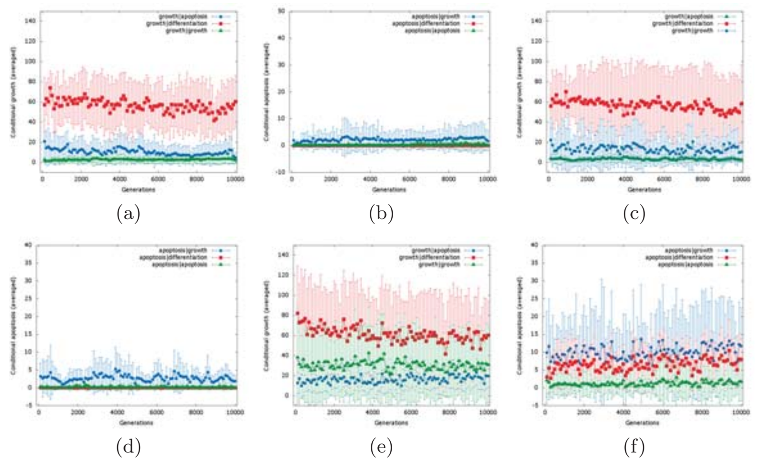


Fig. 12. Conditional developmental processes on multiple-random environment. CA: Conditional growth (a), apoptosis (b). BN: Conditional growth (c), apoptosis (d). Common developmental genomes: Conditional growth (e), apoptosis (f).

6 Conclusion

In this study, we investigated potential relations or patterns that exist between mutation and evolution for the common developmental genomes and how these are limited by the different environmental conditions. Also, we studied the developmental processes and the inherent dynamics involved and how environmental conditions affect the outcome. In this line, we also made a comprehensive work to identify directionalities during ontogeny by looking at conditional developmental processes and identifying which processes are triggered mostly under certain conditions.

Concluding, common genomes showed higher positive and neutral mutation ratios in more complex environments giving an inherent ability to cope with such environments. Also, they acquired higher ratios of growth and differentiation processes as compared to the other architectures. Lastly, high exploitation ratios of *growth|differentiation* and *growth|apoptosis* conditional processes, offered to common genomes the ability to perform in random environments. They have also shown a larger plurality of conditional processes during development. More work is needed towards conditional developmental processes and how these affect evolution and the final phenotype.

As future work, we shall change the way of looking into the architectures, i.e., instead of looking at them as different species, we can consider them as organs of a common developing biological entity. In this case, architectures need to be merged (as is the case in biological organs). The overall goal of this study is to target more adaptive scalable systems able of complex computation. The exploration of these architectures (i.e., hybrid architectures) with common genomes, the current developmental model and the ability to shape the phenotype of the system as modules, to change the dynamic properties of the entire system (i.e., *phenotypic shaping*), seem promising as future research.

References

1. Wallace, A.: Evolution: A Developmental Approach. Wiley-Blackwell (2010)
2. Antonakopoulos, K., and Tufte, G.: On the Evolvability of Different Computational Architectures using a Common Developmental Genome. In: Rosa, A., Dourado, A., Madani, K., Filipe, J., and Kacprzyk J. (eds.) IJCCI 2012, pp. 122–129. SciTePress Publishing (2012)
3. Antonakopoulos, K., and Tufte, G.: Is Common Developmental Genome a Panacea Towards More Complex Problems?. In: 13th IEEE International Symposium on Computational Intelligence and Informatics (CINTI2012), pp. 55–61 (2012)
4. Arthur, W.: Developmental Drive: An Important Determinant of the Direction of Phenotypic Evolution. Evolution & Development. vol. 3, num. 4, pp. 271–278 (2001)
5. Antonakopoulos, K., and Tufte, G.: A Common Genetic Representation Capable Of Developing Distinct Computational Architectures. In: IEEE Congress on Evolutionary Computation (CEC2011), pp. 1264–1271 (2011)
6. Wilkins, J.: What is a species? Essences and generation. Theory in Biosciences. vol. 129, num. 2, pp. 141–148 (2010)
7. Lindenmayer, A., and Prusinkiewicz, P.: Developmental Models of Multicellular Organisms: A Computer Graphics Perspective. In: Langton, C.G. (eds.) Proceedings of ALife. pp. 221–249. Addison-Wesley Publishing (1989)
8. Staffer, A., and Sipper, M.: Modeling Cellular Development Using L-Systems. 2nd Int'l Conference on Evolvable Systems (ICES98). In: LNCS. pp. 196–205. Springer (1998)
9. Tufte, G., and Haddow, P.C.: Achieving environmental tolerance through the initiation and exploitation of external information. In: IEEE Congress on Evolutionary Computation (CEC2007), pp. 2485–2492 (2007)
10. Federici, D.: Evolving a neurocontroller through a process of embryogeny. Simulation of Adaptive Behavior, LNCS, Springer, pp. 373–384 (2004)
11. Gordon, T.G.W., and Bentley, P.J.: Development brings scalability to hardware evolution. In: 2005 NASA/DoD Conference on Evolvable Hardware, pp. 272–279 (2005)

Common Developmental Genomes - Evolution through Adaptation

K. Antonakopoulos

16th European Conference on the Applications
of Evolutionary Computation (EvoApps 2014)
Granada, Spain, April 23-25, 2014

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.

Common Developmental Genomes Revisited - Evolution through Adaptation

Konstantinos Antonakopoulos

Norwegian University of Science and Technology,
Department of Computer and Information Science,
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 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.

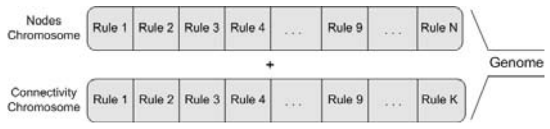


Fig. 1. The genome is split into two chromosomes: Node- and Connectivity-chromosomes.

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.

(a)

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

(b)

Symbol	Description
x	Node (different from y)
y	Node (different from x)
$+$	Connect forward
$-$	Connect backwards
$\rightarrow$	Production

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

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`$\rightarrow$`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 2(a), 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].

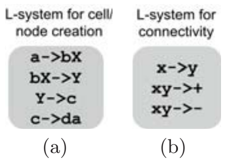


Fig. 2. (a) L-system rule set for node/cell generation, (b) L-system rule set for connectivity.

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 2(b). 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).

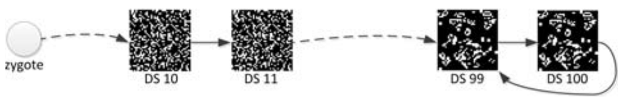


Fig. 3. Developmental path of a structure shown as a trajectory.

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.

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

Table 2. Cell types and functionality.

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.

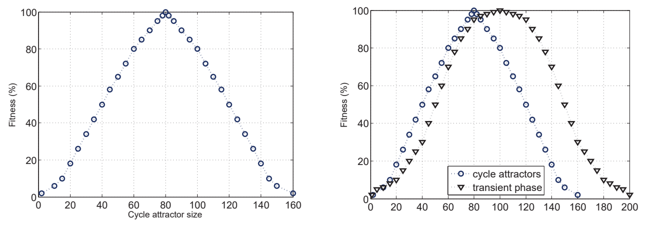
The computational problems chosen here describe some basic dynamic behavior for CA and BN and the goal is generally expected to be reached. Though, the problems as such are of minor importance since we are mainly after the ability of common developmental genomes to adapt during evolution.

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

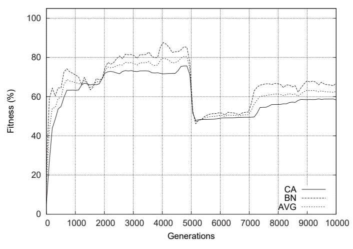


Fig. 5. Fitness evaluation of common developmental genomes (averaged).

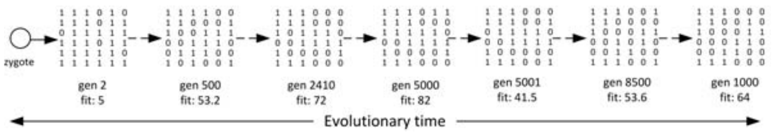


Fig. 6. Some of the best intermediate and final phenotypes of a CA evolution over time.

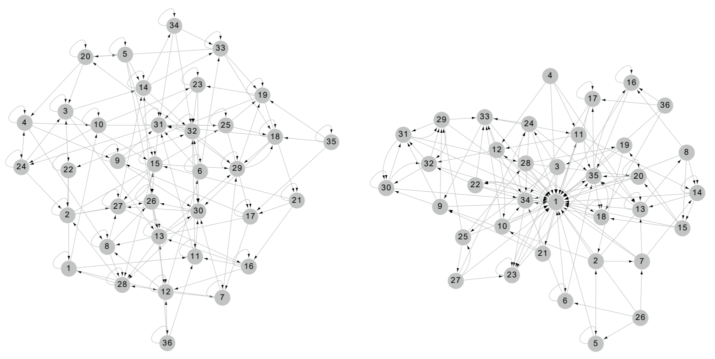
the developmental model with the CA, extended out exploiting the complete CA lattice for both the problems investigated. In addition, development produced maximally big genomes at the very beginning of the process (not shown). As we will see in the next paragraph, this is not the case for the evolved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

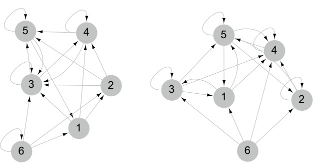
Some of the near-perfect solutions given by evolution (fitness > 80), include networks with a rather small number of nodes (not shown). All perfect solutions (fitness 100), involved networks having the maximum number of nodes allowed by the model (N=36). This suggests that development initially tries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



(a) (b)

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

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.

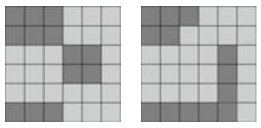


(a) (b)

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

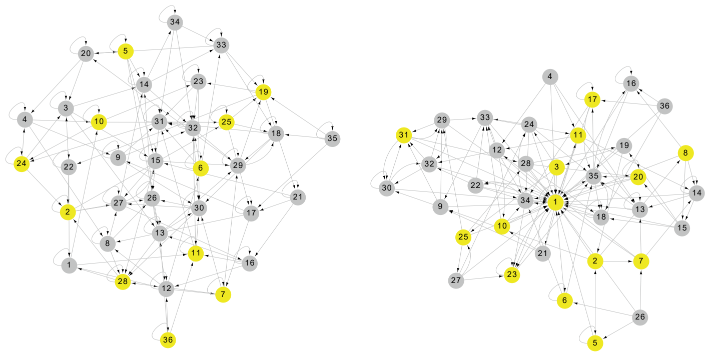
Figure 8 shows the best evolved BN solutions for the second problem at generation 10000. Both solutions have a rather small number of nodes (N=6) and most of the nodes have at least one self-connection. Other, less than perfect solutions provided networks having the max number of nodes (N=36).

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.



(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.

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 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., and Vasicek, M.: Evolution of cellular automata with conditionally matching rules, Congress on Evolutionary Computation (CEC2013), 1178–1185 (2013).

[2] Bull, L.: Artificial symbiogenesis and differing reproduction rates," Artificial Life, vol. 16, no. 1, 65–72 (2010).

[3] Antonakopoulos, K., and Tufte, G.: On the Evolvability of Different Computational Architectures using a Common Developmental Genome. In: Rosa, A., Dourado, A., Madani, K., Filipe, J., and Kacprzyk J. (eds.) IJCCI 2012, 122–129. SciTePress Publishing (2012)

[4] Antonakopoulos, K., and Tufte, G.: Is Common Developmental Genome a Panacea Towards More Complex Problems?. In: 13th IEEE International Symposium on Computational Intelligence and Informatics (CINTI2012), 55–61 (2012)

[5] Antonakopoulos, K., and Tufte, G.: A Common Genetic Representation Capable Of Developing Distinct Computational Architectures. In: IEEE Congress on Evolutionary Computation (CEC2011), 1264–1271 (2011)

[6] Tufte, G.: The discrete dynamics of developmental systems, Evolutionary Computation (CEC09). IEEE Congress on, 2209–2216 (2009).

[7] Lindenmayer, A.: Developmental Systems without Cellular Interactions, their Languages and Grammars, Journal of Theoretical Biology, vol. 30, no. 3, 455–484 (1971)

[8] Lindenmayer, A., and Prusinkiewicz, P.: Developmental Models of Multicellular Organisms: A Computer Graphics Perspective. In: Langton, C.G. (eds.) Proceedings of ALife. 221–249. Addison-Wesley Publishing (1989)

Studying Network Morphology in Common Developmental Genomes

K. Antonakopoulos

IEEE Systems, Man and Cybernetics (SMC 2014)
San Diego, California, USA, October 5-8, 2014

Abstract

In most EvoDevo (Evolutionary Developmental) systems, genotypes are developed and evolved towards a structural or computational goal utilizing some kind of computational architecture (i.e., structures made of connected elements that may compute). Exploiting a common genotype to develop and evolve different classes of computational architectures towards a common goal has previously been successfully implemented, through common developmental genomes. In this work, we focus at how common genomes exploit the underlying architectures during development and build structure (network morphology) in phenotypes for different problem instances and architecture sizes. Common developmental genomes showed an ability to exploit the size of the architecture by actively involving a larger number of nodes/cells while managed to maintain a small number of neutral and static parts in the evolved structures.

Studying Network Morphology in Common Developmental Genomes

Konstantinos Antonakopoulos

Department of Computer and Information Science, NTNU
Sem Sælandsvei 7-9, NO-7491, Trondheim, Norway
Email: kostas@idi.ntnu.no

***Abstract*—In most EvoDevo (Evolutionary Developmental) systems, genotypes are developed and evolved towards a structural or computational goal utilizing some kind of computational architecture (i.e., structures made of connected elements that may compute). Exploiting a common genotype to develop and evolve different classes of computational architectures towards a common goal has previously been successfully implemented, through common developmental genomes. In this work, we focus at how common genomes exploit the underlying architectures during development and build structure (network morphology) in phenotypes for different problem instances and architecture sizes. Common developmental genomes showed an ability to exploit the size of the architecture by actively involving a larger number of nodes/cells while managed to maintain a small number of neutral and static parts in the evolved structures.**

***Index Terms*—Developmental genomes, L-systems, morphology, cellular automata, boolean networks**

I. Introduction

One central challenge in artificial development is to understand how a mapping process could work on a class of computational architectures (species) in a more general way. Computational architectures can for example be cellular automata (CA) [1] or boolean networks (BN) [2]. The hypothesis for going this way forward is that it may be possible to exploit these properties from each computational architecture or combine more than one architectures in a more efficient way (i.e., a true multicellular approach) [3]. This hypothesis has given rise to common developmental genomes (CDG) [4].

Common developmental genomes have shown a potentiality of developing different classes of computational architectures with non-identical structural [4] or computational [5] phenotypic properties. A simple stock market model with varying architecture size ($N = 144$, $N = 169$, and $N = 196$) was studied in [5], using CA and BN as architectural models. Although results showed that CDG are able to find better solutions for certain architecture sizes than single genomes evolved specifically for a one computational architecture only (i.e., CA or BN), no insight was acquired regarding how they manage to evolve networks towards the final phenotypes.

In [6], is mentioned that a sequence (phenotype) and the description of the environment in which that sequence is to be interpreted, is expressed in terms of physical complexity (which technically can be defined as the *shared Kolmogorov complexity*). By borrowing related concepts from Kolmogorov complexity [7], it can be inferred that the more information required to fully describe an object and its environment, the more complex the object is. As a simplified example, let k be the amount of information required to describe i.e., a finite state automaton. For $k = 4$, we suggest that the finite state automaton at timestep t, can be fully described by an amount of information of four. It is assumed that environment information is included in this description. As such, for $k = 5$, the finite state automaton in a later timestep, i.e., $t + \Delta x$, is characterized by an information amount of five. Therefore, the latter automaton is interpreted as a more complex system. This simple notion is used here as a rudimentary metric to describe the complexity of a problem instance.

In this work, we investigate how CDG model exploits the underlying computational architectures during development and builds structure (network morphology) for different problem instances and architecture sizes. The motivation here is to see how common genomes manage to develop and evolve network structures in phenotypes in more scalable conditions. This is supported by a need to understand what dynamic conditions apply during evolution and whether architecture size affects their ability to evolve.

The rest of the article is organized as follows. The developmental model is briefly described at Section II. Experimental design is given at Section III with results at Section IV. Finally, conclusion and future work come at Section V.

II. The Developmental Model

In this section, the genetic representation and the developmental model is described in brief. For a detailed description of the model and analytical examples for cellular automata and boolean network development, see [4].

The genome comprise two parts or *chromosomes* as shown at Figure 1. The first chromosome creates cells/nodes of the species where 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. L-systems are

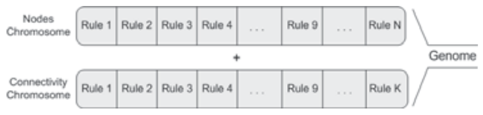


Fig. 1. Genome comprise two chromosomes: Node and Connectivity.

TABLE I
SYMBOL TABLE FOR (A) NODES/CELL CREATION, (B) CONNECTIVITY

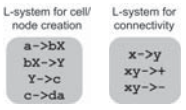
(a)

Symbol	Description
a	growth
b	growth
c	growth
d	apoptosis
X	differentiation
Y	differentiation
$\rightarrow$	Production

(b)

Symbol	Description
x	Node (other than y)
y	Node (other than x)
+	Connect forward
–	Connect backwards
$\rightarrow$	Production

rewriting grammars, able to describe developmental or generative systems and have successfully been used to simulate biological processes [9]. Two separate L-systems are used in the representation as the developmental model; the first L-system develops cell processes of the first chromosome and the second L-system develops the connectivity rules of the second chromosome. Table I show the symbols used by the L-system for the first and second chromosomes.
Symbol *a* is the *axiom*. Apart from symbols *a*, *b*, and *c*, which perform *growth* of the phenotype, symbol *d* performs *apoptosis* i.e., deletion of current rule (cell/node). Symbols X and Y, represent the *differentiation* process, replacing the predecessor cell/node. For example, for X$\rightarrow$Y, outcome will be Y. For node/cell generation, L-system runs for n timesteps and then stops. As such, the intermediate phenotypes generated by development are of variable size. Figure 2, show an example of a L-system grammar that can be used for developing chromosome rules.



(a) (b)

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

A simple example of a 2D-CA development is illustrated at Figure 3. Development starts with any of the symbols a, b or c, representing a cell at developmental step (DS) 0. The next rule triggered is a$\rightarrow$bX. This rule generates two cells b and X, resulting in CA growth, at DS 1. The next rule triggered is bX$\rightarrow$Y. Since X$\rightarrow$Y denotes differentiation, X is replaced by Y at DS 2. For differentiation to occur, rules should either be X$\rightarrow$Y or Y$\rightarrow$X. Next, rule Y$\rightarrow$c triggers causing again CA growth (DS 3). At DS 4, rule c$\rightarrow$da is triggered, causing the death of cell c and growth of CA with the cell a.

A. The Genetic Algorithm

A genetic algorithm (GA) is utilized to evolve the chromosome rules. Since there are two separate L-systems involved in development, evolutionary process consists of two phases: a. creation of cells/nodes and b. generation of connections. Single symbol mutation and single-point crossover are used as genetic operators. Mutation may occur anywhere within the 4-symbol rule, so that production symbol ($\rightarrow$) remains intact. Single-point crossover between two parents takes place at the production symbol location within the rule. The evolutionary cycle ends after a predetermined number of generations.

III. EXPERIMENTS

In [5], it was studied whether the same developmental mapping can favor the evolvability of different computational architectures (i.e., CA and BN), in problem instances where the amount of information required to fully describe the system increases. As explained in Section I, this approach allows for a simple characterization of the complexity of problem instance. Complexity in a problem instance is defined in brief by the number of cell states involved to decide upon the state of the current cell (Section III-A).

Although a great deal of the capabilities of CDG was acquired in [5], [8], no studies were done to understand how the model exploit the underlying architectures and how it builds network structures or morphologies. This becomes more interesting as the developmental model is investigated using problems of varying complexity under scalable conditions. It might be that the model builds different network structures for "easier" problem instances or when architecture size (N) remains small. In addition, it may be possible that CDG exploit architectures in a similar way with any problem instance or that the size of architecture does not really affect the ability of the model to evolve and find some of the best phenotypes in the solution space.

So, the goals of the experiments to follow are: (i) study how the model exploits underlying architectures during development for varying architecture size and problem instance, and (ii) investigate how the developmental model builds network structure (morphology) for the best phenotypic solutions.

A. Experimental Setup

We develop two different computational architectures (2D-CA and BN) with size: a. $N = 36$, b. $N = 64$, and c. $N = 100$ cell/node. The size of architectures is smaller than in [5], but large enough so that dependencies amongst cells/nodes be minimized during state evaluation. In addition, the size chosen should allow for easy, visual explanation of the network structures of the phenotypes.

For architectures to be comparable, they must yield the same state space or number of possible states. For example, an architecture of size $N = 36$, each cell/node can take 2 different distinct values (boolean). The total state space for each architecture would then be 2^{36}. The number of outgoing

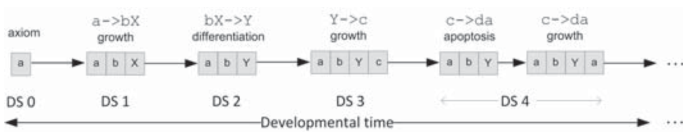


Fig. 3. Development of a 2D-CA architecture based on the example L-system for the first chromosome.

connections per node is $K = 5$. If 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 by the total number of nodes in the network ($N - 1$). For simplicity, the length of each rule is 4 symbols (i.e., 4x8bits=32bits). The total number of rules for cell/node and connectivity is 64 with total size 32x64bits=2048bits. Each rule can be reused during a developmental cycle. The GA drives a single population of 20 individuals. All GA parameters are similar to [5]. Development runs for 100 time steps for each individual in the population. *Generational mixing* is used as the GA's global selection mechanism and *fitness proportionate* for parental selection. Mutation rate is .009 and one-point crossover rate .001. Rates are chosen based on past experience with the model [5], [8].

The computational problem is described at Section III-B. For each problem, instances of different configuration are considered; an instance with no-state memory and instances of 2-, and 5-state memory. Problem instance description is given below (Sections III-C, III-D, III-E).

1) Preliminary Run: We initially run a set of 10 experiments of 1000 generations for each problem instance and architecture size. For each individual, a random initial environment is fed into the architecture (CA and BN). This preliminary experiment resulted in ten final phenotypes. For each architecture size, we chose the two best and two worst solutions, based on their fitness. Since the focus of this work is to investigate how the model builds network structure, phenotypes yielding a distant fitness one should anticipate diverse genetic information in genotypes. Evolving further these genotypes will presumably result in very different network structure morphologies.

2) Main Run: After best and worst phenotypes for each problem instance are chosen, their genotypes are further evolved by an additional set of experiments. Each genotype is fed into the model and run once over 500 generations each, for the same computational task. The final fitness score is the average of the fitness of all four (two best & two worst) genotypes.

B. A Simple Stock Market Model

A simple stock market model is set as the computational task since such models provide interesting dynamics phenomena [10]. 2D-CA and BN lattices are used to represent a typical trading behavior. Each cell/node corresponds to a trader that either *buys* or *sells* on each timestep. The model is based on local interactions and involves simple rules to represent the behavior of the traders. The behavior of a trader X at timestep

TABLE II
RULES TABLE FOR THE STOCK MARKET MODEL

Left neighbor	Right Neighbor	Next state
buy	buy	buy
buy	sell	buy
sell	buy	buy
sell	sell	sell

t is determined by the behavior of its two neighboring traders at timestep $t - 1$. The governing trading rules are given at Table II. If for example the left neighbor buys and the right neighbor sells, the state of the current trader at timestep t is *buy*. The states of the stock market model are translated into binary values for assessing the fitness, that is, $sell = 0$ and $buy = 1$. Fitness is assessed employing the total number of *buy* states in the architecture, as shown in (1).

$$fitness = \left(\frac{\sum buy}{N} \right), \text{ where } N = \text{architecure size} \quad (1)$$

Following is a short description of the problem instances that are used in the experiments:

C. No-State Memory Problem Instance

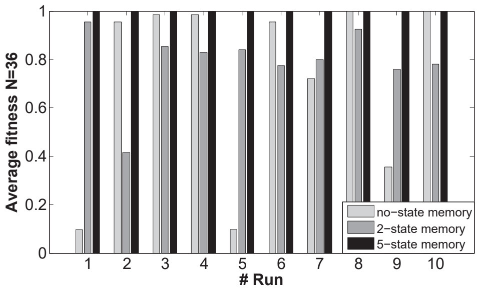
Species hold no state memory (i.e., do not take into account previous states, in order to decide upon the current state of the cell). Since the model uses two neighbors to decide upon the state of the current cell, the instance has a rudimentary complexity $k = 2$.

D. A 2-State Memory Problem Instance

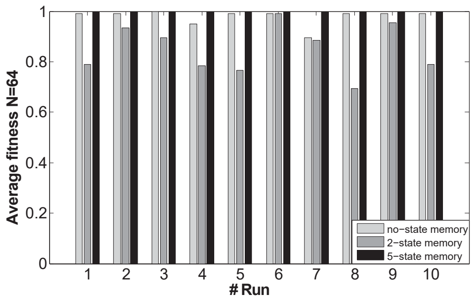
Except the two current neighbors, rules in this case take into consideration the state of the cell during the two previous timesteps (state memory 2), to decide upon the state of the current cell. Rules are based on the trading behavior (buy/sell) that dominate within the environment, using simple majority. As such, the instance would have a rudimentary complexity $k = 4$.

E. A 5-State Memory Problem Instance

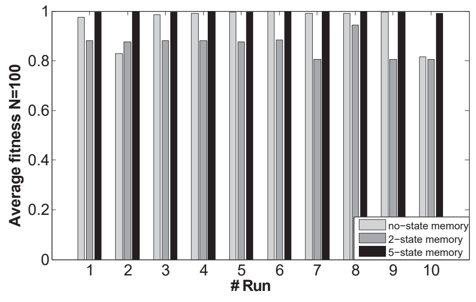
Except the two current neighbors, rules in this case take into consideration the state of the cell during the five previous timesteps (state memory 5), to decide upon the state of the current cell. Rules are based on the behavior (buy/sell) that dominate within the environment, using simple majority. This instance would have a rudimentary complexity $k = 7$.



(a)



(b)



(c)

Fig. 4. Averaged normalized (0-1) runs for the different architecture sizes (a) 36-node/cell, (b) 64-node/cell, (c) 100-node/cell architecture.

IV. Results

Figure 4 shows the average (normalized) fitness results of the preliminary run with ten evolved phenotypes for different architecture sizes (N=36, N=64, and N=100). Each figure shows a combined bar chart for the instance problems (i.e., no-state, 2-state, and 5-state memory). After preliminary run (Section III-A1), two best and two worst phenotypes for each architecture size, are selected for the main experiment. That is, four phenotypes from each problem instance for N=36, N=64, and N=100. Phenotypes chosen for the main experiment (#Run), are shown at Table III.

TABLE III
Phenotypes chosen to include for further development and evolution.

Prob.Instance	36-node	64-node	100-node
no-state	#7,#8,#9,#10	#1,#3,#4,#7	#2,#3,#6,#10
2-state	#1,#2,#8,#9	#1,#6,#8,#9	#6,#8,#9,#10
5-state	#1,#2,#3,#4	#1,#2,#3,#4	#1,#2,#4,#10

A. Rate of Change of the cell/node

To study how CDG exploit the underlying computational architecture, first is investigated how they are deployed during development. That is, monitor and estimate how each cell/node changes with respect to developmental time (i.e., *Rate of Change* or RoC). The rate of change is given by the derivative dy/dt, where y is the change of cell/node over the previous value and t the instantaneous developmental time.

Figure 5 shows the average RoC ($\%$) and standard deviation for each cell/node during development for varying architecture size. The average RoC is estimated by averaging the RoC of each cell/node of the architecture for all generations. Standard deviation shows the dispersion of the max/min RoC from the mean value for each cell/node for all generations.

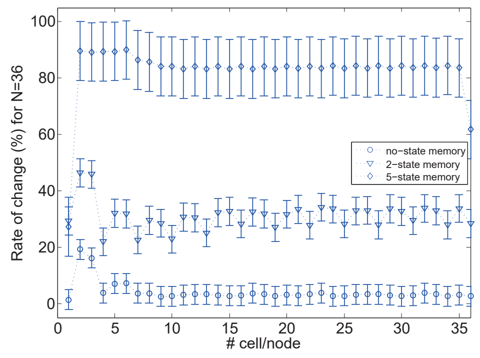
Generally, the first part of the architectures (cells/nodes 1-10), involve a higher RoC for all problem instances. 5-state memory problem yield a considerably higher average RoC, comparing to the other two instances, i.e., the no-state and 2-state memory. The average RoC for the 5-state memory problem, is inversely proportional to the architecture size.

No-state memory RoC, is proportional to the architecture size. At Figure 5(a), we observe that the average RoC is quite low but at figure 5(c) the average RoC is similar to that of the 2-state memory problem. The average RoC of 2-state memory exhibits a similar behavior and is not affected by the size of the underlying architecture.

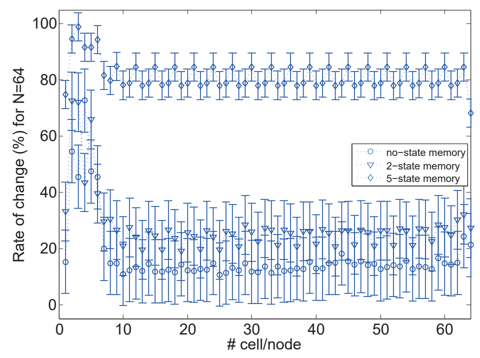
B. Evolved Network Structures

In this section, we investigate how common developmental genomes build the network morphology of the solutions sought. As *network morphology*, we mean parts of network structure that have certain characteristics. These characteristics are based on i. the developmental dynamics and ii. changes in the phenotypic structure. Since any of these factors may have an impact to the "local" fitness of the individual, we record and analyze the best only individuals, by identifying the following conditions:

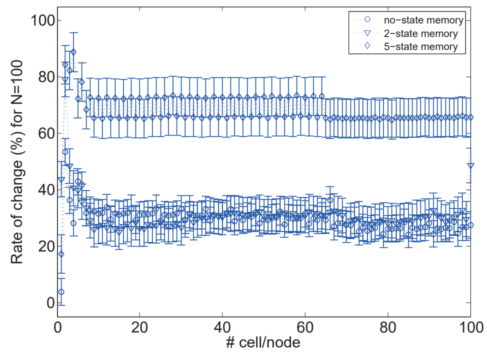
1) Positive impact to the fitness. For example, a new cell/node added or a different connection pattern may assign a greater fitness score than previously.
2) Neutral impact to the fitness (neither positive nor negative). For example, a cell/node with differentiated functionality or a newly deleted connection between two nodes may have no impact to the fitness.
3) Cell/node functionality or connectivity remain static during evolution, i.e., no changes have occurred.



(a)

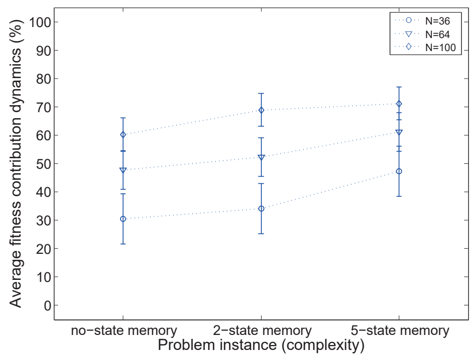


(b)

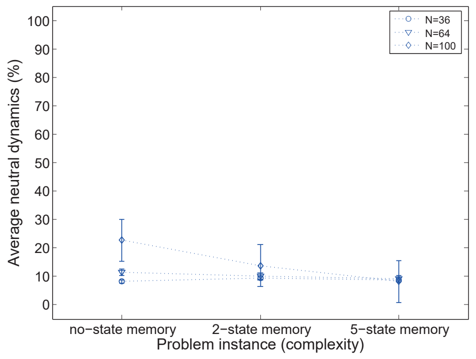


(c)

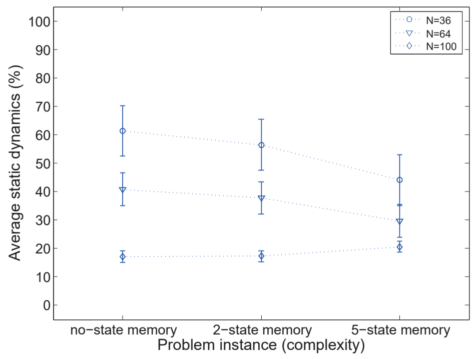
Fig. 5. Average rate of change (%) for (a) N=36, (b) N=64, (c) N=100, node/cell architecture during development.



(a)



(b)



(c)

Fig. 6. Results of the three dynamic conditions for the problem instances during evolution.

Visualization of dynamic evolution of network structures were analyzed with Cytoscape and DynNetwork plugin [13], [11]. Figure 6(a), shows the amount of structure (%) that contributes to the fitness, for different architecture sizes. The number of cell/nodes contributing positively to fitness is generally increasing with problem instance. Similarly, as the size of architecture increases, the number of cell/nodes contributing to the fitness increase. The result for N=100, follows the other two trends for N=36 and N=64, reaching a maximum contribution for 2-state memory problem. Unfortunately, trend failed to keep up since it flattens out for the 5-state memory problem.

Figure 6(b), shows the average amount of the structure (%) that has neutral impact to the fitness. The amount of neutrality in the structure for N=36 and N=64 is almost constant for all problem instances. Structures evolved for N=100, show a decreasing amount of network neutrality in the morphology when the problem instance's complexity increase. Figure 6(c) indicate average amount of static structures. Network structures for N=36, show that the number of static cells/nodes during evolution is almost constant when the problem instance's complexity increases. Conversely, network structures for N=64 and N=100, initially involve a large number of static cells/nodes, but the number of nodes gradually decrease as problem's complexity increase.

Focusing on how network structures are evolved, common genomes initially generate solutions (phenotypes) that exploit the maximum number of nodes/cells available in the architecture. Subsequently, common genomes enter into a phase where evolved phenotypes are characterized by network solutions of smaller size with the same fitness. At later evolutionary stages, the developmental model provides solutions yielding the total number of available nodes/cells in architecture. This behavior was identified only in no-state and 2-state memory problem instances. This is justified by the decreased capacity of the 5-state problem towards neutral dynamics as shown at Figure 6(b).

Figure 7 show an example of emergent behavior for one of the best no-state memory, N=64 boolean network. All boolean networks shown in this figure are perfect solutions, i.e., individuals with best fitness score. Evolution starts exploring a large number of nodes at generation 1. Next generation, the model finds perfect solutions with smaller network structure (N=52). The model continues to provide phenotypes with simple structure also in later evolutionary stages (i.e., gen. 2-423). From generation 424, evolution started to explore solutions with more complex structures. Figure 7(c)-7(e) show some phenotypes with most complex network structure (N=64). Networks are plotted with the force-based graph layout algorithm.

V. Conclusion and Future Work

In this work, we investigated how CDG manage to exploit the underlying computational architectures during development and how they build network structures (morphology) for different problem instance and architecture size. CDG showed an emergent behavior where an increased number of cell/nodes was employed and the number of neutral and static parts of network structures were constantly small. In addition, it was shown that the more complex the problem (i.e., 5-state memory problem), the higher the rate of change. The rate of change was shown to be inversely proportional to architecture size.

Same type of behavior was also shown in [12], where common genomes initially found solutions whose network structures yielded a small number of nodes. At later evolutionary stages solutions employed a larger number of nodes. Among the final solutions were also phenotypes whose network structures were rather simplified, i.e., the model did not need to exploit the total number of nodes available in the underlying architecture to obtain maximum fitness.

Both here and in [12], the model showed emergence since it was not designed to exhibit such behavior. Also, although the target tasks in the latter work were very different in nature, common genomes showed an ability to explore the solution space by providing solution whose structure range from simplified to complex.

Future work involves the generalization of the findings of this work. To draw statistically correct conclusion, results need to be validated with further experimentation. It is also interesting to investigate whether the amount of static/neutral parts of network structures may have a direct influence to the robustness of the evolved system.

Acknowledgment

I would like to thank associate professor Gunnar Tufte, Dept.of Computer and Information Science, NTNU for discussing the ideas presented in this paper.

References

[1] Von Neumann, J.: The Theory of Self-reproducing Automata. University of Illinois Press (1966)

[2] Kauffman, S. A. and Johnsen, S.: Co-evolution to the edge of chaos: Coupled fitness landscapes, poised states and co-evolutionary avalanches. Artificial Life II, pp. 325–370. Addison-Wesley (1992)

[3] Antonakopoulos, K., and Tufte, G.: Possibilities and Constraints of Basic Computational Units in Developmental Systems. In Norsk Informatikkonferanse (NIK), pp. 73–84 (2009)

[4] Antonakopoulos, K., and Tufte, G.: A Common Genetic Representation Capable Of Developing Distinct Computational Architectures. In: CEC 2011, pp. 1264–1271 (2011)

[5] Antonakopoulos, K., and Tufte, G.: Is Common Developmental Genome a Panacea Towards More Complex Problems?. In: CINTI 2012, pp. 55–61 (2012)

[6] Adami, C. and Cerf, N.J.: Physical complexity of symbolic sequences. Physica D., vol. 137, pp. 62–69, (2000)

[7] Li, M., and Vitányi, P.M.B.: An Introduction to Kolmogorov Complexity and Its Applications, Springer-Verlag, 2nd edition (1997)

[8] Antonakopoulos, K.: On the Evolvability of Different Computational Architectures using a Common Developmental Genome. In: Rosa, A., Dourado, A., Madani, K.,Filipe, J., and Kacprzyk, J. (eds.) pp. 122–129. SciTePress Publishing (2012)

[9] Lindenmayer, A.: Developmental Systems without Cellular Interactions, their Languages and Grammars, Journal of Theoretical Biology, vol. 30, no. 3, pp. 455–484 (1971)

[10] Qiu, G., and Kandhai, D., and Sloot, P.M.A.: Understanding the complex dynamics of stock markets through cellular automata. Phys. Rev. vol. 75, issue 4, (2007)

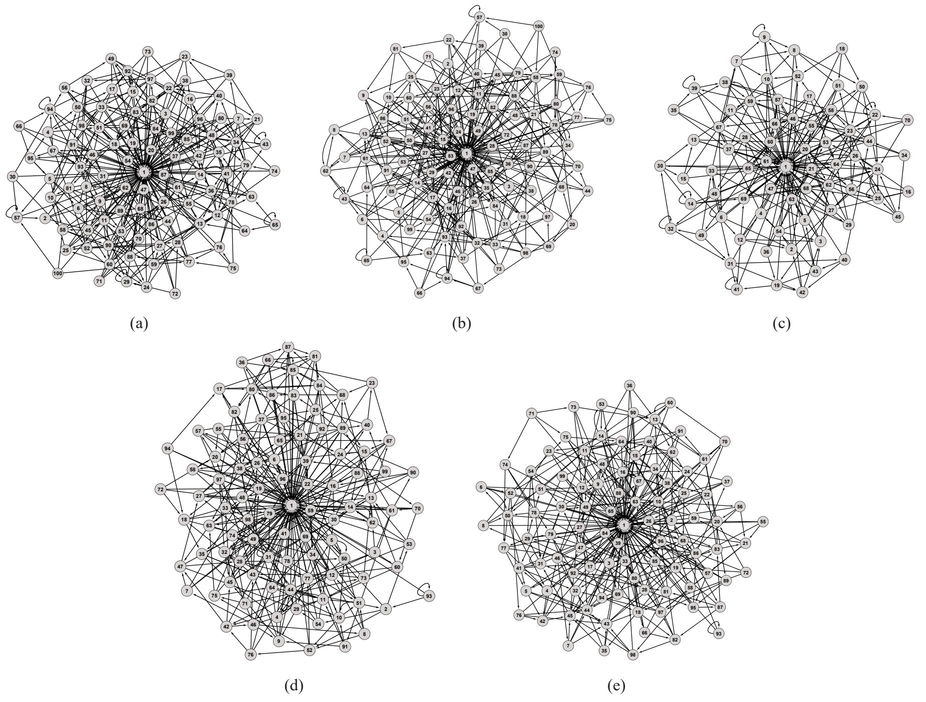


Fig. 7. Evolution stages for one of the best no-state memory, N=64 boolean network. (a) Initial solutions involve phenotypes with complex structure (gen. 1). (b) More simple solutions are found at generations 2-423 (N=52), (c)-(e) Later stages include solutions with more complex phenotypes. Sample phenotypes are shown from generations 500, 999 and 1000 (N=64).

[11] Saito, R., and Smoot, M.E., and Ono, K., and Ruscheinski, J., and Wang, P.L., and Lotia, S., and Pico, A.R., and Bader, G.D., and Ideker, T.: A travel guide to Cytoscape plugins. Nature Methods. vol.9, no.11, pp.1069-76 (2012)

[12] Antonakopoulos, K.: Common Developmetal Genomes Revisited-Evolution through Adaptation. 16^{th} European Conference on the Applications of Evolutionary Computation (EvoAPPS 2014), pp. xxx–xxx (2014)

[13] Shannon, P., and Markiel, A., and Ozier, O., and Baliga, N., and Wang, J., and Ramage, O., and Amin, N., and Schwikowski, B., and Ideker, T.: Cytoscape: A software environment for integrated models of biomolecular interaction networks, Genome Research, vol. 13, no. 11, pp. 2498–504 (2003)