NTNU
Norwegian University of Science and Technology

Evolving Compositional Pattern Producing Networks For Cellular Automata Transition Rules

Mathias Berild Ose

Master of Science in Computer Science
Submission date: June 2017
Supervisor: Gunnar Tufte, IDI
Co-supervisor: Stefano Nichele, IDI

Norwegian University of Science and Technology
Department of Computer Science

Summary

Traditional *Cellular Automata* (CA) transition rules are encoded as tables that grow quickly when the number of cell states or the size of the CA neighborhood increases. For methods that search for good transition rules, such as genetic algorithms, the space of possible encodings also grows rapidly with both parameters. This thesis investigates replacing the traditional encoding with *Compositional Pattern Producing Networks* (CPPNs), a Neural Network-like structure. The search for good CPPN-based transition encodings is performed with the *NeuroEvolution of Augmenting Topologies* (NEAT) genetic algorithm.

A software framework is implemented and CA problem solving experiments are performed. The problems investigated include both morphology problems and computational problems. The results found are diverse, with some problems solved easily, some with moderate difficulty and some not at all.

In another experiment, the new CA framework is also modified to extend the cellular model with environmental information. The model lends itself very easily to extension, and the result is that previously difficult tasks become solvable.

In addition to the task-solving experiments, another experiment is performed to investigate the relationships between the mechanisms of the NEAT algorithm and the properties of the CPPN encoding and the CA model. The results from this experiment can inform decisions about parameters in future experiments.

A variation of the NEAT algorithm called *novelty search* is also implemented. While it is not able to produce any more interesting results than the objective NEAT search is, it does lead to some insights about the algorithm that could lead to success in the future.

The results of the experiments indicate that the new combination of encoding and algorithm has merit. In addition to the CA model, the algorithm and encoding may also have applications in other morphogenetic engineering situations.

Sammendrag

Tradisjonelle *cellulære automater* (*Cellular Automata*, CA) har overgangs-regler kodet som tabeller som vokser hurtig når antallet celle-tilstander eller størrelsen på CA-nabolaget øker. For metoder som søker etter gode overgangs-regler, for eksempel genetiske algoritmer, vokser rommet av mulige kodinger også fort sammen med begge disse parameterene. Denne oppgaven undersøker å erstatte den tradisjonelle kodingen med *Compositional Pattern Producing Networks* (CPPNs), en nevralt nettverk-lignende struktur. Søkingen etter gode CPPN-baserte overgangs-kodinger blir utført med den genetiske algoritmen *NeuroEvolution of Augmenting Topologies* (NEAT).

Et software-rammeverk blir implementert og eksperimenter for å løse CA-oppgaver blir utført. Oppgavene som blir undersøkt inkluderer både morfologi-problemer og beregings-problemer. Resultatene er mangfoldige, da noen oppgaver blir utført enkelt, noen med moderat vanskelighetsgrad, og noen ikke blir løst i det hele tatt.

I et annet eksperiment blir the nye rammeverket modifisert for å utvide celle-modellen med miljø-informasjon. Modelen er veldig enkel å utvide, og resultatet er at oppgaver som tidligere var vanskelige blir mulige å løse.

I tillegg til eksperimenter med å løse oppgaver, undersøker et annet eksperiment forholdet mellom mekanismene i NEAT-algoritmen og egenskapene til CPPN-kodingen og CA-modellen. Resultatene fra dette eksperimente kan bidra til å finne bedre parametere til fremtidige eksperimenter.

En variasjon av NEAT-algoritmen kalt *novelty search* blir også implementert. Selv om denne algoritmen ikke produserer resultater som er mer interresante enn det vanlig NEAT produserer, gir eksperimentet noen innblikk om algoritmen som kan føre til suksess i fremtiden.

Resultatene fra eksperimentene indikerer at den nye kombinasjonen av koding og algoritme kan ha nytte. I tillegg til CA-modellen, kan algoritmen og kodingen også ha bruksområder i andre morfogenetiske domener.

Preface

This thesis project concludes my five-year integrated Master of Science degree at the Department of Computer Science at the Norwegian University of Science and Technology in Trondheim. The thesis project has been supervised by Stefano Nichele. I would like to thank Stefano for coming up with the idea for this interesting project, and for his guidance and feedback during the process.

The work presented within started with the specialization project in the fall semester of 2016, and continued in the spring semester of 2017. Parts of the experiments presented in this thesis (Section 4.2) were performed and analyzed during the specialization project [1]. The report from the specialization project was further refined with co-authors Stefano Nichele, Gunnar Tufte and Sebastian Risi to a paper which at the time of writing has been accepted but not yet published in the *IEEE Transactions on Cognitive and Developmental Systems* [2]. I would also like to thank all my co-authors for their efforts to get this work published.

Mathias Berild Ose
Trondheim, June 2017

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Cellular Automata (CA) were first conceptualized and introduced in the 1940's and 1950's. Around the same time, the idea of the *evolutionary algorithm* was also being developed independently. Both of these concepts take inspiration from nature, and thus fall into the category of *artificial life*. One goal of this field of research is to create artificial systems that are of complexity comparable to that of biological systems found in nature.

One way to try to achieve this goal is to combine these two distinct concepts: Cellular systems designed by evolution. Many different kinds of tasks have been solved by cellular systems that have been created this way. However, more complex tasks and more complex models means greater spaces of possible solutions that the evolutionary algorithm must search. For a traditional genetic algorithm it can be both time consuming and challenging to find good solutions.

One of the possible ways to remedy this problem is to replace the traditional table-based encoding of CA transition rules with a different encoding: an encoding that supports a more complex evolutionary algorithm. This thesis describes the investigation of using *Compositional Pattern Producing Networks* (CPPNs) as the data structure for transition rules, and the *NeuroEvolution of Augmenting Topologies* (NEAT) genetic algorithm for evolving these CPPNs. Both normal NEAT (objective search) and a variation called *novelty search* is investigated.

With CPPN-based transition functions there is not a linear relationship between the input-output size and the size of the encoding. The algorithm starts with a small encoding and iteratively adds features to it and adjusts them until an optimal solution is found. This *complexification* mimics how biologists believe life on Earth developed.

The CPPN structure is very generic and can be easily adapted to support novel cellular models. By augmenting the cellular model with *environmental information*, the new model can emulate how development in nature is affected by the environment it occurs in.

To test this new combination of model, encoding and algorithm, which we call *CA-NEAT*, a custom Python framework was built to run simulations in software. The framework was tasked with solving various CA tasks of different difficulties, with different degrees of success.

1.1 Research Questions

The questions that the thesis aims to answer include:

- What kinds of CA problems can CA-NEAT solve? What kinds of CA problems are difficult, and why?
- Does adding information to the environment that the CA can access help with solving difficult tasks?
- How do the measurable properties of the architecture, encoding and algorithm develop over time? How do they correlate?
- What is the effect of the various mechanisms of NEAT on the end result and the development over time?
- Does changing the algorithm to the novelty search variant give improved results at tasks where objective search struggles?

1.2 Structure of the Thesis

The structure of the thesis is as follows: Chapter 2 details background theory about the topics the thesis is concerned with, the motivation for the project, and some of the recent related work. Chapter 3 describes the development and functionality of the CA-NEAT framework. Chapter 4 describes the experiments performed with CA-NEAT as separate sections, including results and some discussion pertaining only to the particular experiments. Chapter 5 discusses the results from a more overarching perspective, and the possibilities for future work. Chapter 6 offers answers to the research questions and some concluding remarks.

Chapter 2

Background & Motivation

2.1 Complex and Biologically-Inspired Systems

If you try and take a cat apart to see how it works, the first thing you have on your hands is a nonworking cat.

Douglas N. Adams
The Salmon of Doubt [3]

.

Complex systems is an umbrella category consisting of a variety of topics from a variety of domains, such as mathematics, computer science and biology. Figure 2.1 shows one possible ”taxonomy” of complex systems. It is not immediately obvious why these topics should be grouped together. The word *complex* is related to *complicated*, synonymous with *difficult*, *intricate* and *perplexing* [4]. In 1962, Herbert Simon proposed a definition of complex systems as “made up of a large number of parts that interact in a non-simple way” [5]. Hiroki Sayama later elaborated this definition:

> Complex systems are networks made of a number of components that interact with each other, typically in a nonlinear fashion. Complex systems may arise and evolve through self-organization, such that they are neither completely regular nor completely random, permitting the development of emergent behavior at macroscopic scales. [6]

The idea of studying complex systems is that by grouping these topics together, it is possible to start seeing similarities across domains, and find insights about one topic that may be applicable to other topics.

A common property of complex systems is *emergence* over scale. Sayama defines emergence as “a nontrivial relationship between the properties of a system at microscopic

[1] `https://commons.wikimedia.org/wiki/File:Complex_systems_organizational_map.jpg` (CC BY-SA 3.0)

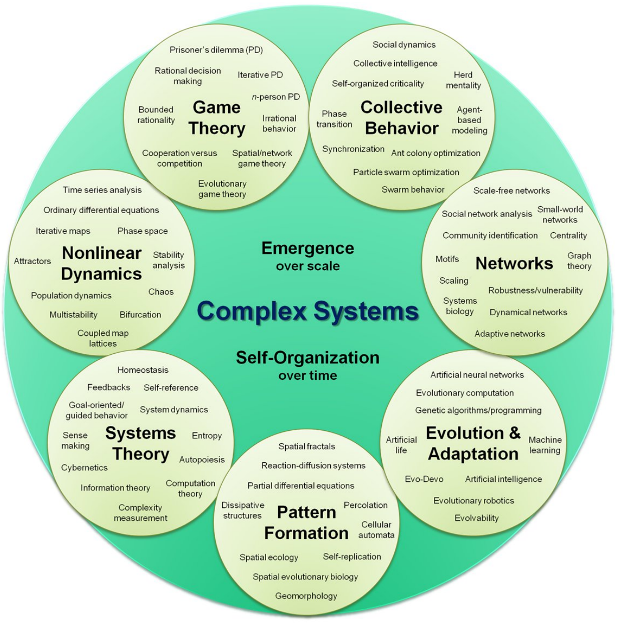


Figure 2.1: Complex systems taxonomy. Created by Hiroki Sayama [1].

and macroscopic level". This means that a process which can be observed at the macroscopic level (e.g. the body functions of a cat) can not be explained by studying the individual components that make up the system (looking at the cells that make up the body). Instead, both the individual components and *how they interact* must be understood.

The other common property of complex systems is *self-organization* over time. Sayama defines it as "a dynamical process by which a system spontaneously forms nontrivial macroscopic structures and/or behaviors over time". One example is magnetization of metals, where the initially random configuration of "spins" (the components of the larger system) orient themselves over time, so that the magnetic vector of all the individual spins and that of the whole system become the same [7].

The behavior seen in complex systems can be characterized by these two properties. Often the behavior is a combination, rather than a clear instance of one or the other.

Many of the topics seen in Figure 2.1 are based on concepts found in nature. These belong to the group of *bio-inspired systems* and to the category of *artificial life*. Since the infancy of modern computing in the 1940s, computers have gradually gained the capability to perform many tasks. Along the way, computer scientists and engineers have sometimes looked to nature for inspiration and goals to reach for. Often, approaching problems from engineering, mathematical and logical perspectives have yielded good results. But some times, the analytical approach leads to a dead end. Some tasks that are trivial for a human to perform, can be practically impossible for a programmer to codify [8]. This is where the bio-inspired approach may help. Since nature has been able to create biological "machines" that can solve these tasks, then perhaps borrowing nature's methods will allow computers to do the same.

2.1.1 Morphogenetic Engineering

The field of *morphogenetic engineering* [9] studies the confluence between biological and artificial systems, and how to create new systems that straddle the the divide. Doursat et. al. defined four categories of morphogenetic engineering that they classified existing techniques by:

Category I: by Constructing
Where individual components precisely assemble themselves together into a larger system for example by physically attaching to each other.

Category II: by Coalescing
Where individual components flock or swarm in a fluid formation, acting as a larger system while the components are still physically separate.

Category III: by Developing
Where a systems starts as a simple model and iteratively adds complexity by division or aggregation.

Category IV: by Generating
Where the systems starts as a simple ”grammar” model and iteratively adds complexity by rewriting itself.

Within the third category we find many of the concepts that this thesis is concerned with, such as evo-devo and morphogenesis.

2.2 Cellular Automata

Cellular Automata (CA) were first invented in the 1940s by John von Neumann and Stanislaw Ulam as mathematical models of computation [10]. They were inspired by biological organisms, and created a model that could emulate some of their interesting and useful properties, such as multi-cellular development (e.g. embryogenesis), reproduction (clonal or sexual) and robustness (e.g. self-repair).

In the following decades, as modern computers emerged, the concept of CA became the basis for the field of *Cellular Computing* (CC). As the performance of ”conventional” computers kept increasing dramatically (as described by *Moore’s law*)[11], CC never became the basis for the mainstream computers that we use today, but CA and CC remained an area of research by mathematicians and computer scientists, and as part of the larger field of artificial life. More recently, as Moore’s law has faltered and this rate of growth of performance has diminished, some have started to look for new methods that can lead to renewed performance growth. Parallel computing has helped a lot, but it has shown itself to be difficult in practice. Some, such as Michael J. Flynn have speculated that CC might be the path forward [12] . Matthew Cook proved that a CA of a certain configuration can be Turing complete [13], giving further credibility to this idea.

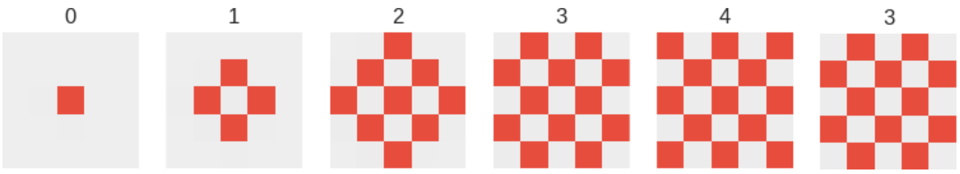


Figure 2.2: An example binary 2D CA. At time step 5 the CA state is the same as that in time step 3, meaning the CA has entered a cyclical attractor with a period of 2 (oscillation).

2.2.1 CA Definition

A CA consists of a grid of very simple units called cells. A cell can be in one out of a finite set of states, and can change between states based on input to the cell. As such the CA can be considered as a grid of identical Finite State Automatons. Sipper [14] described the three core principles of CC, which also apply to CA in general:

Simplicity
A cell is simple and can do very little by itself.

Vast Parallelism
The number of cells is very large, much more than the number of processors in a conventional parallel computer.

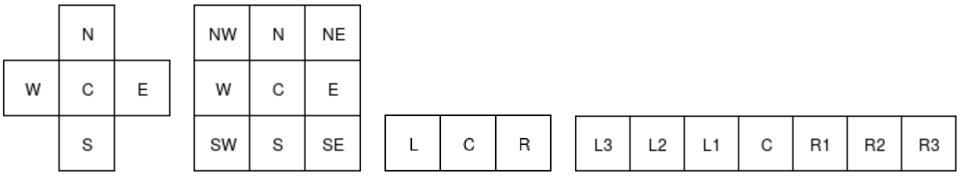
Locality
All interactions between cells take place on a purely local basis. No cell knows or controls the entire system.

The cells in a CA can ”see” only their closest neighboring cells. They use this limited information in conjunction with some set of rules to transition from one cell state to another. Depending on the starting configuration of the whole system and the rules, it is possible to observe interesting emergent or self-organizing behavior over time and space. These interesting CA often find an *attractor* [15, 16]. If a sequence of CA states repeat periodically it is called a *cycle attractor*, and if the CA stabilizes into a permanent, fixed state is is called a *point attractor*. Figure 2.2 shows an example of a CA that enters a cyclical attractor. Binary CA, with only two possible cell states, are the most commonly seen and studied. Greater numbers of cell states is possible, but the increase in degrees of freedom can lead to some difficulties in implementation.

In many CA models, one of the possible state is designated as the *quiescent* state. This state is ”dead”, and can only come alive when something in the neighborhood is already alive. This is called the *quiescent rule*.

2.2.2 Transition Rules

Christopher Langton [17] formally defined finite CA as consisting of a finite set of cell states Σ of size $K = |\Sigma|$, a finite input alphabet α, and a transition function Δ. Each cell has a N-sized neighborhood. The number of possible neighborhood states can be expressed by equation (2.1).



(a) $N = 5$ (Von Neumann) **(b)** $N = 9$ (Moore) **(c)** $N = 3$ **(d)** $N = 7$

Figure 2.3: Some examples of common neighborhood shapes. The two common 2D shapes are named for John von Neumann and Edward F. Moore.

$$|\alpha| = |\Delta| = |\Sigma^N| = K^N \tag{2.1}$$

The transition function for a CA must thus encode $|\alpha|$ different mappings of N inputs to one of K outputs. The number of possible unique transition function behaviors is thus $K^{(K^N)}$.

2.2.3 The λ Parameter and the Edge of Chaos

When Langton studied the elementary CA ($N = 3, K = 2$) [17], he created a metric to help determine if a rule is ordered, chaotic, or something in between, which he called λ. It is defined by K, N and the number of transitions that goes to the quiescent state, n.

$$\lambda = \frac{K^N - n}{K^N} \tag{2.2}$$

Langton found that low elementary CA λ values resulted in ordered behavior, either settling into a static state or repeating periodically. With high λ values, the CA became chaotic, losing all useful information in the noise. But at the critical border region between order and chaos, interesting behaviors and computation could occur. This area has come to be called the *edge of chaos*.

In the study Langton investigated the λ of 1D binary CA, but the measure can be used on any CA. The 2D binary transition function shown in Table 2.1 has 17 input combinations that lead to the quiescent (0) state ($n = 17$), and thus $\lambda \approx 0.47$.

2.2.4 Finding Interesting Transition Rules

Traditionally Δ has been encoded as a complete mapping $\Delta : \Sigma^N \to \Sigma$, which can be implemented as a lookup table. Table 2.1 shows an example of a table encoding for the CA in Figure 2.2. This works very well for smaller cases such as the elementary CA. But when working with non-trivial CA where both K and N can be relatively large numbers, it becomes a problem both to store the mapping Δ in an efficient way, and the space of possible Δ becomes too large to be explored by exhaustive enumeration.

Designing Δ with interesting behavior by hand is possible, but it is time-consuming and impractical for problems of greater dimensions. Using adaptive optimization algorithms to explore the space of possible solutions is more feasible. This is not guaranteed

Table 2.1: An example table-based transition rule that exhibits the same behavior as seen in Figure 2.2. $N = 5, K = 2$ gives $|\alpha| = 32$, the height of the table. The neighborhood shape is "Von Neumann" (Figure 2.3a).

North (t_0)	West (t_0)	Center (t_0)	East (t_0)	South (t_0)	Center (t_1)
0	0	0	0	0	0
0	0	0	0	1	1
0	0	0	1	0	1
0	0	0	1	1	1
0	0	1	0	0	0
0	0	1	0	1	0
0	0	1	1	0	0
0	0	1	1	1	0
0	1	0	0	0	1
0	1	0	0	1	1
0	1	0	1	0	1
0	1	0	1	1	1
0	1	1	0	0	0
0	1	1	0	1	0
0	1	1	1	0	0
0	1	1	1	1	0
1	0	0	0	0	1
1	0	0	0	1	1
1	0	0	1	0	1
1	0	0	1	1	1
1	0	1	0	0	0
1	0	1	0	1	0
1	0	1	1	0	0
1	0	1	1	1	0
1	1	0	0	0	1
1	1	0	0	1	1
1	1	0	1	0	1
1	1	0	1	1	1
1	1	1	0	0	0
1	1	1	0	1	0
1	1	1	1	0	0
1	1	1	1	1	0

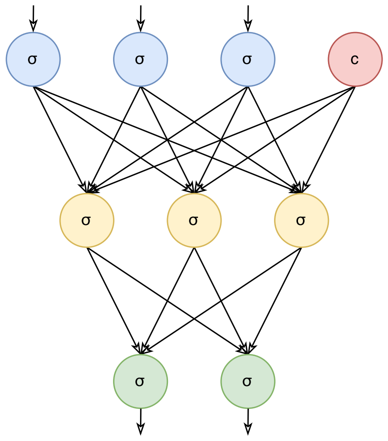


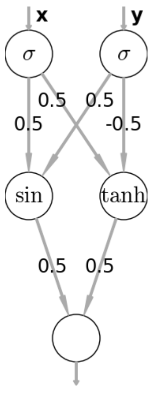
Figure 2.4: An example layered ANN with three input neurons, one bias neuron, three hidden neurons and two output neurons. Each connection between neurons also has some weight that scales the value being passed (not shown).

to produce good results though. The use of table-based encodings also puts certain limitations on the search. Using other encodings may enable new, more powerful search algorithms that can possibly resolve this issue.

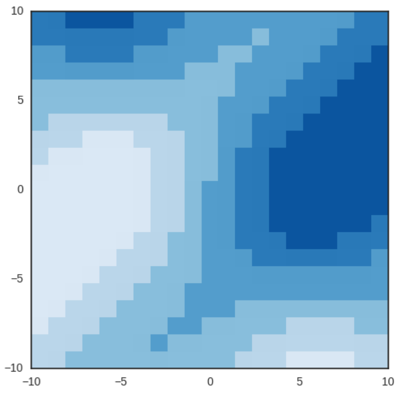
2.3 Artificial Neural Networks

Artificial Neural Networks (ANNs) [18, Chapter 1] have been used in many different applications related to artificial life and intelligence, such as robotics or machine learning. An ANN is a directed graph structure, with vertices (referred to as neurons) and edges (referred to as connections). Input values are fed into the first layer of neurons and passed through the connections to the next layers. All the connections to one neuron is added together and input to the neuron. In each neuron some activation function transforms the input to a new value, and in each connection the value is scaled by some weight. There can also be *bias* neurons, outputting values that are constant, not determined by input.

This is inspired by neuroscience, with the brain consisting of neurons and synapse connections. ANNs are useful because they consists of many discrete parts that can be individually or collectively tuned by some adaptive process, and are easily expanded. The *universal approximation theorem* [19] shows that relatively simple ANNs can approximate a wide variety of functions, and the field of deep learning shows that a large complex structure with enough tuning can perform very complex tasks, such as image classification or natural language processing [18]. Figure 2.4 shows an example ANN with three inputs and two outputs. An example of a use case for this structure could be to control a robot with three sensors and two motors.



(a)



(b)

Figure 2.5: An example composition of the sigmoid, sinusoid and hyperbolic tangent functions. The discrete coordinates of (b) are first normalized to $[-1.0, 1.0]$ and then mapped to various output values through the CPPN (a).

2.3.1 Compositional Pattern Producing Networks

A *Compositional Pattern Producing Network* (CPPN) is an *artificial development encoding* introduced by Kenneth O. Stanley in 2007 [20]. CPPNs are structurally similar to ANNs, but differ in the use case. Various techniques designed for ANN development and analysis may also be used for CPPNs.

Just like an ANN, a CPPN consists of a set of nodes with activation functions, weights and biases, as well as weighted connections between nodes. Also like in an ANN, external values are input to the first layer, then undergo transformation by weights and activation functions before being outputted by the final layer. This can be thought of as a composition of functions producing a pattern, hence the name. An ANN is usually structured with neurons of the same activation functions, arranged in layers, whereas a CPPN has few such restrictions on topology and layer-wise heterogeneity.

Figure 2.5 shows an example CPPN and its output when mapped over a 2D Cartesian grid. A CPPN is able to produce a pattern without multiple steps of development, in contrast to e.g. a CA where local interactions and time is required. CPPNs have been used both to produce patterns for the sake of the patterns, e.g. as evolutionary art [21], but also to create patterns which are used in a larger process, such as machine learning [22] and robot control [23].

2.4 Artificial Evolution and Development

Artificial development and *artificial evolution* (evo-devo) takes inspiration from biology in order to explore large and complex solution spaces for some given problem. *Evolutionary algorithm* (EAs) are a type of algorithms inspired by evolution in nature [24]. The *genetic algorithm* (GA) is an EA that models natural selection. In a typical GA setup [25, 26], relatively simple representations of solutions are encoded as *genotypes*. Through some *development process* a genotype may be transformed into a *phenotype* which can be used to attempt to solve the problem at hand. The performance of the phenotype at solving the problem is the *fitness* of that individual genotype.

The fitnesses of a *population* of different individuals are compared. A selection process picks individuals from the population that get to reproduce. This selection process is usually stochastic, with a bias towards picking the individuals with the highest fitness, but some chance of picking a less fit individual now and then. An optional part of selection can be to eliminate some fraction of the population that performed poorly, excluding them from pair selection entirely. This is analogous to creatures in nature dying before reaching sexual maturity. The combination of these mechanisms creates a *selection pressure* that drives the overall population towards higher average fitness.

Individuals that are selected for reproduction are paired up. The genotypes of the pair are combined in some fashion to create a new genotype. In addition to the combination, random mutations may also be applied in order to produce new features not present in either parent. As new individuals are born, the older parent generation dies out, so that the entire population is replaced. In some cases the very best individuals of the parent generation are cloned directly into the next generation, ensuring that their well-performing genotype continues to be present until some better genotype comes along. This is called *elitism* [27].

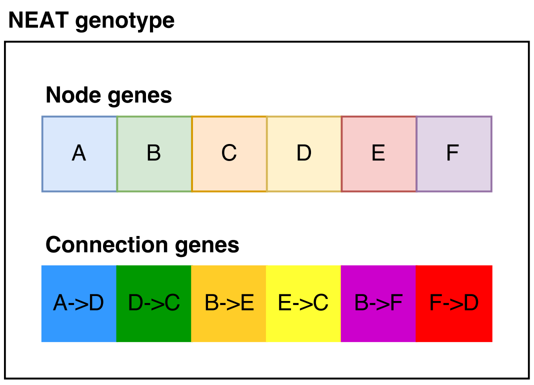
Starting out with an arbitrary initial population and repeating this generational algorithm, it is often possible to find novel genotypes that encode good solution to the problem at hand. Like other optimization algorithms that search a space of solutions, there is a risk of getting stuck in a local maximum and never finding a global maximum, an optimal solution to the problem at hand. There are many kinds of parameters, such as mutation rate or selection function that can be tweaked to try to avoid this.

When a phenotype is developed into a genotype, it is either possible that the genotype encodes the entire phenotype explicitly, or it is possible that the development process augments the information stored in the genotype to create a more complex phenotype. This is called respectively *direct* and *indirect encoding* [28]. Indirect encodings allows the evolutionary search to explore a space of genotypes that is smaller than the space of possible phenotypes. This is the way development happens in nature, where simple genomes are developed into complex lifeforms [28].

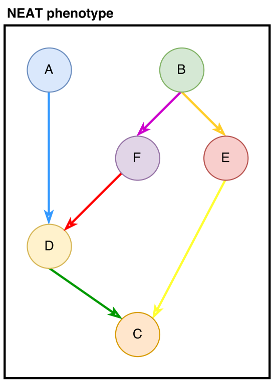
2.4.1 NEAT

NeuroEvolution of Augmenting Topologies (NEAT) is a genetic algorithm variant introduced by Kenneth O. Stanley and Risto Miikkulainen in 2002 [29], designed specifically to evolve ANNs. When introducing CPPNs [20], Stanley also introduced the CPPN-NEAT variation of the algorithm.

A NEAT genome consists of genes that encode nodes and connections between them. Figure 2.6 shows an example genotype to phenotype mapping. NEAT starts with an initial population of very simple networks, typically with just the input and output nodes and connections between them. Over generations, more nodes and vertices are added or disabled, activation functions are changed, and weights are adjusted. The process of gradually expanding the genome is called *complexification*, and reflects how life on earth is believed to have started with simple organisms and gradually evolved into more complex creatures



(a) Genotype



(b) Phenotype

Figure 2.6: An example NEAT genotype and corresponding phenotype. This example only shows the topology that the genotype encodes, leaving out the weights, biases and activation functions.

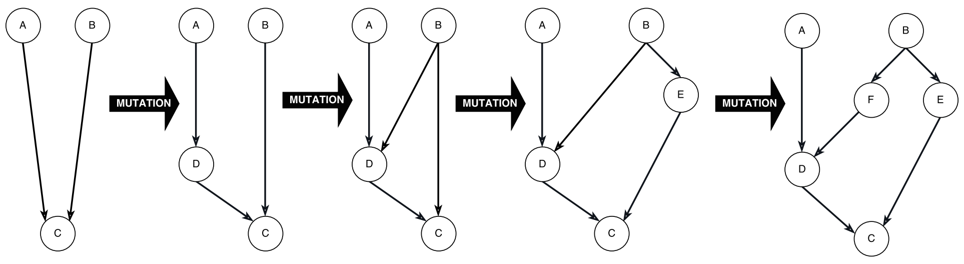


Figure 2.7: An illustrated example of NEAT mutation starting with a basic network of only two inputs and one output. Through the sequence, neurons and connections are added until the network is equal to that in Figure 2.6. This example shows only mutation, leaving out the crossover operation which is also part of NEAT.

[30, 31]. Figure 2.7 shows an example of how mutation could gradually complexify a NEAT phenotype network.

The genes that make up a NEAT genome are marked with an *innovation number* so that they may be recognized as the same gene in different individuals. As new features are added to the genomes, the individuals making up the population become gradually less similar. The degree of similarity is measured through a measure called the *compatibility distance*. When the distance between individuals pass a certain threshold, they are segregated into separate species. This process is called *speciation*. Pair selection for reproduction happens within species. Typically the species that have the most fit individuals will produce more children, while the less fit species will produce fewer (but not 0) children.

When a new species appears with a new feature, the feature will not be tuned and likely affect the fitness of the individuals negatively. NEAT protects new species for a certain amount of time, allowing them time to adjust before being evaluated and, if performing poorly, being extincted to make more room for the more fit species.

One notable use case of NEAT is called *HyperNEAT* [32]. In this process, NEAT is used to evolve CPPNs whose output determine the topology of ANNs. This is useful because it allows the ANN to scale easily, since the CPPN can just take more input and output more topology. If the evolved CPPN has useful output at a small scale, it should also have a useful output at a large scale. HyperNEAT has been used to create scalable neural networks used in applications such as playing Go [33] with different board sizes, controlling an ”octopus” arm with variable number of segments [34] and multi-agent learning [22].

2.4.2 Novelty Search

Novelty search is another genetic algorithm variant, introduced by Joel Lehmann and Kenneth O. Stanley in 2008 [35]. It is designed to be good at *deceptive* tasks, where local maximas in the fitness landscape can ”deceive” conventional algorithms and prevent the search from finding the global maximum.

Novelty search avoids this by eschewing the fitness measure entirely during the search, and instead rewards *innovation*, giving higher scores to individuals that exhibit previously unseen behavior. To find what behavior is new, an archive of seen behaviors is maintained. A *distance metric* must be selected that is appropriate for the problem at hand. For example, if the behavior of the phenotypes produces strings, an *edit distance* measure such as the *Levenshtein distance* can be used. For each genotype the k nearest neighbors (in terms of behavior) are found, and the average of these distances can be used as the *novelty metric*. If a new behavior is sufficiently novel, it is added to the archive.

The algorithm is otherwise equal to NEAT, with the novelty score substituting for the objective fitness score. This produces a population with a large variation of behaviors. The hope is that this way, some individual ”stumbles” into the global maxima. Whether this has occurred can be determined by using the objective fitness test on the individuals of the archive.

Novelty search as been applied to several types of problems successfully, including agent path-finding in deceptive mazes [35], machine learning clustering and classification [36, 36] and swarm robotics [37].

2.5 Motivation

As mentioned in Section 2.2.4, using a more advanced encoding for CA tasks may enable a more advanced search algorithm. This combination may then lead to successfully solving tasks that are considered difficult with the classical encoding and algorithm.

In fields such as neuroevolution, NEAT has been shown to produce useful patterns, without using temporal development and local interactions. However, these tools *are* used by nature in processes such as embryogenesis, that we can model with CA. In the quest to advance cellular systems towards biological levels of complexity, it is worth investigating if this technique, which is proven in one field, lends itself to being adapted as a part of a process in another field.

By testing a new model on a few select tasks, we may not only learn whether the model can accomplish the task, but we may gain more general insights into the components (CA, the architecture; CPPN, the encoding; NEAT, the algorithm) that make up the model, shedding further light on domains such as artificial life and morphogenetic engineering. If something works well, we can try replacing a component with a new one to see if it still works, or if something does not work well, we can try replacing a component to see if that changes anything.

2.6 Related Work

In [38], Wolper and Abraham used evolution and CPPNs to find seed patterns for Conway's Game of Life [39], which is a type of CA. The Game of Life has a set of defined transition rules, so CPPN-NEAT was not used for exploration of transitions. They tried both normal CPPN-NEAT (objective search) and novelty search. The results were varied, but the conclusion was in support of further research into using CPPNs for CA problems.

Many different kinds of CA encodings have been investigated previously with positive results. With *Conditionally matching rules* [40, 41, 42], the table of transition rules is not a complete enumeration of all possible inputs, but a sequence of conditions that, if all asserted, determine the next state of the cell. In *instruction-based development* [43, 44, 45, 46, 47], the transition function encodes a set of instructions (a small program if you will) that transforms act on the input to produce the output. With *self-modifying cartesian genetic programming* [48], the transition function is a genetic program. In *variable length gene regulatory networks* [49], the genomes encode networks that mimic the network of cells found in nature.

Nichele and Tufte [46, 47, 45] have studied complexification during evolution of CA transition rules, both with table-based encodings and with instruction-based encodings.

Chapter 3

Implementation

In order to conduct the experiments presented in this thesis, a custom framework has been created in Python. The source code is available at Github[1]. The system consists of a mix of library components and self-made components.

The CA subsystem was built from scratch. It supports 1D and 2D grids with various border conditions (finite, toroidal, expanding), and should be easily extensible for other cases in the future. For each problem there is a problem-specific fitness function which receives a genotype as input from the NEAT subsystem, develops the transition function and iterates the CA before evaluating the performance and returning a fitness value to the NEAT system.

The NEAT portion of the system is mostly based on the library `neat-python`[2]. Data structures for genomes and networks as well as various functions have been used without modifications. The main evolutionary loop was re-implemented with modifications. This was done for multiple reasons, including to take advantage of parallelism using `Celery`[3] and to store the results in a database using `SQLAlchemy`[4]. Other software dependencies include `matplotlib`[5] and `seaborn`[6] for visualization, and `dill`[7] for data and code serialization.

3.1 CA-NEAT

Since the `neat-python` library provides a NEAT implementation, the majority of the development in this project consisted of creating the CA subsystem and the interfacing between CA code and NEAT code.

[1]`https://github.com/mathiasose/CA-NEAT/`
[2]`https://github.com/CodeReclaimers/neat-python/`
[3]`http://celeryproject.org/`
[4]`http://sqlalchemy.org/`
[5]`http://matplotlib.org/`
[6]`http://seaborn.pydata.org/`
[7]`https://github.com/uqfoundation/dill`

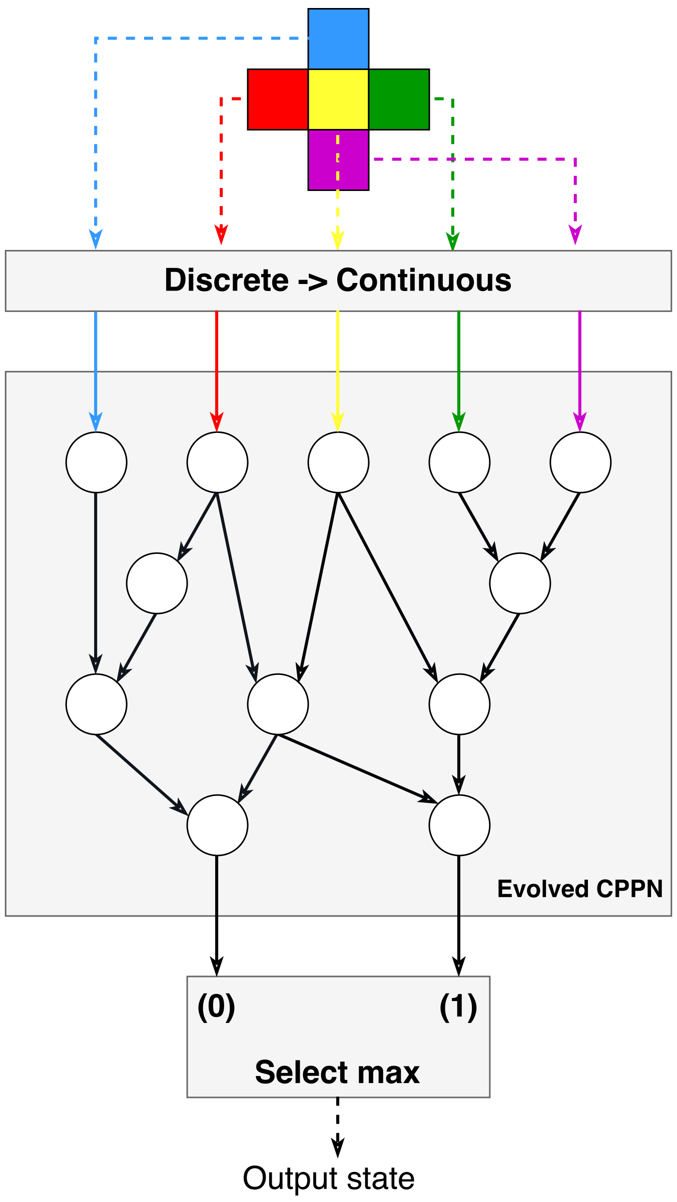


Figure 3.1: Overview of how to use a CPPN as a CA transition rule.

Table 3.1: Possible activation functions

Type	Equation
Sigmoid	$f(x) = \frac{1}{1+e^{-x}}$
Hyperbolic tangent	$f(x) = tanh(x)$
Sinusoid	$f(x) = sin(x)$
Gaussian	$f(x) = ae^{-\frac{(x-b)^2}{2c^2}}$ a, b, and c are constants
Rectified linear unit	$f(x) = max(0, x)$
Identity	$f(x) = x$
Clamped	$f(x) = \begin{cases} 0 & x \leq 0 \\ x & 0 < x < 1 \\ 1 & x \geq 1 \end{cases}$
Inverse	$f(x) = \frac{1}{x}$
Logarithmic	$f(x) = log(x)$
Exponential	$f(x) = e^x$
Absolute value	$f(x) = \|x\|$
Hat	$f(x) = \begin{cases} 1 - \lvert x\rvert & \lvert x\rvert < 1 \\ 0 & otherwise \end{cases}$
Square	$f(x) = x^2$
Cube	$f(x) = x^3$

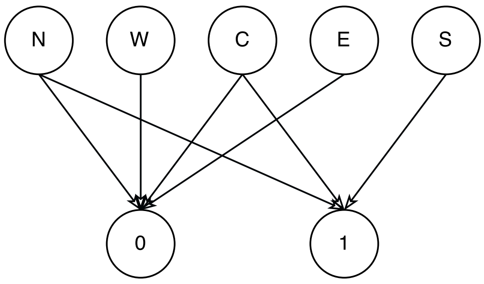


Figure 3.2: Example first-generation CPPN with 7 out of 10 possible connections. $N = 5$ input nodes corresponds to the "Von Neumann" neighborhood shape, and $K = 2$ output nodes correspond to a binary CA.

Figure 3.1 gives an illustrated overview of the mapping from neighborhood input to a new cell state. The cell states are discrete values from a finite set, but the activation functions used in the CPPN expect inputs and outputs from the real numbers ($\mathbb{R}$). Therefore there a mapping is performed before the CPPN input layer, assigning a continuous value to each possible cell state. The mapping assigns each value in the finite state set to a value in the range $[-1.0, 1.0]$, evenly distanced.

The mapped values are then sent to the CPPN as input. After the N values have propagated through the network there are K different outputs. These are each paired with one of the cell states, and the cell state corresponding to the CPPN output with the highest activation value is selected as the output and becomes the next state of the cell.

The activation functions used in the experiments in this thesis are listed in Table 3.1.

The fitness evaluation function is specific to each problem. In all experiments in this thesis, the fitness values returned are between 0 and 1, with 0 representing a complete failure and 1.0 representing a perfectly accomplished task.

3.1.1 Mapping CA-NEAT Rules to Traditional Rules

The transfer between the discrete and continuous number domains that happens before and after the CPPN is activated, means that many CPPNs that are have different topologies, activation functions and weights, will actually exhibit the exact same behavior as CA transition rules. Enumerating all the possible inputs and recording the corresponding output of the transition functions creates a classical table/string representation of the transition functions. We will call this the *behavior* of the transition function, and use this representation in analysis.

3.1.2 Identifying Vestigial Structures

With NEAT adding and removing nodes and connections, sometimes nodes end up not being connected to any of the output nodes. They thus have no effect on the output of the network, but are still present in the structure. In biology, features that are present but do not serve any purpose are called *vestigial features* [50].

To determine which CPPN nodes are in use and which are vestigial, a backwards graph traversal can be done from the output nodes, following the enabled connections and marking all the nodes encountered. This can be used both for analysis of genotypes, as well as to simplify the visualizations of networks.

3.1.3 Extending CA-NEAT with Environmental Information

Another aspect of CA-NEAT that can be explored, is the possibility of using additional inputs that are not the cell states of the neighbors. A table-based transition function is a complete mapping from all the possible inputs to a specific output, meaning it must be possible to enumerate all the possible inputs. A CPPN structure such as a CA-NEAT phenotype is less restricted in what kinds of inputs it can accept.

In biological cell systems, external factors can have large effects on the development. For instance, a growing plant will choose which direction to expand in based on where it can find light or water in its environment. In embryogenesis of animals, spatial information

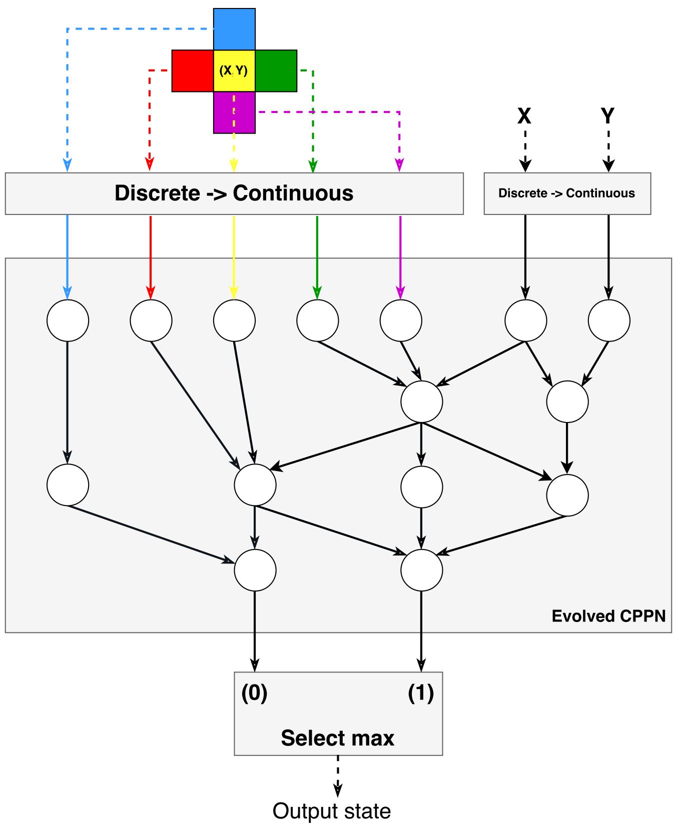


Figure 3.3: An example extension of CA-NEAT.

is used to produce the correct layout of the body, with brain cells forming in the head, skin cells on the surface of the body, and so on. This is not all encoded in the genome of the organism, but arises from the interaction between the genome rules and the environment.

In a CA experiment, adding environmental information to the transition introduces a factor of indirect encoding, since the same genotype can then produce different phenotypes in different environments.

Extending CA-NEAT to accept new kinds of inputs is very simple. The NEAT part only needs to be told how many input nodes the networks should have. Then the fitness evaluation function, which must be custom for each problem anyway, must be programmed to extract the necessary values from the cellular model and input them to the CPPN.

A concrete example is the extension used in the experiment in Section 4.3. Figure 3.3 illustrates this. In addition to the neighborhood information, the coordinate values of the cell in question is also input to the network, which therefore has two additional input nodes. The coordinate values are also mapped to the $[-1.0, 1.0]$ domain, but by a separate mapping function. Other than at the input to the transition function, the extended CA-NEAT functions exactly like the unextended version.

3.2 Novelty Search

Novelty search has been added in as an extension to the existing CA-NEAT framework. Many CA problems can be ”deceptive” to a GA and to the programmer tasked with creating an appropriate fitness function. The CA must transition through a sequence of intermediate states before arriving at the desired target state, but it is not necessarily clear what intermediate behavior should be rewarded in order to find the final result. This makes a good case for testing novelty search for these problems.

In order to extend CA-NEAT for novelty search, some phenotype value needs to be measured and compared in order to calculate novelty. The enumerated ”string” representation of the transition function was selected for this. The *Hamming distance* between different strings can then be calculated and used as a measure of novelty distance. The distances are normalized to the $[0.0, 1.0]$ range and the mean of the $k = 15$ closest distances is used for the innovation score of the individuals.

The threshold for adding a genotype to the innovation archive is initialized at 0.5, but is dynamically adjusted throughout the run. If 0 genotypes are added in a generation, the algorithm will pick one at random to add anyway, and also decrease the threshold by multiplying with a factor between 0.95 and 1.0. If more than 5 individuals are added in a generation, they will all be added, and afterwards the threshold is increased by multiplying with a factor between 1.0 and 1.05. The adjusting factors are picked at random from a uniform distribution. This should allow the threshold to eventually settle into an appropriate value, and the randomness should prevent it from oscillating between two different values.

Chapter 4

Experiments

4.1 Overarching Methodology

The experiments about to be presented in Sections 4.2, 4.3 and 4.4 have some commonalities. They are about selecting a CA problem and letting CA-NEAT have a go at solving it. In these experiments, the same basic configuration of CA-NEAT is used. The size of the CA neighborhood (equal to the number of CPPN inputs) and the number of cell-states (equal to the number of CPPN outputs) is changed to be appropriate to the problem at hand. The fitness evaluation function is also custom to every problem.

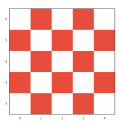
During development of the system, a variation of different configurations were tried for different problems. When deciding on the experiments to collect results from for this paper, a deliberate choice was made to use the same CPPN-NEAT configuration and as close to the same CA configuration as possible for all problems. This makes it easier to make comparison between experiments, but means that the settings chosen may favor some experiments over others. Appendix A lists the NEAT parameters used in all the experiments. Notably, the mutation weights are biased towards adding nodes and connections more so than removing them, leading to the average network size growing over time.

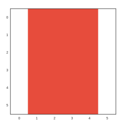
Each experiment consists of 100 independent trials with the same parameters but different initial populations. All experiments have a population size of 200 individuals and elitism degree of 1. Each generation-population is segregated into species by NEAT, with selection and reproduction happening within these groups. *Sigma scaled selection* [51] is used to select pairs for reproduction.

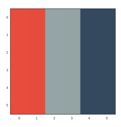
4.2 Morphogenesis and Replication of 2D Patterns

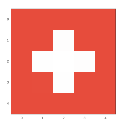
The first class of problems CA-NEAT was tested on was *morphology problems*, in the form of the two tasks of morphogenesis and replication. The patterns investigated in these experiments are shown in Figure 4.1

These are the same problems and patterns as studied in [46], which used an instruction-


(a) 5x5 ”Mosaic”


(b) 6x6 ”Border”


(c) 6x6 ”Tricolor”


(d) 5x5 ”Swiss”

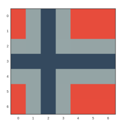
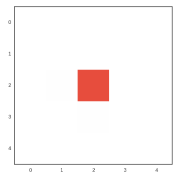
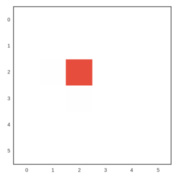

(e) 7x7 ”Nordic”

Figure 4.1: Patterns being investigated for morphogenesis and replication.


(a) 5x5


(b) 6x6

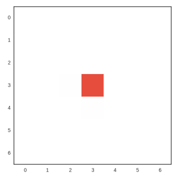

(c) 7x7

Figure 4.2: Seed patterns for morphogenesis. For the 6x6 patterns there is no central cell, so the seed is not symmetric.

based encoding and also tested table-based encoding for comparison. This allowed the results of [46] to act as a benchmark for testing the CA-NEAT framework during development, and to make comparisons between the results for analysis.

4.2.1 Morphogenesis Problems

In CA terms, morphogenesis is the construction of a (more) complex pattern from a simple ”seed” pattern. A biological analogy and inspiration is *embryonic development*, with the seed pattern also sometimes called a *zygote*. Figure 4.2 shows the seed patterns used in these experiments.

The fitness evaluation function used in the morphogenesis experiments consists of the following steps:

1. Develop seed pattern for 30 iterations
2. For each stage
   (a) Compare cell by cell with target pattern
   (b) Calculate ratio of correct out of total cells
3. Pick the highest of the values from step 2
4. Use function (4.1) with value from step 3 as x

$$f(x) = x * \frac{e^{5*x}}{e^5} \tag{4.1}$$

In cases such as the ”Mosaic” pattern (Figure 4.1a), a completely dead CA (all cells in the quiescent state) would have a ”correct cell” ratio of 0.52. Function (4.1) is used to reduce the score for such cases, while ensuring that $f(1.0) = 1.0$.

Because every iteration of the CA is counted equally and separately, the fitness evaluation does not care if the CA becomes stable, enters a cycle, or neither within the 30 allotted iterations. If the target pattern occurs at any point, that is enough to get a perfect score.

4.2.2 Replication Problems

In a CA replication problem, the initial state has some complex pattern present. The goal is to produce multiple copies of this initial patterns within the allotted time. Biological analogies of this is cell division (mitosis) and asexual (clonal) reproduction. For the replication problem the seed pattern is thus one copy of the target pattern in a larger grid.

The fitness evaluation for a replication phenotype is as follows:

1. Develop seed pattern for 30 iterations
2. For each stage
   (a) For each region of target pattern size
      i. Compare cell by cell with target pattern
      ii. Calculate ratio of correct out of total cells
   (b) Pick the highest 3 values from (a)
   (c) Multiply any value less than 1.0 by a penalty factor of 0.9
   (d) Calculate mean of three values
3. Pick the highest value from stage (2)

In this case the number of replicas sought is three. There is no further contribution to the score if there are more than three perfect replicas. But it follows logically that if one instance can be duplicated once, then each of the duplicates should be able to duplicate again, leading to exponential growth if time and space is not limited. Once again a penalty is applied, this time to penalize the contribution from any imperfect replica pattern, hopefully driving the selection pressure towards perfect replication.

Compared to the evaluation of morphogenesis of the same pattern, the replication evaluation is much more computationally expensive. Therefore it will always take longer to collect results for a replication problem than the same-pattern morphogenesis problem.

4.2.3 Cellular Model

For both morphology problem categories a 2D CA model is used. For the morphogenesis problem the grid is of fixed size with toroidal border conditions. For the replication problem the grid is automatically expanding to accommodate growth in any direction. In theory this means an infinite grid, but since the CA may only iterate 30 times, there is a practical limit to how large it may grow. For both problem types the Von Neumann neighborhood (Figure 2.3a) is used.

Table 4.1: Summary of results. The metrics shown are the success rate and the mean number of generations until a solution is found, with standard deviation also shown. In the case of 100% success rate, the number of generations column shows how many generations it took until the final solution was found. In the case of less than 100% success rate, the column shows how many generations were run until the experiment was stopped.

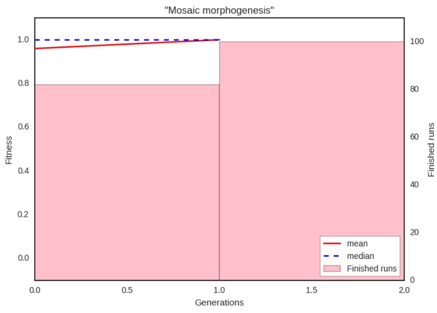
Problem	Success rate %	Mean gens.	σ gens.	Gens. until stop
Mosaic morphogenesis	100	1.2	0.4	2
Border morphogenesis	1	270	0	509
Tricolor morphogenesis	100	56.5	228.8	2189
Swiss morphogenesis	76	147.7	158.9	600
Mosaic replication	100	4.2	10.6	99
Swiss replication	100	7.7	5	20
Tricolor replication	55	55.8	52.6	200
Nordic replication	0	-	-	200

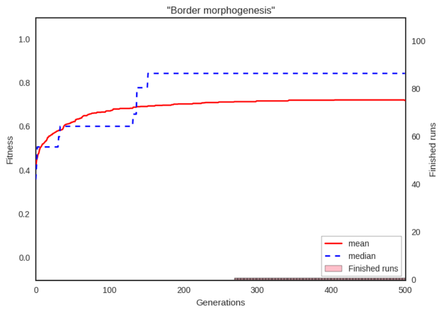
4.2.4 Results

The results of the experiments were varied, with some problems being easily solved with CA-NEAT, some being slowly solved, and some not being consistently solved at all. Table 4.1 summarizes the results in terms of success rate and generations of evolution. In addition to the varied success rate, there was also a large variation in how many generations of evolution was required to find solutions. In many cases there was at least one optimal solution among the 20000 individuals generated as part of initial populations. This means there exists a simple solution consisting of only the input and output layers with connections. The most extreme of these cases is the ”Mosaic” morphogenesis where 80 runs complete in the initial generation and the last 20 in the second generation. This result is understandable, since the pattern has so much symmetry and repetitiveness. For more complex patterns, more generations of evolution is required in order to bring the success rate nearer 100%.

It is somewhat surprising which problems are easily solved and which ones are difficult. The ”Swiss” replication is much easier than the ”Swiss” morphogenesis, but the ”Tricolor” morphogenesis is easier than the replication of the same pattern. The fact that the ”Border” morphogenesis is much more difficult than the ”Tricolor” morphogenesis is not intuitive, since the ”Border” pattern has both fewer colors and one more symmetry. Perhaps it is the symmetry that is the ”trap” which leads to a local maxima, and the ”Tricolor” experiment avoids this, since symmetry in solutions will not give great scores in that case. Since the other morphogenesis experiments succeed, and one ”Border” solution is found, there is little reason to suspect that there is any technical error causing poor results. So the reason must be that the combination of algorithms, problem and parameters cause the problem to be very difficult.

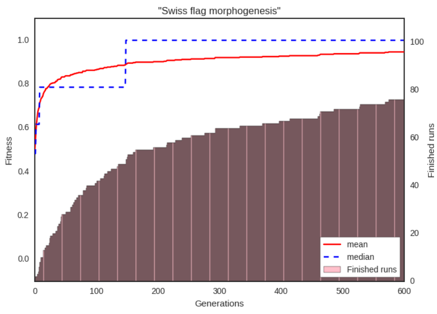
CA-NEAT does quite well for three out of four replication problems, but fails completely at the ”Nordic” replication. This problem can be expected to be quite difficult, since the pattern is rather complex. But the instruction-based encoding in [46] did well at the task, so it is certainly possible to solve the problem with this particular cellular model. The development of the mean and median lines shown in Figure 4.4d indicate that the

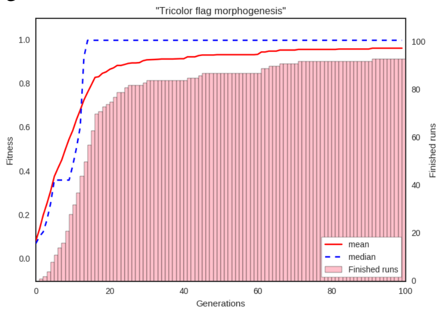




(a) Mosaic pattern morphogenesis, all generations.

(b) Border pattern morphogenesis, 500 first generations.

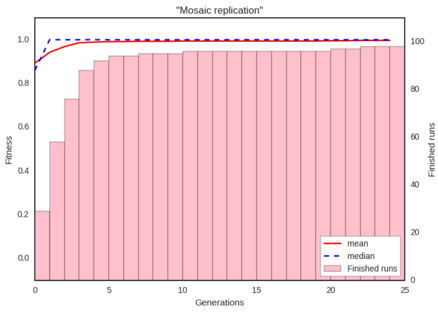




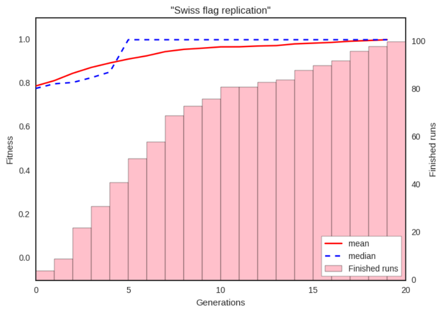
(c) Swiss flag pattern morphogenesis, 600 first generations.

(d) Tricolor flag pattern morphogenesis, 100 first generations.

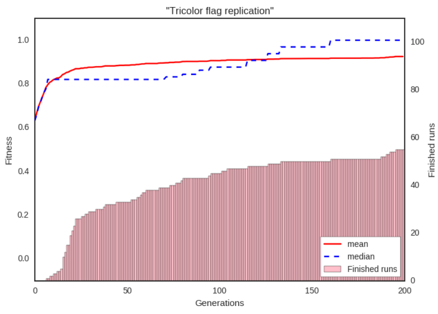
Figure 4.3: Success rate (cumulative histogram) of the morphogenesis experiments. Also shows the mean and median of the max fitness in each run.



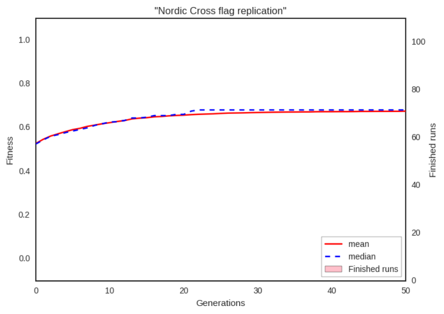
(a) Mosaic pattern replication, 25 first generations.



(b) Swiss flag pattern replication, all generations.



(c) Tricolor flag pattern replication, 200 first generations.



(d) Nordic cross pattern replication, 50 first generations.

Figure 4.4: Success rate (cumulative histogram) of the replication experiments. Also shows the mean and median of the max fitness in each run.

Table 4.2: Success rate at morphology problems for table-based, instruction-based and CA-NEAT transition functions found by GA.

Problem	Table-based	Instruction-based	CA-NEAT
Mosaic morphogenesis	55%	98%	100%
Swiss morphogenesis	23%	100%	76%
Border morphogenesis	69%	98%	1%
Tricolor morphogenesis	19%	46%	100%
Mosaic replication	85%	100%	100%
Swiss replication	1%	100%	100%
Tricolor replication	8%	100%	45%
Nordic replication	0%	100%	0%

evolutionary searches find local maxima from which they can't escape.

These results were compared with the results of [46]. Table 4.2 shows the success rates of the different encodings at the different tasks. It was found that CA-NEAT was able to significantly outperform instruction- and table-based encodings at some problems, while also performing much worse at other problems. At the morphogenesis tasks, CA-NEAT outperformed the table-based evolution at 3/4 tasks and the instruction-based evolution at 2/4 tasks. For the replication tasks, CA-NEAT outperformed the table-based evolution at 3/4 tasks and tied at 0% for the last task. Instruction-based evolution had a 100% success rate at all replication tasks. CA-NEAT equaled this rate at two of the tasks, had some success at one task, and failed completely at the last task.

In addition to the number of completed runs, another interesting result is the sizes of the genotypes of optimal solutions. NEAT genotypes consists of a fixed number of input and output nodes $N + K$, 0 or more hidden nodes, and some number of connections between nodes. When evaluating NEAT genotypes it is interesting to consider these numbers both separately and combined.

Table 4.3 shows some measures of the sizes of optimal genotypes for each experiment. Since some runs finish with a generation where there is more than one optimal solution present, the number of optimal genotypes may be higher than the number of finished runs.

Figures 4.5, 4.6 and 4.7 shows visualizations of some results found by evolution. A larger selection of visualizations is available in Appendix C.

4.3 2D Morphogenesis with Coordinate Input

Adding environmental information to the CA may make it possible to solve a task that is difficult with only the neighborhood information. To test how CA-NEAT performs with this modification, we use the morphogenesis task as a test case.

For this experiment, the cellular model is extended to include coordinate information. The "Border" and "Nordic" patterns (Figure 4.1) are targeted in separate experiments. The "Border" morphogenesis in the previous experiment did not work very well, so it makes for a good comparison. The "Nordic" pattern is also difficult to generate with the previous cellular model, because of the shifted symmetry, so it was not attempted in the previous

Table 4.3: Sizes of genomes of optimal solutions. Genomes consist of node genes and connection genes, which may be counted considered separately or combined. Each genome has a fixed number $N + K$ input and output nodes, plus some number (possibly 0) of hidden nodes. When considering genome size, only the hidden nodes are counted.

		Min	Max	Mean	Median	σ
Mosaic morphogenesis (241 results)	Hidden nodes	0	1	0.1	0	0.3
	Connections	4	11	7.1	7	1.5
	Combined	4	12	7.2	7	1.6
Border morphogenesis (1 result)	Hidden nodes	7	7	7	7	0
	Connections	16	16	16	16	0
	Combined	23	23	23	23	0
Tricolor morphogenesis (119 results)	Hidden nodes	0	14	2	2	2.1
	Connections	6	32	15.1	15	4.3
	Combined	6	46	17.1	17	6
Swiss morphogenesis (61 results)	Hidden nodes	0	13	2.9	2	3.1
	Connections	5	22	10.2	9.5	3.7
	Combined	6	32	13.1	11	6.5
Mosaic replication (136 results)	Hidden nodes	0	10	0.6	0	1.2
	Connections	4	21	7.6	7	2
	Combined	4	31	8.2	8	3
Swiss replication (114 results)	Hidden nodes	0	3	0.5	0	0.7
	Connections	7	14	9.5	9	1.5
	Combined	7	16	10	10	2
Tricolor replication (47 results)	Hidden nodes	0	20	4.8	4	4.2
	Connections	8	41	14.7	14	5.7
	Combined	8	61	19.5	17	9.5

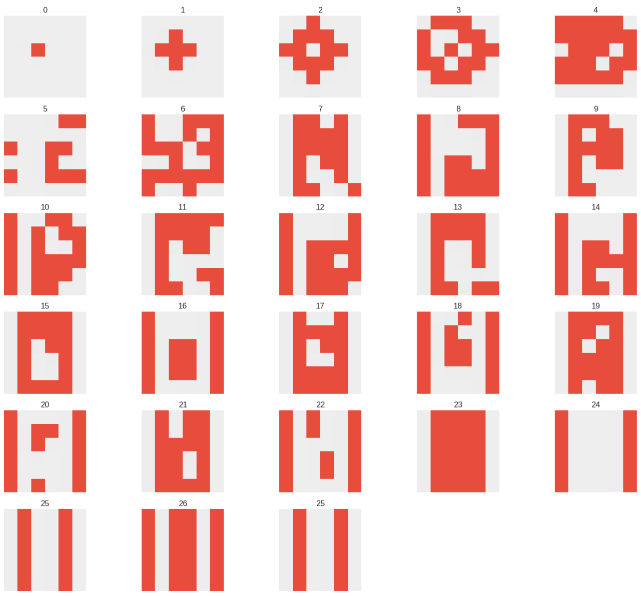


Figure 4.5: The only solution that was found for the ”Border” morphogenesis. After finding the target state in iteration 23, the CA goes in to a two-step cycle which does not include the target state.

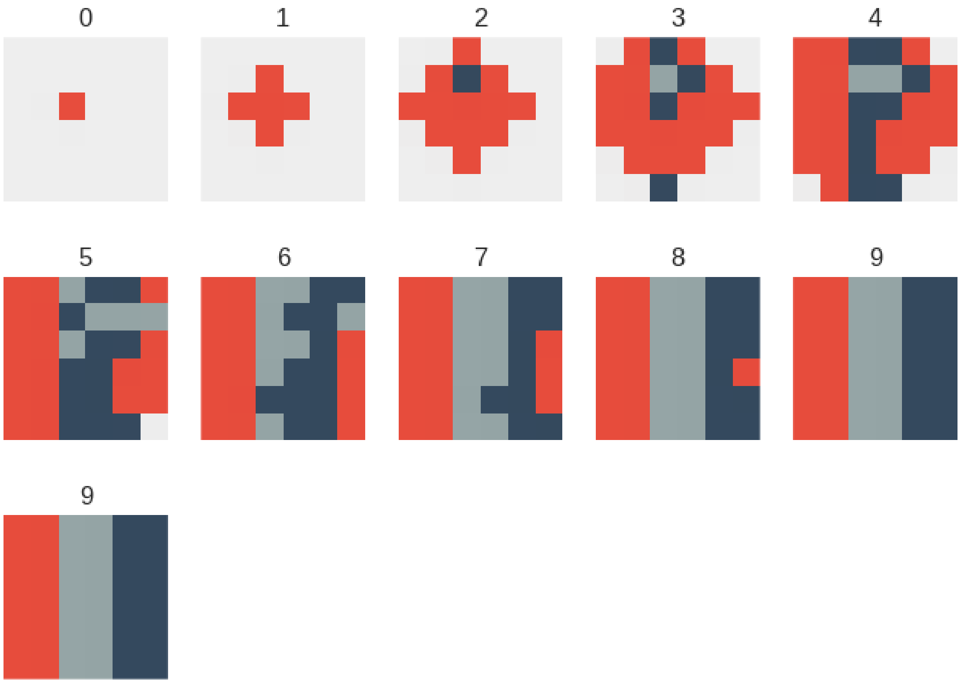


Figure 4.6: A solution to the ”Tricolor” morphogenesis that finds a point attractor equal to the target pattern. Most solutions seen did not stabilize like this, but instead found a variety of cycles.

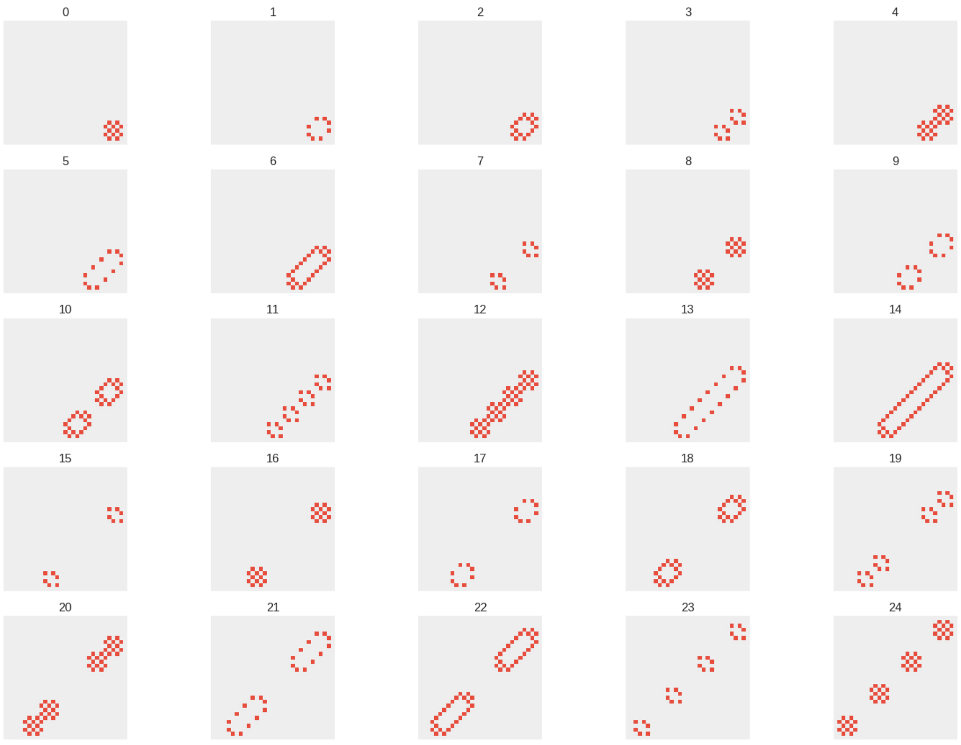


Figure 4.7: A solution to the ”Mosaic” replication that shows multiple stages of replication. First the original replicates into two copies. Then each copy tries to replicate, but they interfere with each other and instead return to one copy each, but at a greater distance. Then they each succeed in replicating, producing four copies total.

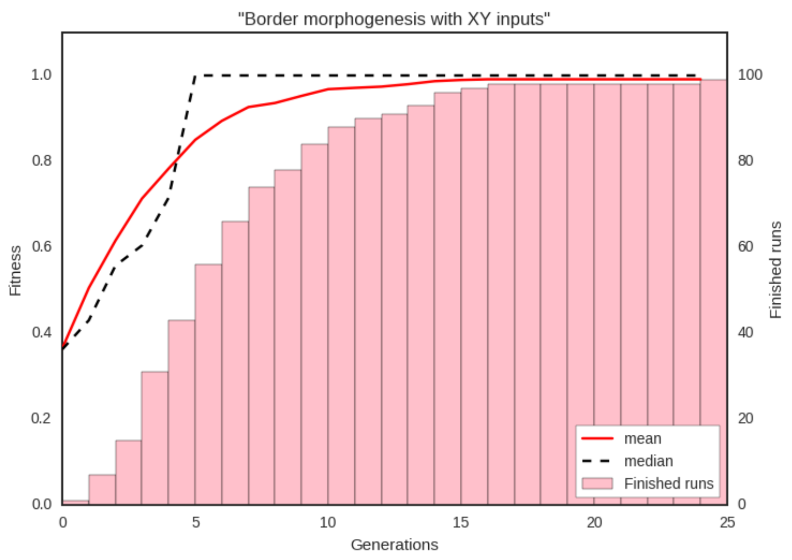


Figure 4.8: Success rate at the ”Border” morphogenesis with coordinate input. 99/100 trials finished by 25 generations, but the last one was unable to finish before it was stopped after 1000 generations.

experiment, but it is attempted now.

The extension to add coordinate information to CA-NEAT is described in Section 3.1.3. The quiescent rule is still in place: a cell may not change state if all the neighbors are quiescent, even if the information from the coordinates is available to inform a decision. The coordinate information is meant to be used in conjunction with the neighborhood information, not to replace it. If this was not the case, evolution could come up with a CPPN where all the neighborhood inputs were disconnected from the output, and a static pattern was produced in one time step, like in Figure 2.5. That is not a result that we are interested in.

Like in the previous experiments, 100 independent trials were performed for each target pattern. Only the input to the transition function is changed from before, so the fitness evaluation function is still the same as described in Section 4.2.2.

4.3.1 Results

Figure 4.8 shows the success rate of the ”Border” morphogenesis evolution. It was able to find a solution for 99/100 trials in 25 generations. This shows how a task that was very difficult (but not impossible) with only neighborhood input becomes easy with the additional spatial information available. There was even one solution present in the initial population of one of the trials, meaning a simple CPPN structure with no hidden nodes can solve the task.

Figure 4.9 shows the success rate of the ”Nordic” morphogenesis evolution. By 100 generations, only 3 of the runs have succeeded. Letting the evolution go on until 1000

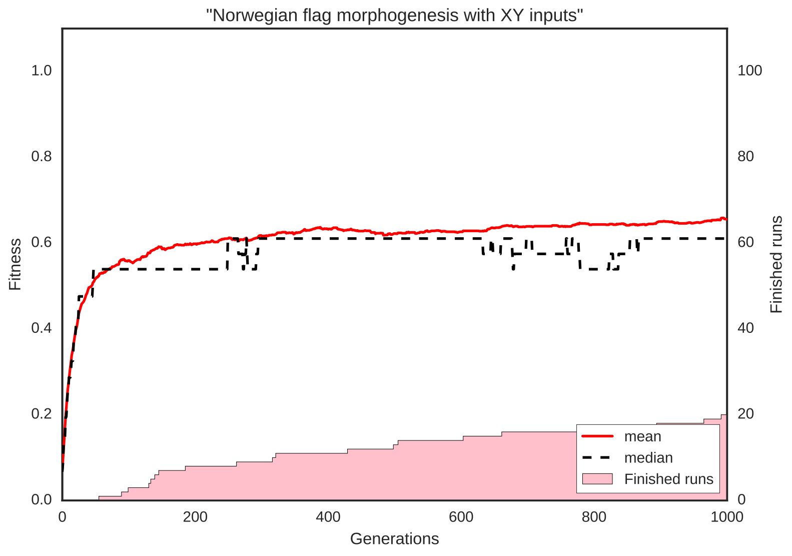


Figure 4.9: Success rate at the ”Nordic” morphogenesis with coordinate input.

generations results in a total of 20 successful runs. While not as successful as the ”Border” morphogenesis, this is still a very positive result, since the pattern is so much more complex.

Figure 4.10 shows one of the found behaviors for the ”Border” morphogenesis that finds a point attractor equal to the target pattern. Point attractors were quite common in the optimal behaviors found by CA-NEAT with coordinate information, unlike in the experiments without coordinate information, where they were rare. Figures 4.11 and 4.12 show visualizations of one of the networks found for the ”Nordic” morphogenesis. The first shows the minimal structure of only the nodes and connections that affect the output. The second shows the full network, including all the nodes that are disconnected from the output layer and all the disabled connections.

4.4 Majority and Synchronization Problems

Another class of CA problems are computational problems that have to do with the transmitting and coordination of information through the grid. This category includes the *majority problem* and the *synchronization problem*, both problems for 1D binary CA. This problem has been studied extensively with table-based transition functions found by genetic algorithm [52, 53, 26]. The design of the experiments in this section takes inspiration from these sources, but also does some things differently.

The *majority problem* (also called the *density classification task* in literature [52, 53]) is about figuring out which of the states is more common in the initial configuration (IC). The CA must have some arbitrary IC where one ”color” (black/white) is more common

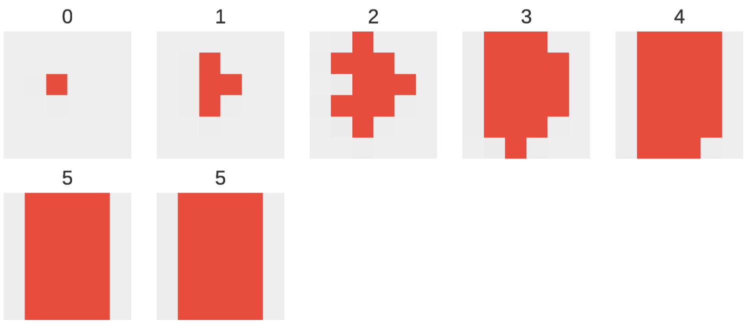


Figure 4.10: One of the found solutions to the ”Border” morphogenesis with coordinate input. This is the shortest solution that was found, needing just five timesteps, and going into a point attractor.

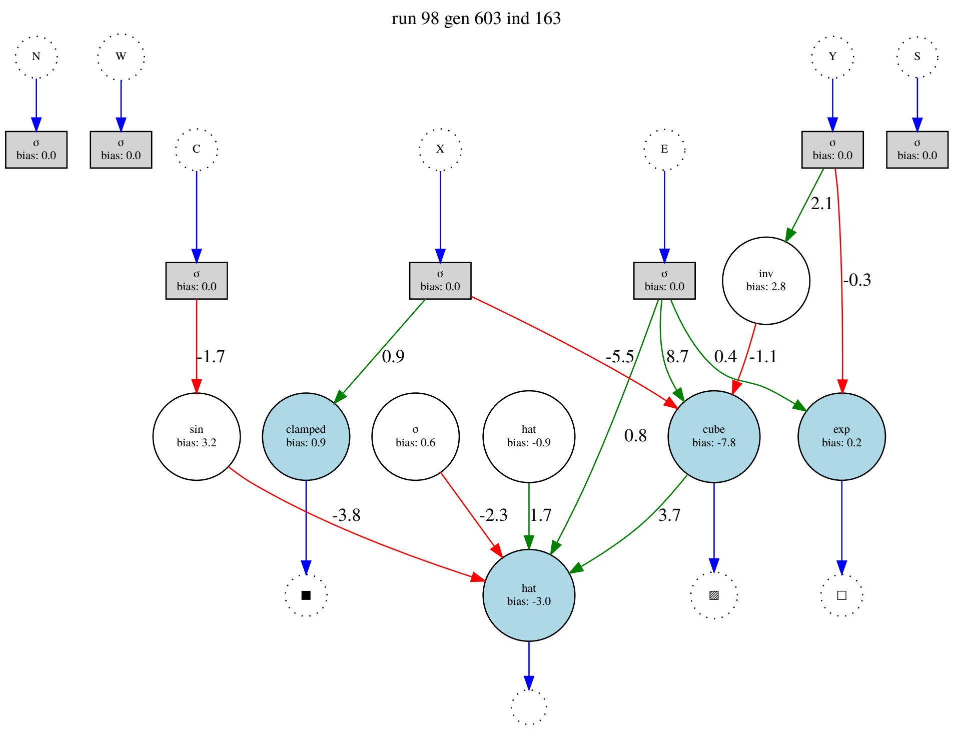


Figure 4.11: One of the found solutions to ”Nordic” morphogenesis with coordinate information. This one occurred after 603 generations of evolution. This particular solution uses only four out of the seven inputs available: both the coordinate values, plus the center and east input from the neighborhood. Vestigial hidden nodes and disabled connections have been pruned away from the visualization.

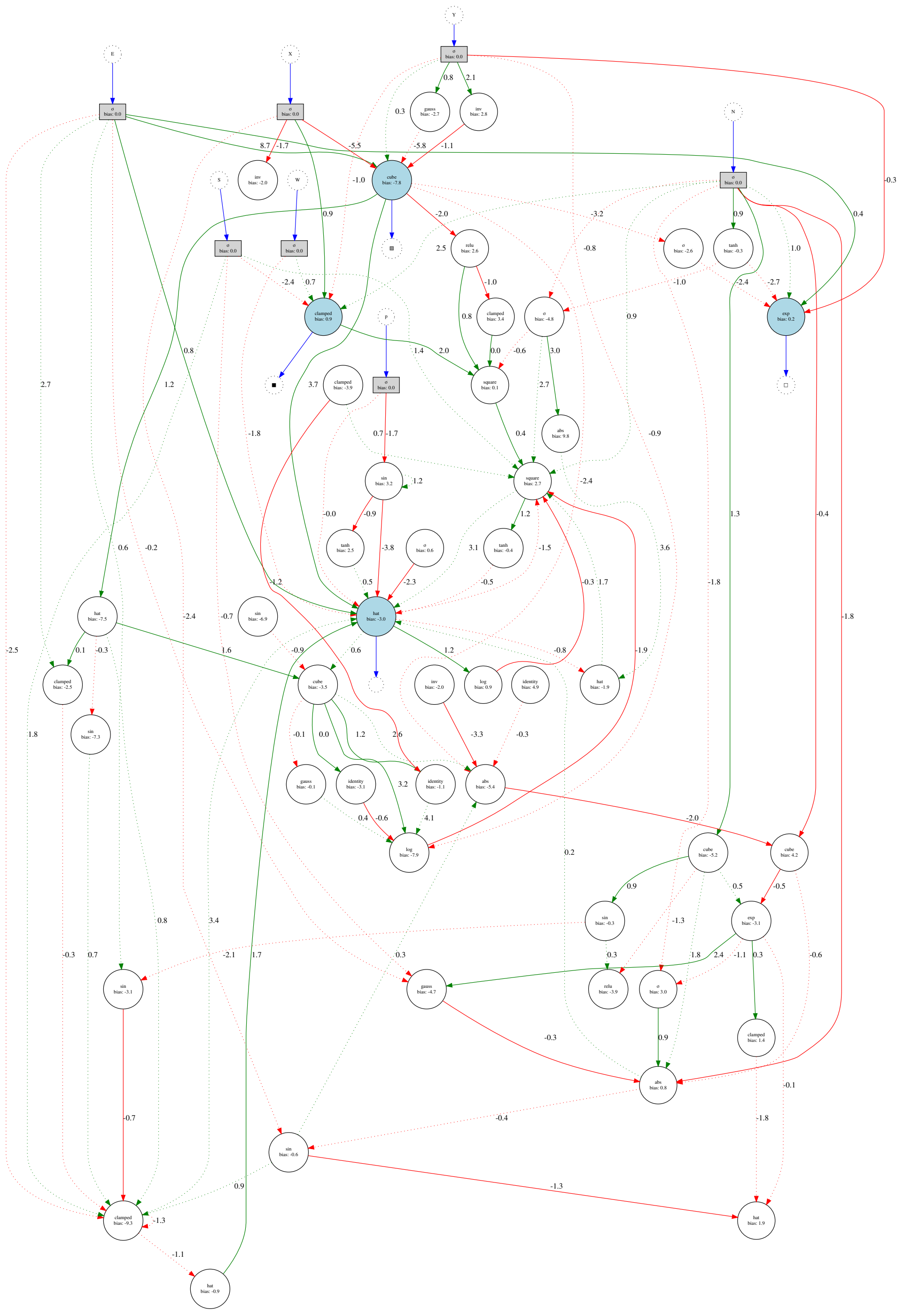


Figure 4.12: The same network as in Figure 4.12 without pruning reveals an enormous number of vestigial structures.

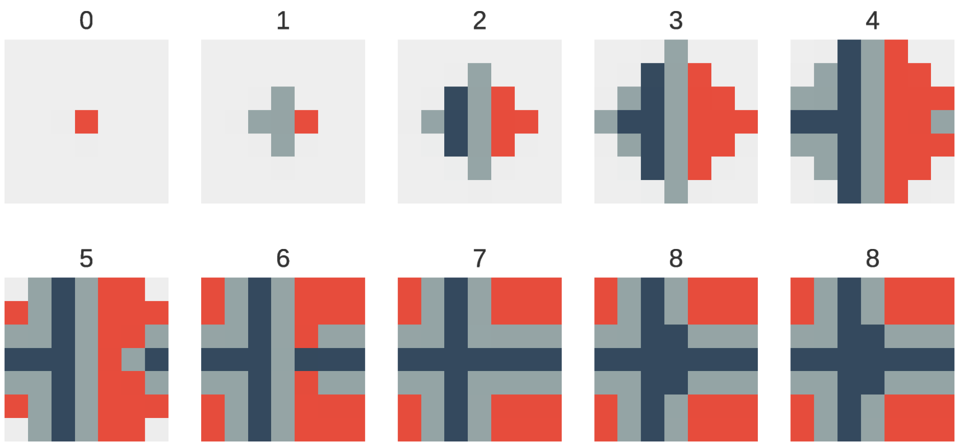


Figure 4.13: The CA behavior encoded by the network shown in Figures 4.11 and 4.12. This one visits the target pattern at time step 7, then ends up in a different configuration point attractor.

than the other (i.e. not a 1:1 ratio). During the CA development iterations it must then figure out which color is dominant and end up in a point attractor state where all the cells are of this color. Depending on the IC this can be easy, or very difficult, so finding a general solution that can figure out any IC is difficult. Even if the majority of the configuration is black, there might be a sub-region that is majority white. The lack of global overview means that the CA needs to send information around the grid and ”negotiate” a consensus about the color.

The *synchronization problem* is a similar problem, but in this case the CA should find its way to a two-step cyclic attractor where all cells share the same state in one timestep, then all share the other state in the next timestep. It is therefore similar to the majority problem in the way information must be transmitted across the CA in order to coordinate the cells, but instead of having to ”count” cells and landing in a specific point attractor, it can find a cyclic attractor without concern for which of the two states it visits first.

The difficulty of both problem is also dependent on the size of the neighborhood used in the cellular model. For both experiments in this section, the neighborhood is of size $N = 7$ (Figure 2.3. The grid wraps around at the ends (toroidal border conditions).

4.4.1 Fitness Evaluation

For the majority problem, fitness evaluation for a genotype is as follows:

1. Create k ICs of size n, with some ratio r of black/white cells (in a random order). There should be both black-dominant and white-dominant ICs.

2. For each IC:

   (a) Record which color is dominant.

   (b) Develop the CA for $i = 2 * n$ iterations, or until a cycle is detected.

   (c) Calculate the ratio of cells in the final configuration that is of the IC-dominant color.

3. Calculate the mean of the ratios from (2). This value is the fitness for the individual.

For the synchronization problem, fitness evaluation for a genotype is as follows:

1. Create k ICs of size n, with some ratio r of black/white cells (in a random order). There should be both black-dominant and white-dominant ICs.
2. For each IC:
   (a) Develop the CA for $i = 2 * n$ iterations, or until a cycle is detected.
   (b) Check whether the last two iterations match the condition of being all black and all white (the order does not matter).
3. Calculate the ratio of ICs from (2) that lead to the desired behavior. This value is the fitness for the individual.

The notable difference in the fitness evaluation methods is that the majority fitness evaluation gives partial credit for all attempts at solution, whereas the synchronization evaluation only credits attempts that lead to the exact desired behavior.

4.4.2 Evolution

For the majority problem $k = 10$ ICs of size $n = 49$ are created, all with a ratio $r = 1/3$ of black and white cells. Five are white-dominated and five are black-dominated.

For the synchronization problem, $k = 100$ ICs of size $n = 49$ are created, with individual ratios r selected from a uniform distribution. The uniform distribution creates a variation of ICs, some very white-dominant, some very black-dominant, and some more mixed. The majority problem can have a smaller k with challenging r because of the partial credit, while the synchronization problem needs a larger k with both easy and challenging r values to get a nuanced fitness score.

For both experiments, 100 independent trials of different initial populations are performed. The same set of ICs is used in all the trials.

4.4.3 Comprehensive Testing

The evolution populations only get to train on a small set of ICs. The performance at these give an indication, but not an absolute measure of how the individual would perform at *any* arbitrary IC. For this reason a more comprehensive test should be performed on the best individuals produced by evolution. This test can show both how the individuals perform on ICs they have not seen before, and also how they perform on ICs of greater size n than they trained on.

The comprehensive testing method is the same for both the majority and synchronization problems. Three different sets of $k = 1000$ ICs are generated from a uniform distribution. The sets have different IC sizes, $k \in \{49, 99, 149\}$. A selection of the best performing individuals from the evolution are tested on all the 3000 ICs using the same fitness function as during evolution.

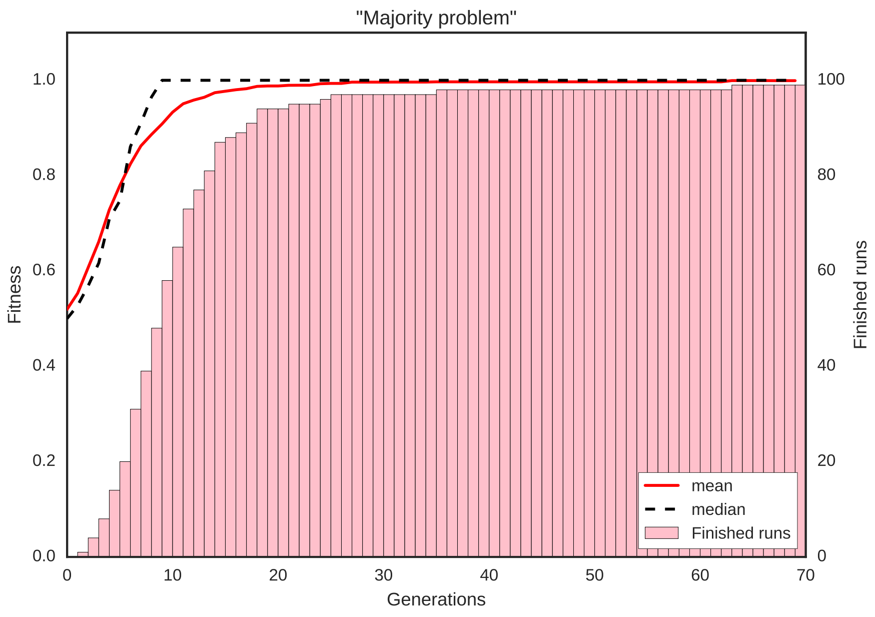


Figure 4.14: Success rate development when training on the $k = 10$ set of initial configurations. Within the first 10 generations over 50% of the runs had already found an individual that could solve all the training configurations. Within the first 79 generations, 99/100 runs had found a solution. The last run finished at 371 generations. This statistical outlier has been omitted from the figure to make it clearer.

4.4.4 Results

Majority Problem

Figure 4.14 shows the success rate of the majority problem evolution. Every run was eventually able to find at least one individual that could solve all the training ICs. Across all 100 runs there were in total 129 individuals with fitness 1.0. Categorizing these by CA enumerated table behavior and removing duplicates resulted in 111 individuals with distinct behavior.

These 111 candidates were selected for comprehensive testing. Figure 4.15 shows the performance of the candidates at the new test ICs. None were able to achieve a perfect score on the new tests, but some came quite close at over 90% score. Even though all the candidates had $f = 1.0$ at the training ICs, there was a large variation of performance at the test ICs, with the lowest mean score at less than 65%.

Another pattern that the figure reveals is that the variance between the scores on the different n is wider for the worst-performing candidates, and narrower for the best performing candidates. The worst-performing candidate had an about 0.2 difference between the best and worst sets, while the best candidate had only about 0.05 difference. The worst-performing candidates were also clearly better at the $n = 49$ ICs than the others,

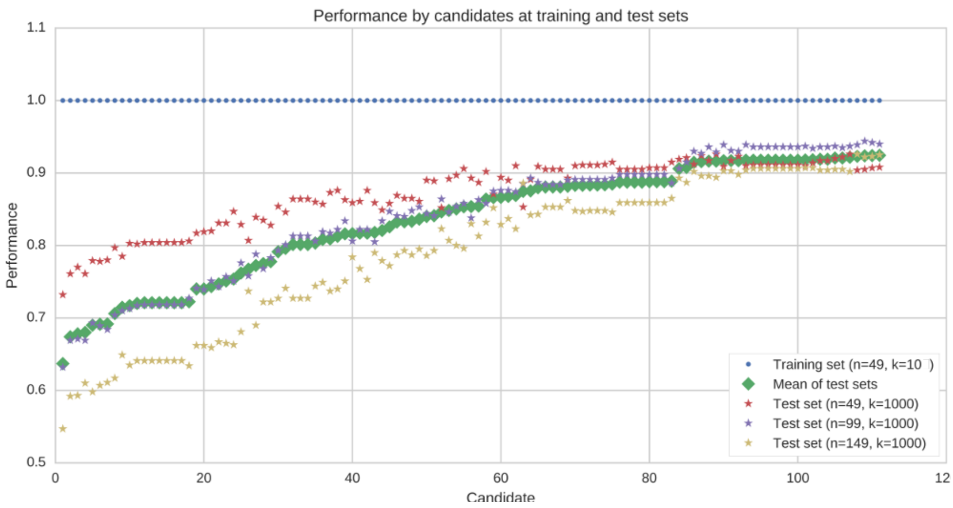


Figure 4.15: Performance at more comprehensive testing of the 111 distinct individuals that scored $f = 1.0$ on the training configurations. The individuals have been ordered by their mean test performance.

Rate	#
0.94	1
0.95	2
0.96	2
0.97	65
0.98	27
0.99	3
1.0	0

Table 4.4: Distribution of max fitness achieved in 100 independent trials of synchronization evolution

but the best-performing ones were, while all close, actually performing best at $n = 99$ and worst at $n = 49$.

Figures 4.16 and 4.17 show the different behaviors of two found individuals solving the same IC.

Synchronization Problem

In 100 generations of evolution none of the trials were able to find any individual that could solve all $k = 100$ training ICs. Some did come very close though. Table 4.4 shows the distribution of the best achieved fitness in the 100 independent trials. While none of the trials achieved 100% coverage of the configurations, three managed to solve all the ICs except one. 95 out of 100 trials achieved a score of 0.97 or greater. Because of the uniform distribution the ICs were sampled from, it is to be expected that a lot of the configurations

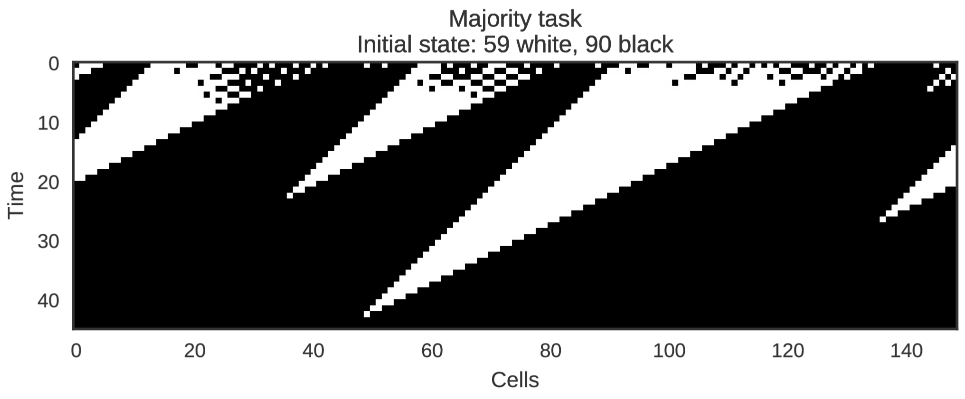


Figure 4.16: A successful solution to the majority problem for an IC of size $n = 149$ and ratio $59 : 90$.

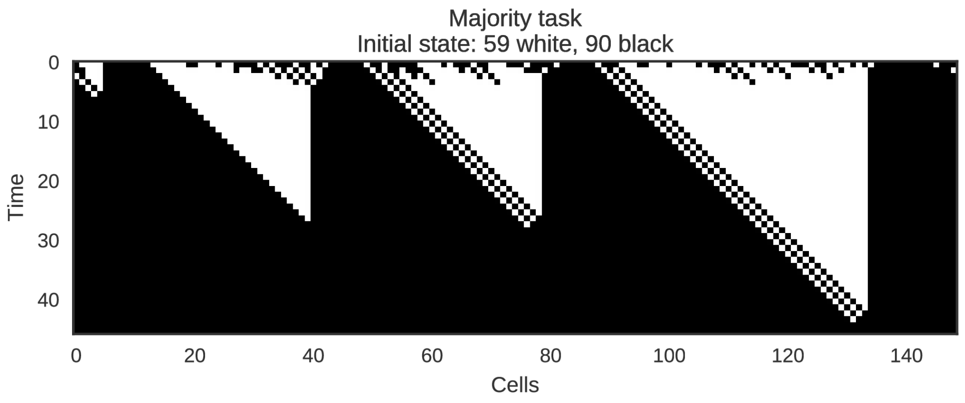


Figure 4.17: Another solution to the majority problem for an IC of size $n = 149$ and ratio $59 : 90$.

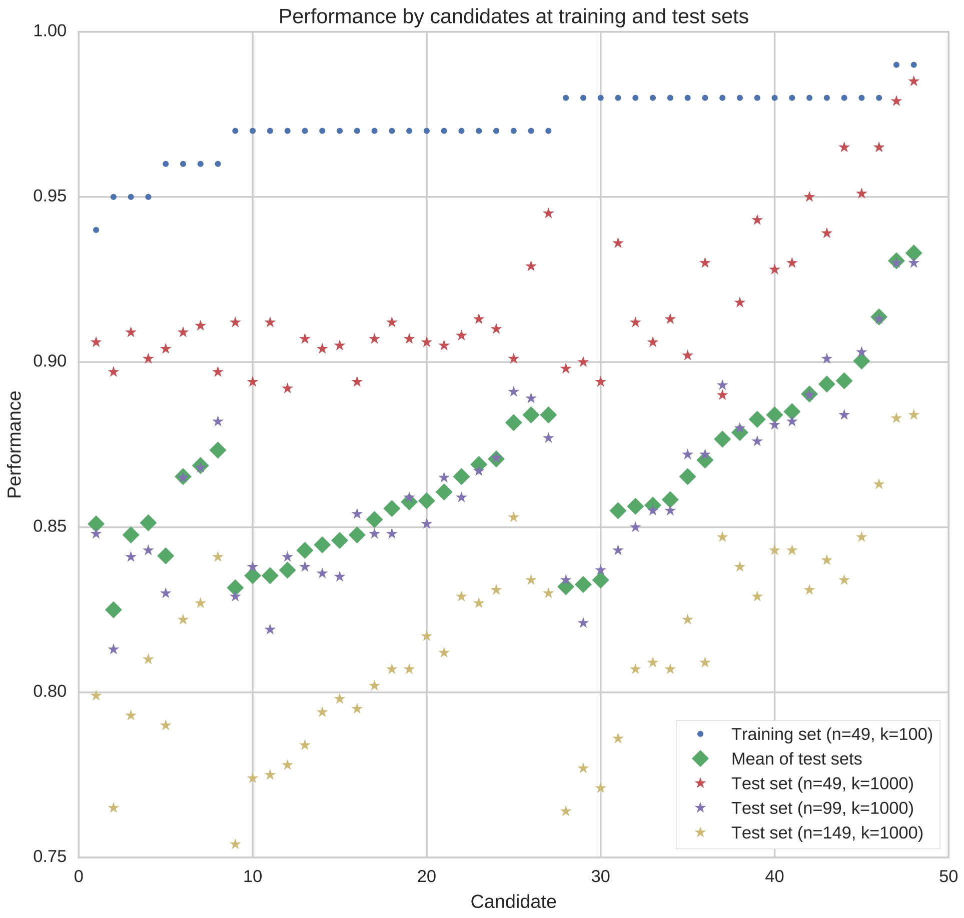


Figure 4.18: Performance of the 48 candidate solutions at the training and test sets. The candidates have been ordered by training performance first, and average test performance second.

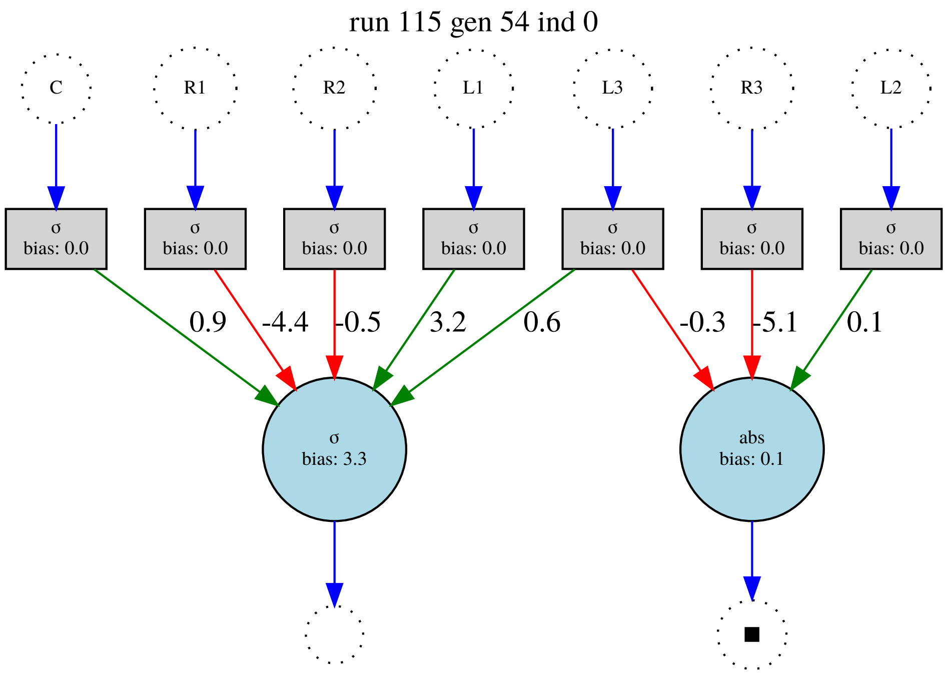


Figure 4.19: One of the found networks for the synchronization task.

are similar in that a behavior that can solve one of them can solve most of them. It is therefore not surprising to be able to get scores greater than 0.9. The challenge is to be able to solve *all* the configurations, both the simple ones and the tricky ones. There was no IC that was never solved in any of the trials. Most of the ICs were simple and were solved in every trials. The most difficult IC was solved in 22 of the trials.

Since no individuals achieved perfect scores in the evolution phase, the sample of the population selected for exhaustive testing was selected by picking the top 10 performing individuals from each run. These 1000 individuals were then categorized by CA behavior and the duplicates removed, leaving 48 distinct behaviors.

Figure 4.18 shows the performance of the 48 candidates at the training set and each of the test sets. There is a correlation between the average score of the test sets and n, with the $n = 49$ test performing closest to the $n = 49$ training performance. It is also clear that the two candidates that performed the best ($f = 0.99$) at the training set, also performed the best at each of the test sets.

Figures 4.19 and 4.20 show an evolved network and its behavior when performing the synchronization task on a non-trivial IC. This simple network with no hidden nodes (when pruned of vestigial structures) is actually one of the networks that tied for the best performance, with a fitness score of 0.99. This shows that very simple network structures can be tuned to perform very well at this kind of task.

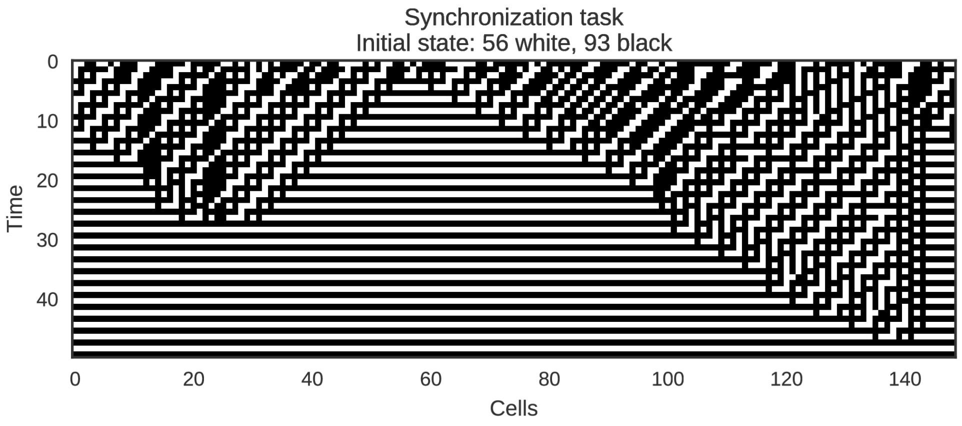


Figure 4.20: The network from Figure 4.19 solving the synchronization task. The IC used here is of size $n = 149$ and has an initial ratio of cells $56 : 93$.

4.5 Investigation of Genome Properties

The process of running a task-solving experiments such as in the preceding sections is quite opaque. The experiment is configured by human, but after it is started the system runs itself with no human input, and the human observer can only wait and hope a useful result emerges at the end.

In order to gain a better understanding of the process, an experiment was designed with the goal of investigating various properties of the population and their development over time, rather than to solve a specific task. This involves storing every single individual from the whole run, then analyzing them quantitatively afterwards.

CA-NEAT can be studied from multiple ”angles”: as a GA experiment there are properties such as fitness that can be studied, as a CA experiment there is for example the λ parameterization, and as a graph-based encoding properties such as the number of nodes can be investigated. Inspecting these, individually and trying to see correlation, should help with understand the system better, and selecting better configurations for future experiments.

There are also multiple different mechanisms of the algorithm that can enabled or disabled by configuration. Running multiple GA runs with different combinations of mechanisms enabled should give some indication about the effects of the mechanism.

4.5.1 Experiment Design

The ”Swiss” morphogenesis task is used as the basis for this experiment. The previous experiment with this task showed it to be consistently solvable by CA-NEAT, but not trivially simple. The fitness evaluation for morphogenesis problems is also very fast compared to other problems, making it possible to run tests with large populations in a reasonable amount of time. The CA for this problem has $K = 2$ cell states and the neighborhood shape ”Von Neumann” ($N = 5$) is used. An initial population of size $P = 1000$ was

Table 4.5: Mechanisms enabled in different scenarios

Scenario	Mutation	Crossover	Selection pressure	Speciation	Elitism
A	✓				
B	✓	✓			
C	✓	✓	✓		
D	✓	✓	✓	✓	
E	✓	✓	✓	✓	✓

created, with N input nodes, K output nodes and no initial hidden nodes. This same population was then used as the initial population for five independent ”scenarios” of $G = 100$ generations, with different mechanisms of NEAT in use. Table 4.5 shows which mechanisms were used in which scenario. From top to bottom, the table can be read as gradually enabling mechanisms, until arriving at the full algorithm, as used in other experiments.

Scenario A

Scenario A is different from the rest, since it is not a GA run, but more of a random walk in the search space using the mutation mechanism from the GA. The individuals present in the initial population are mutated once per generation, and the mutated versions are recorded as the next ”generation”. With no crossover and no selection pressure, the individuals will be completely independent from each other throughout the entire run.

Scenario B

Scenario B uses NEAT, however the selection mechanism is completely randomized, so there is no selection pressure towards the goal of accomplishing the morphogenesis task.

Scenario C

Scenario C has selection pressure appropriate for the morphogenesis task, but does not use the speciation mechanism of NEAT.

Scenario D

Scenario D uses NEAT with selection pressure like C does, but also uses the speciation mechanism.

Scenario E

Scenario E uses all mechanisms of NEAT, including a per-species elitism degree of $E = 1$.

Unlike the problem-solving experiments, this experiment does not involve multiple trials of the same configurations. The reasoning is that this would mean we would be studying averages of averages in the following figures, and some relationships between properties could be hidden by this. We therefore should not draw any strong conclusions from the observations, but use it to find hypotheses that can be more rigorously investigated later.

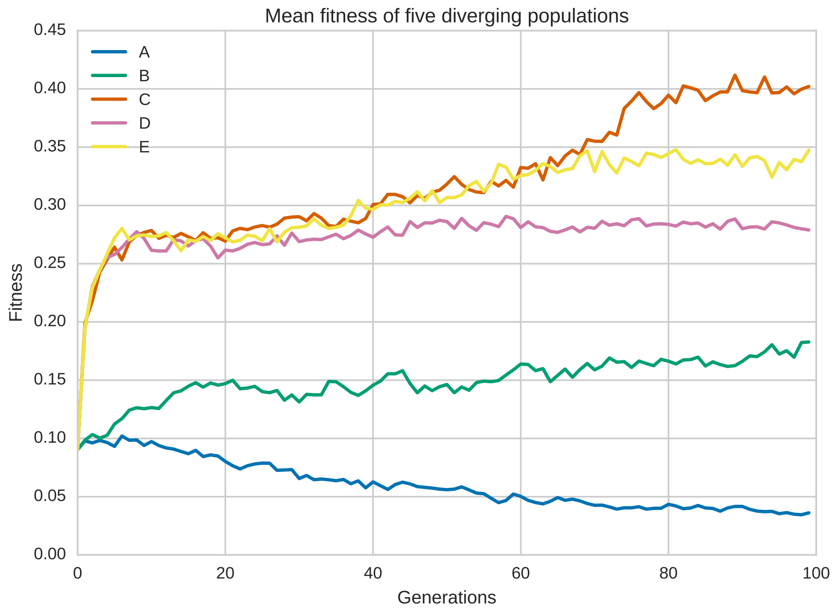


Figure 4.21: The development over time of the mean fitness of the populations

4.5.2 Fitness

When working with genetic algorithms, the most obvious property of a population to study is perhaps the average fitness over time. Figure 4.21 shows the mean fitness development across the whole population for each of the five scenarios. As one would expect, scenario A and B do not show any considerable improvement over time. A declines steadily, while B has a slight, but not very significant increase. Among C, D and E, which have selection pressure, there is a sharp increase in the first 10 generations. D then flattens out for the remaining time, while C and E improves some more, at a slower rate.

Averaging hides one important aspect of the fitness distribution, namely the maximum value, which indicates ”success” when it reaches 1.0. Looking at the maximum fitness development over time in Figure 4.22, both A and B appears to decline over time, but B less so than A. C and D act similar in that they hover around the upper half of the scale, sometimes finding a 1.0 score, but are unable to stay stable there. Whereas E, with the elitism mechanism is able to stay at 1.0 once it finds it. D finds a perfect solution quite early, but this seems likely to be a ”fluke”, since it loses it again and takes a long time to find another. This is consistent with the results of the ”Swiss” morphogenesis in Section 4.2, where in 100 independent trials, a few of them chanced upon an early solution.

The fact that the C average fitness overtakes both the D and E averages can hypothetically be explained by the difference the speciation mechanism makes. In scenario C, the search can only optimize for fitness, so when it finds a good candidate it will make a large number of children for that candidate. Scenarios D and E can optimize for diversity as

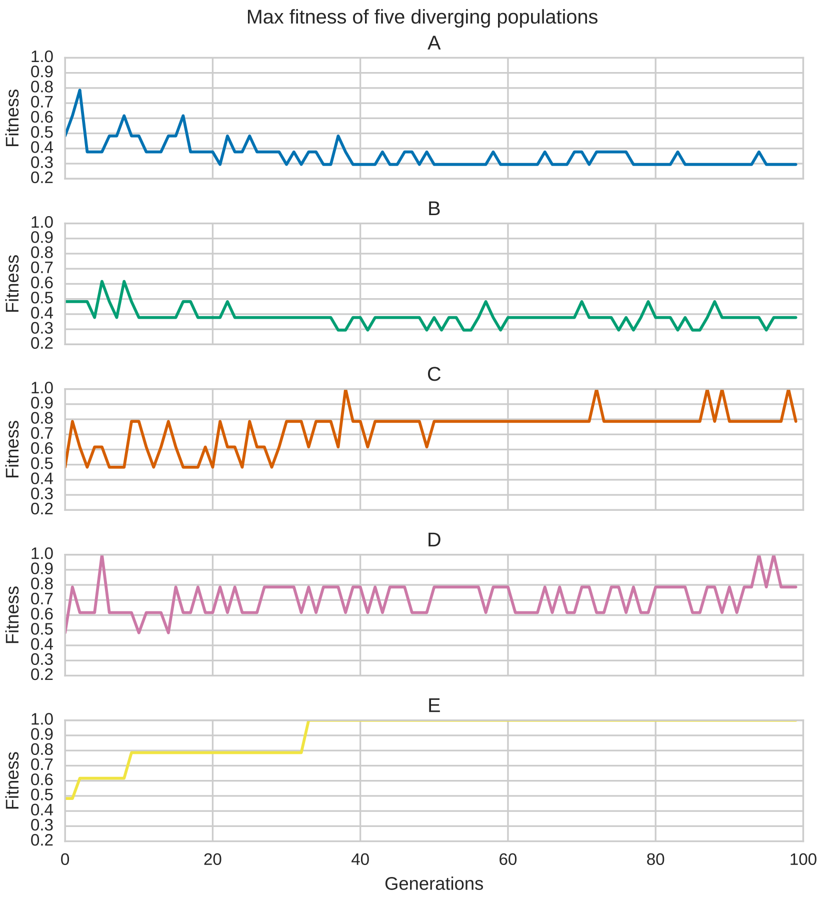


Figure 4.22: The development over time of the max fitness of the populations

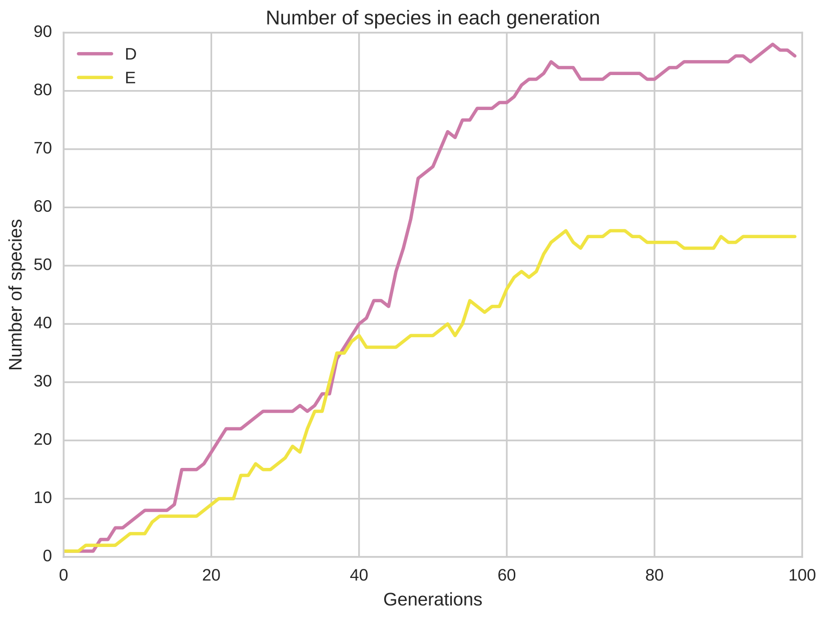


Figure 4.23: The number of species in scenarios D and E over time

well, searching in multiple directions. When a good candidate is found in a species, its children will dominate only in that species, while the other species continue their search unaffected. Another possible factor is the stagnation check which is also part of the speciation mechanism. This eliminates species that have not improved average fitness in 15 generations. If a species has ”peaked”, then it will be eliminated eventually, even if it’s average fitness is above the population average. In the task-solving experiments, the run is stopped when a ”perfect” solution is found, but in this experiment it is allowed to continue, so this effect can happen.

4.5.3 Speciation

Another GA property that can be studied is the NEAT-specific speciation in scenarios D and E. Figure 4.23 shows the number of different species over time. Both follow approximately the same development in the first 40 generations, before D overtakes E and they stabilize at different levels. The difference in levels is quite significant. A speculative reason for this is that the lack of elitism in D means that a species there is more likely to split into multiple species by chance. Whereas with elitism in E, the members of the species may be more likely to remain more similar to the elite in the population, leading to less diversity *within* the species.

Figure 4.24 shows the mean number of members per species over time. The development is as one would expect with the growing number of species seen in Figure 4.23, declining more rapidly at first then gradually stabilizing. The accompanying plot of the

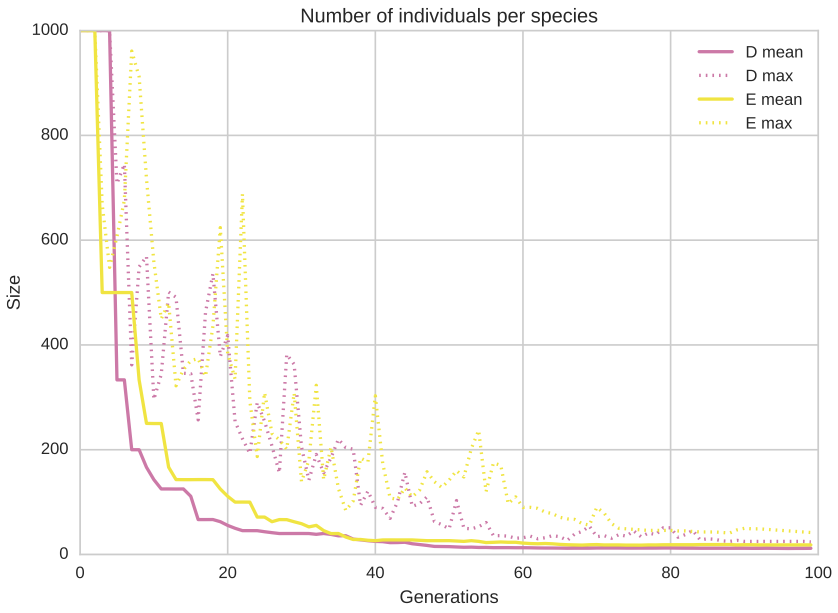


Figure 4.24: The mean and max number of members in the species of scenarios D and E over time

max number of species members gives an indication of the variance of the underlying data. It fluctuates a lot at first, but as 100 generations approach, the max stabilizes and is only slightly higher than the mean, indicating that the population consists of many species approximately the same (low) number of members.

In the case where there is elitism, as the species size approaches the number of elites ($E = 1$) there is less room for innovation in that species, since one member of each species is always a copy of a previous member. If the species size were to reach 1, there would be no innovation at all happening. Since the population of scenario E seems to have stabilized around 55 species, a mean species size around $1000/55 \approx 18$ should avoid this problem and leave room for innovation.

4.5.4 λ

The λ parameter is an interesting property of a CA transition function to study. Figure 4.25 shows the mean λ over time for the five scenarios. The shapes of the curves have a lot of similarities with the shapes of the fitness curves in Figure 4.21. The random mutation in scenario A drives the mean λ towards 0, while the combination of mutation and crossover without selection pressure of scenario B causes the λ to fluctuate a lot and increase slightly, but not very significantly. Scenarios C, D and E have roughly the same development: a sharp rise to about 0.9, then flattening out.

Figure 4.26 shows the scenarios broken down individually into stack plots, illustrating the distribution of values changing over time. The values are sorted into six bins, two

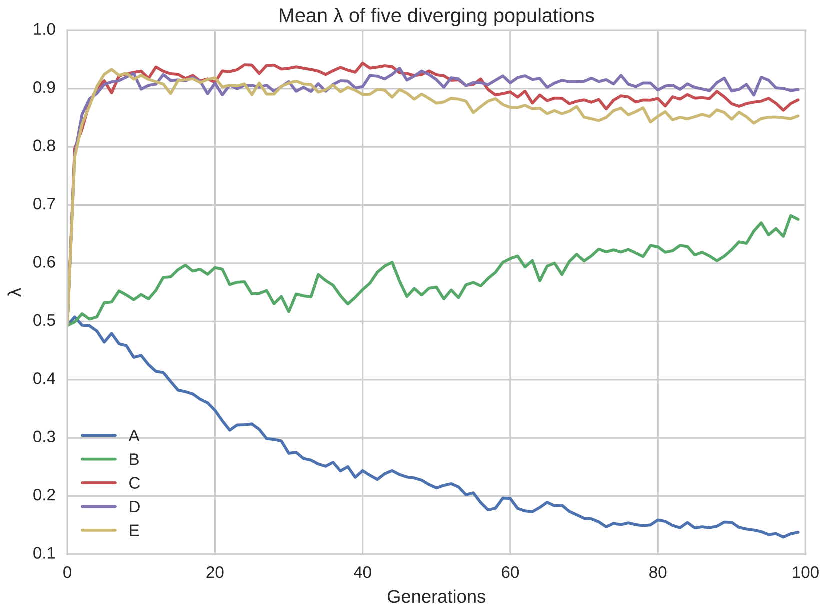


Figure 4.25: Development of the mean λ of five scenarios over time

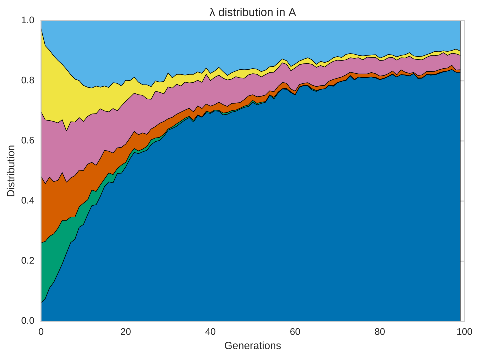
special ones for λ exactly equal to 0.0 and 1.0 and four for the equal intervals in between. In earlier experiments it was observed that the two extreme λ occur often, so they get special bins in this visualization.

It can be seen that scenario A has a very strong bias towards producing $\lambda = 0.0$, while the other scenarios skew more towards producing higher λ values. This would suggest that crossover mechanism is at least partially responsible for producing higher λ values. There is also a strong resemblance between C and E, less so between D and the others. Why this is so is not entirely obvious, but it could just be a consequence of the randomness of the trial. If the experiment was repeated with multiple trials, we could determine if this is just a fluke or not. In any case, it is clear that all the scenarios with selection pressure are strongly biased towards producing $\lambda > 0.75$.

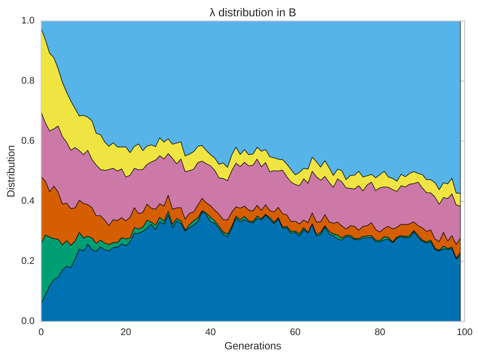
The task at hand and the implementation of the fitness function must also be considered when analyzing the λ result. The fitness evaluation function is described in detail in Section 4.2.1. Notably, when trying to produce a "Swiss flag" pattern, $20/25$ cells are active in the target state, meaning that the algorithm can get *some* score easily by producing a transition function with $\lambda = 1.0$. This can explain why the algorithm produces very many $\lambda = 1.0$ solutions early on.

4.5.5 Distinct Behaviors

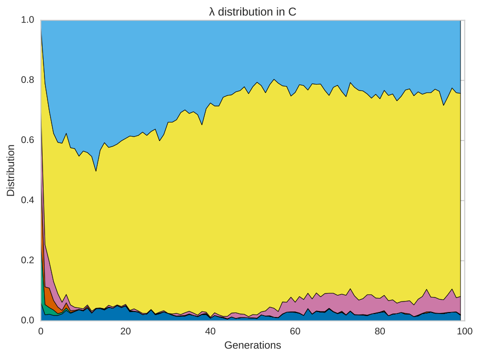
Another way to analyze the population is to look at the diversity of the enumerated behavior "strings" (described in Section 3.1.1) over time. Figure 4.27 shows how many unique



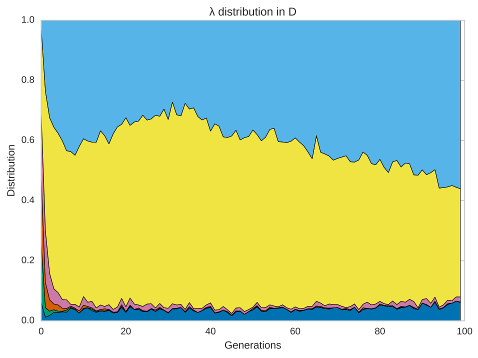
(a) Scenario A



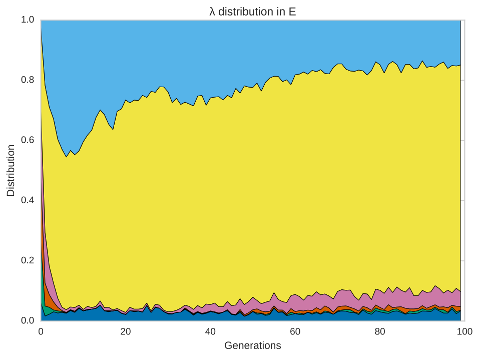
(b) Scenario B



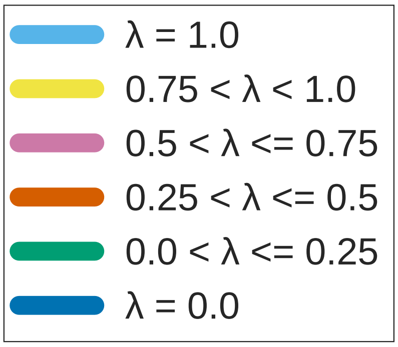
(c) Scenario C



(d) Scenario D



(e) Scenario E



(f) Legend

Figure 4.26: Breakdown of λ distribution over time in five scenarios

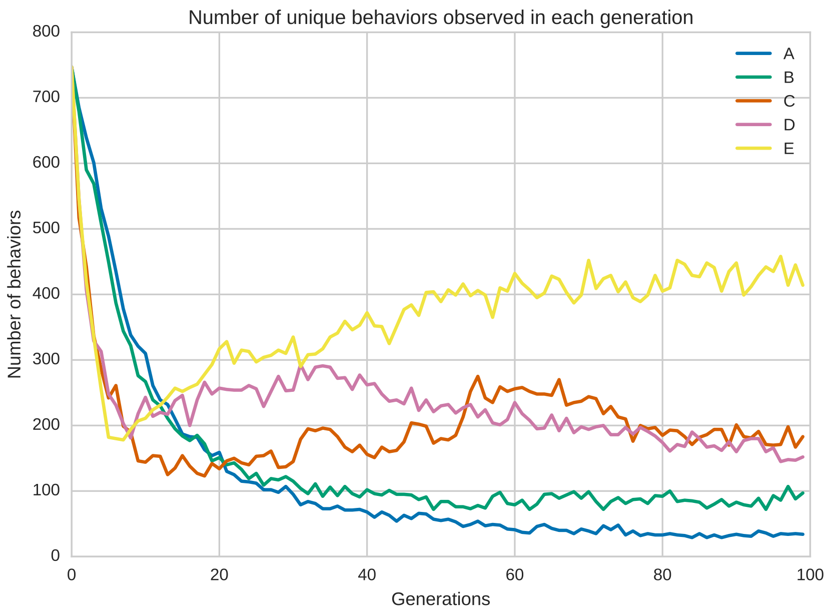


Figure 4.27: Number of unique behaviors in each generation

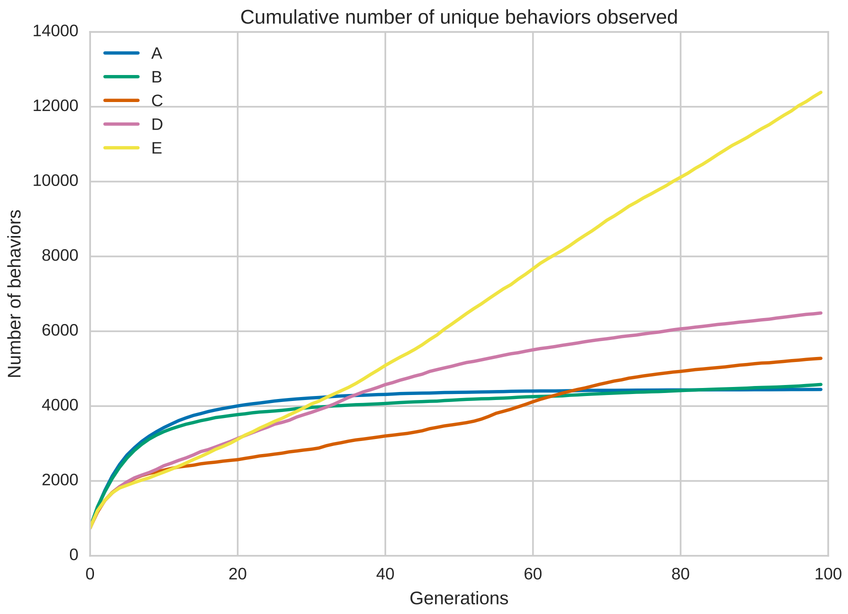


Figure 4.28: Number of unique behaviors seen over time, cumulative.

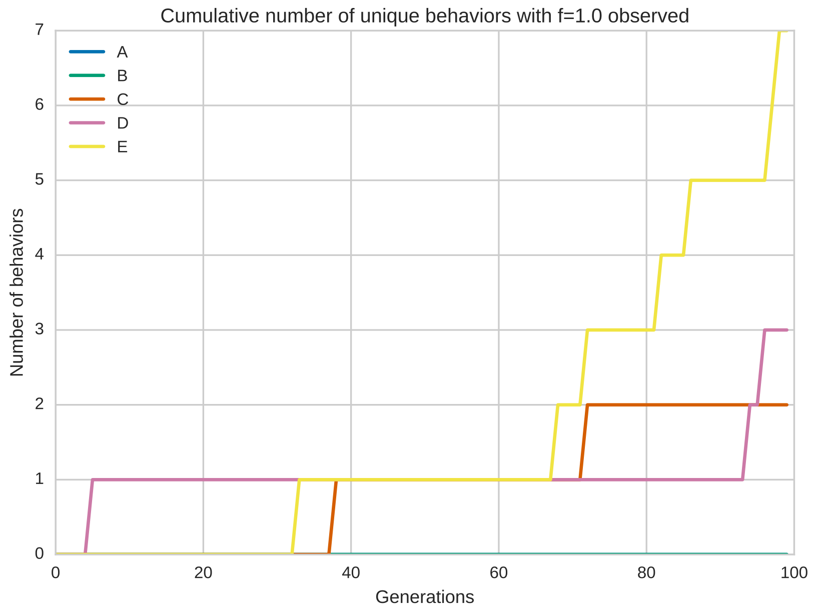


Figure 4.29: Number of unique $f = 1.0$ behaviors seen over time, cumulative.

behaviors are present in each generation of each scenario. The initial population is clearly very diverse, but the diversity drops very rapidly in each of the scenarios. Scenarios A and B decrease slightly slower than the rest at the beginning. The rate of decreasing slows down and they eventually stabilize at low levels. Scenarios C, D and E drop down fast at first, but then they do a sharp turn. C and D fluctuate a it up and down, but overall seem to stabilize. E rises steadily again and stabilizes at a significantly higher level than the others.

Figure 4.28 visualizes the number of unique behaviors seen in the entire lifetime of the scenario. A and B are almost identical, showing almost no increase after the first 20 generations. C and D climb steadily and overtake A and B at different points. The contrast between E and the rest is stark. There is a much higher rate of increase, and it is almost linear over time, reaching about twice the value that the next best does in 100 generations. These result have some big implications. First of all, it is clear that a random search such as A and B gets ”stuck” quite quickly and stops producing innovative results. With selection pressure, the search is slower at first, needing quite some time to overtake the initial flood of innovation produced by randomness. But it keeps a steady increase where the random searches stagnate. And the presence of elitism increases the innovation by a very considerable degree.

In a ”best-case” where every behavior in every generation is distinct, the number of unique behaviors observed would be $P * G = 100000$. Scenario E passes 12000 unique behaviors observed, which is ”only” 12% of the theoretical maximum. And for a $K = 2, N = 5$ CA, the number of total possible behaviors is $K^{K^N} = 2^{32} \approx 4.3 * 10^9$. This

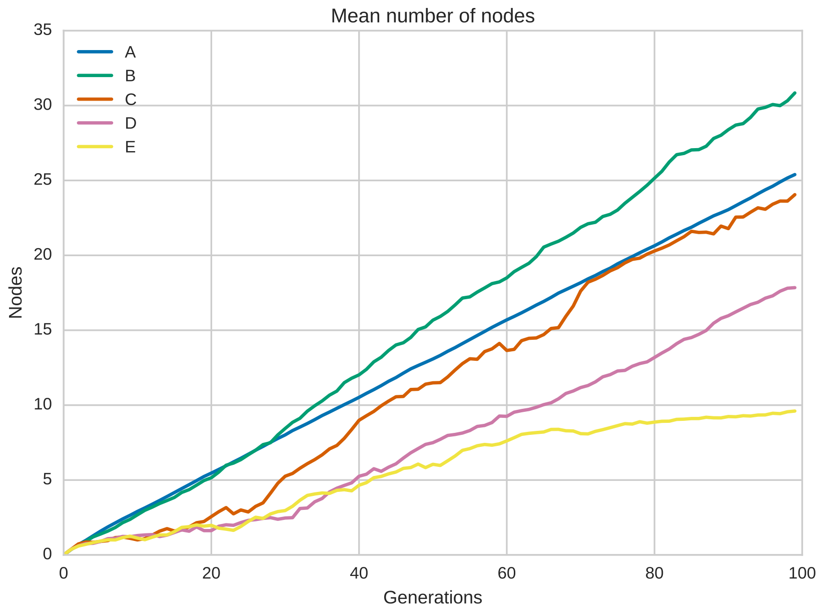


Figure 4.30: The mean number of nodes in each scenario

illustrates the futility of attempting to solve advanced CA tasks by exhaustive enumeration of behaviors. Luckily, while the proverbial haystack may be humongous, there are many needles to be found within it, and you only need to find one to be successful. Figure 4.29 illustrates the number of distinct behaviors observed with a perfect fitness score. As one might expect, the diversity of the optimal behaviors is higher when the diversity of all behaviors is higher. Scenario E is able to find 7 distinct solutions to the task in 100 generations.

4.5.6 Network Topology

The individuals of the populations are graph structures, and can also be studied as such. Appendix B showcase a selection of structures CA-NEAT has found. Figure 4.30 shows the development of the mean number of nodes in the networks of each scenario. The growth of scenario A is approximately linear. Since the individuals of the population are completely independent from each other, the law of large numbers means that the curve should fit the expected value given by the probabilities of adding and removing nodes, $(P_{\text{add}} - P_{\text{remove}})x = (0.5 - 0.25)x = 0.25x$. The full listing of mutation probabilities is given in in Appendix A. This is not a very useful insight by itself, but it does give a baseline which the other scenarios can be compared to. Scenario B has a slightly higher growth than A. This makes sense considering the crossover mechanism. If two parents independently are likely to gain a new node, then their child will have both of the new nodes, leading to a bigger growth rate overall. The three scenarios with selection pressure

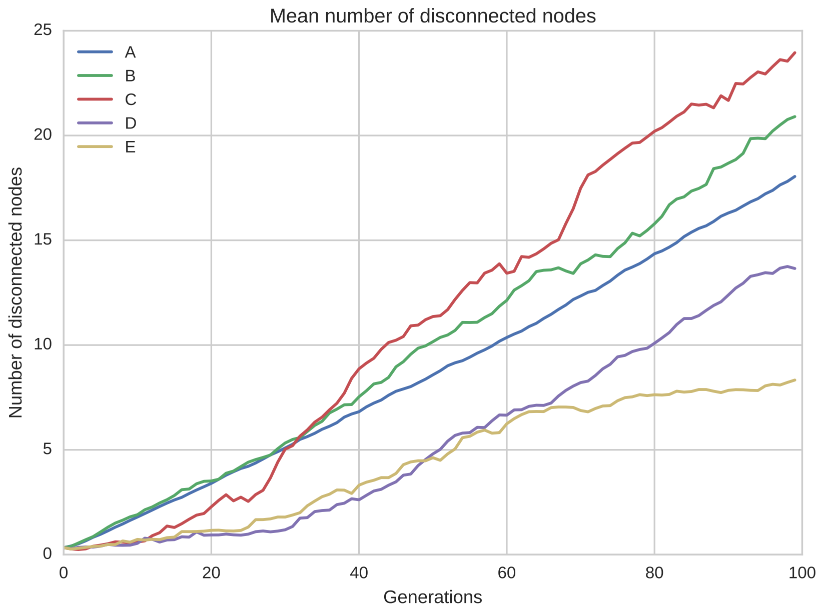


Figure 4.31: The mean number of disconnected nodes in each scenario

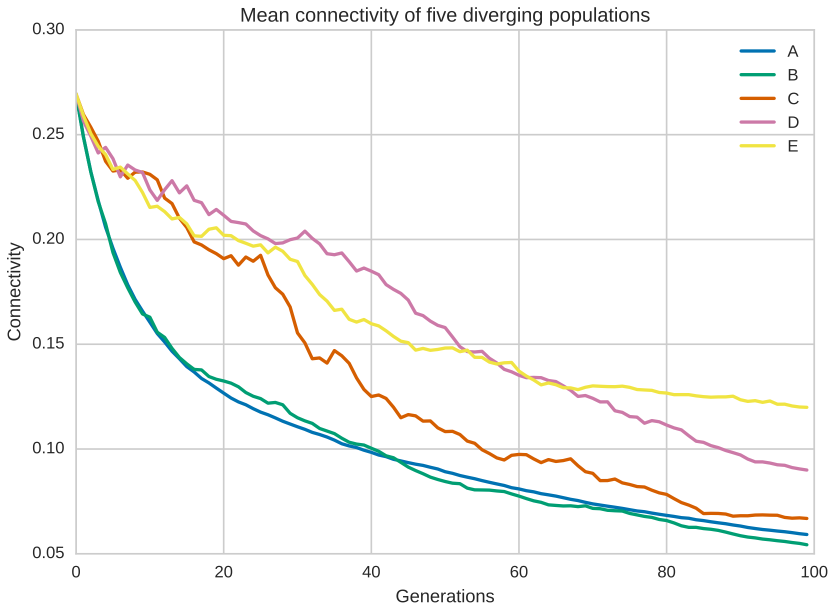


Figure 4.32: The mean connectivity degree in each population

at first follow the same pattern, with much lower growth than the baseline. Then they diverge around 30 generations. Scenario C ”catches up” with scenario A and settles into a near-linear growth like scenario A. Scenario D also goes into a near-linear growth close to the same as scenario A, but without ”catching up” first. Scenario E continues with almost the same growth rate for all 100 generations, ending up at a much lower number than the rest.

Figure 4.31 shows the number of vestigial nodes that do not connect to the output layer. The significance of these is that they are equally likely to be affected by mutation as any other node, but they do not affect the output of the network. The more vestigial nodes there are, the more likely that the mutation operation does ”nothing”.

Scenarios A and B have curve shapes very similar to those they had in Figure 4.30, suggesting a simple proportional correlation when the process is completely random. More interestingly, the curve of scenario C does a sudden turn to overtake both A and B. The curve of C reaches almost the same value as that in Figure 4.30, indicating the population members have a very large number of disconnected nodes. The curves of D and E are similar to their counterparts in Figure 4.30. Again it is clear that E has a distinctly different development than the rest.

Figure 4.32 shows another measure of connectedness which is often used in graph theory, connectivity $= \frac{c}{\sum_{x=1}^{n} x}$, where c is the number of connections and n is the number of nodes. In this perspective, C follows D and E at first, but diverges and drops much quicker to the level of A and B.

What these observations seem to indicate, is that the population of C comes to be dominated by networks with many nodes and fewer connections. It is likely that in repeated trial this would not occur every time, but that it is just part of the randomness of the single trial. This is perhaps more likely to occur with the scenario C parameters than the others. Scenarios A and B are somewhat predictable systems with behavior determined by the constant parameters set before the experiment. The selection pressure of C can possibly act as a feedback loop, causing a random feature to dominate the whole population. Scenarios D and E have speciation which automatically create (somewhat) independent trials within the population.

4.6 Novelty Search

Section 3.2 describes how CA-NEAT was extended to support novelty search.

Because novelty search disregards the objective, a search for a specific neighborhood size N and number of states K creates population that can be used on any CA with that N or K. For example, the same population could contain solutions to both morphogenesis and replication of both the 6x6 ”Tricolor” and 7x7 ”Nordic” patterns (Figure 4.1).

4.6.1 Results

Several different N and K combinations were tested and the resulting innovations archives checked against the objective function of appropriate tasks. Table 4.6 gives an overview of the combinations tested.

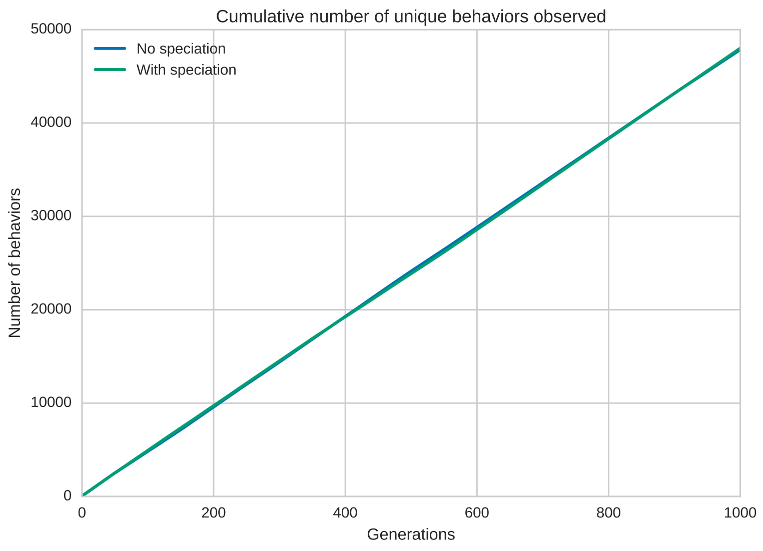


Figure 4.33: Cumulative number of unique behaviors observed. The two runs share the properties $N = 5, K = 4, P = 50, G = 1000$, and differ in whether they have speciation enabled.

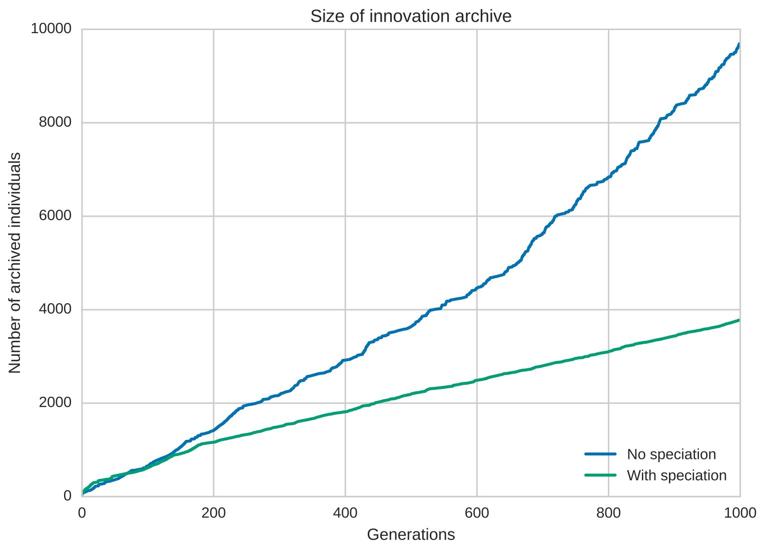


Figure 4.34: Size of the innovation archives. The archives are a subset of their corresponding populations.

Table 4.6: Overview of the novelty search runs attempted

N	K	P	G	Speciation	Objectives tested
5	4	50	1000	Yes	{Tricolor, Nordic} {morphogenesis, replication}
5	4	50	1000	No	{Tricolor, Nordic} {morphogenesis, replication}
5	2	50	1000	No	Border morphogenesis
7	2	200	1000	No	Majority, synchronization

The tasks that were tested for were selected because they were sufficiently challenging for the objective search (as described in Sections 4.2, 4.4). Other tasks such as ”Mosaic” morphogenesis had multiple solutions present in the initial population. Using novelty search in such cases would not tell us anything about the effectiveness of the search algorithm.

When testing against the objective functions of the sufficiently challenging tasks, the results were not particularly useful, with no optimal solutions found for any of the tasks tested. Detailed results about the fitnesses found are not too interesting and is omitted from this report. Suffice it to say, the success rate was 0% across the board.

One configuration that was tested was $N = 5$, $K = 4$, a population size of $P = 50$, for $G = 1000$ generations. We will take closer look at the results, assuming them to be representative for the results of all the experiments. This configuration was tested in two independent runs, with and without speciation. Figure 4.33 shows the cumulative number of unique behaviors observed in the two runs. The figure is created from the full population of each run, not the innovation archive, which is a subset of the full population. It shows that the novelty search is correctly implemented and does in fact produce many novel genotypes in each generation. It can be compared to Figure 4.28 in Section 4.5, taking into account the different population size. The curves are approximately equal, showing no particular effect from the presence or lack of speciation. At 1000 generations, the runs are at 47806 unique individuals (no speciation) and 47969 (with speciation). This is as good as equal, and it is close to the curve of the maximum possible unique behaviors $y(x) = 50x$. Still, 50000 unique individuals would only be $\frac{50000}{2^{2^5}} \approx 0.0012\%$ of the search space.

Figure 4.34 shows the development of the sizes of the innovation archives. The two archive sizes are approximately the same in the first 150 generations, but then diverge considerably. The population without speciation adds far fewer innovations to the archive than the population with speciation does. Over 1000 generations, the population with speciation adds on average almost 10 individuals to the archive per generation, while the population with speciation adds on average 3-4 innovations per generation. Since the speciation threshold is supposed to dynamically adjust to keep the number of added innovations between 1-5, this must mean that the average innovation metric in the speciated population is changing continuously, so that the threshold is not able to keep up. This could happen because the average innovation degree is continuously increasing, or because it is fluctuating around some value. The species extinction may be the cause of this, since it will be removing whole groups of individuals that may affect the innovation degree drastically. In either case the threshold adjusting algorithm is always a step behind.

Because of performance issues with the implementation, it was not feasible to run

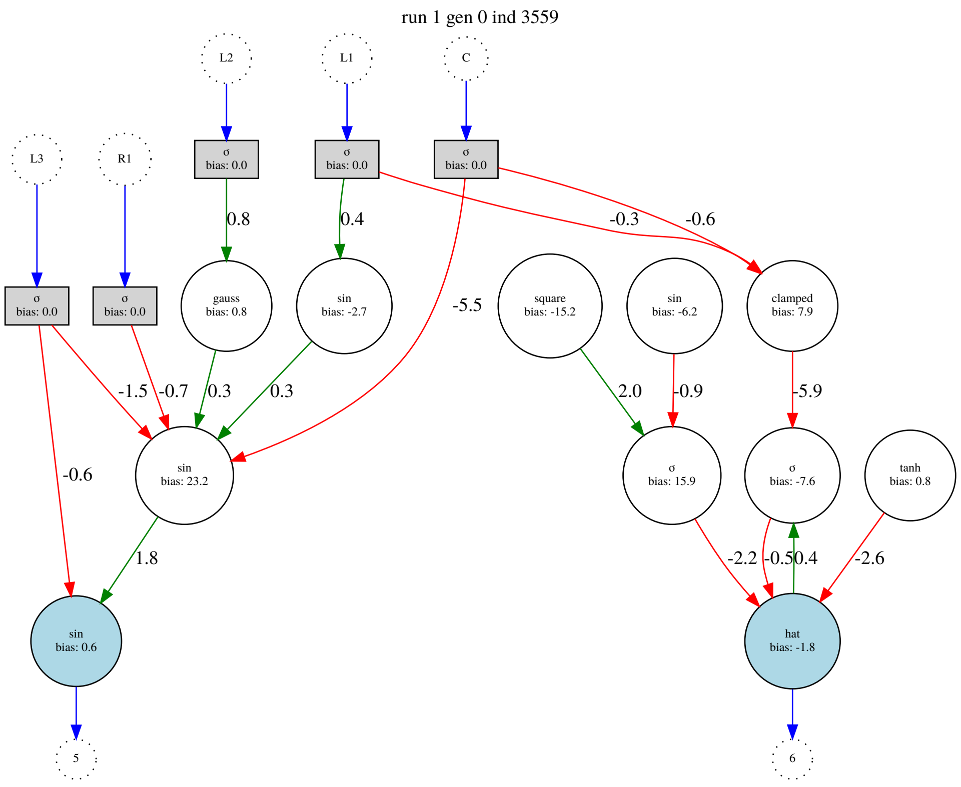


Figure 4.35: Network found by novelty search

many independent trials of the same configuration. There is a possibility that in a larger number of trials, some might succeed. But a more performant implementation is necessary to test this.

These results suggest that this approach to novelty search is not going to lead to produce anything that is more interesting than what objective search produces. This does not necessarily mean that novelty search is never going to work for CA problems, only that the setup used in these experiments is flawed and must be reconsidered.

Figures 4.35 and 4.36 show an example of a network found in the $N = 5, K = 2$ novelty search and the behavior when tested in a 5x5 morphogenesis cellular model.

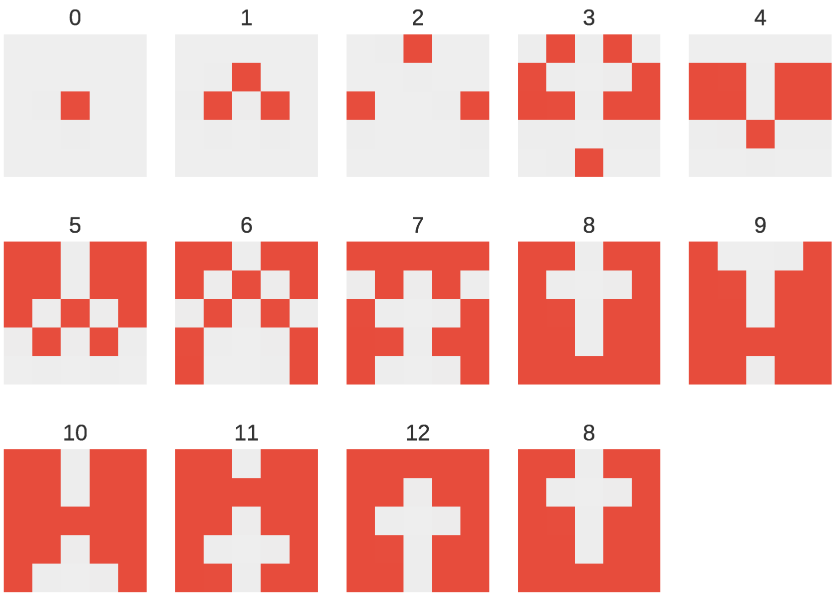


Figure 4.36: The behavior of the network shown in Figure 4.35

Chapter 5

Discussion & Future Work

The individual experiments are discussed in their sections in Chapter 4. This chapter discusses the overarching themes, what went well and not so well, and the possibilities for future work.

5.1 Tasks Solved

Morphogenesis of all the patterns listed in Figure 4.1 has been achieved, either using only neighborhood information or using neighborhood information in conjunction with coordinate information. In comparison with table-based and instruction-based evolution, CA-NEAT did better than either in most cases. This is a very promising result with potential applications in different morphogenetic engineering situations, for example constructing micro- or nanoscale structures in carbon nanotubes or bio-matter. Such tasks can be difficult to perform by machine, but might be possible if the substrate can be stimulated to assemble itself.

Replication of four patterns was tested with a standard neighborhood information cellular model. The results were in general better than the corresponding results with table-based evolution, though not as good as the results of instruction-based evolution.

With positive results from the morphogenesis with coordinate information, it seems likely that the replication performance can also be improved by adding information to the environment. For example, the environment could be augmented with ”anchor points” in cells at intervals, giving the CA something a ”scaffold” when producing duplicates, similar to how some crystals form in nature (heterogeneous nucleation).

For both the majority and synchronization problems some phenotypes were found that performed quite well. While the results were not in any way revolutionary, they do not exclude the possibility that a more fine-tuned model could perform well. Optimizing the parameters of the algorithm may be crucial to achieve this. It would also be interesting to investigate whether different parameters are optimal for different problem categories.

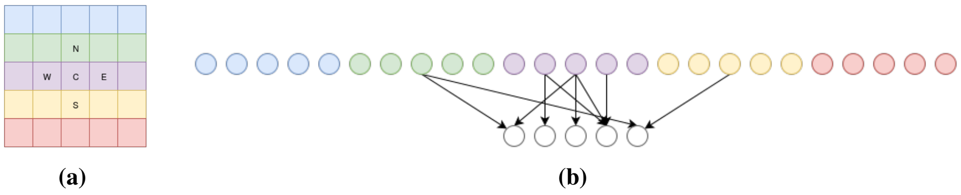


Figure 5.1: An example CPPN with a large available input neighborhood, but only the Von Neumann sub-neighborhood connected to the output layer.

5.2 Neighborhood Definitions

The experiments in this thesis have only used the Von Neumann 2D neighborhood definition and the size 7 1D neighborhood definition. The neighborhood definition of the cellular model can be expanded to greater sizes easily in CA-NEAT, since it is as simple as adding new input nodes to the CPPN. Adding another input node is not any more complicated than adding another hidden node is, nor does it affect the run time performance of the algorithm any more. Larger neighborhood definitions would create much more diversity in the initial population, but could also possibly create genotypes that are more complex than what is required for the problem.

Figure 5.1 shows one possible way to implement larger neighborhoods while trying to avoid over-complication. Since NEAT can add connections by mutation, there could be many input nodes *available*, but only a small selection connected in the initial population. When mutating new genotypes, new inputs could be connected, and over time it could be determined if this was a good innovation. This is similar to *FS-NEAT*, a NEAT variation which has been shown to solve certain tasks more efficiently than regular NEAT [54].

5.3 Network Size

In many of the morphology tasks, both with and without environmental information, there were solutions present in the initial populations of some of the trials. This means that those tasks could be solved with only the input and output neurons and the connections between them.

The results of the analysis of network size also indicate that the complexification by network growth can be detrimental. The scenario that was able to find the greatest number of solutions was the one that had the lowest average number of nodes, and the lowest average number of disconnected nodes.

If a problem *can* be solved with a smaller structure, then the possibility of adding to the size may be detrimental to the performance, since it introduces more components that need to be tuned. It might be worthwhile in the future to try an experiment where the mutation probabilities of adding and removing nodes is set to 0 and see how NEAT does when tuning a fixed-size structure. Another possibility could be to allow the adding of nodes, but to have the fitness evaluation penalize it. The selection pressure would then be optimizing both for task solving and for smaller genomes.

5.4 Exploring the CA Behavior Space with Objective and Novelty Search

With both objective search NEAT and novelty search, the algorithms are able to explore the space of CA behaviors quite effectively. However, the search spaces are very large, and fully exploring them would take very large populations over very many generations. A given space may have multiple optimal solutions present, but if they are very sparse or the fitness evaluation is deceitful, then objective search can struggle to succeed. The ”Nordic flag” replication task is like this. It has the same behavior space as the other four-color tasks, but it is a ”tricky” task because of the shifted symmetry of the pattern. The task was solved by the instruction-based encoding in [46], but CA-NEAT could not find any solutions with the same cellular model.

While the universal approximation theorem says that an ANN-structure such as the CA-NEAT encoding can encode any function, it does not tell us which kind of functions are easy to encode and which are difficult. CPPN may be predisposed to approximate some types of functions, and be less disposed to other types. A potential future experiment could try to categorize the found behaviors, to see if some types of behaviors are dominant or missing. One possible taxonomy that could be used is Wolfram’s classification [16].

Novelty search was tried on non-trivial problems. While a large number of behaviors were investigated, the search could not find behaviors that were objective-optimal. Novelty search explores more efficiently than objective search, but without being able to direct the search towards an optimum it just meanders around the behavior space.

The choice of the enumerated mapping string to measure the innovation distance between phenotypes in novelty search has disadvantages. The novelty search was able to explore the space at an almost optimal rate, yet 1000 generations of search could only explore about a $\frac{1}{100000}$th of the space. The use of the behavior string also ”reintroduces” the restrictions of table-based transition functions to CA-NEAT. It can’t handle other types of input that cannot be enumerated (e.g. environment information like in Section 4.3), and as the size of the neighborhood or the number of cell states increases, so does the computational complexity of creating the strings and comparing them.

This does not mean CA-NEAT and novelty search are incompatible. It should be possible to use other more suitable properties for the innovation measure that have smaller spaces. These properties could be task-specific. For example, in a morphogenesis task where finding a point attractor is required, the final (stable) state of the CA could be used as a vector, and distance calculated appropriately.

Another possible option is to use the innovation archive population as a diverse initial population for objective search. Instead of balancing the trade-off between exploration and exploitation, the search would have two distinct phases for exploration and exploitation. It is possible some of the genotypes in the archive are close to, but not quite optimal, and that some generation of objective search evolution could lead to optimal results. It would probably be necessary to sample the archive though, as it may be much larger than the size of a population in NEAT.

5.5 The Role of NEAT Mechanisms and Parameters

The investigation of the underlying properties of the populations revealed a lot of interesting patterns. The utility of elitism was very apparent. The scenario with elitism was able to explore the CA behavior space much more effectively, and found many more solutions than the other scenarios. The population with elitism had fewer nodes in total, and thereby also fewer vestigial nodes and a higher connectivity degree. Having a larger number of nodes appeared to be detrimental to the performance of the other populations. With more nodes there were also more vestigial nodes, and so more of the mutation operations would have been ineffective, leading to less innovation.

The property analysis performed in this project has only touched upon a few of the properties of the encoding. Future experiments could explore more of them to gain an even deeper understanding. There are decades of research on network structures and network growth to draw from [55]. There are also other properties and parameterizations of CA transition rules that can also be studied, such as the *sensitivity* measure μ or the *pre-image* degree Z [56].

5.6 CPPN Domain and Activation Functions

The set of possible neuron activation functions used in these experiments (Table 3.1) are the ones provided by `python-neat` by default. No analysis of how appropriate these are for CA problems was performed. This set of activation functions is meant to be used with $\mathbb{R}$ input values, so the cell states are mapped to $\mathbb{R}$ before they are input to the CPPN. A potential future experiment could explore other activation function sets, potentially in the cell state domain. For example, for a binary CA problem the possible activation functions could be binary functions like $\{\vee, \wedge, \oplus\}$ and the connection weights could be 1 or -1 ($\neg$). Instead of K network outputs of which one is selected, there would be one output which would give the next state. Binarized neural networks have been shown to perform well at some machine learning applications, for example those relating to image processing of black and white images [57].

5.7 Implementation Critique

The basic CA-NEAT framework was not terribly difficult to implement, especially since `python-neat` provided an excellent basis. The biggest technical challenge was to take advantage of the parallelism at the fitness evaluation stage. This was implemented using `celery`, and was crucial to being able to run multiple concurrent experiments each with multiple concurrent fitness evaluations, which allowed for a large volume of results.

The CA-NEAT model lended itself very easily to extension with environmental input. Programming the extension was only a matter of rewriting the fitness evaluation function and specifying a bigger number of input nodes for the CPPNs. The extension did not have a noticeable effect on the run time performance of the program.

When implementing CA-NEAT, the choice of database for persisting results was a relational (SQL) database. When CA-NEAT was extended to novelty search, this meant that the innovation archive was also implemented as relational table. This had detrimental effects on the performance of the system that made gathering results take much longer than it did for objective search. As the innovation archive grew larger during experiments, it became obvious that this had a large detrimental effect on the performance of the algorithm, both in terms of time spent calculating and in terms of memory used during calculations. In every generation the whole innovation archive was queried from the database and loaded into memory, and the Hamming distances were calculated between the whole generation of individuals and the whole of the archive. This also prevented concurrency in fitness evaluation, further slowing the process. If implementing novelty search from scratch, a more suitable database structure could be used instead. Inspiration could be taken from the techniques used for embedding spaces in machine learning applications.

Chapter 6

Conclusion

CA-NEAT has been tested on both morphology problems (morphogenesis, replication) and computational problems (density classification, synchronization). Some tasks were more difficult than others, but in each category CA-NEAT was able to produce some good behaviors. Some of the morphology tasks were easily solved by using a simple cellular model with only neighborhood information, and some tasks that were difficult with the simple model became solvable when extending the model with coordinate information.

The success of the extended model at task performance, and the ease of adding the extension to the encoding is a very promising result. The CPPN-based encoding could help make many difficult tasks possible, not only CA tasks, but also potentially in other morphogenetic engineering models.

The analysis of the relationship between NEAT mechanisms and properties of the population revealed some interesting trends. It definitely showed the advantage of NEAT with speciation and per-species elitism in exploring the space of CA behaviors. It also showed that the genetic algorithm can reach high fitnesses and specific λ values in just a few generations, but that sometimes a large number of generations of fine-tuning is needed before finding the perfect solution.

It was also apparent that the network growth could be detrimental to the performance of the population. With high probability of mutation adding to the network, the networks grew very large. The more components of the network, the more tuning is required. With mutation also sometimes removing connections, many of the nodes would be disconnected from the output nodes. With nodes that do not affect the network output, but are still targets of mutation, the effectiveness of mutation to explore the space is diminished. Parameters and mechanisms that limit the growth of the network may bring about better results.

While the novelty search implementation used in this thesis did not produce any optimal results for any of the tasks investigated, it was very efficient at exploring the space of CA behaviors. But because of the size of the space, much more time would have to be spent searching to find optimal results consistently. But novelty search does not to be written off entirely for CA problems. If smaller problem-specific search spaces can be defined and searched, it might be possible to use novelty search successfully for CA problems.

Bibliography

[1] Mathias Berild Ose. *Evolved Compositional Pattern Producing Networks As Cellular Automata Transition Rules*. Research rep. Department of Computer Science Norwegian University of Science and Technology, 2016.

[2] Stefano Nichele et al. “CA-NEAT: Evolved Compositional Pattern Producing Networks for Cellular Automata Morphogenesis and Replication”. In: *IEEE Transactions on Cognitive and Developmental Systems* (2017). Forthcoming.

[3] Douglas Adams. *The Salmon of Doubt: Hitchhiking the Universe One Last Time*. Harmony, 2002.

[4] “Roget’s 21st Century Thesaurus, Third Edition”. In: (Mar. 2017). URL: `http://www.thesaurus.com/browse/detailed`.

[5] Herbert A Simon. “The architecture of complexity”. In: *Proceedings of the American philosophical society* 106.6 (1962), pp. 467–482.

[6] Hiroki Sayama. *Introduction to the modeling and analysis of complex systems*. Open SUNY Textbooks, 2015.

[7] Francis Heylighen et al. “The science of self-organization and adaptivity”. In: *The encyclopedia of life support systems* 5.3 (2001), pp. 253–280.

[8] Hans Moravec. *Mind children: The future of robot and human intelligence*. Harvard University Press, 1988.

[9] René Doursat, Hiroki Sayama, and Olivier Michel. “A review of morphogenetic engineering”. In: *Natural Computing* 12.4 (2013), pp. 517–535.

[10] John von Neumann. *Theory of Self-Reproducing Automata*. Ed. by Arthur W. Burks. Champaign, IL, USA: University of Illinois Press, 1966.

[11] Robert R Schaller. “Moore’s law: past, present and future”. In: *IEEE spectrum* 34.6 (1997), pp. 52–59.

[12] Michael J. Flynn. “Parallel processors were the future... and may yet be”. In: *Computer Magazine* (1996).

[13] Matthew Cook. “Universality in elementary cellular automata”. In: *Complex systems* 15.1 (2004), pp. 1–40.

[14] Moshe Sipper. “The Emergence of Cellular Computing”. In: *Computer Magazine* (1999).

[15] Carlos Gershenson. “Introduction to random Boolean networks”. In: *arXiv preprint nlin/0408006* (2004).

[16] Stephen Wolfram et al. *Theory and applications of cellular automata*. Vol. 1. World scientific Singapore, 1986.

[17] Chris G Langton. “Computation at the edge of chaos: phase transitions and emergent computation”. In: *Physica D: Nonlinear Phenomena* 42.1 (1990), pp. 12–37.

[18] Yoshua Bengio, Ian J Goodfellow, and Aaron Courville. “Deep learning”. In: *An MIT Press book in preparation. Draft chapters available at http://www. iro. umontreal. ca/ bengioy/dlbook* (2015).

[19] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. “Multilayer feedforward networks are universal approximators”. In: *Neural Networks* 2.5 (1989), pp. 359–366. ISSN: 0893-6080. DOI: `http://dx.doi.org/10.1016/0893-6080(89)90020-8`. URL: `http://www.sciencedirect.com/science/article/pii/0893608089900208`.

[20] Kenneth O Stanley. “Compositional pattern producing networks: A novel abstraction of development”. In: *Genetic programming and evolvable machines* 8.2 (2007), pp. 131–162.

[21] Kenneth O Stanley. “Exploiting regularity without development”. In: *Proceedings of the AAAI Fall Symposium on Developmental Systems*. AAAI Press Menlo Park, CA. 2006, p. 37.

[22] David B D’Ambrosio and Kenneth O Stanley. “Generative encoding for multiagent learning”. In: *Proceedings of the 10th annual conference on Genetic and evolutionary computation*. ACM. 2008, pp. 819–826.

[23] Sebastian Risi and Kenneth O Stanley. “Confronting the challenge of learning a flexible neural controller for a diversity of morphologies”. In: *Proceedings of the 15th annual conference on Genetic and evolutionary computation*. ACM. 2013, pp. 255–262.

[24] C. Darwin. *On the Origin of Species by Means of Natural Selection*. London: John Murray, 1859.

[25] John H Holland. “Genetic algorithms”. In: *Scientific american* 267.1 (1992), pp. 66–72.

[26] Melanie Mitchell. “Life and evolution in computers”. In: *History and philosophy of the life sciences* (2001), pp. 361–383.

[27] JA Vasconcelos et al. “Improvements in genetic algorithms”. In: *IEEE Transactions on magnetics* 37.5 (2001), pp. 3414–3417.

[28] Jeff Clune et al. “On the performance of indirect encoding across the continuum of regularity”. In: *IEEE Transactions on Evolutionary Computation* 15.3 (2011), pp. 346–367.

[29] Kenneth O Stanley and Risto Miikkulainen. “Evolving neural networks through augmenting topologies”. In: *Evolutionary computation* 10.2 (2002), pp. 99–127.

[30] JE Darnell and WF Doolittle. “Speculations on the early course of evolution”. In: *Proceedings of the National Academy of Sciences* 83.5 (1986), pp. 1271–1275.

[31] Addy Pross. “On the emergence of biological complexity: life as a kinetic state of matter”. In: *Origins of Life and Evolution of Biospheres* 35.2 (2005), pp. 151–166.

[32] Kenneth O Stanley, David B D’Ambrosio, and Jason Gauci. “A hypercube-based encoding for evolving large-scale neural networks”. In: *Artificial life* 15.2 (2009), pp. 185–212.

[33] Jason Gauci and Kenneth Stanley. “Indirect encoding of neural networks for scalable go”. In: *Parallel Problem Solving from Nature, PPSN XI* (2010), pp. 354–363.

[34] Brian Woolley and Kenneth Stanley. “Evolving a single scalable controller for an octopus arm with a variable number of segments”. In: *Parallel Problem Solving from Nature, PPSN XI* (2010), pp. 270–279.

[35] Joel Lehman and Kenneth O Stanley. “Exploiting Open-Endedness to Solve Problems Through the Search for Novelty.” In: *ALIFE*. 2008, pp. 329–336.

[36] Enrique Naredo, Leonardo Trujillo, and Yuliana Martı’nez. “Searching for novel classifiers”. In: *European Conference on Genetic Programming*. Springer. 2013, pp. 145–156.

[37] Jorge Gomes, Paulo Urbano, and Anders Lyhne Christensen. “Evolution of swarm robotics systems with novelty search”. In: *Swarm Intelligence* 7.2-3 (2013), pp. 115–144.

[38] Josh Wolper and George Abraham. “Evolving Novel Cellular Automaton Seeds Using Computational Pattern Producing Networks (CPPN)”. In: (2015).

[39] Elwyn R Berlekamp, John Horton Conway, and Richard K Guy. *Winning Ways, for Your Mathematical Plays: Games in particular*. Vol. 2. Academic Pr, 1982.

[40] Michal Bidlo and Zdenek Vasicek. “Evolution of cellular automata with conditionally matching rules”. In: *2013 IEEE Congress on Evolutionary Computation*. IEEE. 2013, pp. 1178–1185.

[41] Michal Bidlo. “Investigation of Replicating Tiles in Cellular Automata Designed by Evolution Using Conditionally Matching Rules”. In: *Computational Intelligence, 2015 IEEE Symposium Series on*. IEEE. 2015, pp. 1506–1513.

[42] Michal Bidlo. “On routine evolution of new replicating structures in cellular automata”. In: *7th International Conference on Evolutionary Computation Theory and Applications. SCITEPRESS*. 2015.

[43] Michal Bidlo and Jaroslav Škarvada. “Instruction-based development: From evolution to generic structures of digital circuits”. In: *International Journal of Knowledge-Based and Intelligent Engineering Systems* 12.3 (2008), pp. 221–236.

[44] Michal Bidlo and Zdenek Vasicek. “Evolution of cellular automata using instruction-based approach”. In: *2012 IEEE Congress on Evolutionary Computation*. IEEE. 2012, pp. 1–8.

[45] Stefano Nichele, Tom Eivind Glover, and Gunnar Tufte. “Genotype Regulation by Self-modifying Instruction-Based Development on Cellular Automata”. In: *International Conference on Parallel Problem Solving from Nature*. Springer. 2016, pp. 14–25.

[46] Stefano Nichele and Gunnar Tufte. “Evolutionary growth of genomes for the development and replication of multicellular organisms with indirect encoding”. In: *Evolvable Systems (ICES), 2014 IEEE International Conference on*. IEEE. 2014, pp. 141–148.

[47] Stefano Nichele, Andreas Giskeødegård, and Gunnar Tufte. “Evolutionary growth of genome representations on artificial cellular organisms with indirect encodings”. In: *Artificial life* (2016).

[48] Simon L Harding, Julian F Miller, and Wolfgang Banzhaf. “Self-modifying cartesian genetic programming”. In: *Cartesian Genetic Programming*. Springer, 2011, pp. 101–124.

[49] Martin A Trefzer et al. “On the advantages of variable length GRNs for the evolution of multicellular developmental systems”. In: *IEEE Transactions on Evolutionary Computation* 17.1 (2013), pp. 100–121.

[50] GB Muller. “Vestigial organs and structures”. In: *Encyclopedia of Evolution* 2 (2002), pp. 1131–1133.

[51] Peter JB Hancock. “An empirical comparison of selection methods in evolutionary algorithms”. In: *AISB Workshop on Evolutionary Computing*. Springer. 1994, pp. 80–94.

[52] Melanie Mitchell, Peter Hraber, and James P Crutchfield. “Revisiting the edge of chaos: Evolving cellular automata to perform computations”. In: *arXiv preprint adap-org/9303003* (1993).

[53] Melanie Mitchell, James P Crutchfield, Rajarshi Das, et al. “Evolving cellular automata with genetic algorithms: A review of recent work”. In: *Proceedings of the First International Conference on Evolutionary Computation and Its Applications (EvCA’96)*. Moscow. 1996.

[54] Aksel Ethembabaoglu, Shimon Whiteson, et al. “Automatic feature selection using FS-NEAT”. In: (2008).

[55] Mark EJ Newman. “The structure and function of complex networks”. In: *SIAM review* 45.2 (2003), pp. 167–256.

[56] Gina Maira Barbosa de Oliveira, Pedro PB de Oliveira, and Nizam Omar. “Guidelines for dynamics-based parameterization of one-dimensional cellular automata rule spaces”. In: *Complexity* 6.2 (2000), pp. 63–71.

[57] Itay Hubara et al. “Binarized neural networks”. In: *Advances in Neural Information Processing Systems*. 2016, pp. 4107–4115.

Appendix A

CA-NEAT Parameters

CA-NEAT has a large number of parameters that affect the outcome of the experiments. This appendix lists some notable ones that would be needed to reproduce the results in this thesis consistently.

Listing A.1: Parameters that define the initial population. Notably, for each generated individual, the ratio of connectivity is determined individually to be a value between 0.5 and 1.0.

```
initial_hidden_nodes = 0
initial_connection = 'partial'
connection_fraction = lambda *args: 0.5 * (1 + random.random())
```

Listing A.2: Parameters that have to do with selection and speciation

```
survival_threshold = 0.2
stagnation_limit = 15
compatibility_threshold = 3.0
excess_coefficient = 1.0
disjoint_coefficient = 1.0
weight_coefficient = 0.5
```

Listing A.3: Parameters that have to do with mutation

```
max_weight = 30
min_weight = −30
weight_stdev = 1.0
prob_add_conn = 0.5
prob_add_node = 0.5
prob_delete_conn = 0.25
prob_delete_node = 0.25
prob_mutate_bias = 0.8
bias_mutation_power = 0.5
prob_mutate_response = 0.8
response_mutation_power = 0.5
prob_mutate_weight = 0.8
prob_replace_weight = 0.1
weight_mutation_power = 0.5
prob_mutate_activation = 0.002
prob_toggle_link = 0.01
```

Appendix B

CPPN Visualizations

This appendix contains visualizations of CPPN structures found by CA-NEAT. They range from small networks that a human can reason about with some effort, to huge networks that are difficult if not impossible to reason about. The intent of this appendix is not to have the reader try to parse the structures, but to showcase the diversity of the populations evolved by NEAT.

The visualizations are created programatically from the genome data structures using Graphviz[1]. All the networks shown are pruned of vestigial structures and disabled connections.

[1] `http://www.graphviz.org/`

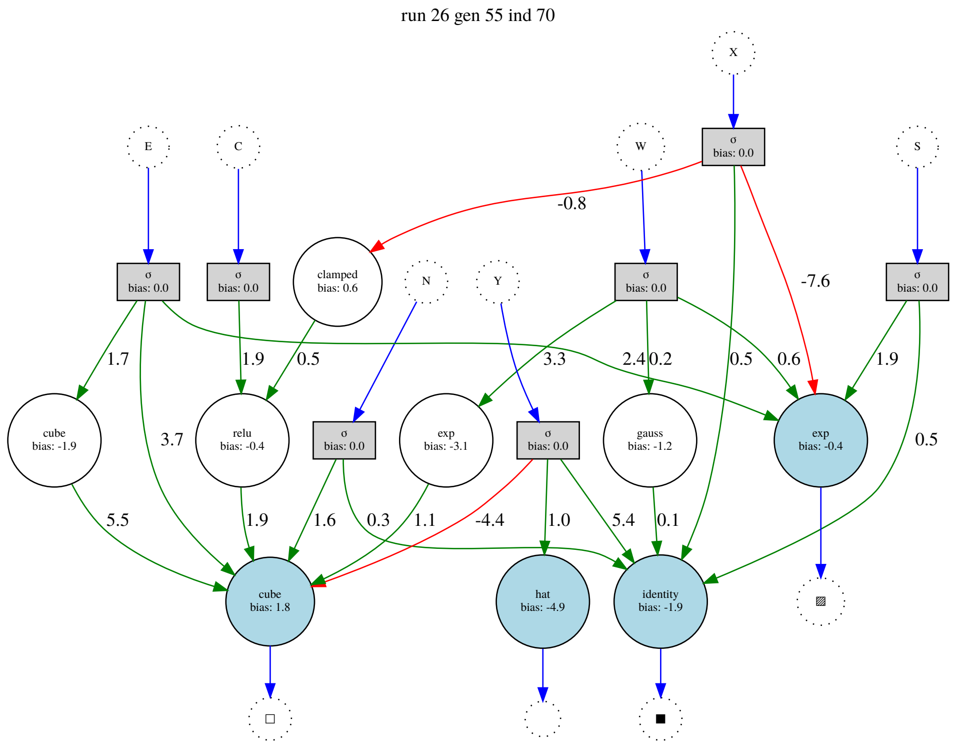


Figure B.1: A network evolved for the ”Nordic” morphogenesis with coordinate information task.

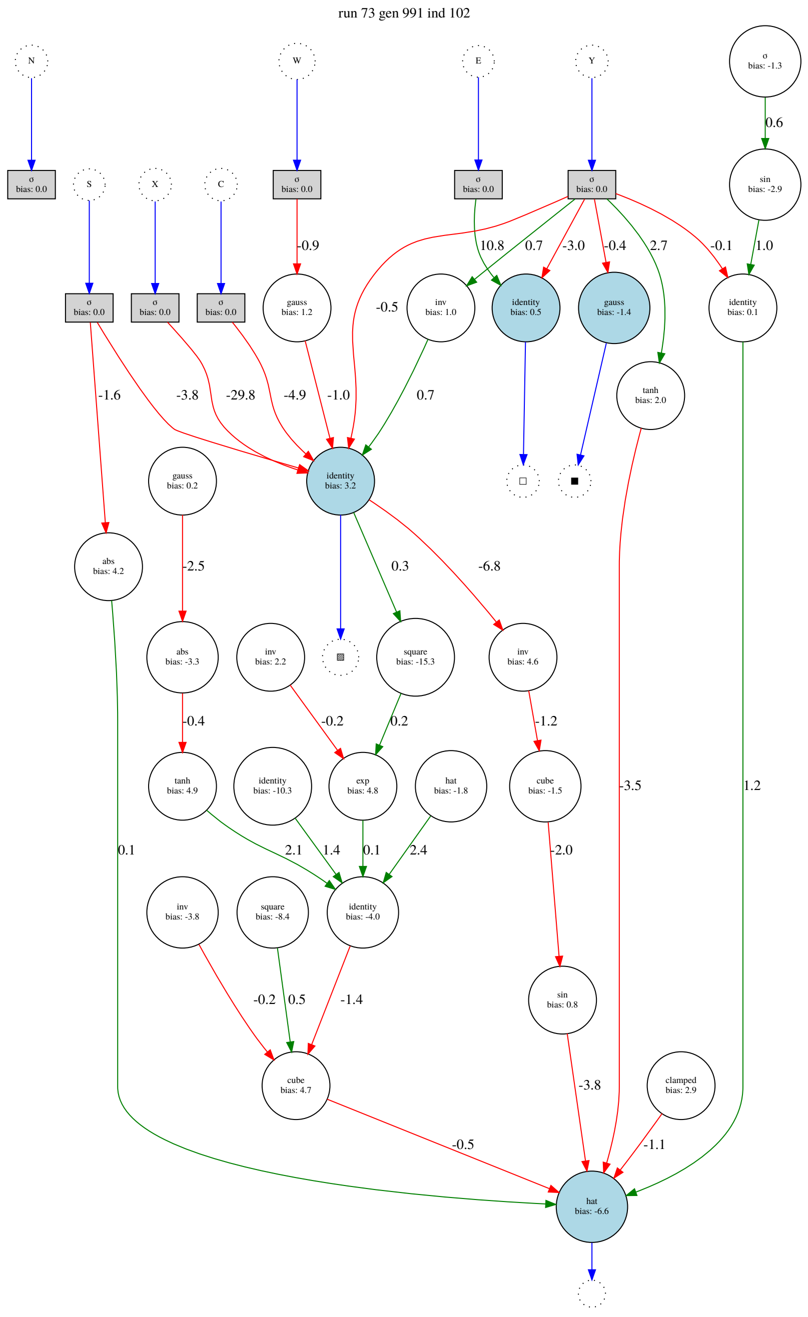


Figure B.2: A network evolved for the "Nordic" morphogenesis with coordinate information task.

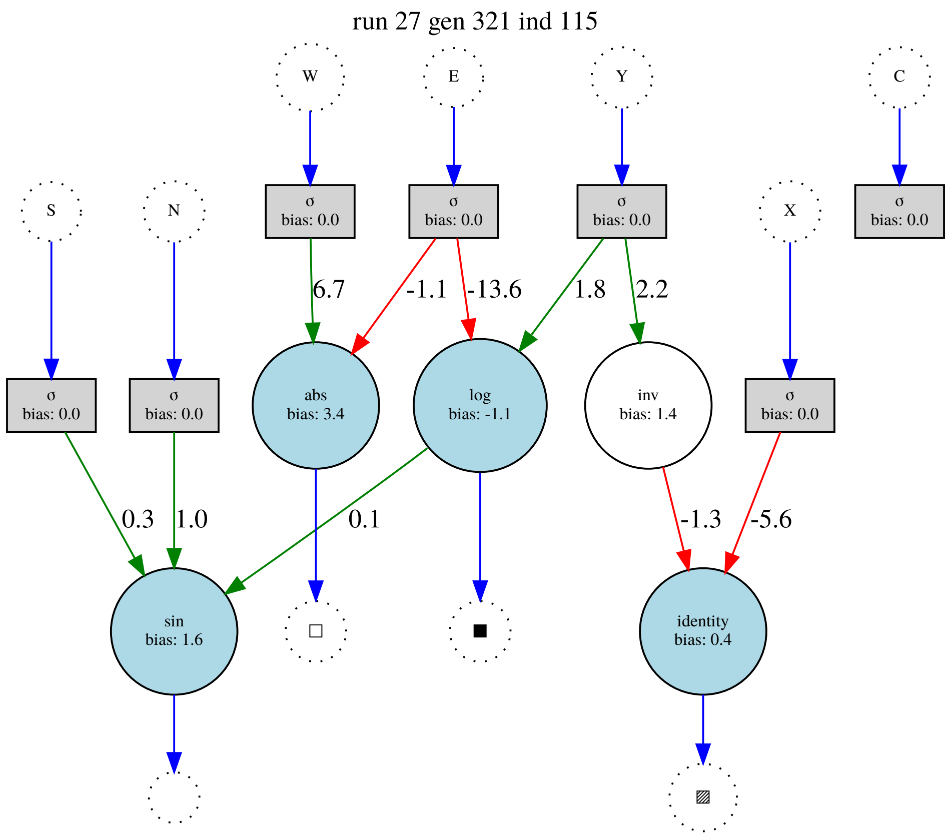


Figure B.3: A network evolved for the ”Nordic” morphogenesis with coordinate information task.

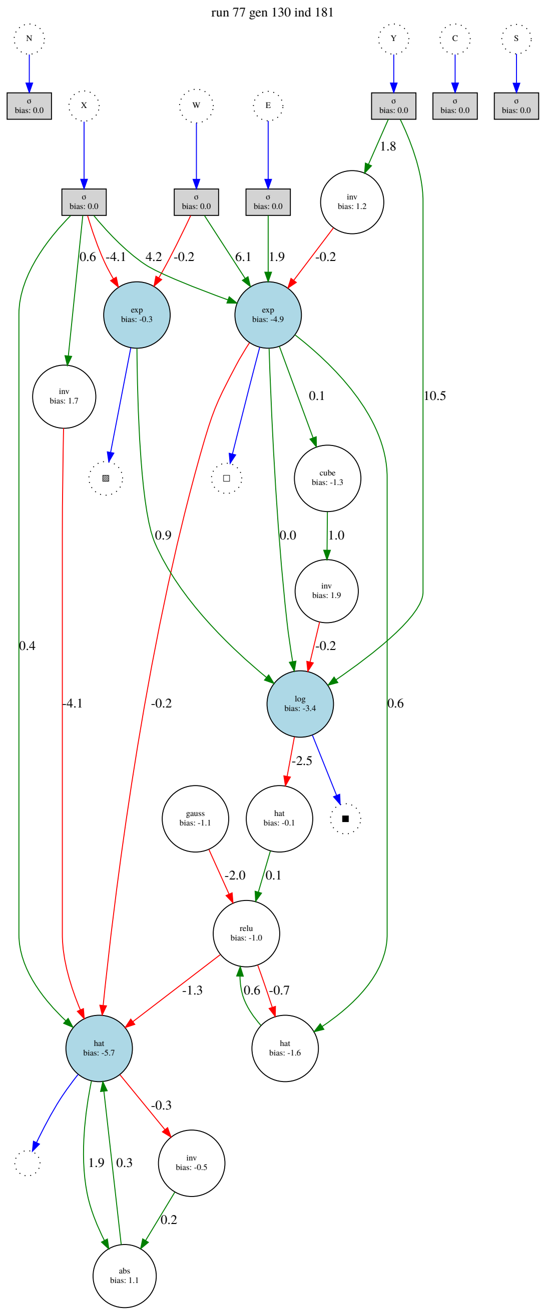


Figure B.4: A network evolved for the ”Nordic” morphogenesis with coordinate information task.

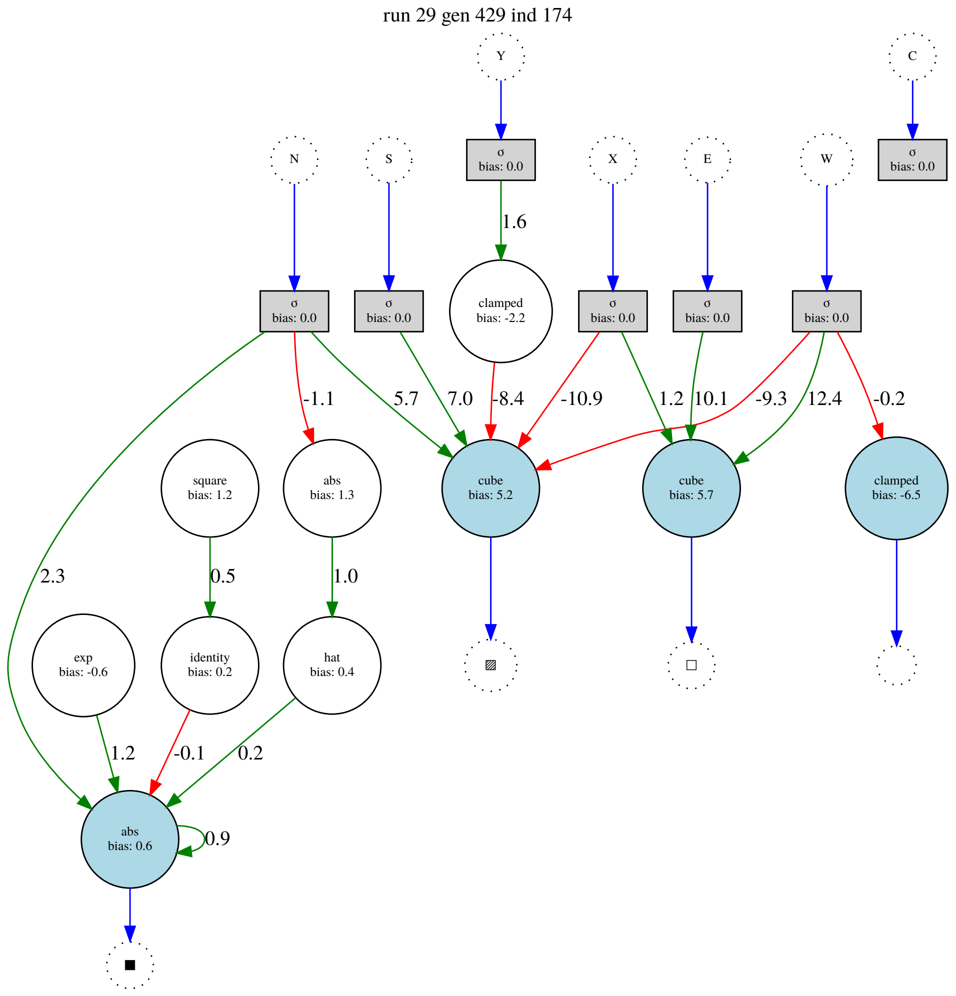


Figure B.5: A network evolved for the ”Nordic” morphogenesis with coordinate information task.

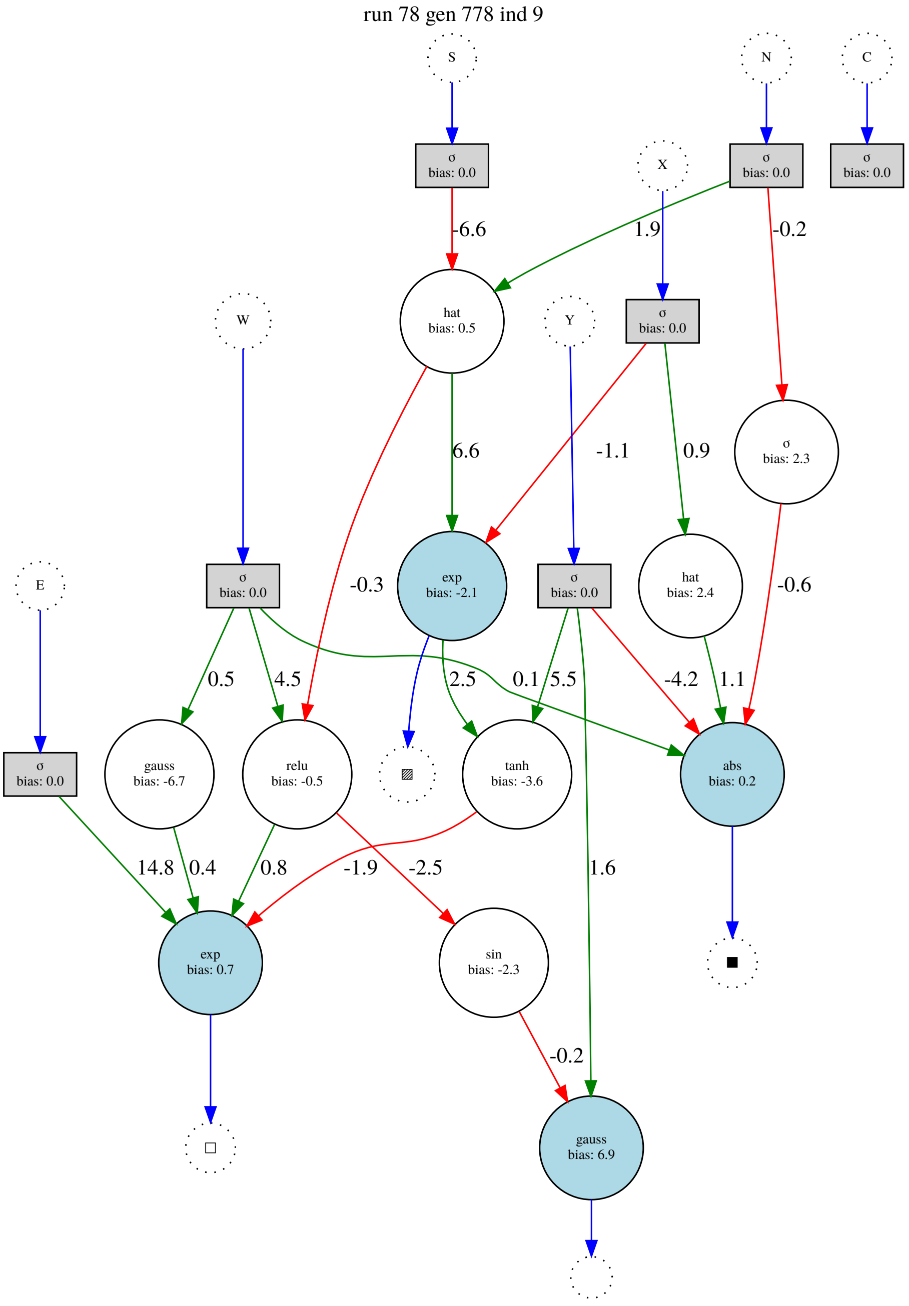


Figure B.6: A network evolved for the ”Nordic” morphogenesis with coordinate information task.

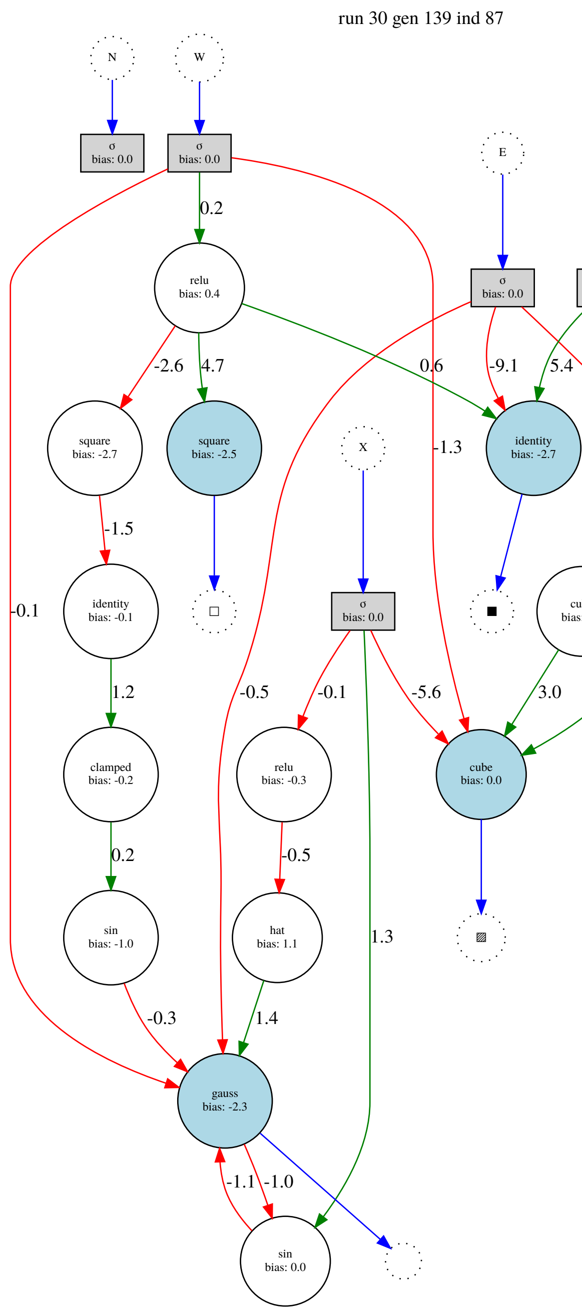


Figure B.7: A network evolved for the ”Nordic” morphogenesis with coordinate information task.

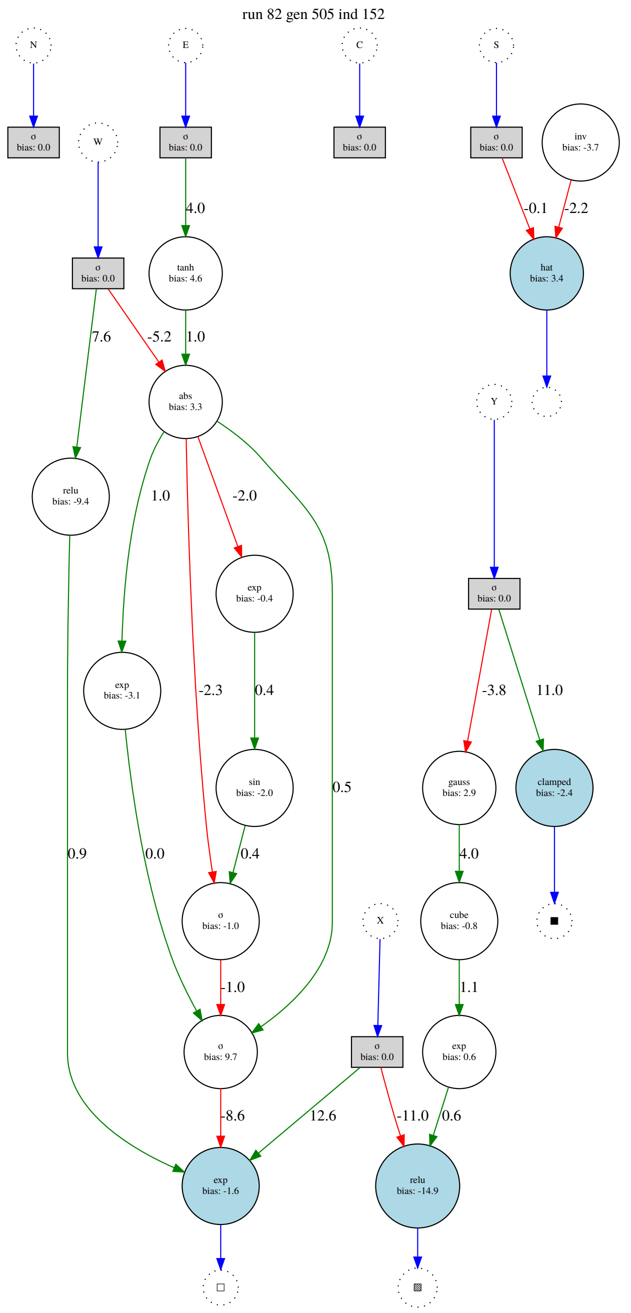


Figure B.8: A network evolved for the ”Nordic” morphogenesis with coordinate information task.

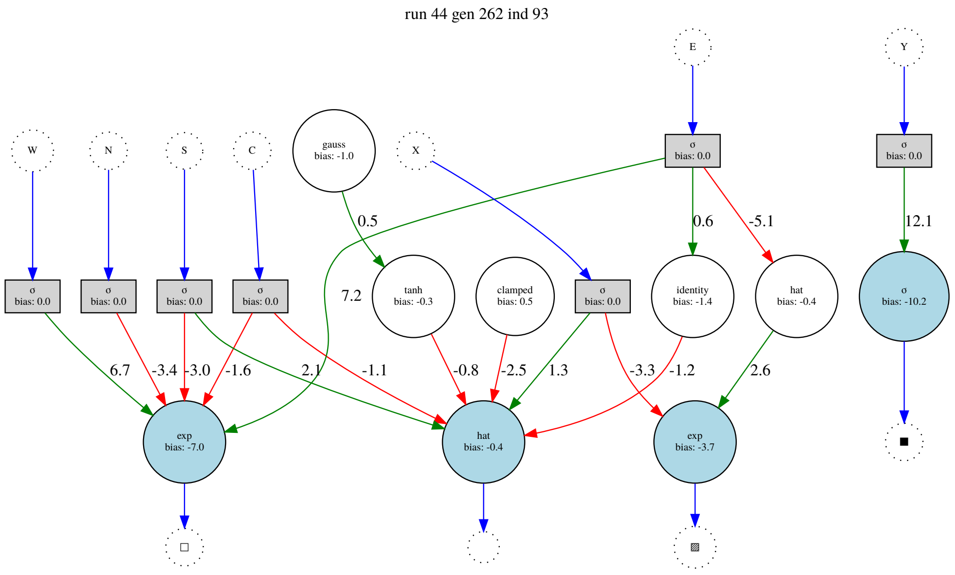


Figure B.9: A network evolved for the "Nordic" morphogenesis with coordinate information task.

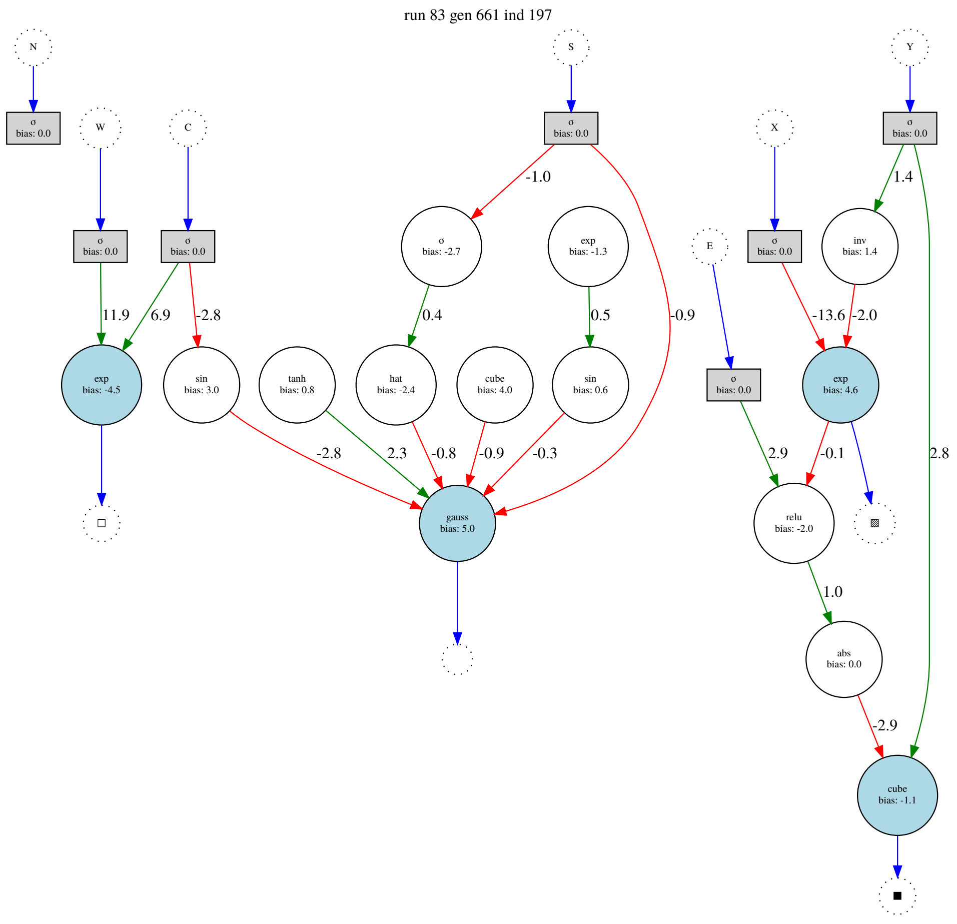


Figure B.10: A network evolved for the ”Nordic” morphogenesis with coordinate information task.

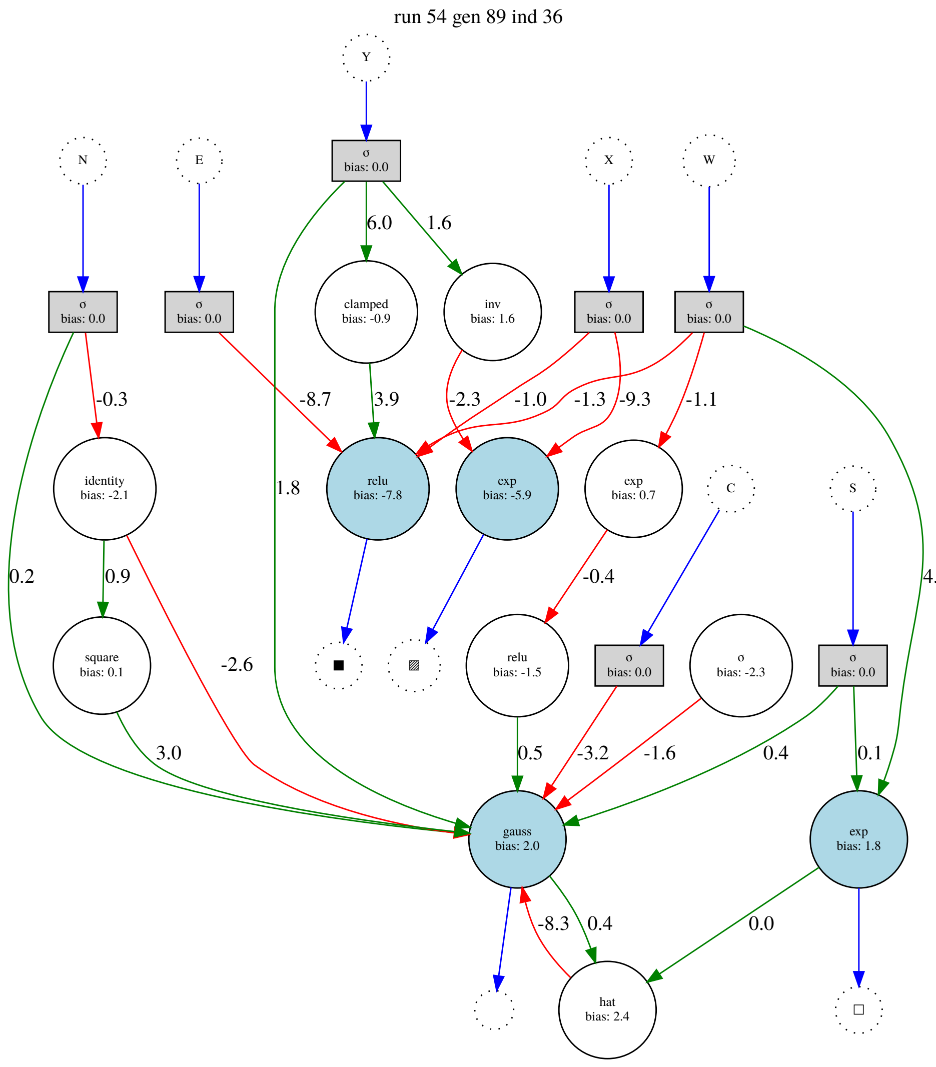


Figure B.11: A network evolved for the ”Nordic” morphogenesis with coordinate information task.

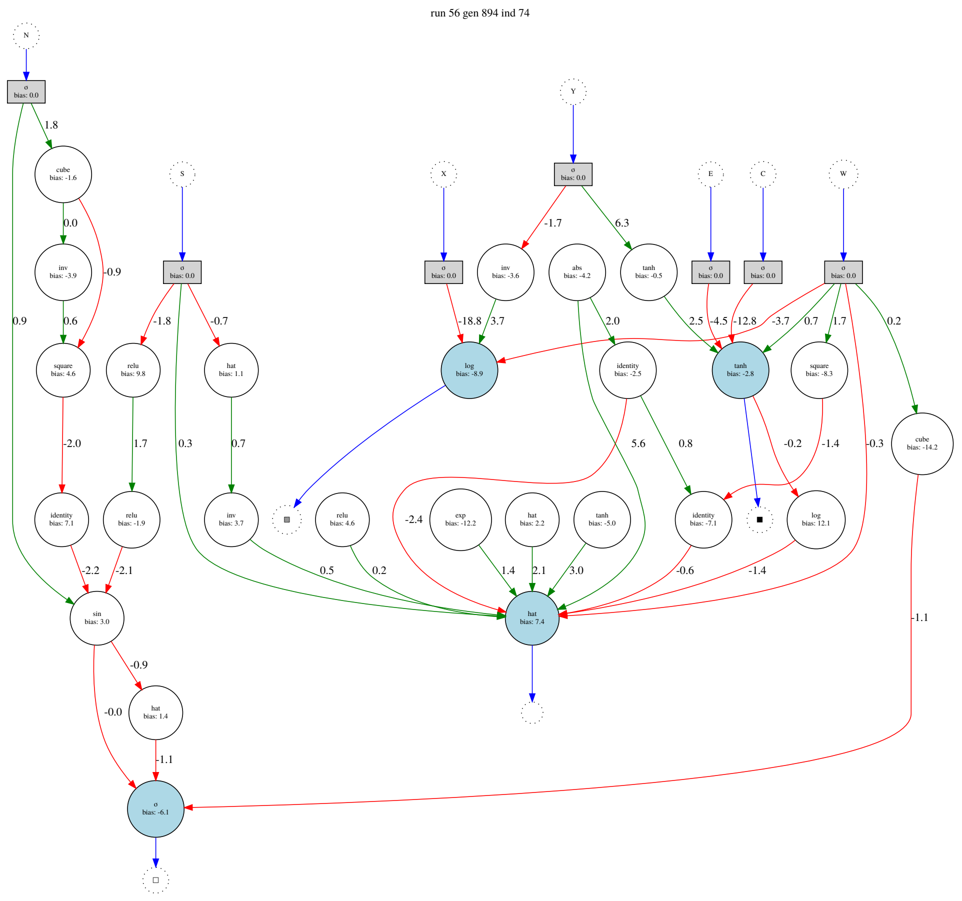


Figure B.12: A network evolved for the ”Nordic” morphogenesis with coordinate information task.

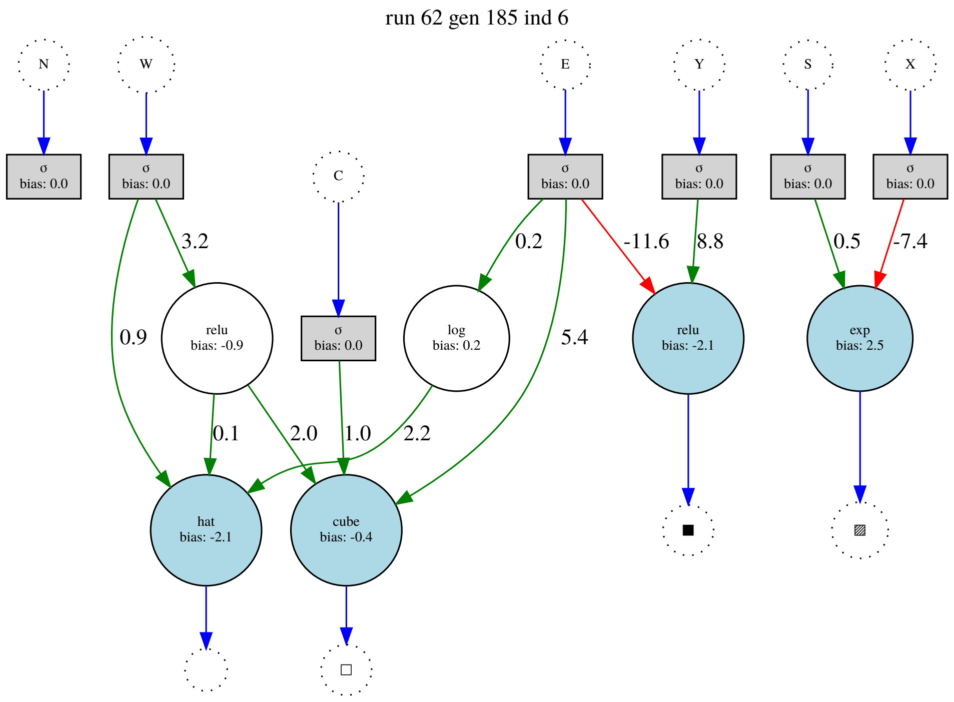


Figure B.13: A network evolved for the "Nordic" morphogenesis with coordinate information task.

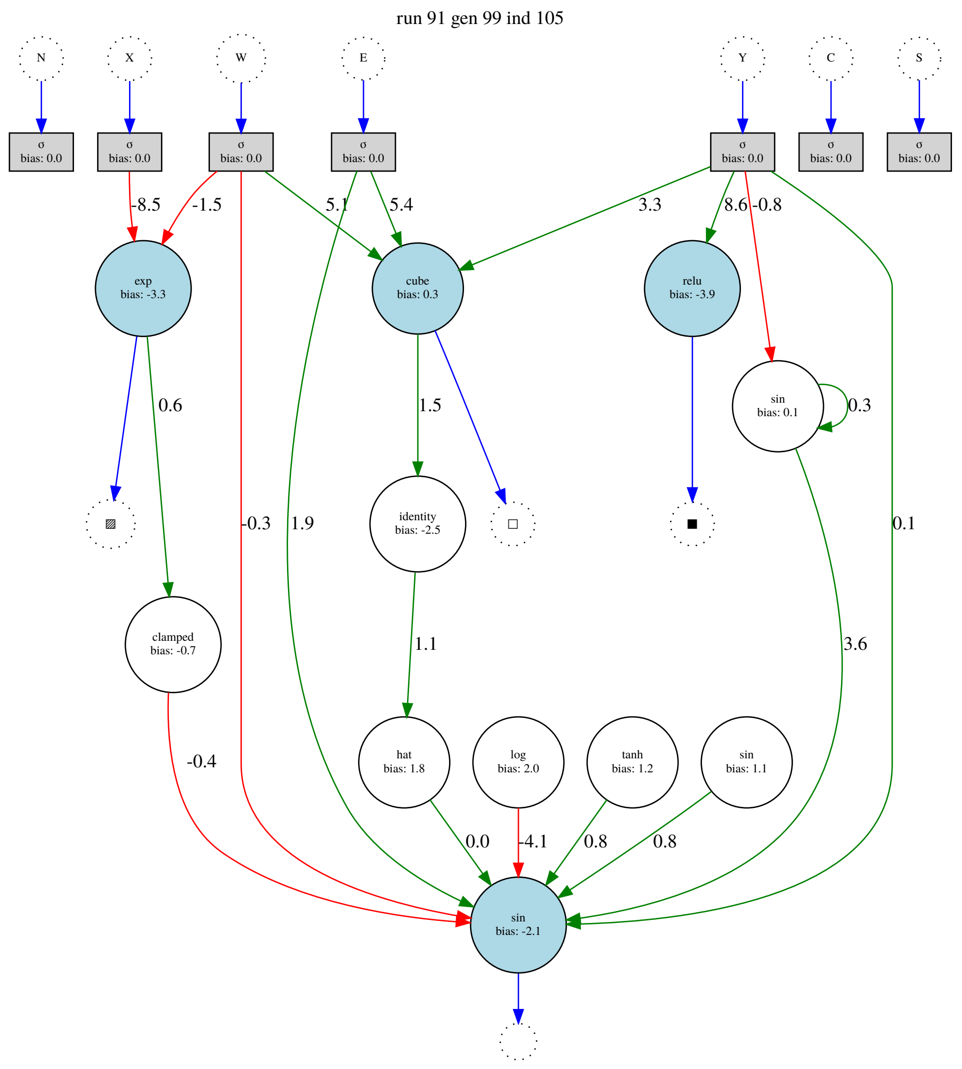


Figure B.14: A network evolved for the ”Nordic” morphogenesis with coordinate information task.

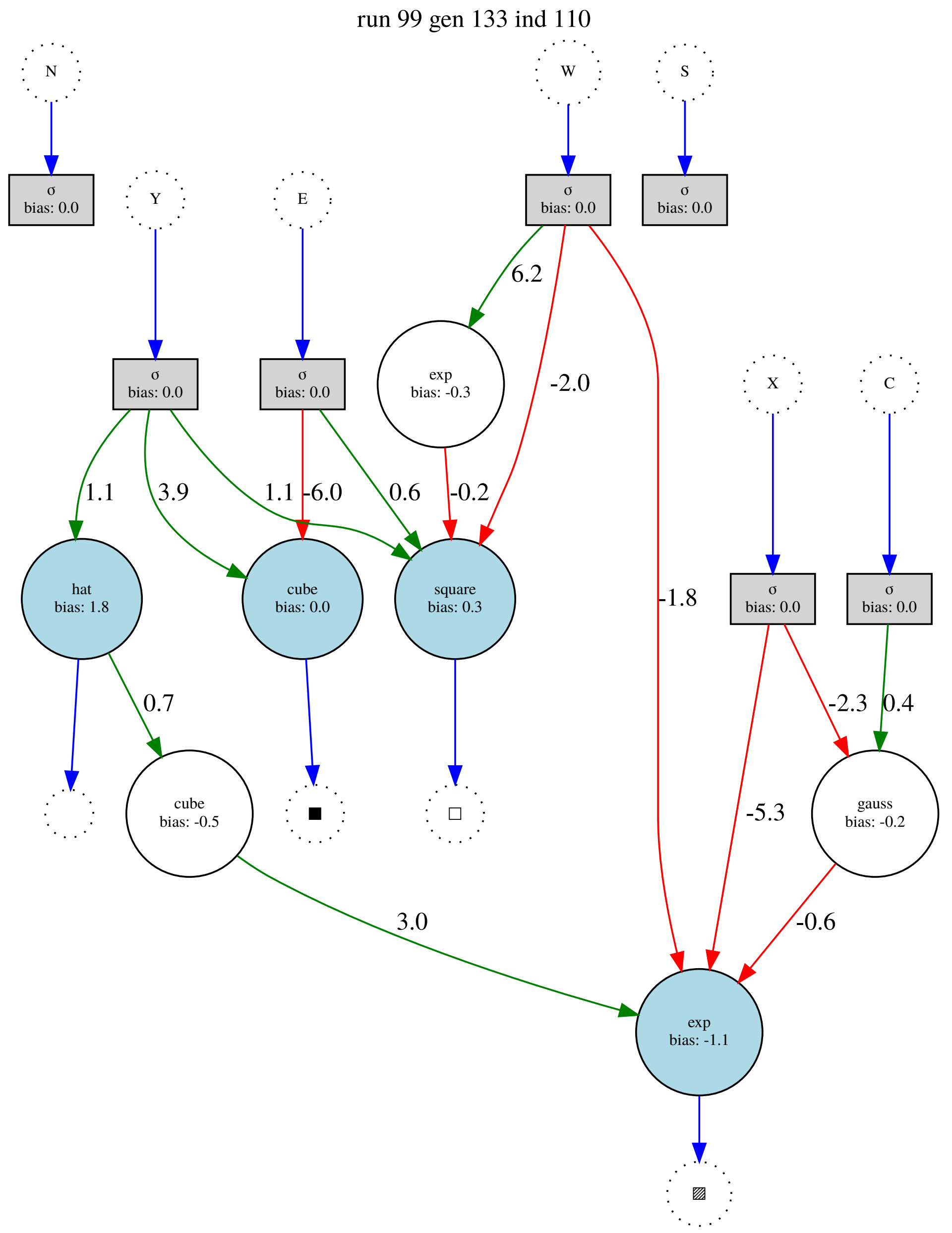


Figure B.15: A network evolved for the ”Nordic” morphogenesis with coordinate information task.

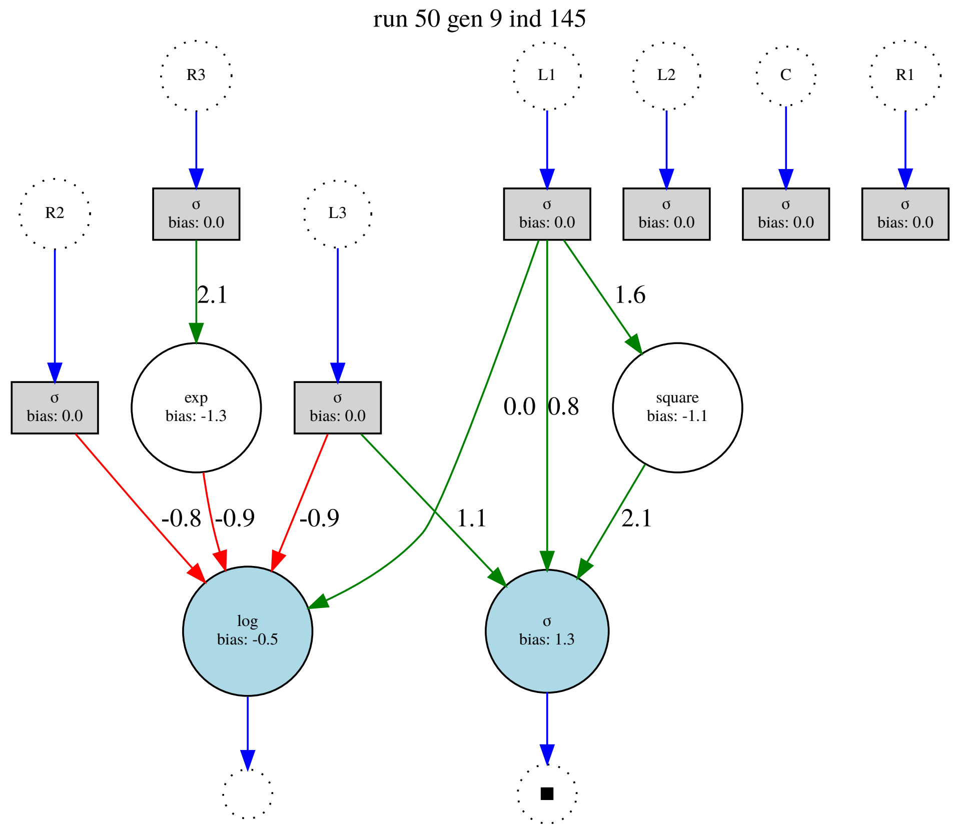


Figure B.16: A network evolved for the majority problem. This one uses only 4/7 inputs, yet is able to solve all the ICs in the training set.

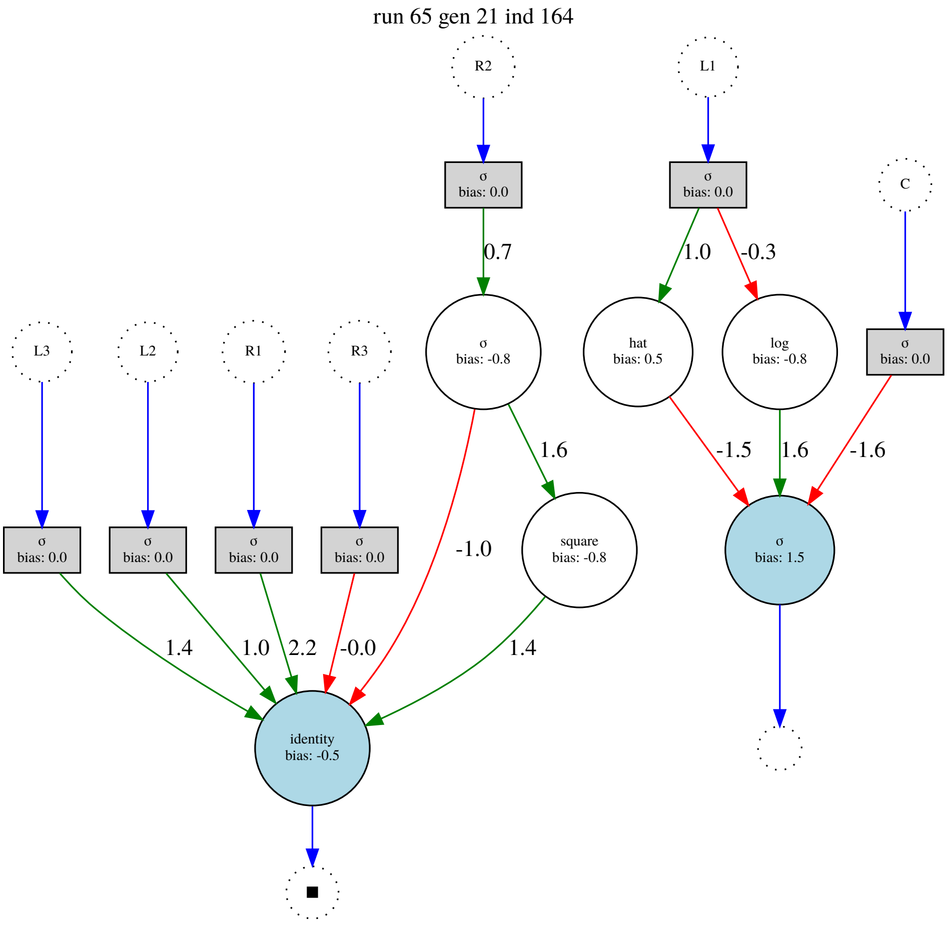


Figure B.17: A network evolved for the majority problem. This one has separated into two sub-networks, each deciding the activation value of one of the outputs.

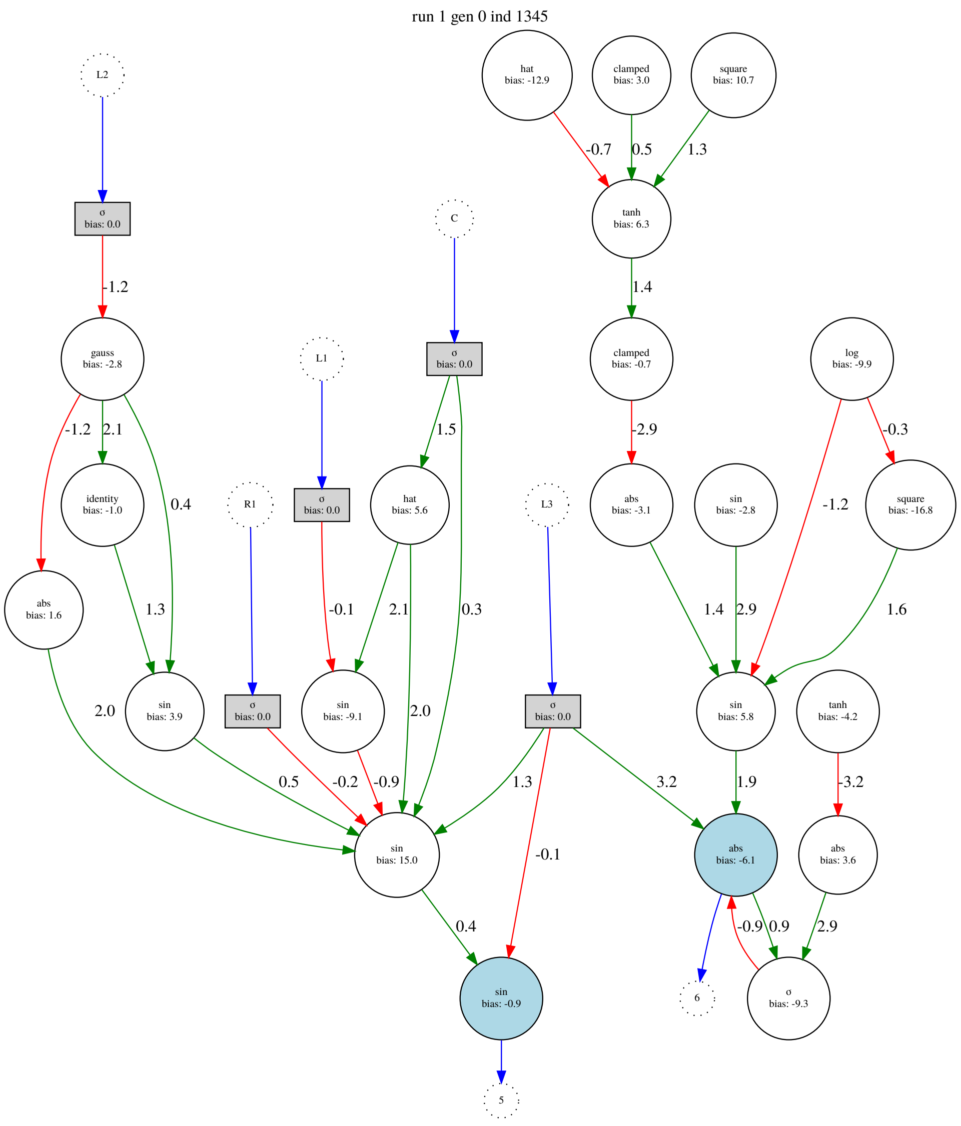


Figure B.18: A network evolved with novelty search ($N = 5$, $K = 2$).

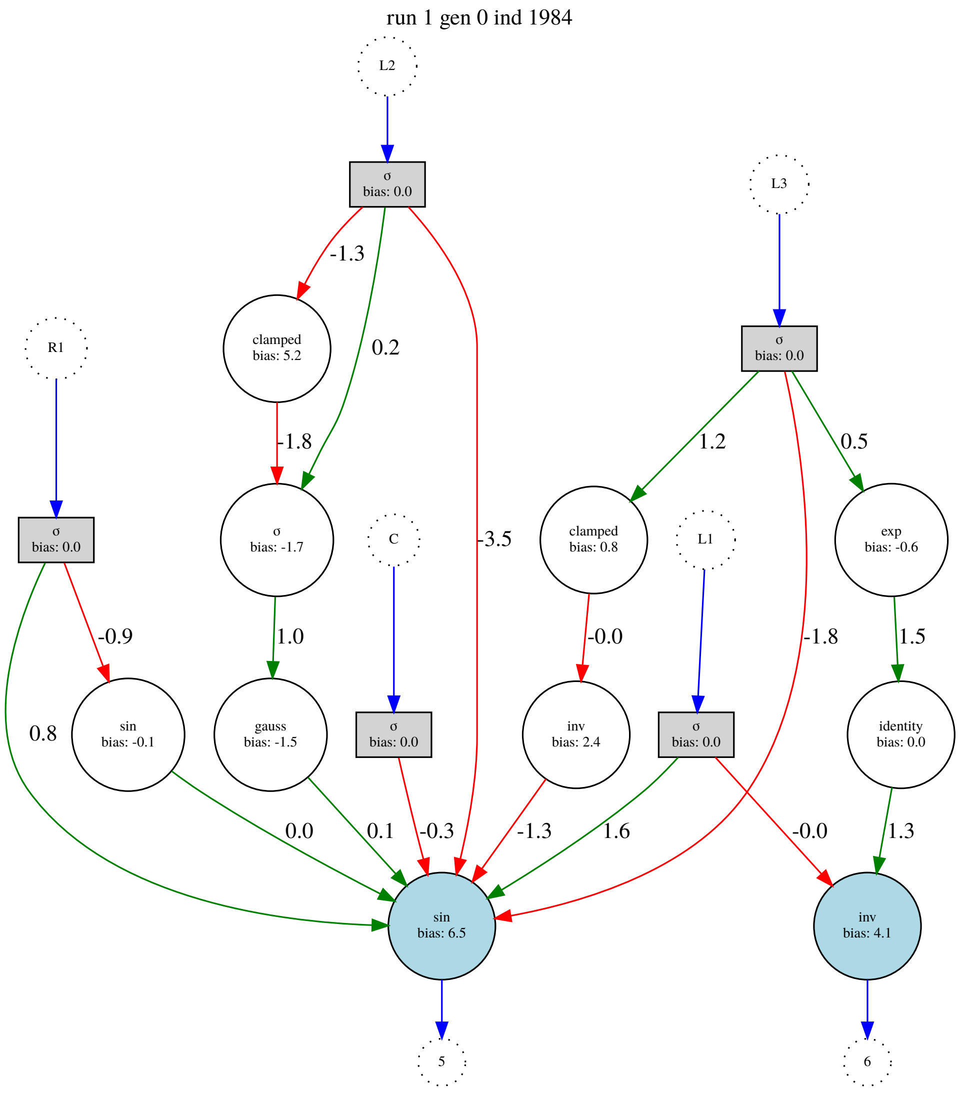


Figure B.19: A network evolved with novelty search ($N = 5$, $K = 2$).

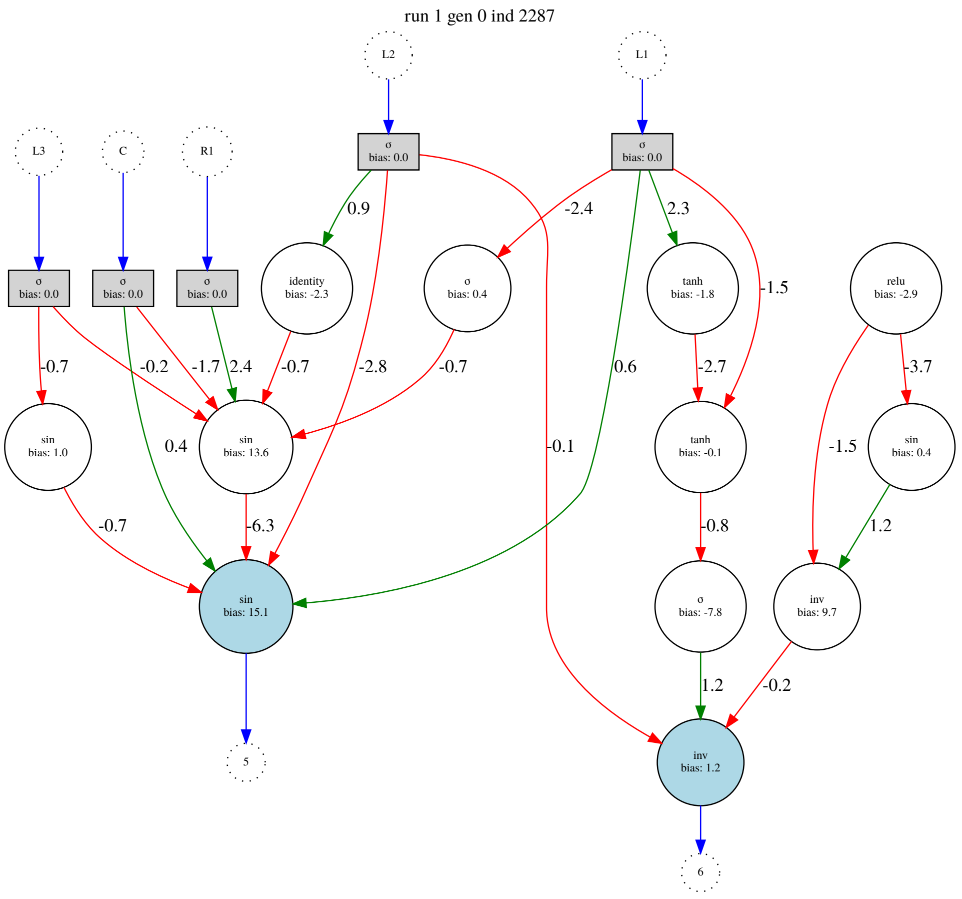


Figure B.20: A network evolved with novelty search ($N = 5$, $K = 2$).

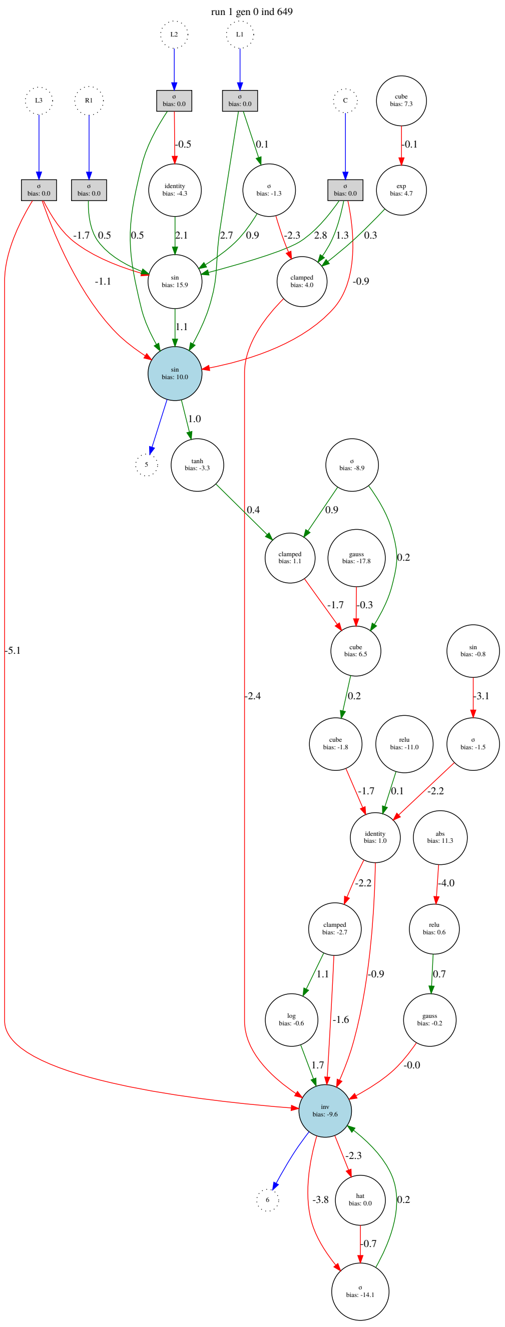


Figure B.21: A network evolved with novelty search ($N = 5$, $K = 2$).

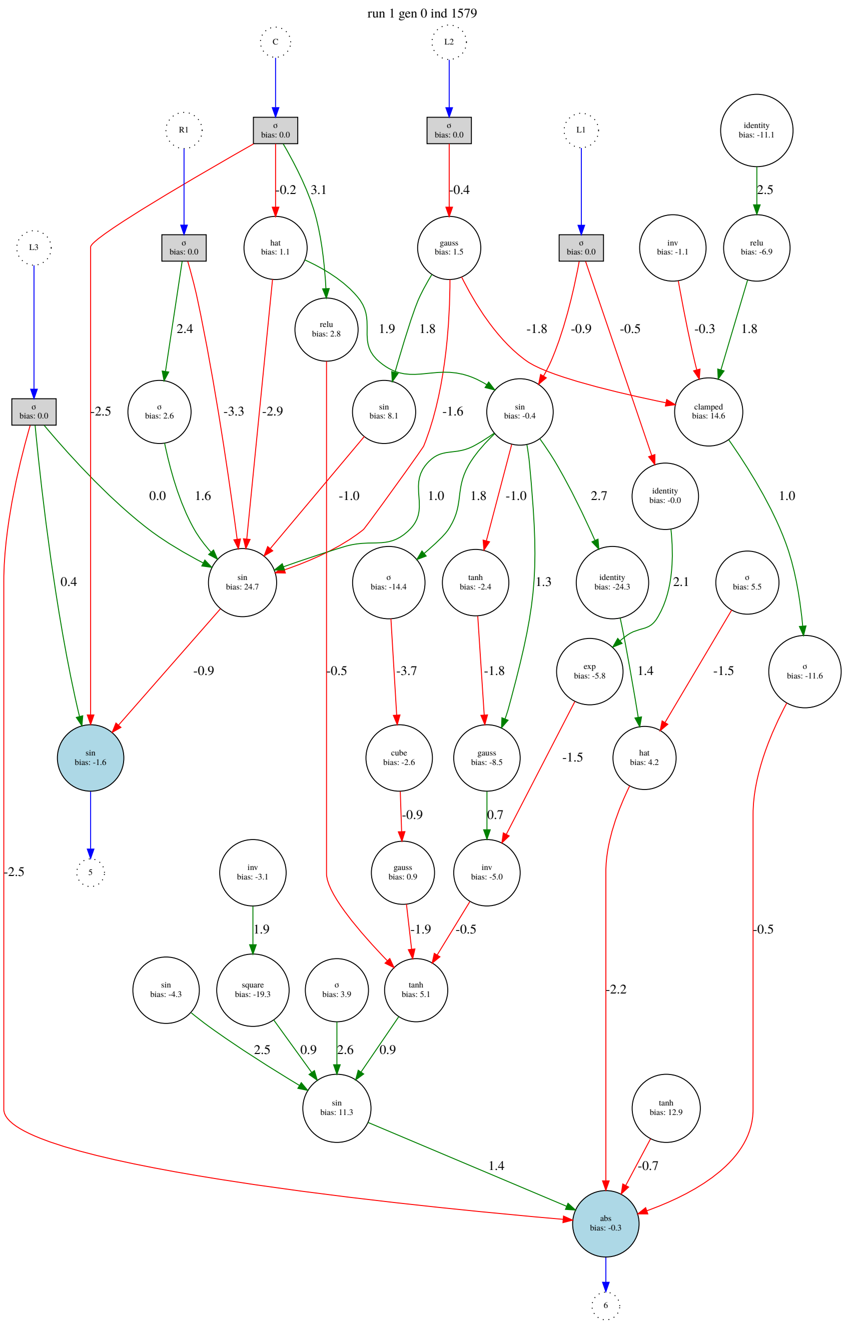


Figure B.22: A network evolved with novelty search ($N = 5$, $K = 2$).

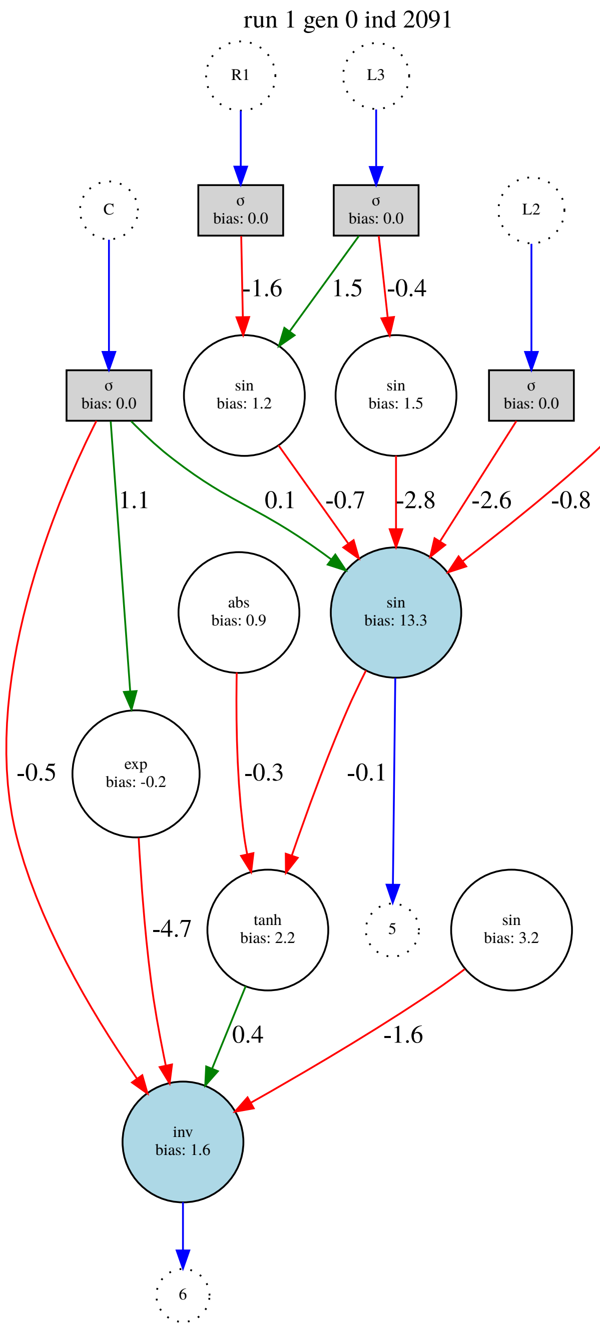


Figure B.23: A network evolved with novelty search ($N = 5, K = 2$).

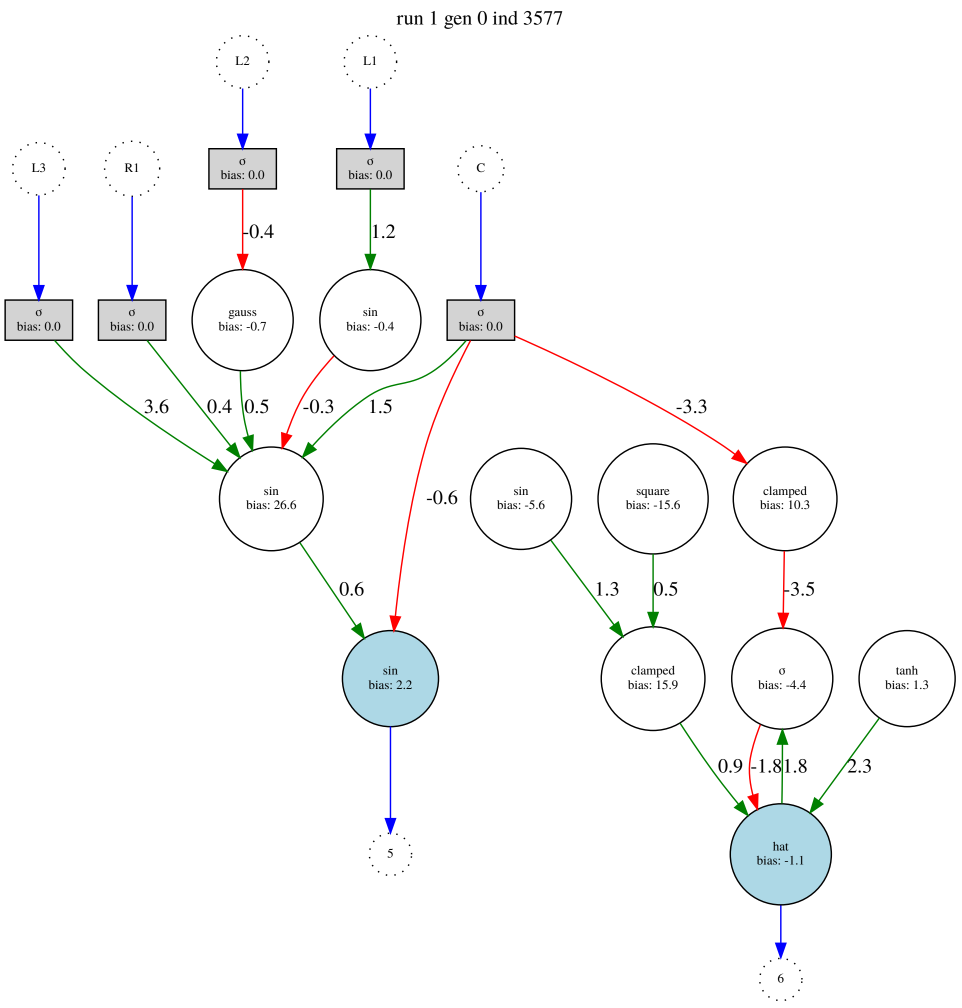


Figure B.24: A network evolved with novelty search ($N = 5$, $K = 2$).

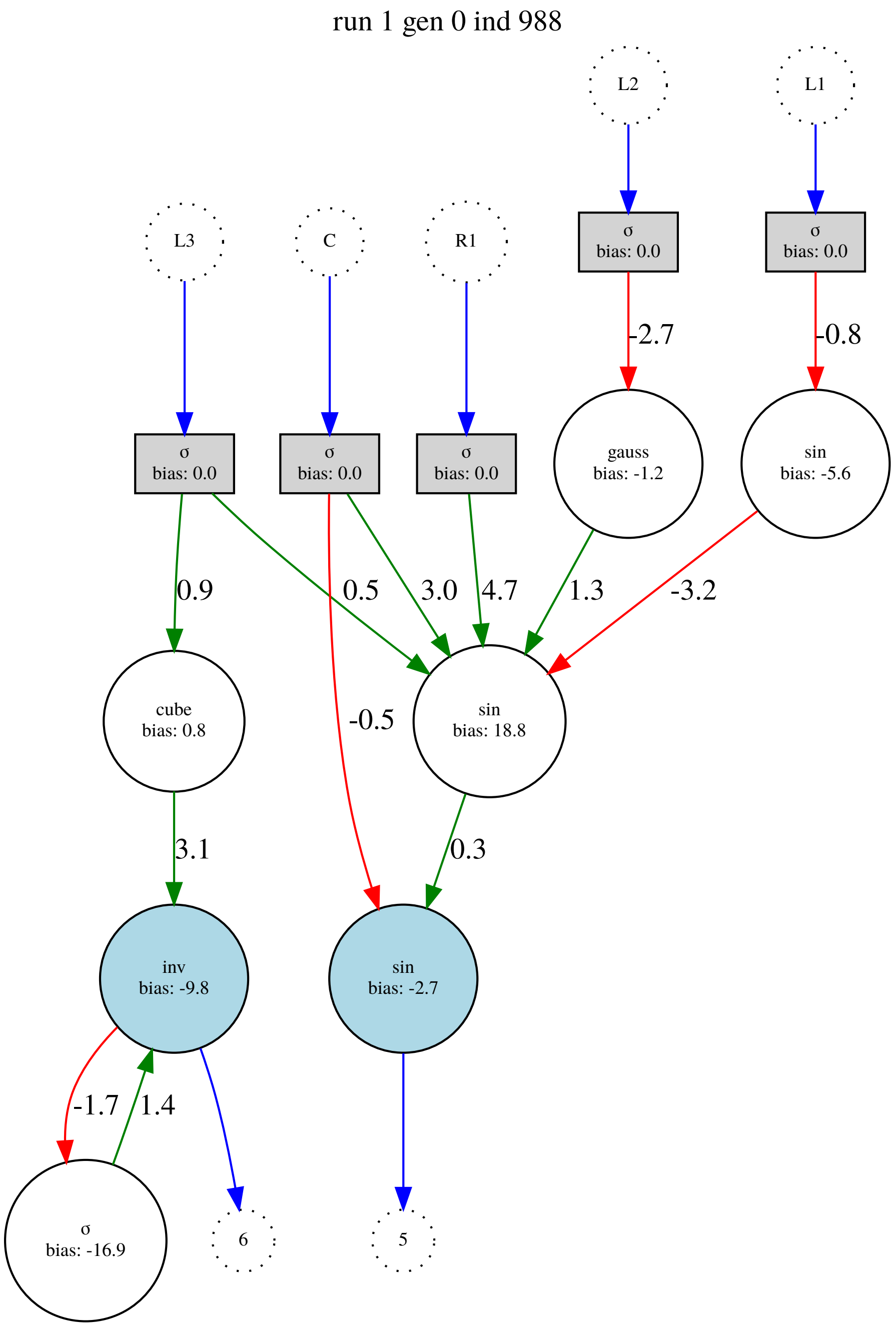


Figure B.25: A network evolved with novelty search ($N = 5, K = 2$).

Appendix C

Sample Solutions

This appendix contains some visualizations of hand-picked example solutions that were found.

In morphogenesis cases where a previous state is revisited (a cycle is found), the number above the last state shown indicates which previous state it is equal to.

For the morphogenesis examples up to 30 states are shown, including states after the target pattern is seen. For replication examples, the visualizations stop after an optimal (3 copies) solution is seen.

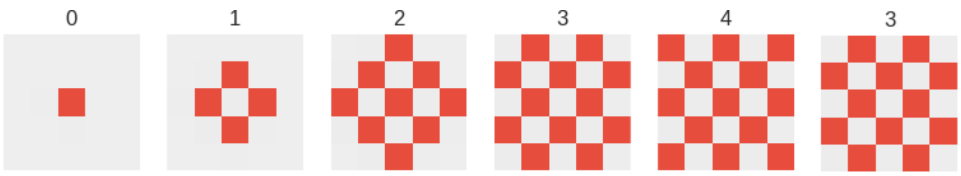


Figure C.1: A solution to the ”Mosaic” morphogenesis where the CA finds a two-step cycle that includes the target pattern.

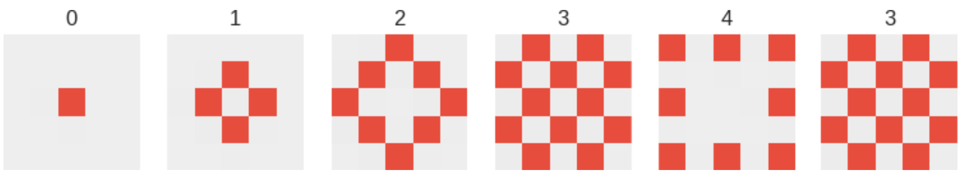


Figure C.2: Another two-state cycle that includes the target state.

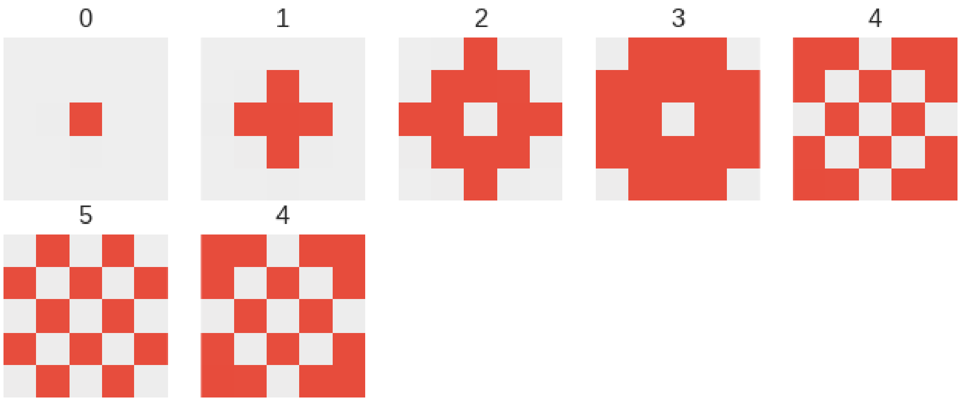


Figure C.3: Another two-state cycle that includes the target state.

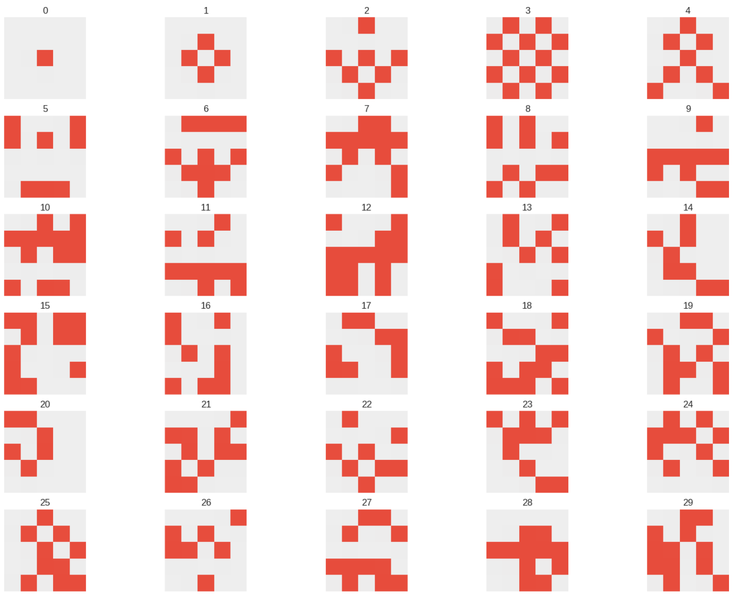


Figure C.4: A solution where the target pattern can be seen early, but when the CA continues it becomes chaotic.

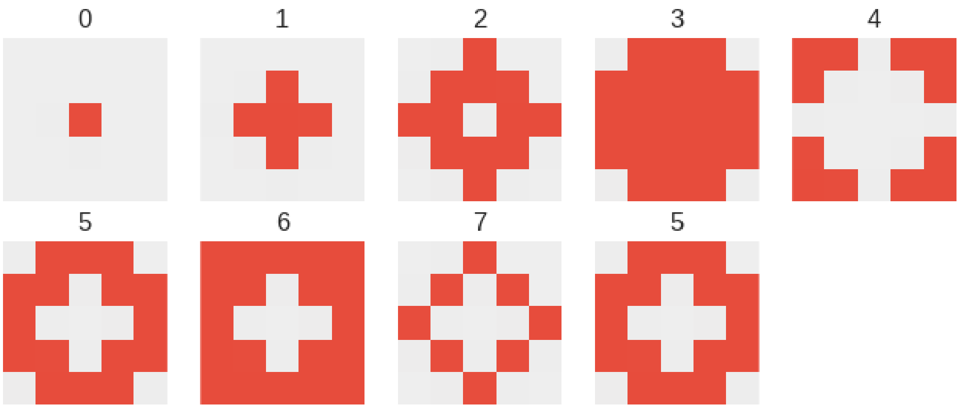


Figure C.5: A solution to the ”Swiss” morphogenesis where the CA goes in to a three-state cycle that contains the target pattern.

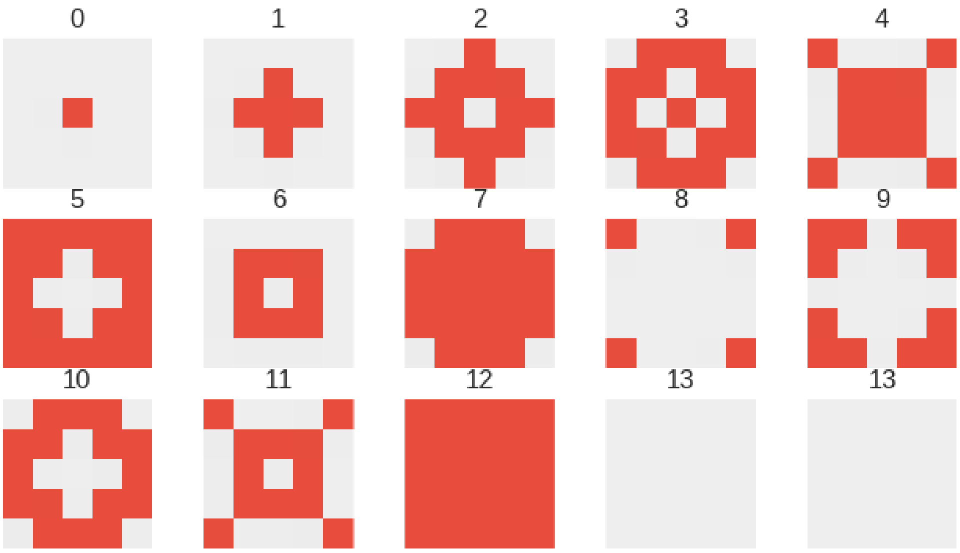


Figure C.6: Another solution, where after visiting the target state the CA eventually annihilates itself completely.

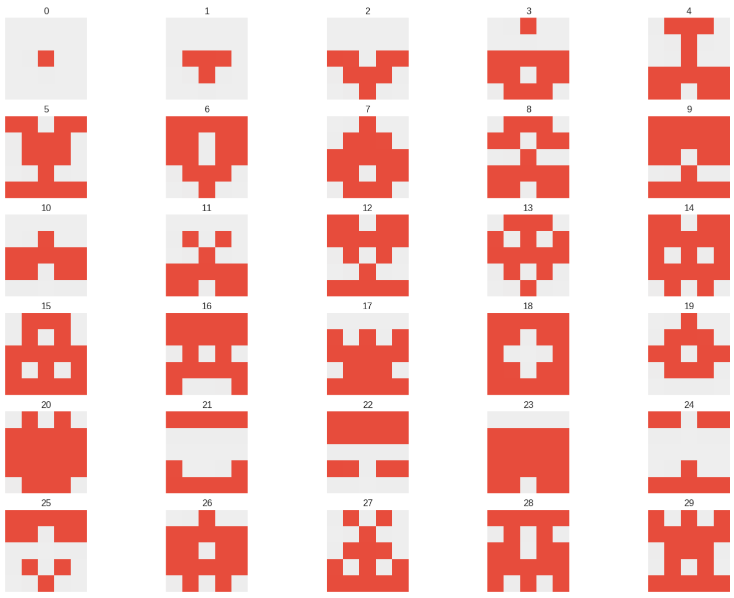


Figure C.7: A solution where the CA develops with one of two possible symmetries, but still finds the solution that has two symmetries.

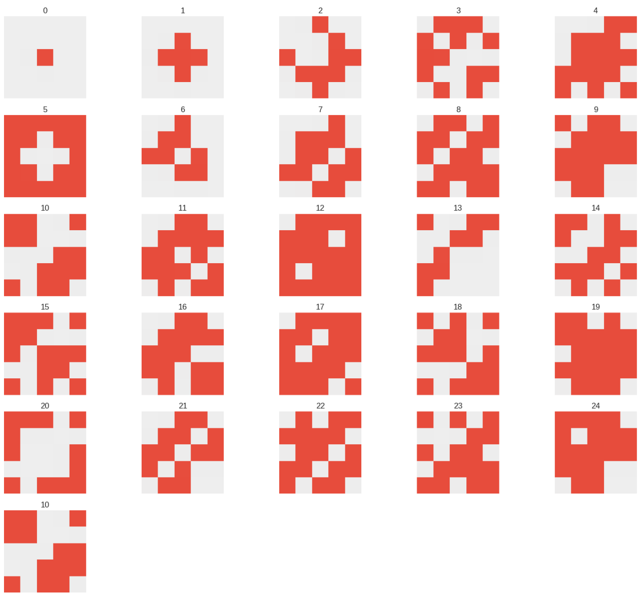


Figure C.8: A solution that quickly finds the target pattern, but then keeps developing and finds a longer cycle that does not include the target pattern.

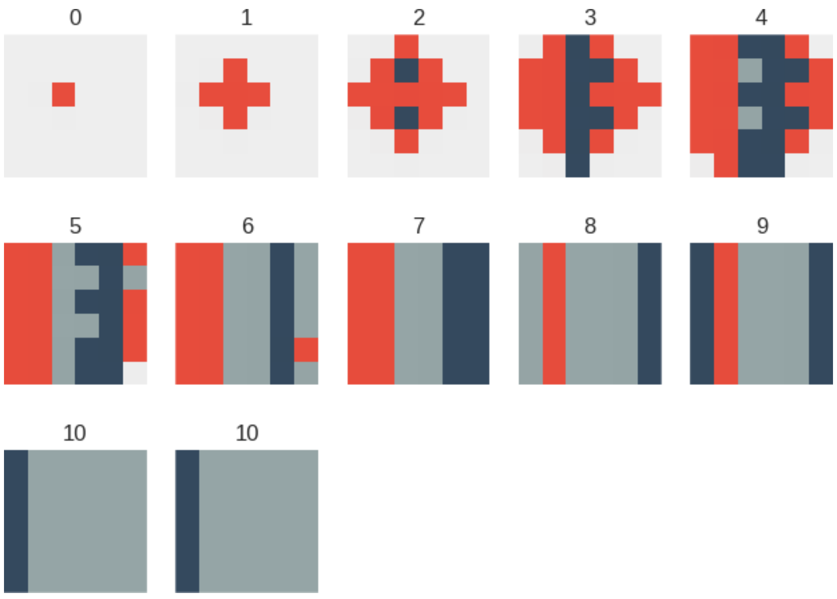


Figure C.9: Another solution which visits the target pattern, but stabilizes in a point attractor not equal to the target.

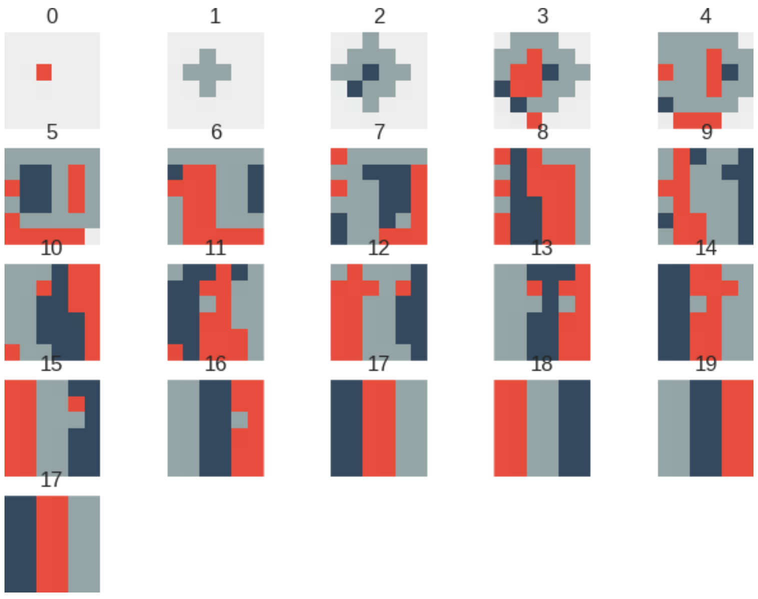


Figure C.10: A solution which stabilizes into a ”Tricolor” pattern and then cycles through different permutation of the colors.

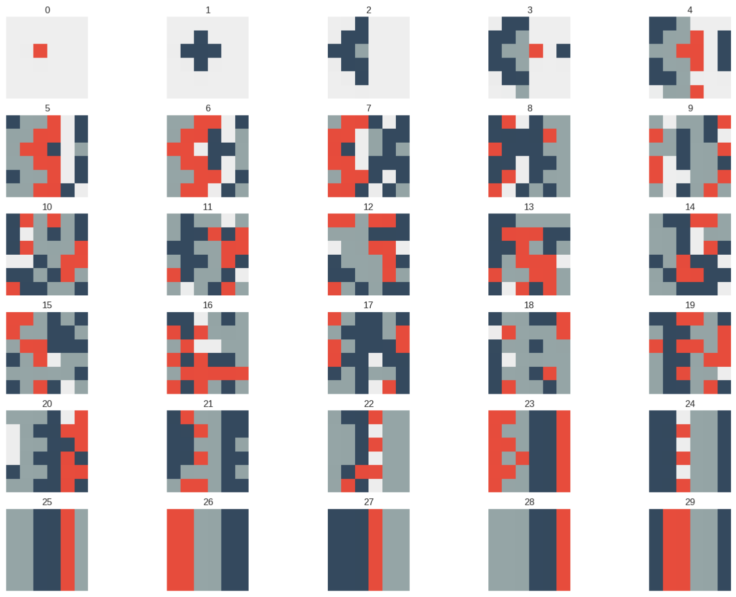


Figure C.11: A solution that looks very chaotic for a very long time, but eventually manages to recover and create patterns with vertical lines.

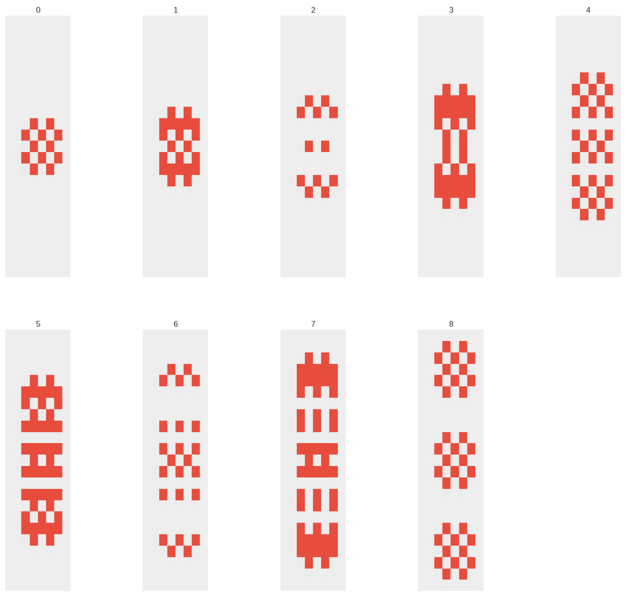


Figure C.12: A solution to the "Mosaic" replication, where expansion happens only along one axis.

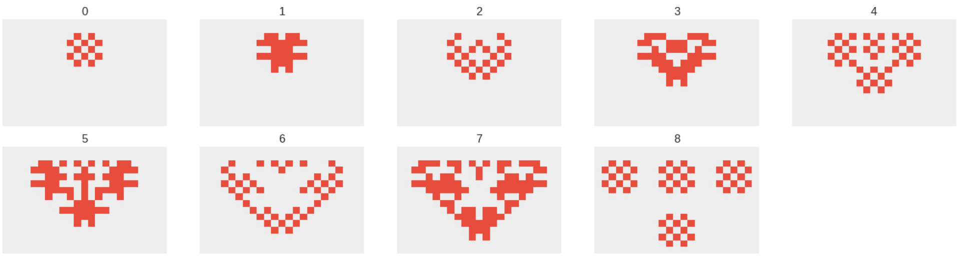


Figure C.13: Another solution, where replication is done in three directions while retaining the original.

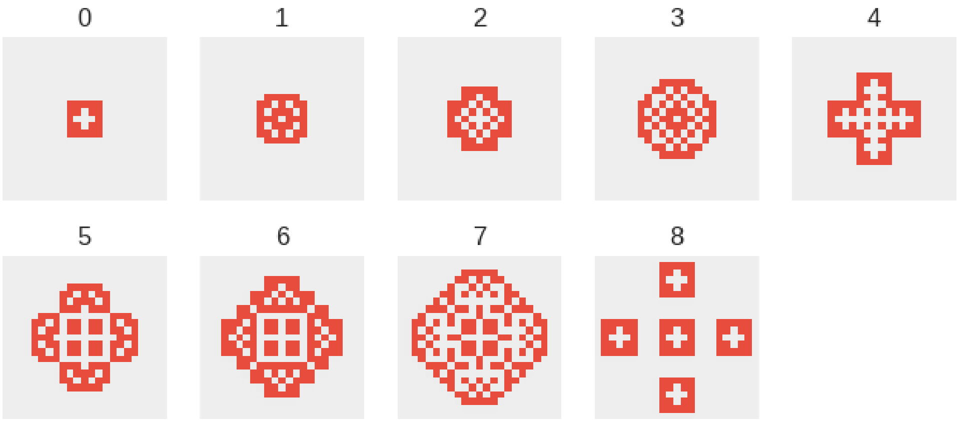


Figure C.14: A solution to the ”Swiss” replication that expands symmetrically along both axes.

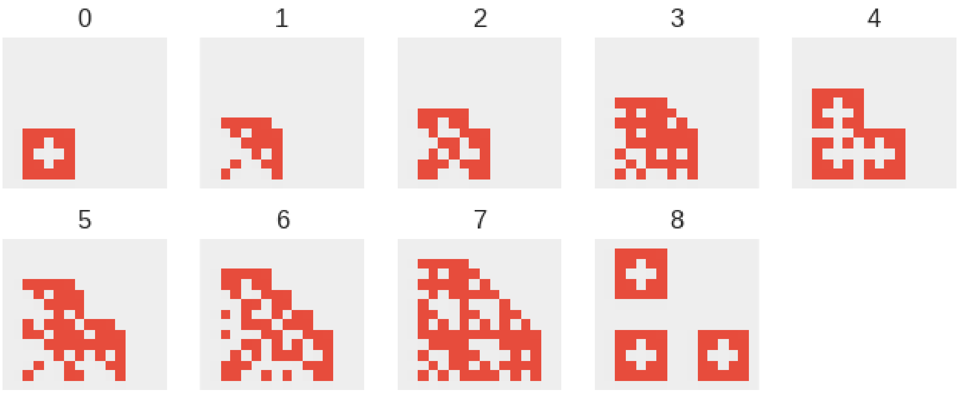


Figure C.15: A solution which expands in two orthogonal directions.

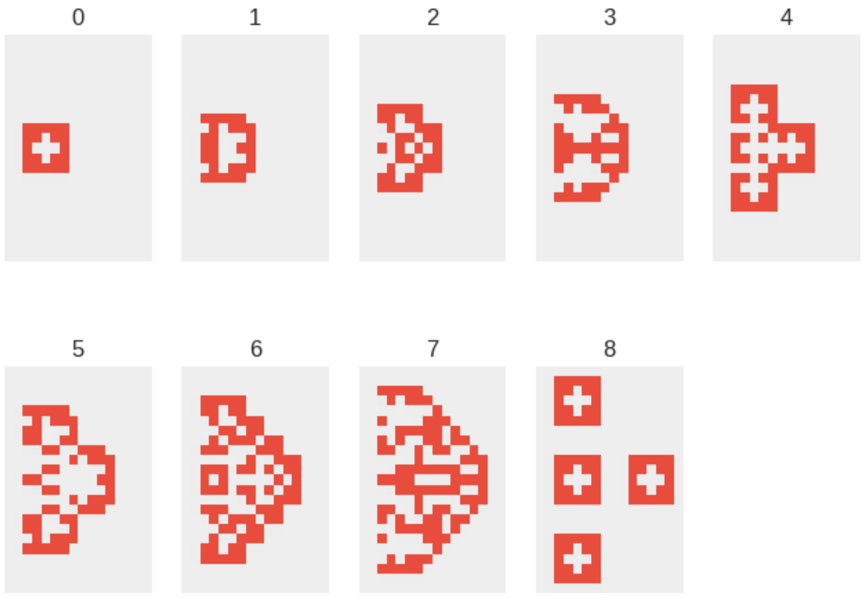


Figure C.16: A solution which is like that of Figure C.14, except the CA ”forgot” one direction.

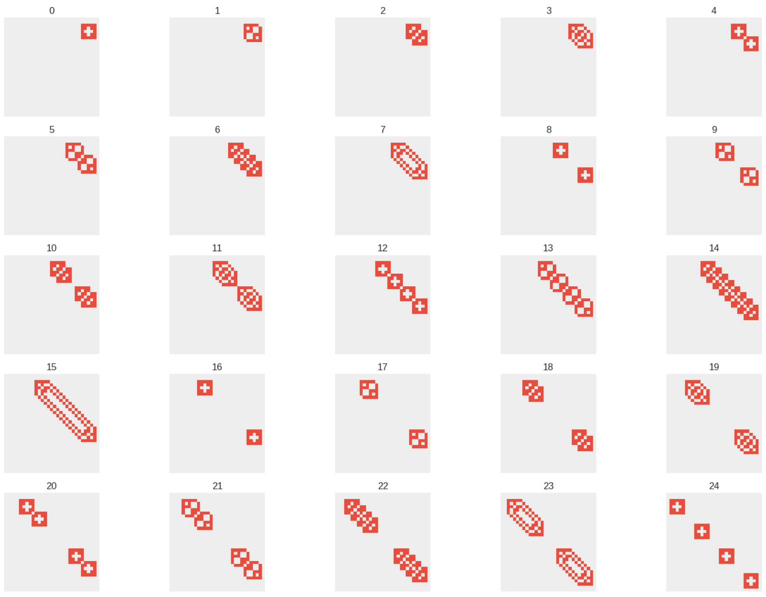


Figure C.17: A solution that has the same behavior as that of the ”Mosaic” replication in Figure 4.7.

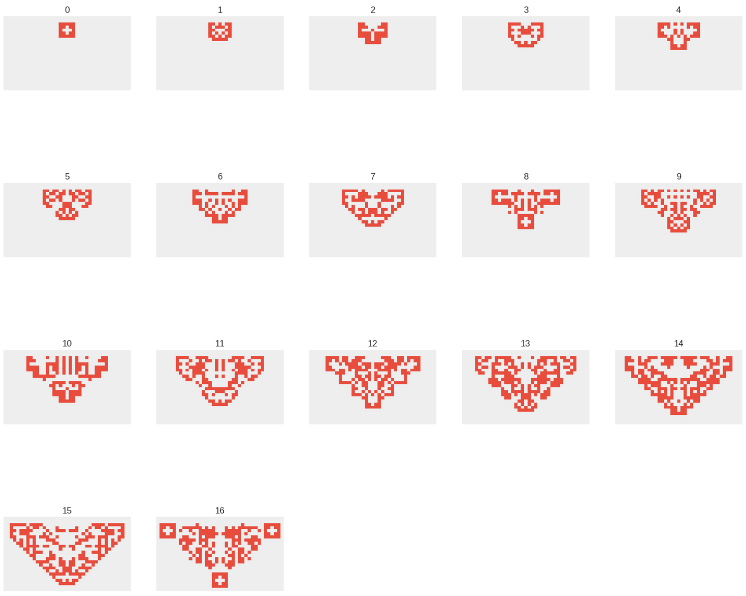


Figure C.18: A solution which illustrates that the fitness function doesn't care about noisy "background" patterns as long as it finds at least three perfect replicas.

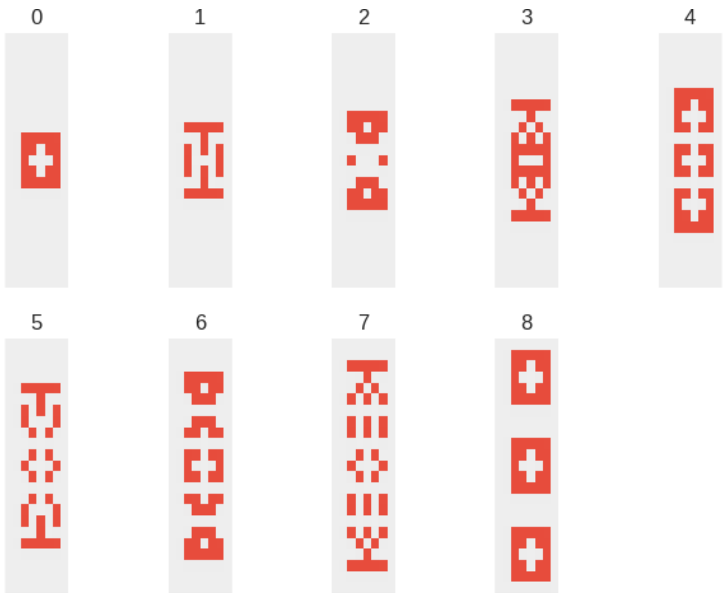


Figure C.19: A solution which expands only along one axis.

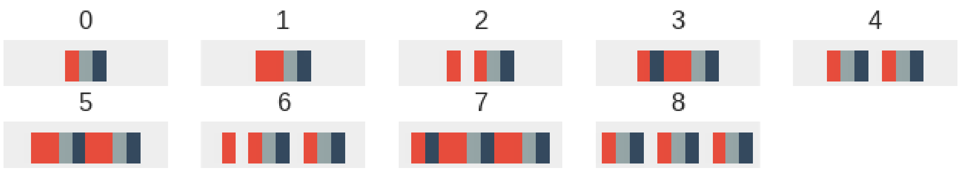


Figure C.20: A solution to the ”Tricolor” replication. All the found solutions expanded only along the horizontal axis.

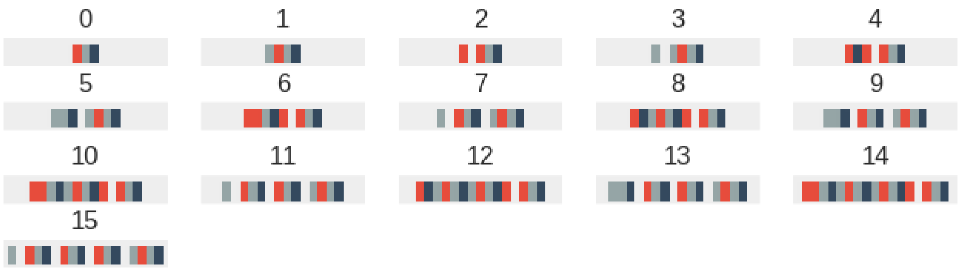


Figure C.21: Another solution to the ”Tricolor” replication.

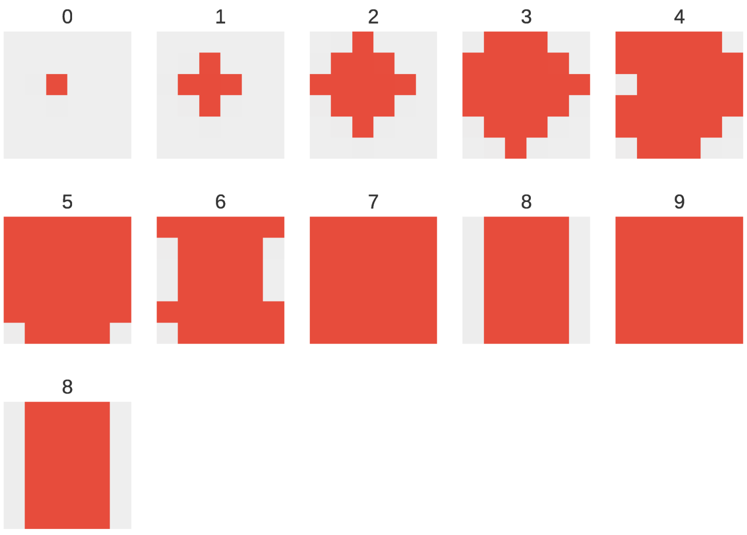


Figure C.22: A solution to the ”Border” morphogenesis with coordinate input. Finding a point attractor (like Figure 4.10) or a two-step cycle attractor containing the target pattern like this was quite common among the found solutions.

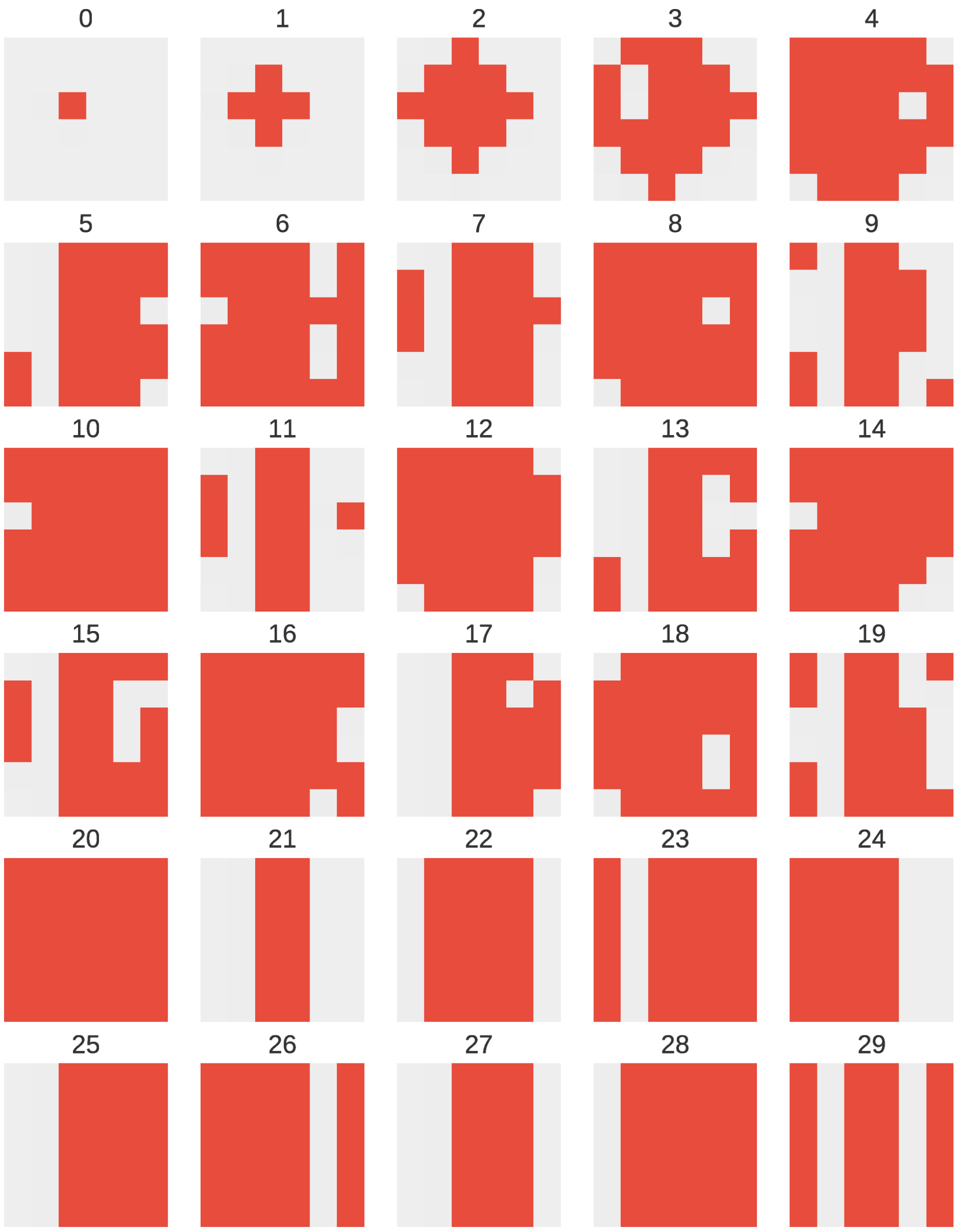


Figure C.23: A ”solution” to the ”Border” morphogenesis with coordinate input. This illustrates a weakness of the fitness evaluation, since the ”solution” does not appear to try to target the wanted pattern, but does 30 different things and just happened to visit the target pattern at time step 22. This did not occur too often though, most morphogenesis results found a point or cyclic attractor within 30 time steps.

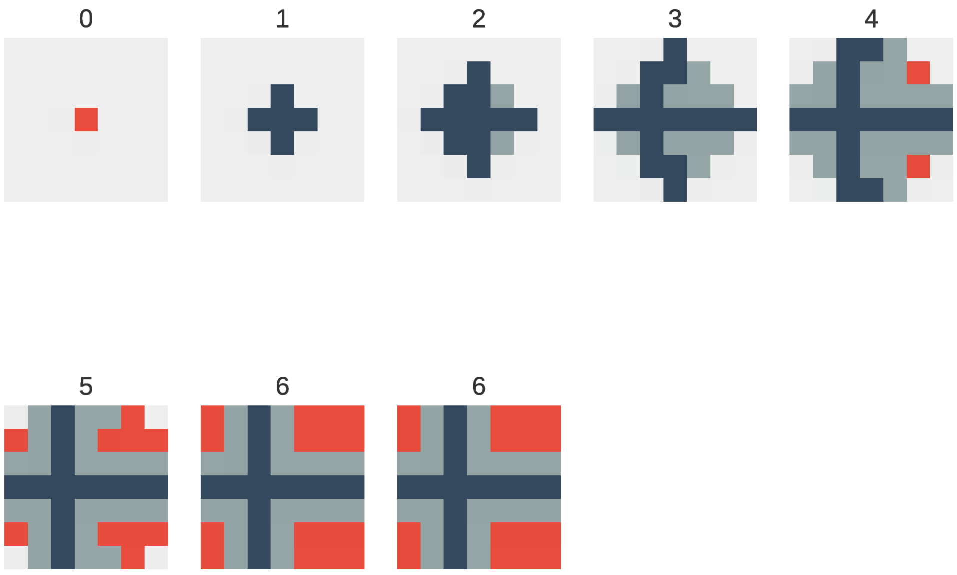


Figure C.24: A solution to the ”Nordic” replication with coordinate input. This is the ”quickest” solution (tied with others), finding a point attractor in time step 6.

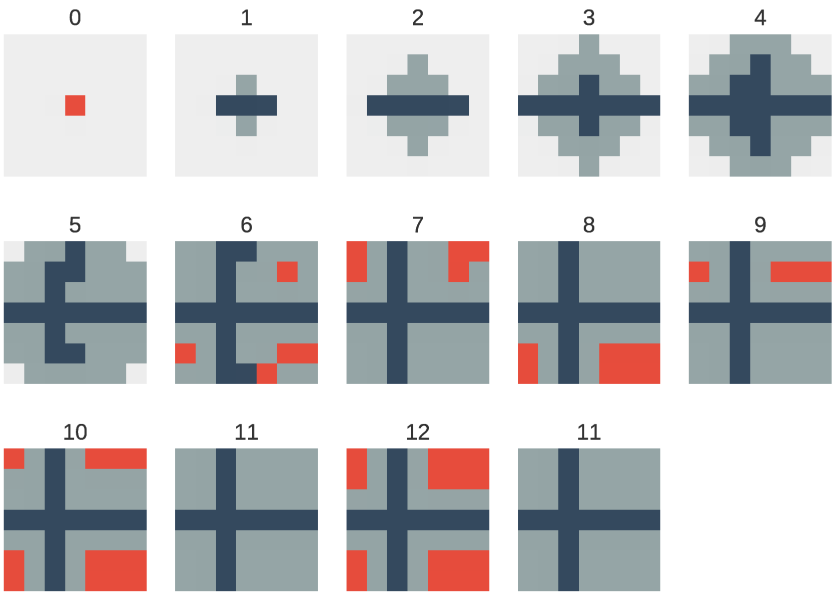


Figure C.25: A solution to the ”Nordic” replication with coordinate input.