Comenius University
Faculty of Mathematics,
Physics and Informatics

Cellular Embryogenic Representations in Evolutionary Design

Jana Hlavačiková 2010

Comenius University

Faculty of Mathematics, Physics and Informatics

Cellular Embryogenic Representations in Evolutionary Design

(Master thesis)

Jana Hlavačiková

Study programme: Cognitive Science
Branch of study: 9.2.8 Artificial Intelligence
Department: Department of Applied Informatics
Supervisor: Mgr. Pavel Petrovič, Ph.D.

Bratislava, 2010

Abstract

Evolutionary Algorithms are robust tools for the optimization of complex systems. They have given rise to the modern area of automated engineering - Evolutionary Design. Inspired by biological systems, several approaches to evolving complex phenotypes have been proposed. This work focuses on a developmental approach with indirect genetic encodings in the representation of cellular automata capable of optimizing multiple complex target flags. More specifically, simultaneous optimization of multiple color and structure patterns was tested using cellular representations. Cellular grid topology, diffusion-reaction model, feedforward neural network, and evolutionary algorithm were used to optimize the cellular representation and decode its genotype into the phenotype, thus into a color picture similar to the optimized target flag. A target specific impulse was encoded into border cells' values in order to optimize multiple color patterns. A variable cellular grid has been used for the structure optimization purpose. The results are promising because they showed the capability of the cellular representation to perform multiobjective optimization of complex patterns.

Keywords:

Cellular Representations, Cellular Automata, Evolutionary Design, Evolution Strategies, Diffusion-reaction model, Multiobjective optimization, Variable Cells Grid, Indirect Genetic Encodings, Neural Networks

Abstrakt

Evolučné algoritmy sú robustným nástrojom na optimalizáciu komplexných systémov. Podnietili vznik modernej éry automatického inžinierstva - evolučného dizajnu. Niekoľko prístupov inšpirovaných biologickými systémami bolo navrhnutých na evolvovanie komplexných fenotypov. Táto práca sa zaoberá jedným z nich, konkrétne sa zameriava na vývojový prístup s nepriamym genetickým kódovaním v reprezentáciach celulárnych automatov schopných optimalizovať viaceré komplexné vzory. Optimalizácia farby a štruktúry bola testovaná použitím celuárnej reprezentácie. Topológia v podobe celulárnej mriežky, difúzno-reakčný model, dopredná neurónová sieť a evolučný algoritmus boli použité na optimalizáciu celulárnej reprezentácie a preklad jej genotypu na fenotyp, teda na farebný obrázok podobný tomu optimalizovanému. Impulz špecifický pre daný cieľ bol zakódovaný do hodnotových premenných hraničných buniek za účelom optimalizácie viacerých farebných vzorov. Premenná celulárna mriežka bola použitá na optimalizovanie štruktúry. Výsledky sú perspektívne, pretože poukazujú na schopnosti celulárnej reprezentácie optimalizovať viaceré komplexné vzory.

Kľúčové slová:

celulárne reprezentácie, celulárne automaty, evolučný dizajn, evolučné stratégie, difúzno-reakčný model, optimalizácia viacerých vzorov, premenná celulárna mriežka, nepriame genetické kódovanie, neurónové siete

Contents

Art is the Queen of all sciences communicating knowledge to all the generations of the world.

Leonardo da Vinci

Acknowledgments

I thank Pavel Petrovic for the supervision of my master thesis, Marc Schoenauer and Alexandre Devert for their advice and kind hospitality during my research stay at Universite Paris Sud, and Richard Baran for technical help. My research stay at Universite Paris Sud was funded by Erasmus Mobility Programme.

Chapter 1

Introduction

Nature provides numerous examples of evolutionary optimization of organisms to different habitats and lifestyles. Nature has therefore been an inspiration in many engineering areas, especially in evolutionary computation and consequently in evolutionary design.

This work demonstrates how evolutionary design by computers could be capable of a complex optimization using a novel and promising approach - genesis of cellular representations. Furthermore, it explains the reasons why such approach could be creative and facilitate an innovative and novel evolutionary design.

In this section, general concepts of the embryogenic flag problem are introduced. Embryogenic cellular representation is discussed in more detail in Chapter 2, followed by the description of experimental methods in Chapter 4, results in Chapter 5, and concluding remarks in Chapter 6.

1.1 Flag Problem

Flag problem is a well known structured programming problem in computer science. It was proposed by Edsger Dijkstra as the 'Dutch national flag problem'. The task can be viewed as rearranging elements of a given representation in order to construct the Dutch flag, which has three stripes of red, white, and blue (Figure 1.1). The flag is represented as an array, and the problem is about sorting or partitioning, i.e. to sort variables in the array into sections, each section representing a color, and thus obtain the target flag. Flag pseudocode is outlined in Figure 1.2.

Over time, the flag problem has been solved by various methods (Chapter 2.2), and nowadays it serves as a benchmark problem in multiple areas including evolutionary design.

Figure 1.1: Dutch flag as the pattern for reconstruction.

1.2 Neural Networks

Neural networks (NN) are not only the biological processors, memories, and hard drives of human beings and other higher creatures, but they can be designed in a simplified approximation in computers. The artificial neural networks (ANN) or simulated neural network (SNN) are made up of artificial neurons. The artificial neuron is an object that tends to simulate a biological neuron (by using an activation function and sometimes a firing frequency also). Each neuron has at least one input and one output. The neurons are connected together via weighted connections to form a network of nodes. While the topology of a neural network does not have to be variable, the weights usually are. NN have a high degree of plasticity thanks to their continuous search space and thus they can adapt smoothly. The alternation of the strength of the connection weights allows an adaptation to the desired signal flow for the purpose of learning and memory. Learning means that the NN are able to infer a function from observations, memory means that the NN are able to use the inferred function later and repeatedly. This is particularly useful in applications where the complexity of the data or the task makes a manual design of such a function very hard or impractical.

Special types of neural networks are used in neuroscience. These networks are designed to simulate biological neural networks as precisely as possible. The goal is to simulate biological processes, understand them better, find ways how to treat various diseases, and/or design effective human-interface machines.

On the other hand, NN can be used in economics or computer science. In this case, the artificial networks adopt the key feature of the biological NN - parallel signal processing.

There are three major learning paradigms for neural networks:

- **supervised learning**: Sample pairs of input/output data are given and the goal is to find a function that performs the input/output mapping specified by the examples, e.g. classification tasks.

In the problem with three colors, here ‘0’, ‘1’, and ‘2’, the array is divided into four sections:

1. a[1..Lo-1] zeros (red)
2. a[Lo..Mid-] ones (white)
3. a[Mid..Hi] unknown
4. a[Hi+1..N] twos (blue)

The unknown region is shrunk while maintaining these conditions

1. Lo := 1; Mid := 1; Hi := N;
2. while Mid $\leq$ Hi do
   (a) Invariant: a[1..Lo-1]=0 and a[Lo..Mid-1]=1 and a[Hi+1..N]=2; a[Mid..Hi] are unknown.
   (b) case a[Mid] in
      - 0: swap a[Lo] and a[Mid]; Lo++; Mid++
      - 1: Mid++
      - 2: swap a[Mid] and a[Hi]; Hi−−

Figure 1.2: Algorithm for the ‘Dutch national flag problem’[1].

- **unsupervised learning**: The aim is to learn about the structure of the data for the purpose of further processing. Input data and a cost function which is to be minimized are given. The cost function is a function of the input and the network's output. A task of the network is to adapt its weights thus obtain the output which minimizes the function. The applications include compression tasks, clustering, statistical data analysis, etc.

- **reinforcement learning**: The input data for processing is usually not given, but generated by interactions with the environment, e.g. motor control tasks.

Each paradigm corresponds to a particular abstract learning task and usually is not dependent on the NN architecture.

There are several various NN architectures or models that are suitable for a particular abstract task. Basically, the bigger the network is, the more weights combinations it is able to generate and therefore the more hard problem it is able to solve. Such models of NN are: feedforward NN, recurrent NN, echo state networks (ESN), self-organizing maps (SOM), cellular NN (CNN) and other. They differ either in topology and/or signal processing. For more information about NN models and learning, see for example [34].

The simplest type of the feedforward NN is a binary linear classifier - **perceptron**. A linear classifier can separate inputs only with a (hyper)plane. The perceptron maps an input x (a real-valued vector) to an output value $f(x)$ (a binary value) across the weight matrix w (Equation 1.1). Such transformation of the input values to the output values is called **activation function**. The perceptron has a single layer of neuron(s), and each neuron applies the activation function to the input(s) and provides the output.

$$f(x) = \begin{cases} 1 & \text{if } w \cdot x + b > 0 \\ 0 & \text{else} \end{cases} \tag{1.1}$$

where w is a vector of real-valued weights, $w \cdot x$ is the dot product (a weighted sum), b is 'bias' or 'treshold', a constant term.

Multi layer perceptron (MLP) is a feed-forward NN derived from a simple form of a single layer perceptron (Figure 1.3). MLP is more powerful because its hidden layer(s) insure the non-linear character and therefore can solve non-linearly separable problems. MLP is characterized by its topology (number of inputs, outputs, number of hidden layers versus number of neurons), connectivity, and an activation function.

A differentiable logistic function is often used instead of the step activation function. This allows to derive efficient learning rules, and improves the

network plasticity: small alteration in the weights results in small alteration of the output.

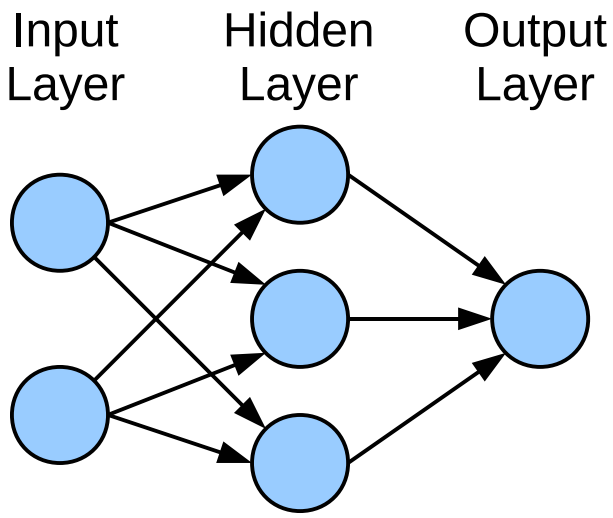


Figure 1.3: Example topology of a multi layer perceptron with one hidden layer.

1.3 Evolutionary Algorithms

Evolutionary algorithms tend to simulate Darwinian evolution. Evolutionary algorithms are broad algorithmic principles, rather than specific algorithms for a narrow range of problems, which can be used to solve a wide variety of hard problems. These problems can be divided into two types. First are little known problems, for which solution algorithms are not available. For the second type of problems, no deterministic polynomial time solution exists (including NP hard problems). Evolutionary algorithms do not require a special type of representation. They are flexible and can explore any type of search space such as trees, graphs, automata, grammars, etc. The search space is called **genotype space** with elements - genotypes. Thus **genotype** is an internal representation of an individual within an evolutionary algorithm. A decoded genotype represents the individual's observable state and is called **phenotype**. Candidate solutions are evaluated by an evaluation function that takes phenotype or genotype as input and returns a **fitness** value. The fitness matches the solution's phenotype quality. For instance, a phenotype can be a color and the genotype a string of values (6-digit hexadecimal code encoding the resulting color). An evolutionary algorithm optimizes (maximizes or minimizes) the fitness value by searching the genotype space.

In general evolutionary algorithms, individuals are completely determined by their genotype. Some experimental algorithms combine the evolution with learning during the life-time of the individual, i.e. the evolved individual further adapts based on its interaction with the environment. This makes the fitness landscape smoother, which in turn improves the search process.

1.3.1 Basic Principles

Candidate solutions - individuals in a population are evaluated in parallel and compete with each other for survival. The competition leads to the selection of stronger individuals, i.e. individuals with better fitness, i.e. better adapted individuals. Stronger individuals have a higher chance to become parents of a new generation and create offspring via recombination and mutation. The fitness of the offspring is evaluated and another round of competition takes place. This competition is for survival in a new generation. At this stage, the offspring compete among themselves or also with parents. In some cases this round of competition does not take place, but for instance all of the offspring can create the new generation. While the selection reduces diversity, it increases quality, and the fitness of the whole population improves over time. This process goes on in a loop until termination condition is satisfied (Figure 1.4).

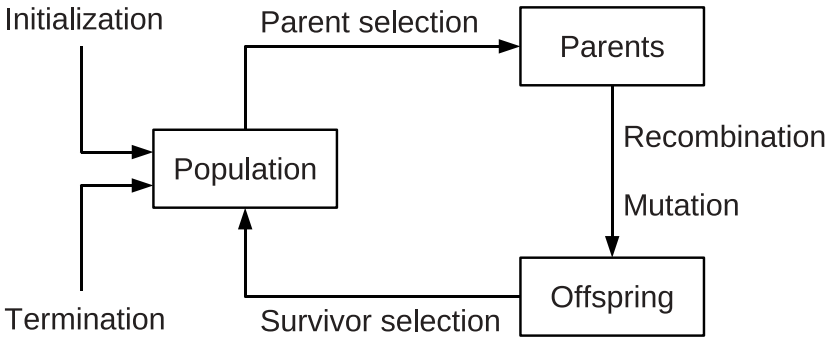


Figure 1.4: General scheme of an Evolutionary Algorithm [2].

1.3.2 Evaluation Function

Evaluation Function is the objective function also called quality or fitness function. It decodes the individuals into the phenotype space and assigns a fitness value to each individual. Thanks to the fitness value, the individuals' genotypes can be compared to each other and ranked.

A good fitness function corresponds with the algorithm's goal, and moreover, it is computed as quickly as possible. The speed is important, because a typical evolutionary algorithm runs many times until the candidate solutions converge to an optimum and a good result is produced. The fitness function is the most computationally demanding part of the algorithm (it has to be evaluated a large number of times: generations * number of individuals in a population). Consequently, the long evaluation time is the main drawback of evolutionary algorithms in some applications.

A fitness function can be visualized (the complexity depends on the number of dimensions). A simple fitness function *f(x)* of one variable x can be visualized on a planar graph. The goal of the optimization is either the maximization or the minimization of the fitness value.

The fitness function transforms the search space into a **fitness landscape** (plot of all possible solutions), where the height of a point corresponds to its own fitness value. The distance of two points characterizes their degree of similarity. The points that are close to each other, have more similar genotypes than the points that are further from each other. The landscape has peaks (maxima of the fitness function) and valleys (minima of the fitness function). If the landscape has many local extrema - maxima or minima, it is called a **multimodal function** (Figure 1.5). It is difficult to find a global optimum of a multimodal function as the optimization process may become trapped in a local optimum. Sometimes, it is even not desirable to obtain a global optimum because such solution can be highly unstable. Local optima can be more robust to perturbations (Figure 1.6). Thus by solving a multimodal problem a number of co-evolving populations is usually maintained to preserve the diversity and approximate multiple different solutions.

A different kind of problem is to optimize multiobjective functions thus optimize more than one objective value. During the general multiobjective optimization, we are looking for a solution for which each objective value can be optimized until the other objective value(s) would start getting worse. Such solution is called **Pareto point**. Finding such a solution, and calculating how much better this solution is compared to other solutions is the goal when solving a multiobjective optimization problem. Basically, we search simultaneously for a set of Pareto points approximating the **Pareto front**. Pareto front (Pareto set)[1] is a set of such solutions that are **nondominated**. We say that a solution x dominates a solution y if it is at least as good on all criteria and better on at least one. Then, if we are able to approximate the Pareto front, we are able to compare and rank each dominated solution according to

[1]The terms Pareto front and Pareto point are named after Vilfredo Pareto, an Italian economist who used the concept of Pareto optimality in his economic studies.

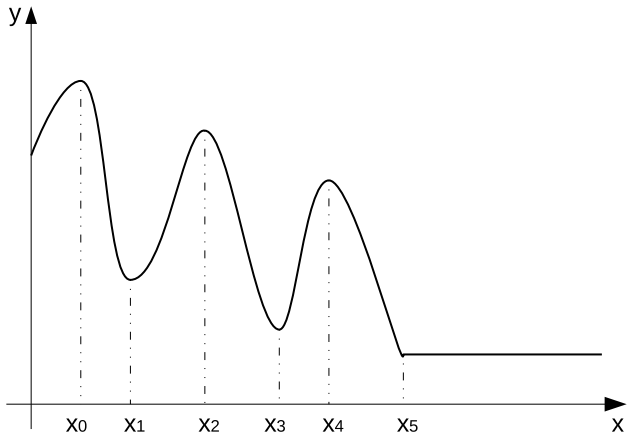


Figure 1.5: A simple example of a planar multimodal function with multiple local maxima in x_2 and x_4, multiple local minima in x_1 and x_3 and a global minimum in x_5, and a global maximum in x_0.

its distance to the Pareto front. It means that we are able to sort and obtain a partial order of the solutions in terms of the **Pareto dominance**. An example of the Pareto dominance is shown in Figure 1.7.

For more information about the multimodal and multiobjectives functions see e.g. [2].

Two main types of fitness function are used in evolutionary algorithms: fixed and mutable functions. A fixed objective function is modeled a priori and stays static during the program run. A mutable function is often used in co-evolution of multiple solutions or when parameters of fitness function or the fitness function itself is not known and it is a subject of search.

1.3.3 Parent Selection

Artificial selection is an intentional breeding of certain individuals into next generations. The selection of an individual is usually highly dependent on the individual's fitness value.

Selection can be implemented as a **roulette wheel algorithm** where the probability of selection of an individual is proportional to its assigned fitness. Such selection is dependent on the absolute fitness values. This is often not effective.

Other methods with relative dependencies are usually applied. In a **rank selection**, the population is sorted according to the individuals' fitness values. This method preserves selection pressure. The worst individual has the lowest rank and the best individual has the highest rank. The final fitness assigned to

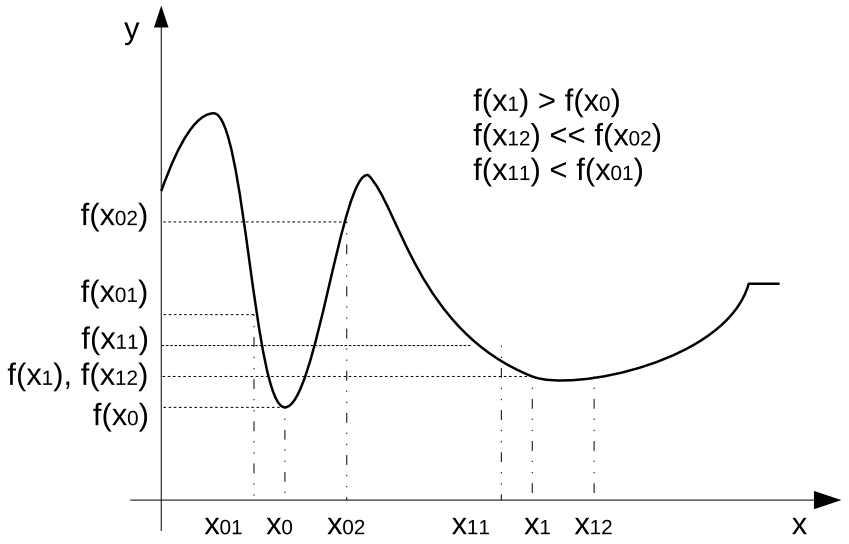


Figure 1.6: An example of a multimodal function, in which solution x_1 laying in a local optimum is more stable than solution x_0 in a global optimum (in a minimization task). A small perturbation (shifting on the x-axis) significantly declines the quality of the fitness value.

the individual depends on the position in the rank thus the fitness is relative rather than absolute.

An alternative selection method, which uses the sorted population, is **truncation**. It just selects k best individuals from the population.

Another alternative is **tournament**. Tournament is useful when the population is very large or there is no overall knowledge about the population. Competition does not take place between all individuals but only between a number of them - the tournament size. The competing individuals are chosen with a uniform probability and an individual with the best fitness is the winner.

1.3.4 Variation Operators

Variation operators maintain the diversity. Mutation and/or recombination operators can be combined in various ways. The variation operators are dependent on the used representation.

Recombination

Two parents create one offspring by combining their genotypes. A new gene cannot arise by the recombination, but a novel combination of previously ex-

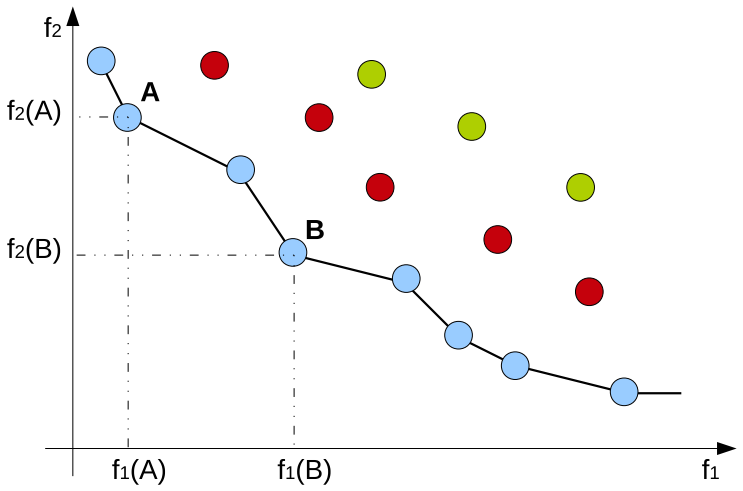


Figure 1.7: An example of the Pareto dominance. Blue points are the nondominated Pareto points and form the Pareto front. Solution A has a better fitness values f_1 than solution B, but solution B has a better fitness values f_2 than solution A. We say that solutions A and B are not strictly dominated by each other, thus lie on the front. Red points are dominated by blue points, but red points are still better solutions than green points. Green points are dominated by red points and blue points also.

isting genes can lead to an improved fitness. Therefore, recombination refers to the exploitation of the search space.

Mutation

One parent undergoes mutation and creates one offspring. Mutation makes changes on the particular gene(s) and by these random genotype changes the mutation shifts the population into new unexplored dimensions. Therefore, mutation refers to the exploration of the search space.

1.3.5 Survivor Selection

Survivor selection (replacement) can be performed in the same way as parent selection. However, there are other specific ways how to select survivors.

Survivor selection can be divided into two approaches:

- **Age-Based Selection**. In steady-state population model, this method can be implemented as FIFO (first in first out), i.e. the oldest individual is deleted. In generational population model, all individuals are replaced by new offspring.

- **Fitness-Based Selection**. Such selection uses one of the methods described in Section 1.3.3 or a different method, for example:
  - **Elitist** $(\mu + \lambda)$ **replacement**. At every generation, μ parents are selected (via a parent selection method) to create λ offspring. The best μ among parents and offspring become the next generation. At least one copy of the fittest solution is always kept.
  - **Non-elitist** (μ, λ) **replacement**. At each generation, μ parents create λ offspring. The best μ out of the λ offspring build up the new generation.
  - **Genitor** also known as "delete the worst".

1.3.6 Initialization

The initialization is the very start of the algorithm. Its task is to sample the search space as uniformly as possible. With a good sampling, the population has a bigger chance to explore promising areas of the search space.

Initialization is usually performed at random or uses some known problem-specific heuristics.

1.3.7 Termination

The stopping criterion can be the reaching of the desired fitness of the population, the lack of significant improvement of fitness, or the exceeding of the pre-specified time or generation limit.

More information about evolutionary algorithms can be found for instance in [2].

1.4 Evolutionary Design

Evolutionary design has its roots in computer science, design, and evolutionary biology. It is one of the application domains of evolutionary computation, it extends and combines Computer-Aided Design (CAD) and analysis software, and it borrows ideas from natural evolution (Figure 1.8) [15].

Evolutionary design has been applied in many different areas over the last 25 years. Design optimization has been used in mechanical engineering to optimize structures (flywheels, wind turbines, supersonic aircrafts, etc.), in electrical engineering to optimize circuits, or in computer engineering to optimize hard problems (for example non-polynomial problems) [14, 15, 17, 19, 25, 38]. Evolutionary design has been used not only in engineering areas but also in

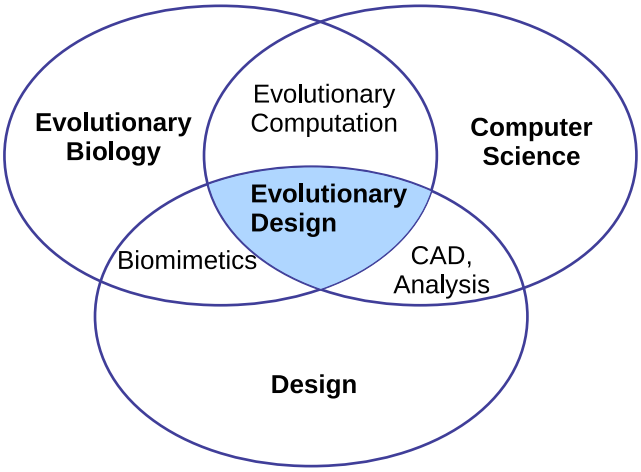


Figure 1.8: The origin of Evolutionary design.

neuroscience (neural simulations of a generation of a diversity of shell structures and pigmentation patterns [33]) or biology (protein folding simulation [32]). Even more, evolutionary design was used as not only the optimizer but as a creator, especially in different kinds of art, modern architecture, or in computer science to evolve artificial life [14, 16].

In general, the types of evolutionary design can be divided into four main categories: evolutionary design optimization, creative evolutionary design, evolutionary art, and evolutionary artificial life forms. These categories overlap and create four additional types of evolutionary design: integral evolutionary design, aesthetic evolutionary design, artificial life-based evolutionary design, and aesthetic evolutionary artificial life (AL) [15]. Figure 1.9 shows all types of evolutionary design and how they relate to each other. For information about particular types, see [15].

The main difference between evolutionary design in an optimization task and an innovative (creative) task is that in the first case we work with a target 'design'. The target 'design' is clearly defined by optimization criteria that form the objective function(s). The criteria are usually a set of constraints given upon the desired state, e.g. dimensions, centroid range, color, materials, weight, price, and other. Such constraints are used by a fitness function to compute the difference between a desired design and the evolved evolutionary system. The goal is to satisfy the constraints and optimize the target design. A hypothetical example of an evolutionary design optimization task is shown in Figure 2.8.

In the case of an innovative task, the criteria are less detailed, the objective function is more behavioral than functional, thus the holistic criterion leaves

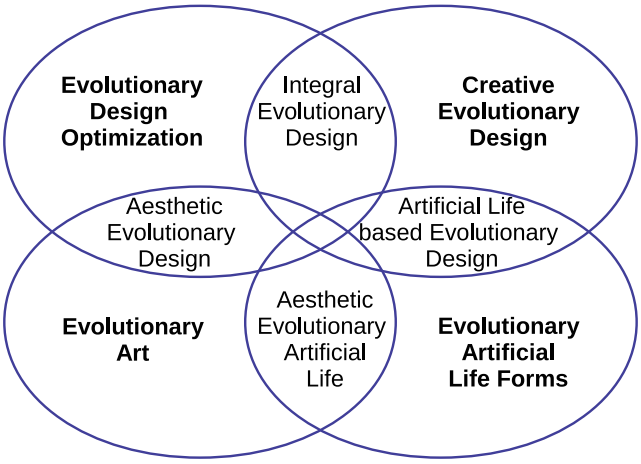


Figure 1.9: Types of evolutionary design

space for innovative solutions found in the search space defined by the genotype representation.

Another option in evolutionary design is interactive evolutionary system, which receives feedback from humans and accounts the feedback in the fitness function.

1.4.1 Advantages of Evolutionary Systems

A simple and common question could be, why should evolutionary design be used? What are the advantages of evolutionary systems over humans in design?

Evolutionary approach uses a population of solutions rather than a single solution. It means that a bigger search space is explored and more promising areas could be exploited. The artificial system is intended to help the designer in the exploration of structures that are otherwise difficult to assess. Moreover, if the program is implemented efficiently, the design time decreases rapidly.

As it was already mentioned earlier, evolutionary design mimics natural evolution. The nature has proven that it is able to design efficient things and creatures.

Now, following the introduction into evolutionary algorithms and the evolutionary design, technical aspects of evolutionary systems will be described.

1.5 Representations

There is a number of challenges facing the implementation of an efficient software to perform evolutionary design. Some challenges are related to the

(a) A target chair described with constraints: material: wood, color: brown, maximum weight: 10 lbs, minimum sitting surface = 200 inch^2, sitting height = 18 inch, etc. (from [35])

(b) Resulting design of the chair (from [36]).

Figure 1.10: A hypothetical example of an evolutionary design optimization task.

properties of the objective function. For instance, noisy, discontinuous, multimodal, or multiobjective functions are difficult to optimize. Other challenges are connected to the choice of a suitable representation (genotype and phenotype), its effective implementation, evaluation and possibly its embryogeny[2]. Most common representations in evolutionary methods include:

- *Binary strings* : **Genetic Algorithms (GA)**
- *Real valued vectors* : **Evolution Strategies (ES)**
- *Finite state machines* : **Evolutionary Programming (EP)**
- *Program trees* : **Genetic Programming (GP)**

Historically, different methods arised together with their respective representations. However, the representations are not limited to individual methods and

[2]Embryogeny is a process of growth that defines how a genotype is mapped onto a phenotype. In the word 'embryogeny' or 'embryogenic representation', the 'genic' is derived from genesis and not genetic. In some evolutionary methods, the goal is to simulate the natural 'embryogenesis'.

can be combined with different design variation operators and fitness functions. Differentiation of representations is thus not strict but rather informative.

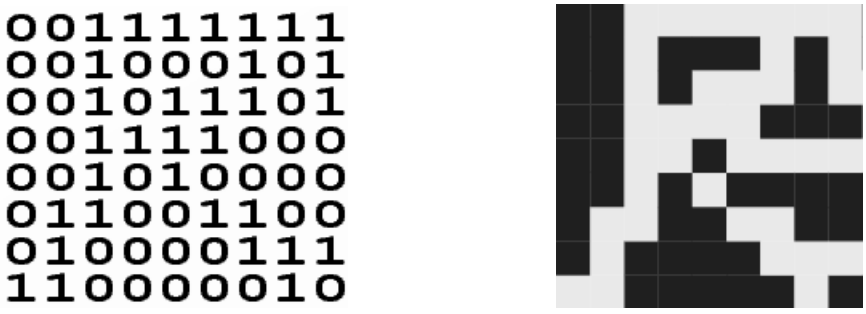
A crucial step in the design of an evolutionary solver is the selection of an effective representation for the current problem. The very important consideration is that if two individuals are closely related on the genotype level then they should be closely related on the phenotype level as well. Otherwise, two solutions are incomparable. In other words, a minor change in the genotype should not cause a major change in the phenotype, otherwise the fine tuning of the system becomes difficult. Another important issue to consider before choosing an appropriate representation/embryogeny is the dimensionality of the search space and the level of complexity. An efficient representation/embryogeny can provide the following benefits: reduction of the search space, complex phenotype solutions, constraint handling, adaptation and repetition. The advantages and drawbacks will be further discussed for each representation category. Representations versus embryogenies:[3]:

- **direct representations** (no embryogeny or external embryogeny)
- **indirect or generative representations** (explicit or implicit embryogeny)
- **cellular representations** (implicit embryogeny)

1.5.1 Direct Representations

Direct representations are the simplest type of representations where the genotype directly encodes the phenotype. The simplest direct representation is a binary representation, so called **bitarray** [12, 13], where each bit represents a single unit of the design pattern, a pixel or voxel (Figure 1.11). A bit can hold the information 'true' or 'false', where 'true' or the value 1 means that the pixel is white and 'false' means that the pixel is black. The main drawback of 'bitarray' representation is the dependency of its complexity on the underlying mesh. Unfortunately, according to both theoretical results and empirical considerations, the critical population size required for convergence should increase at least linearly with the size of the individuals (the number of bits encoding a structure) [17]. Moreover, a bigger number of generations is generally required for convergence of larger populations.

[3]The division is not strict but informative. Examples of different classifications are described in [23, 24]



(a) 'Bitarray' genotype.

(b) Resulting phenotype topology.

Figure 1.11: An example of 'bitarray' representation.

1.5.2 Generative Representations

More complex and sophisticated representations are indirect representations. In this case, the genotype is optimized, which is different from the resulting phenotype. The phenotype is determined by the genotype, but the genotype does not encode the phenotype directly, but rather encodes instructions how to construct the phenotype result. The translation is realized by a finite automaton or by a grammar. Indirect representations have variable length of the genotype representation. Generally, the representation start with one gene and creates more during the evolution. Hence, the complexity of the representation is self-adaptive. The inspiration for indirect representations comes again from nature, which uses the grammar rules in translation of DNA[4] into amino acids, then proteins, and eventually into an embryo[5] and a final organism.

Many works show that indirect representations make search more robust, improve the evolvability of the organism, and obtain better results [23, 24]. An example of generative representations are L-systems, see for example [23] or [31], unordered lists, program trees, or graphs. More specific examples

[4]In DNA, a combination of three letters (DNA bases: adenine (A), thymine (T), guanine (G), cytosine (C)) encodes one amino acid. For example, CT*, TTA or TTG encode Leucine; GC* Alanine; AAA or AAG Lysine; etc.

[5]In nature, the fertilized egg cell leads to a development of an embryo. Embryo undergoes a complex developmental process known as embryogenesis, other *genesis and metamorphosis until finally becomes a mature phenotype.

include Voronoi diagrams[6], dipoles representations[7], and Iterated Function System (IFS) [16, 25]. Voronoi diagrams and dipoles representations can be implemented as the unordered lists, where each item in the list is a Voronoi site or dipole respectively. The Voronoi and dipoles representations showed good results on cantilever benchmark problems[8] [25]. In the same work, the IFS representations did not show very good results - the authors's hypothesis is that the problem was too simple for such complex representation. However, the IFS has the potential to bring fruitful results in evolutionary art [16].

L-Systems or Lindenmayer systems are grammatical parallel rewriting systems introduced to model biological development of organisms, most famously plants. L-systems are similar to Chomsky grammars. Both grammars have an initial string of symbols and a set of production rules that may be applied to the symbols to generate new strings. The grammars differ in the way, in which production rules are applied. Chomsky grammars change symbols sequentially, while Lindenmayer grammars apply many rules at the same time in parallel.

For example we can have rules: $A \rightarrow AB$; $B \rightarrow A$.
If we start at the symbol A, we obtain the following derivations: $A \rightarrow AB \rightarrow ABA \rightarrow ABAAB \rightarrow ABAABABA \rightarrow \ldots$

For further details about L-systems see for example [23, 31].

In the Bentley's and Kumar's work [24], grammar rules are used as 'implicit embryogeny'. In [24], four evolutionary systems were employed for comparison:

- GA with no embryogeny (binary strings representations)
- GA with an external embryogeny (genotypes define the coordinates of a set of predefined shapes in the phenotype grid)
- GP system with an explicit embryogeny (program trees)

[6]Voronoi diagram is a special kind of decomposition of a metric space determined by distances to a specified discrete set of points (Voronoi sites) in the space. Each site is surrounded by points that form a Voronoi cell. These points are closer to the site than to any other site in the diagram. The segments of the Voronoi diagram are borders between the Voronoi cells thus all points that are equidistant to the two nearest sites.

[7]A dipole is a set of two Voronoi sites whose median has a prescribed angle in the plane. Thus medians of sites form Voronoi cells and consequently Voronoi segments.

[8]A cantilever benchmark problem is a popular benchmark problem in design optimization. It is an optimization of a shape of a cantilever rectangular plate fixed at one side, which is subjected to a unit load applied in the middle of the opposite side.

- GA with an implicit embryogeny (grammar rules)

In both works, indirect representations show better experimental results in comparison to direct representations. It should be clear that the main advantage over the direct representation is the reduction of the search space. When the genotype does not encode the phenotype 'one to one', the size of the genotype can be reduced significantly.

Iterated Function Systems The Iterated Function Systems are a method of constructing fractals[9]. The IFS are able to create very complex realistic fractals similar to natural fractals. For all fractal methods, the prescription is simple but it can encode a scene of almost any level of complexity, thus achieves a high ratio of compression from the phenotype to the genotype. The IFS prescription is actually a set of affine transformations (Equation 1.2).

$$x \mapsto Ax + b \tag{1.2}$$

Affine transformation is given by a matrix A and a vector b.

In general, an affine transformation is a linear transformation (rotation, scaling, shear, etc.) followed by a translation (shift). In two dimensions, the matrix form of affine transformation (Equation 1.2) can be rewritten into two easily implemented equations (Equation 1.3).

$$x_{n+1} = a * x_n + b * y_n + e \tag{1.3}$$

$$y_{n+1} = c * x_n + d * y_n + f \tag{1.4}$$

Then, the IFS representation genotype is a set of sets of the coefficients *a, b, c, d, e, f* and a p value. The p value is a probability, with which the particular transformation will be uniformly randomly chosen and applied to the search space (initially a single point). Such iterated application of the recursive transformation equations will eventually create a complex fractal. A very well known example is the Barnsley fern generated by an IFS with four transforms (Table 1.1, Figure 1.12).

1.5.3 Cellular Representations

Cellular representations are a very promising representation type used in evolutionary design. They combine the properties of direct and indirect representations in the sense that the cellular topology (for instance a grid) maps

[9]A fractal is a complex system, which consists of self-similar smaller parts, i.e. reduced-size copies of each other and the whole. Many natural objects are fractals, for instance crystals, clouds, plants (cauliflower, broccoli, fern, ...), or even landscapes. However, fractals can be easily described by recursive function(s) and constructed iteratively.

a	b	c	d	e	f	p	fern transform
0.00	0.00	0.00	0.16	0.00	0.00	0.01	stem
0.85	0.04	-0.04	0.85	0.00	1.60	0.85	main branch
0.20	-0.26	0.23	0.22	0.00	0.16	0.07	left leaves
-0.15	0.28	0.26	0.24	0.00	0.44	0.07	right leaves

Table 1.1: Set of affine transformations generating the Barnsley fern.

Figure 1.12: Barnsley fractal fern.

to the phenotype topology directly. For instance, the cellular grid can map one to one with the resulting picture, thus a cell can match a pixel. On the other hand, the color or other properties of the cells are gained implicitly thus indirectly.

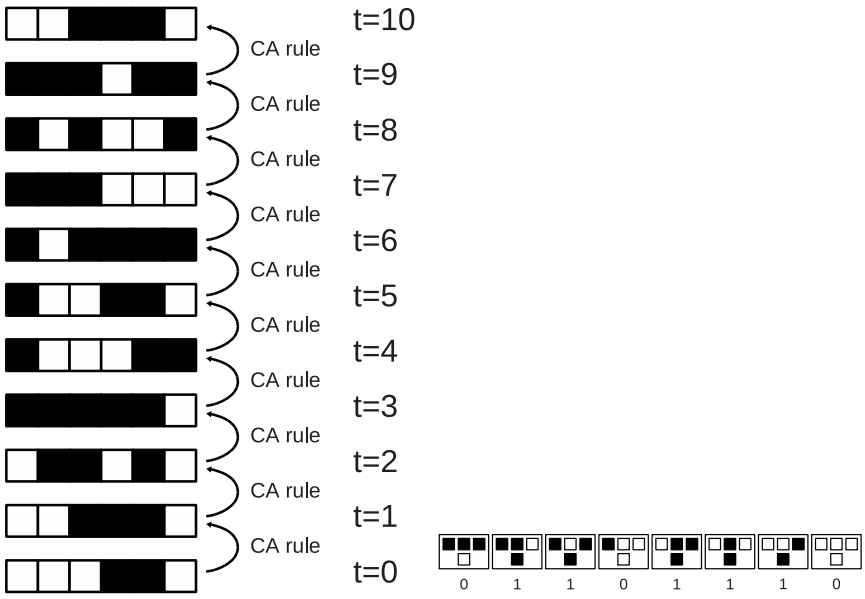
Cellular Automata (CA) are one of the simplest models of complex systems. They are good idealizations of the dynamical behavior of various systems. As also S. Wolfram suggests, cellular automata might be a more efficient tool in modeling natural mechanisms than mathematical equations [21, 22, 28].

Cellular Automata are discrete dynamical systems. Cellular Automaton consists of a finite grid of cells (each cell is a finite automaton) that are in one of their finite number of states.

Cell's neighborhood is defined. Commonly used forms of neighborhood are the Moore neighborhood (8 neighbors) or the Von Neumann neighborhood (4 neighbors). Initial states of the cells and an update rule (function) are specified. Then every next state of the cell is guided by the rule and depends on its current state and the states of its neighbors. Usually, the rule is the same for all cells and it is not changed over time. A CA is a deterministic system completely defined by its update rule(s) and an initial configuration of cells' state values. When the development is finished, one can evaluate the stability of the final state of the CA. A CA can be in a steady state when any

next state is the same as the current. Sometimes CA does not reach such a fixed point and keeps cycling or can even behave chaotically.

The simplest class of one-dimensional CA are elementary CA. Elementary CA have two possible values for each cell (0 or 1), and rules that depend on the nearest neighbors (left and right neighbor) values and its own value. Since there are $2^3 = 8$ possible binary states combinations for the three cells, there are a total of $2^8 = 256$ different elementary cellular automata, each defined by a 8-bit binary string. For example, the rule no. 110 ($110 = 01101110_2$) and its evolution is illustrated in Figure 1.13.



(a) First 10 generations of applying the CA rule.

(b) Rule no. 110 [30].

Figure 1.13: Illustration of the elementary cellular automaton. The possible values of the three neighboring cells are shown in the top row of each panel, and the resulting value that the central cell is assigned in the next generation is shown below in the center [30].

Best-known and a simple application of a CA is Conway's Game of Life [3]. The illustration of the first six states is shown in Figure 1.14. It is worth noting that interactions between elements are usually represented explicitly, but the embryogeny is implicit. The diffusion of the cells' property values further into their neighborhood and the following reaction of the cells is given implicitly (more about diffusion-reaction in Section 2.2). Such cellular automata with implicit behavior have emergent properties that could lead to surprising, novel, and innovative results. Thus the cellular representation can be used

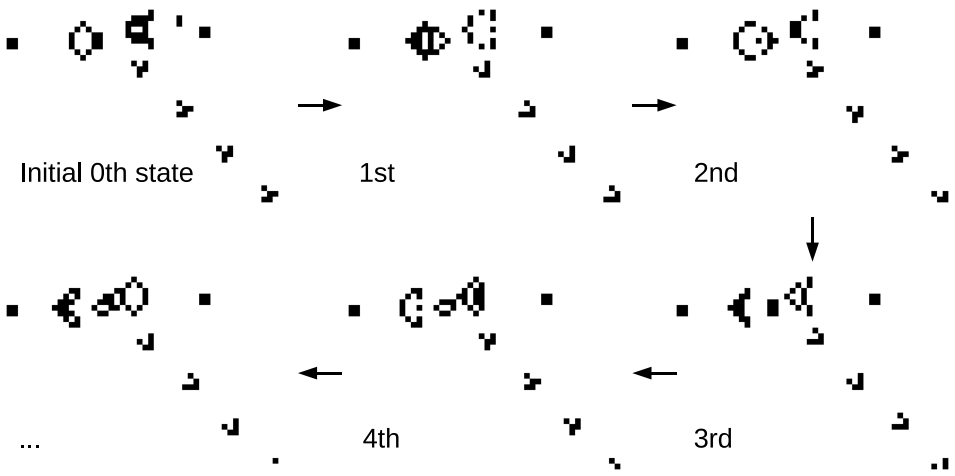


Figure 1.14: Crossing the states in the Conway's Game of Life.

in creative design. CA are one of the promising and still largely unexplored ways of representations in evolutionary design, both in three [5, 19, 26, 27] and two dimensional [5, 19, 26, 27], as well as both natural [29, 32, 33] and artificial [5, 19, 26, 27] engineering. Direct representations have been mostly used for evolutionary design optimization [15], indirect encodings for evolving creative/novel design concepts [23, 25, 16].

Indirect and cellular representations show a high level of adaptability and space reducibility. One drawback of both representations is that it is hard if not even impossible to use them reproducibly. 'Cutting' and reusing a part of a genotype of a direct representation has a predictable effect on the phenotype. This is not the case in implicit embryogenies.

1.6 Cellular Embryogenic Representation

Because cellular representations demonstrate good evolvability and modularity, they are a good candidate for representations in evolutionary design. In our work, the focus is on the evolutionary design optimization but not at the application level. We are testing the possibilities of a novel representation in evolutionary design. The representation is a cellular automaton which combines advantages of implicit embryogeny of the genotype with direct topology mapping to the phenotype. To obtain a good problem solver we have to look for an update function of the automaton capable to render the given pattern from an initial state. Such update is controlled by a neural network. The evolutionary strategy is used to guide the overall optimization, thus guide the

evolutionary cycle of a population of automata and finally obtain the best evolved automaton. As described below, the cellular embryogenic representations appear to be very promising not only in evolutionary design optimization, but in creative evolutionary design as well.

Chapter 2

Methodology

This Chapter describes the architecture of the implemented embryogenic cellular automata - **'embryo'**. Basically, the embryo is a two-dimensional cellular automaton, which consists of a grid of cells. The cells communicate with each other through chemicals, and results of the communication are processed by a multi layer perceptron (MLP). MLP weights create the genotype of the embryo. The MLP updates the chemicals of the embryo cells, thus it dictates a transfer function of the embryo as the cellular automaton. Besides the chemicals, a cell has also internal states. Internal states are also updated by the MLP. From the states and chemicals of embryo's cells, the resulting phenotype (picture) of the embryo is computed. The goal is to reproduce the desired target pattern. Such problem can be seen as a basic problem in the domain of design optimization. An embryo is an individual in the sense of evolutionary strategy. The embryo is evaluated by the MLP from the initial configuration until evaluation is not stopped. Because we are using an evolutionary strategy approach, the search space is not one individual, but a number of individuals - a population. While approximation of the embryo is guided by MLP (Section 2.2), the optimization of the search space is a task for an evolution strategy (Section 2.6). The flow chart of the whole process (Figure 2.1) extends the one from the previous Chapter (Figure 1.4).

The system is inspired by the flag problem (Section 1.1) and the target flag is a two dimensional image pattern. The pattern should be an end result of an iterating cellular automaton. In experiments with static grid, each pixel (px) of the image matches one cell of the automaton. In experiments with dynamic (variable) grid, the cells can grow and match several pixels. We are looking for such a transfer function for a cell that from the initial zero configuration subsequently creates a robust stable target pattern. The goal is to obtain

an attractor[1], which should be able to repair the target pattern after local perturbations (Gaussian noise). However, the maximum amount of noise is limited and becomes an important indicator of the self-repairing feature. The amount of the noise, with which the automaton is able to deal and restore the flag, determine the quality of the solution.

This work is based on the previous work of A. Devert [5] where the goal was to optimize colors of one two-dimensional image pattern by using a cellular automaton with static structure. In this work, the embryo optimizes multiple complex (structure and color) patterns.

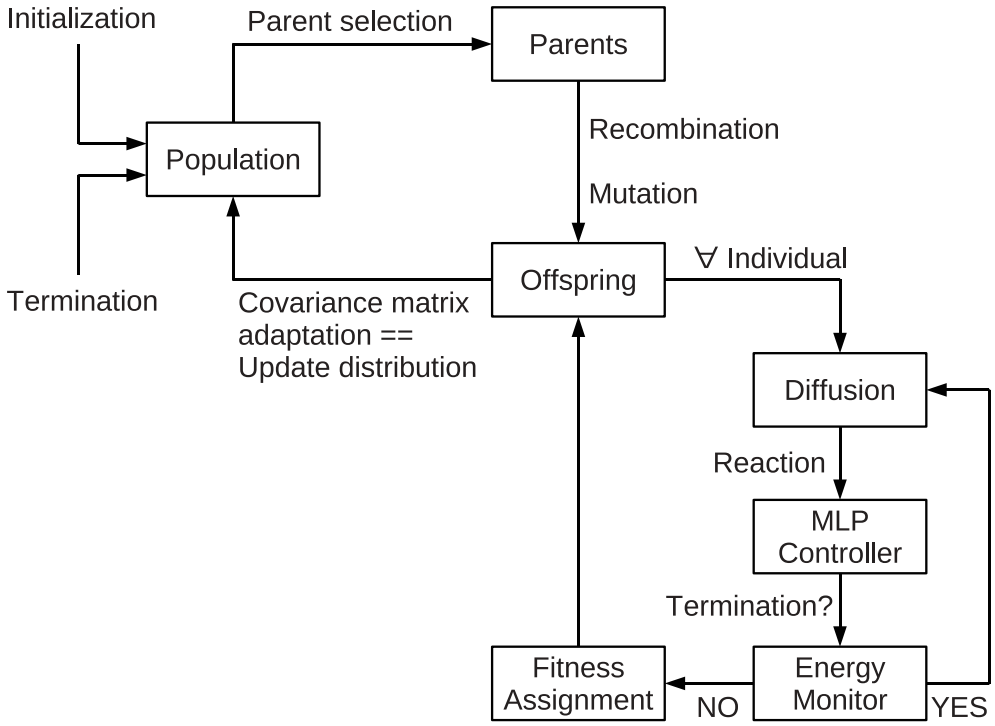


Figure 2.1: Combining evolution strategies and multi layer perceptron in the cellular embryogenic flag problem.

2.1 Cells

The cellular topology of an **embryo** is a full rectangular grid of cells. A cell is specified by its position, size and state variables. These property form the

[1]Attractor is a steady state to which the system tends to restore after a perturbation, i.e. is capable of self-repair. In the image representation, the perturbation is introduced mainly via Gaussian noise.

genotype of the cell. A set of genotypes of all cells in the grid form the genotype of the embryo. The cellular grid maps to the phenotype topology directly. The grid maps one to one with the resulting picture, thus a cell corresponds to a square in the picture, at least a square of size one pixel. On the other hand, the topology and the color of cells are gained implicitly thus indirectly.

Every cell in the grid has two different sets of **state variables**:

- states
- chemicals

Chemicals are external properties that play the main role in the neighborhood communication, i.e. diffusion. States are internal variables that together with the chemicals correspond to a response to the communication, i.e. MLP update.

If we imagine that cells form a two dimensional grid and each cell has state variables as a third dimension then it is possible to sketch the system as a three-dimensional (Figure 2.2). The cell's neighborhood is a 'von Neumann' type neighborhood. It means that each cell is able to communicate with its neighbors on north, south, east and west sides[2]. Basically, when a static grid is used, each cell has four neighbors, but when a dynamic (variable) grid is used, the cell can have any number of neighbors (Figure 2.3).

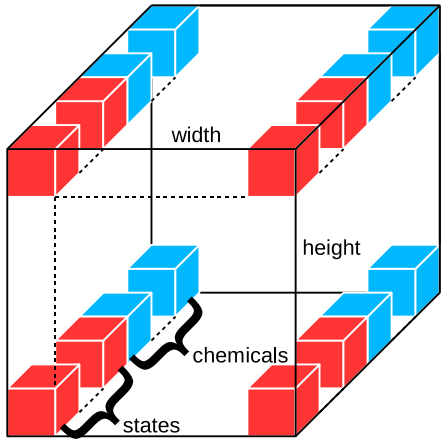


Figure 2.2: A sketch of cellular embryogenic flag structure.

[2]other common type of neighborhood is 'Moore neighborhood' where the neighbors are the cells in direct and diagonal directions also.

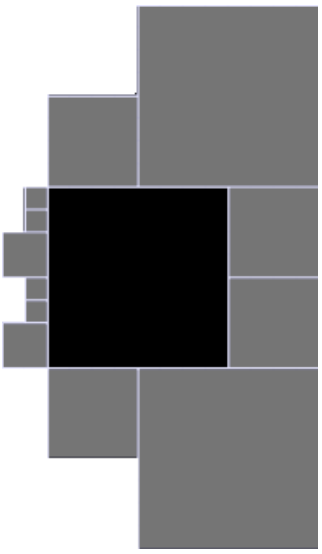
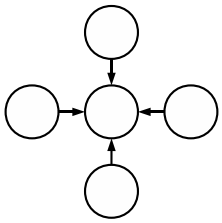
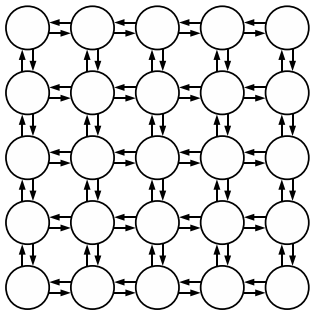


Figure 2.3: An example of a neighborhood (gray cells) of a cell (black square).

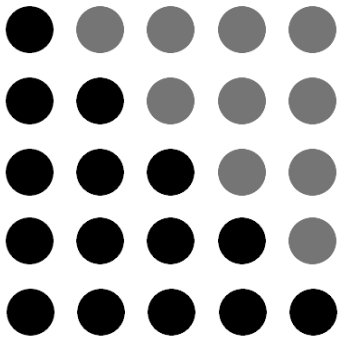
The communication of a cell is a diffusion of the state information to the neighborhood (Figure 2.4a). After the communication process of all cells is finished, each cell is updated by the embryo's MLP (one shared neural network for all cells). A combination of the state variables of a cell and chemicals of the cell's neighbors is used by the MLP update function to compute new state variables of the cell. All cells (the genotype) are updated in the same way (Figure 2.4b), until stopping criteria are met, i.e. the candidate picture (the phenotype) is similar enough to the target picture (Figure 2.4c). The similarity measure is reflected by the objective function.



(a) Cell's state update.



(b) Genotype's state update.



(c) Resulting phenotype.

Figure 2.4: A cellular representation.

2.2 Update Function

The update function tends to simulate **Reaction-diffusion model** proposed by Turing [9]. The model is a system which depends on chemicals transported over space by diffusion guided by local chemical reactions.

2.2.1 Diffusion of Chemicals

On the phenotype level (picture representation) the diffusion is equivalent to the Gaussian Blur filter. Gaussian Blur smudges the image by the Gaussian function in a specified radius (Figure 2.5). Here, the radius number refers to the size of an affected neighborhood, i.e. how far the information about the cell's chemicals reaches. The Gaussian Blur is linearly separated and is applied in two passes, vertical and horizontal direction. For further information about Gaussian Blur filter see [8].

On the genotype level, the diffusion is applied to every cell sequentially, first in the horizontal pass and then in the vertical pass. A cell undergoing the diffusion asks for the chemicals of its neighbors (either west and east neighbors during the horizontal pass or north and south neighbors during the vertical pass). The cell then computes its new chemicals, where i-th chemical in the chemical vector is computed as $(neighborsChemicals_1[i] + 2 * previousChemicals[i] + neighborsChemicals_2[i])/4$, where $neighborsChemicals_1$ is a weighted chemical vector from either eastern side (in horizontal pass) or southern side (in vertical pass) and $neighborsChemicals_2$ is a weighted chemical vector from either western side (in horizontal pass) or northern side (in vertical pass). The weighted chemical vector from one side is computed as $\sum_{i=1}^{neighbors}(chemicals_i * sharedLength_i/sideLength)$, where $sideLength$ is the length of the cell's side and $sharedLength_i$ is the length of the edge that is shared by the cell and its i-th neighbor. If the cell does not have any neighbors on the first or second side, the $neighborsChemicals_1$ or $neighborsChemicals_2$ respectively is substituted by zero value or static border color value.

2.2.2 Reaction

The diffusion step is followed by a reaction step of the embryo's neural network. The neural network is a MLP, so called '**MLP Controller**[3]'. The MLP Controller optimizes the embryo by the unsupervised learning paradigm because it has the input (target pattern), the cost (fitness) function (Section 2.5), and

[3]MLP Controller is MLP that controls the embryo state, hence it is referred to as the MLP Controller.

Figure 2.5: Example of a Blur filter.

the output (the phenotype of the embryo). The fitness function is a function of the input and the output and the MLP tries to minimize the function.

In the reaction to the diffusion, new states and chemicals of the cells are computed by MLP Controller in discrete time. From the new state variables, the input values for objective function are computed by MLP Controller as well. Thus the resulting phenotype of the embryo is implicitly computed from the MLP weights (real values) that form the genotype of the embryo. Thus optimization of the MLP Controller means optimization of the '**MLP weights**' thus optimization the cells' state variables and eventually optimization of the fitness value. The MLP Controller weights are evolutionarily optimized by CMAES evolution strategy (Section 2.6).

The number of weights in MLP depends on the number of state variables of the embryo . The number of parameters of the controller depends on the number of state variables and the required number of neurons in the hidden layer of the controller perceptron (Equation 2.1). The number of states, chemicals, and hidden neurons are system properties and their appropriate values depend on the complexity of an evolved pattern. The specific values were found empirically and are a compromise between accuracy and speed of the optimization process.

The MLP weights could be formally divided into three groups. $(4 * c + s + 1) * hn$ weights belong to the first group. These weights are used by the update function for computing the values of hidden neurons. The second group contains $(hn + 1) * (c + s)$ weights that are used for computing new states and chemicals from the hidden layer. The last group contains output weights. A half of the output weights $(s + c + 1)$ refers to so called '**color weights**' used for expressing the color value from the new state variables. The resulting color value of a cell is a weighted combination of states and chemicals of the cell (Equation 2.5). The remaining part of the output weights $(s + c + 1)$ refers to so called '**structure weights**' used for expressing a 'growing action' value (Section 2.3) from the new state variables. The

computation of the growing action value is similar to the computation of the color value (Equation 2.6).

MLP Controller's update function runs for each cell in the synchronous way until termination condition is met (Section 2.7). The input for the Controller is the chemical vector consisting of the chemicals of the current cell with chemicals of its neighbors, and the state vector consisting of the states of the current cell. The inputs to the hidden layer are the states, and neighbors' chemicals but not the chemicals of the current updating cell. All input values are connected to all hidden neurons. Similarly, all hidden neurons are connected to all output values. Output values consist of new states and chemicals of the cell. Furthermore, the layer of neurons computing the output chemical values has another input also - the chemicals of the current updating cell that were omitted for the hidden layer (Figure 2.6). The output value for each of the hidden neurons and perceptron final outputs are computed by applying the activation function - hyperbolic tangent of the sum of all its weighted inputs together with the neuron's previous value.

It is worth noting that the MLP Controller modifies states and chemicals of the embryo thus it optimizes the diffusion coefficients also.

$$colorWeights = s + c + 1 \tag{2.1}$$

$$structureWeights = s + c + 1 \tag{2.2}$$

$$mlpWeights = (4 * c + s + 1) * hn + (hn + 1) * (c + s) \tag{2.3}$$

$$parameters = colorWeights + structureWeights + mlpWeights \tag{2.4}$$

where s is the number of states,

c is the number of chemicals,

hn is the number of hidden neurons.

$structureWeights = 0$ WHEN ONLY THE COLOR IS OPTIMIZED,

$colorWeights = 0$ WHEN ONLY THE STRUCTURE IS OPTIMIZED.

$$color_i = \frac{1 + \tanh\left(w_{c_i} c_i + w_{s_i} s_i\right)}{2} \tag{2.5}$$

where i is the i-th cell,

c_i is its chemical vector,

s_i is its state vector,

w refer to weights.

$$action_i = \frac{1 + \tanh\left(w_{c_i} c_i + w_{s_i} s_i\right)}{2} \tag{2.6}$$

where i is the i-th cell,
c_i is its chemical vector,
s_i is its state vector,
w refer to weights.

To summarize, the transfer function of the automaton is guided by the neural

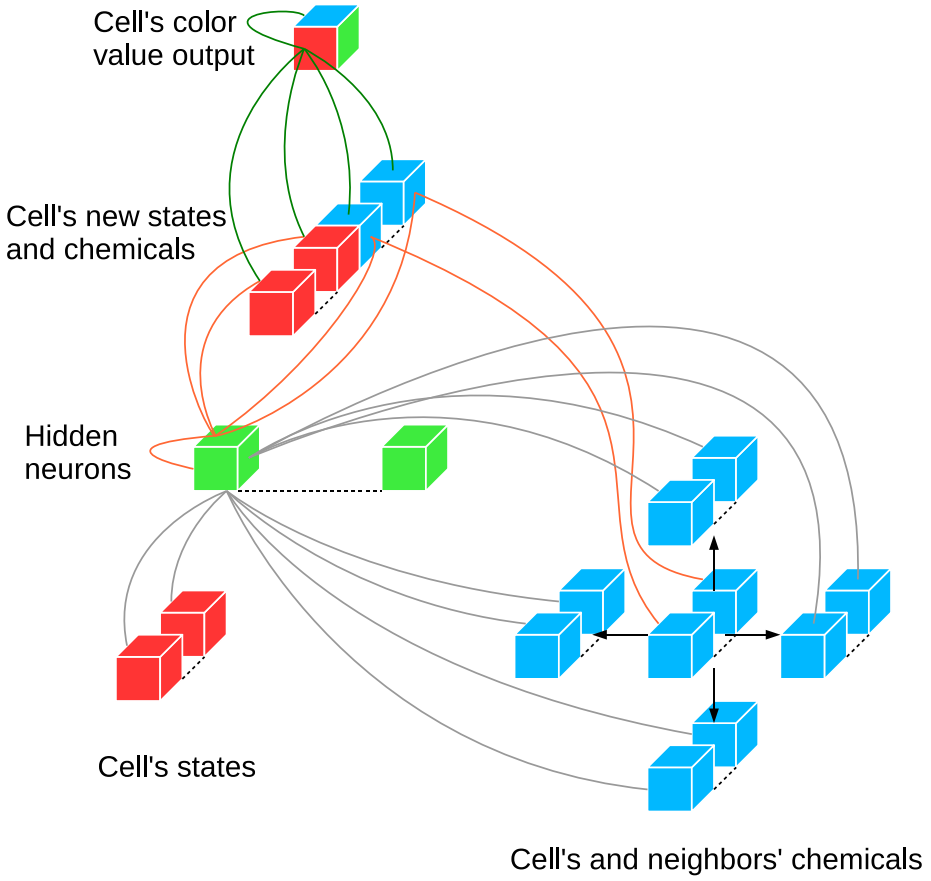


Figure 2.6: A scheme of the MLP Controller computing values of one cell.

network (MLP) and thus the success of the automaton evaluation depends on accurate values of weights of the neural network. Therefore the weights create the space for searching and optimizing by the evolution strategy.

Other applicable approaches to update the CA, which were not tested in this work, are for example NEAT (Neuroevolution of Augmenting Topologies)

and ESN (Echo State Network) Controller with combination of CMAES optimizer [5].

NEAT is an evolutionary algorithm designed to optimize both parameters and topologies of feed-forward or recurrent neural networks. It is based on applying three key techniques: tracking genes with history markers to allow crossover among topologies, applying speciation to preserve innovations, and developing topologies incrementally from simple initial structures.

ESN is a recurrent neural network with a sparsely connected hidden layer. Such sparsely connected networks are often referred to as a reservoir computing. The topologies and weights of hidden neural layer are assigned randomly and fixed. The weights of the output neurons can be trained and the network can learn a specific pattern.

2.3 Variable Cells Grid

Initially, the grid of cells was static. It means that the cells were changing only their state variables and thus the color according to the optimized MLP weights. Later on, more complex requirement was set upon the embryo - to optimize the structure. Optimizing the structure of a picture means adaptation of cells to the structure of the picture. The structure of the picture are contours in the picture, the borders of various components in the picture. Thus the optimal case would be that the cells would adapt to the picture's various color components, and each cell would match a different component at the end. However, our grid is not so flexible because the grid is allowed to consist of only square cells. Thus if the picture has not such optimal distribution of components that each cell could match a different component, it is not possible to obtain a perfect result.

There were two main reasons to optimize the structure. The first reason was that we wanted to find out whether the cellular representation is capable of structural optimization. If so, the second reason was a hypothesis that if the topology of the cellular representation would adapt to the target, the genotype should shrink and speed up the latter color optimization. Namely, it could speed up the evaluation run of an embryo because less cells would perform less operations than more cells, thus less cells would insist less computation time. It is worth noting that the final goal was multiobjective, namely to optimize both the color and the structure of a desired target pattern.

The inspiration for a dynamic structure of cells came from J. F. Miller who introduced Cartesian Genetic Programming in the French flag problem [11].

He used similar cellular representations and the updating engine. In his model, each cell's program decides on the amount of produced chemical, whether it will live, die, or change to a different cell type at the next time step, and how it will grow. Growing into another cell means overwriting its properties completely. Besides the three color states, there is also another state - dead. When a cell dies, it means that it does not act any more. Miller's work has inspired us to do experiments with a dynamic grid of cells.

H. de Garis in his work 'Artificial Embryology and Cellular Differentiation' [15] used two dimensional shapes formed by a colony of cells in reproductive cellular automata as embryos. The idea was to evolve reproduction rules for CA, such that the final shape of a colony of cells would match a desired shape as closely as possible. Each cell contains a differentiable chromosome, which consists of four 'operons'. Each operon contains a condition field and an action field. These operons can switch on and off over time. The sequential operon switching controls the growth of an embryo. If a cell matches one of the conditions of an operon, then the corresponding action is activated and its instructions executed. The matching is computed from the cell's state which is computed from the cell's previous state and from states of the cell's neighbors. The state of a cell was defined in terms of the configuration of its neighborless side(s) because only such cells can reproduce (there are 14 different states).

Both these works showed the capability of a dynamic irregular cells structure to learn a desired shape, i.e. optimize the target flag. In our work, we used a dynamic structure but we did not use the irregular grid with "holes" (the "holes" are the dead cells in Miller's work and the empty spaces in Garis' work). The grid is a full rectangle with the specified dimensions. At the beginning, each cell corresponds to one pixel of a target picture. The cells are allowed to do one of the three actions: merge together, divide, or keep still (do nothing). The desired growing action of a cell is determined by state variables and computed by the MLP Controller (Equation 2.6). The default desired action of the cell - 'do nothing' is then changed to 'dividing' action, if the condition according to Equation 2.7 is valid or set to 'growing' action, if the condition according to Equation 2.8 is valid.

$$action < minimum + \frac{range}{3} \tag{2.7}$$

where $action$ is the action value returned by the MLP Controller,

$minimum$ is the lower bound of $range$,

$range$ is the range of values the MLP Controller can return.

$$action >= maximum - \frac{range}{3} \tag{2.8}$$

where *action* is the action value returned by the MLP Controller,
maximum is the upper bound of *range*,
range is the range of values the MLP Controller can return.

If the action of a cell is set to dividing, the cell divides into 4 identical square cells where surface of each cell is one fourth of the surface of the parent cell. Each cell inherits the state variables identical with the parent cell.

If the action of a cell is set to growing, the cell asks its neighbors if they would like to grow as well and if so, they merge together. Four neighboring cells are allowed to merge together if they have identical dimensions and the result would be a square cell. Resulting state variable is computed as the average of the merging cells state variables.

2.4 Static Edge Cells

Edge cells or border cells are special cells that lie at the borders of the flag. They cannot change their dimensions (they are of size one all the time) and have fewer neighbors than the remaining cells. In the experiment [5] with one flag, the border cells' chemicals were assigned to *null* and such *null* values were skipped by the update function. Because the focus of this work is optimization of more than one pattern, the border was given a new role - it gives the embryo a specific impulse different for each target flag. The impulse or signal remains the same while optimizing a flag. In order to be able to send such distinct and distinguishable signals, the initialization of the border must be done efficiently.

A hypothesis is that the border information could be sufficient for the embryo to distinguish and learn to optimize multiple targets.

2.4.1 Border Initialization Methods

The values of the border cells remain constant for each flag optimization. We used several ways of initialization of the border. One way is to initialize the border to the same value in the range (0,1). This 'Scaling' method assigns the border values to zero for the 0-th picture and for any next j-th picture to $j\frac{1}{n-1}$ where n is the number of pictures (Figure 2.7a).

Another approach was to initialize only some of the border cells with values 0 or 1. There are several options. One option is so called 'alternation'. Alternation assigns i-th border values to 1 if $i\%n = j$ where n is the number of target pictures, j is the order of the current target picture and $\%$ is the modular division (Figure 2.7b).

Another option is so called 'cutting'. Cutting assigns i-th border values to 1 if $i < j\frac{size}{n-1}$ where n, j is the same as before and $size$ is the size of the border, i.e. number of border values (Figure 2.7c). The condition is the same as in the first option and thus if the number of pictures is one, all the border values are assigned to 0.

The last and first option assign values of 1 to all border cells for the last picture and values of 0 to all border cells of the first picture.

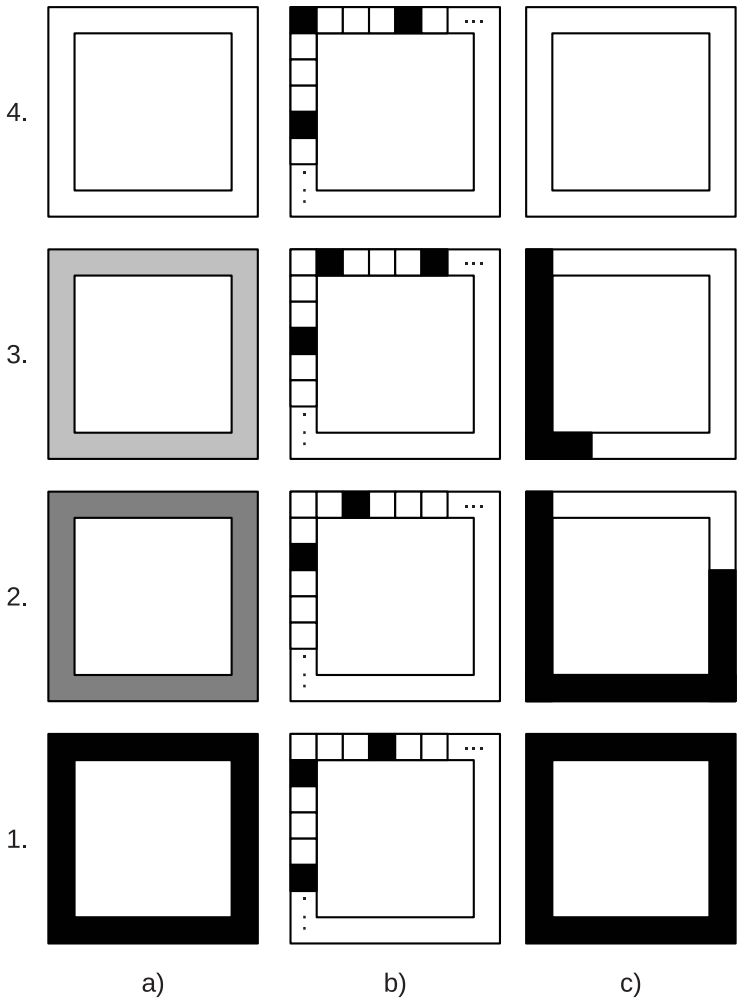


Figure 2.7: Various border cells' values assignments

2.5 Objective Function

Objective (fitness) function and its resulting fitness value reflect an optimizing variable (color or structure). In numerous experiments, we performed a multiobjective optimization. Namely, we either optimized the color of multiple flags or the structure and the color of one flag. We did not use the usual multiobjective approach and optimized all the objectives simultaneously, but we optimized the multiple objectives sequentially. It means that when we optimized multiple color patterns, we optimized the first flag, then we optimized the next one, etc. However, the optimization of one flag is likely to be at the expense of the optimization of the other flag(s). Despite the fact, the effort remained to do the multiobjective optimization with a sequential approach. However, we have explored several heuristic functions that were counting with all or at least some of the fitness values of the optimized patterns. Therefore we disrupted the classical sequential optimization because we were not neglecting remaining objective values while optimizing one objective.

When we used the dynamic grid and optimized both the structure and the color, we performed the optimization sequentially again, namely the structure first and then the color. However, in the structure-color optimization, the optimization of the structure should not be at the expense of the color, as it was in the case of the optimization of multiple color patterns. Firstly, it is because the structure optimization shrinks the genotype size thus accelerates the search process, and secondly, we maintained two sets of MLP weights thus the color optimization was not overwriting the MLP weights previously optimized for the structure, but it optimized its own set of MLP weights dedicated to the color optimization.

During the sequential optimization, the embryo optimizes the structure fitness value first. Eventually, the optimization ends and the best MLP weights are saved. The embryo loads these weights and evaluates itself to get the optimized structure. The embryo fixes the structure (the cells cannot change their dimensions any more) and starts to optimize the color. After the color optimization, another set of optimized MLP weights is stored.

During the simultaneous optimization we kept two ordered vectors of fitness values, the color fitness values vector and the structure fitness values vector. The final fitness values vector (coming into the CMAES optimization process (Section 2.6) is a partial order of the Pareto dominance.

While optimizing the color pattern, we used the color fitness function described in Section 2.5.1. While optimizing the structure pattern, we used the structure fitness function described in Section 2.5.2. While optimizing both - the color and the structure pattern, we used both these fitness functions.

2.5.1 Color Fitness

While optimizing the color, the resulting phenotype of the embryo is a color picture. Fitness value of the embryo is determined by a distance between the target picture and the phenotype of the embryo. The picture expressed from the embryo is represented by the color values computed by the MLP Controller for each cell as in Equation 2.5.

The distance d between pictures A and B is a normalized squared distance error between their pixels' color values as in Equation 2.9. The fitness value (Equation 2.11) is the distance value normalized by so called 'maximal distance' which is a property defined for any picture as in Equation 2.10.

$$d_{AB} = \sum_{i=1}^{pixels_A} \sum_{j=1}^{pixels_B} (A_{i,j} - B_{i,j})^2 \tag{2.9}$$

$$d_A^{max} = \sum_{i=1}^{pixels} \sum_{j=1}^{pixels} (max\,(A_{i,j}, 1 - A_{i,j}))^2 \tag{2.10}$$

$$fitness_B = d_{AB}/d_A^{max} \tag{2.11}$$

Color Fitness in Sequential Optimization of the Dynamic Grid

We pick up a random pixel in the target picture that the pixel is in the range of coordinates of a cell. For the cell we compute the color distance as a distance between the random pixel and the cell's color value (Equation 2.5). The square of the distance is multiplied by the cell's size what is equivalent to the color distance computed pixel by pixel under condition that the cell is covering a single color component. Such assumption is taken while optimizing sequentially, thus we assume that the cell grid should be optimally aligned to the color components of the target picture. The color fitness of the embryo is a normalized sum over all particular cells fitness values (Equation 2.12). The denominator in the Equation 2.12 is the normalization component. Because we sum over all cells, we normalize the fitness by the number of cells ($cells$). Because a cell color distance ($(colorCell_i - colorPixel_{random})^2$) is multiplied by the cell's size ($size_i$), we normalize with a maximum size of a cell in the

embryo ($size_{max}$). The resulting normalized fitness value is in interval (0, 1).

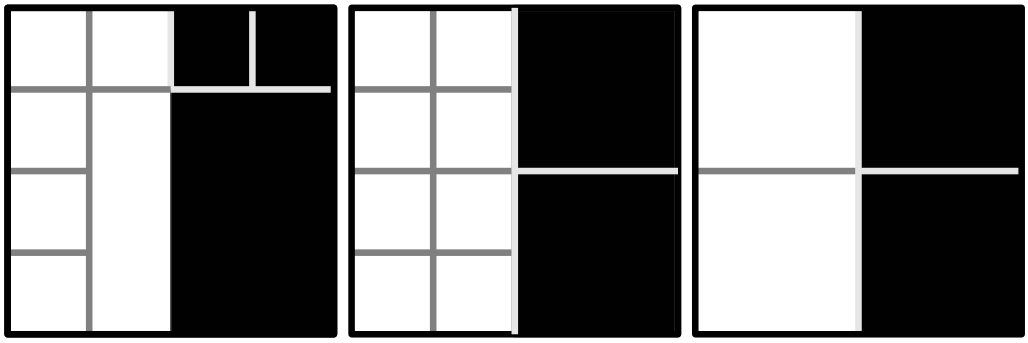
$$d = \frac{\sum_{i=1}^{cells} \left(colorCell_i - colorPixel_{random}\right)^2 * size_i}{size_{max} * cells} \tag{2.12}$$

where $cells$ is a number of cells in the embryo,
$colorCell_i$, is a color value of the cell,
$colorPixel_i$, is a random pixel in the picture covered by the cell,
$size_{max}$ is the maximum size of a cell in the embryo.

2.5.2 Structure Fitness

While optimizing the Structures, the resulting phenotype of the embryo is a color picture. The optimal final grid structure of the embryo is such a structure where borders of cells are aligned with contours in the picture, i.e. where borders of cells cover individual color components. However, the optimal alignment is limited by square proportions of cells. While optimizing only the structure, the color is neglected, thus the phenotype picture is formed only by the cells' edges. While optimizing both the color and the structure, the phenotype picture is a color picture with enhanced contours thus the cells edges.

If a cell is not aligned properly within components, thus for instance the cell lies in two components of different color, it is desirable to penalize the cell. The size of the penalty depends on the number of pixels of the cell that are outside of the component. In the computation of the structure fitness, we worked only with monochromatic pictures, thus the cell can cover only one or two components (black component and white component). Therefore, we count the number of white and black pixels that the cell is covering in the target picture. The smaller number of them represents the part of the cell which is outside of the desired component and forms the first component of the cell's penalty (Figure 2.8a). The other component of the cell's penalty counts with the overall number of cells. The effort is to minimize the number of cells, thus embryos with a wasting amount of cells are penalized (Figure 2.8b). An example of the optimally evaluated structure is in Figure 2.8c. The number of black and white pixels was counted for each cell and the total number of pixels for the embryo was counted as well. Then the structural distance d between the embryo and the picture was calculated according to Equation 2.13. The denominator in the Equation 2.13 is a normalization component. The first part of the denominator ($min\left(\sum_{i=1}^{cells} black, \sum_{i=1}^{cells} white\right)$) normalizes the first component of the objective functions ($\sum_{i=1}^{cells} min\left(black_i, white_i\right)$): The number of pixels of a cell that are off an original picture's component, cannot



(a) Example of optimized embryo structure consisting of one big cell 3x3 px and 7 small 1x1 px cells. The big cell consists of 3 white pixels and 6 black pixels. The smaller part - 3 white pixels form the cell's penalty.

(b) Example of a penalization when 8 small cells cover a single component.

(c) Example of an optimally evaluated embryo structure.

Figure 2.8: A hypothetical example of an evolutionary design optimization task.

be bigger than the minimum number of pixels of both colors. The second part of the denominator (*height*) normalizes the second component of the objective functions ($\frac{cells}{width}$): The overall number of cells cannot be bigger than the overall number of pixels. The resulting normalized fitness value is in interval (0, 1).

$$d = \frac{\sum_{i=1}^{cells} min\left(black_i, white_i\right) + \frac{cells}{width}}{min\left(\sum_{i=1}^{cells} black, \sum_{i=1}^{cells} white\right) + height} \tag{2.13}$$

where *cells* is the number of cells in the embryo, *width*, *height* are dimensions of the embryo, $black_i$, $white_i$ is the number of black/white pixels respectively per each cell, *black*, *white* is the total number of black/white pixels respectively in the embryo.

The first part of the sum represents the minimum number of pixels that are conflicting, i.e. they must be of a different color than the color of the cell thus such cell should divide. The second part of the sum aims to minimize the number of cells in the CA in order to favor larger cells over smaller if they cover an area of the same color.

The fitness values described above are raw values and are tuned by a penalty. The penalty depends on the number of evaluations of the embryo

update function (Equation 2.14). The final fitness value is computed as in Equation 2.15.

$$penalty = \frac{evaluations}{evaluations_max_allowed} \tag{2.14}$$

$$fitness = fitness * (penalty * penalty + 1) \tag{2.15}$$

The distance (fitness) minimization was performed using the MLP Controller (Section 2.2).

2.6 CMAES Optimizer

CMAES (Covariance Matrix Adaptation Evolution Strategy) is an algorithm for continuous optimization. It is suited to solve ill conditioned and non-separable problems. The principle is similar to the Principal Component Analysis but the set of search points is not specified but sampled iteratively. Using evolution strategy, CMAES learns the covariance matrix of the search points, which is proportional to learning the inverse Hessian matrix in the Quasi-Newton method $C \propto H^{-1}$ and it is equivalent to reducing any ellipsoid function to a spherical one. Estimation of the covariance matrix is the estimation of the maximum likelihood.

CMAES plays an important role in our embryogenic flag implementation because it optimizes the search space and samples new points within it.

CMAES uses $\left(\mu/\mu_{\{I,W\}}, \lambda\right)$ survival selection evolution strategy with μ parents, with recombination of all μ parents, either Intermediate (I) or Weighted (W) recombination, and λ offspring. For further details about CMAES refer to the tutorial of the CMAES' author N. Hansen [6] available on-line at [7] with the CMAES framework also.

2.6.1 Sampling New Generation

New search points for generation g are sampled from multi-variate normal distribution with covariance matrix C and mean m zero (Equation 2.16). In the language of the Evolution Strategies, the sampling equals to the mutation operator. Each of the λ best individuals undergoes a mutation where a random Gaussian vector x_i of dimension N is added to the individual. Dimension N equals to the number of controller parameters. All the variables are updated

during the optimization process.

$$x_i \sim \mathcal{N}_i\left(m, \sigma^2 C\right) = m + \sigma\mathcal{N}_i\left(0, C\right) \quad for\ i = 1, ..., \lambda \tag{2.16}$$

where $x_i, m \in \mathbb{R}^n, \sigma \in \mathbb{R}_+, C \in \mathbb{R}^{n \times n}$,
the x_i is a random Gaussian vector,
the mean vector m represents the current fittest solution,
the so-called step-size σ determines the scale of the search,
the covariance matrix C determines the shape (main directions)
of the search.

2.6.2 Selection

The population was sorted according to the fitness of its individuals. The best μ parents are selected from the newly sampled population and weighted intermediate (arithmetic) recombination was applied upon them. The size of population λ (Equation 2.20), and parent number for selection μ (Equation 2.21), depend on the number of controller parameters (if their values are not specified explicitly). The weights used in Equation 2.18, in Equation 2.24, or in Equation 2.27 are assigned either equal for no selection pressure, or linearly decreasing with fitness value (Equation 2.22), or logarithmically decreasing with fitness value (Equation 2.23).

$$y_{i:\lambda}^{g+1} = \frac{x_{i:\lambda}^{g+1} - m^g}{\sigma^g} \tag{2.17}$$

$$m \longleftarrow \sum_{i=1}^{\mu} w_i x_{i:\lambda} = m + \sigma \sum_{i=1}^{\mu} w_i y_{i:\lambda} \tag{2.18}$$

$$w_i \geq ... \geq w_\mu > 0, \quad \sum_{i=1}^{\mu} w_i = 1 \tag{2.19}$$

$$\lambda = 4 + \lfloor 3 * \ln N \rfloor \tag{2.20}$$

$$\mu = \frac{\lambda}{2} \tag{2.21}$$

$$w_i = \mu - 1 \tag{2.22}$$

$$w_i = \log \mu + 1 - \log i + 1 \tag{2.23}$$

2.6.3 Covariance Matrix Adaptation

Covariance Matrix of a random vector x is $C = \mathbb{E}\left[(x - \mathbb{E}(x))(x - \mathbb{E}(x))^T\right]$.

Rank μ update

By incorporating the selected μ best parents, we estimated a new covariance matrix as in Equation 2.24.

By estimating a covariance matrix for a new generation, the term $(1 - c_{cov})$ was exponentially increasing for each older generation. Because $(1 - c_{cov}) \in \langle 0, 1\rangle$, the weights of older generations were lower than the weights of recent generations. Such covariance matrix update is called rank μ update because the sum of outer products in Equation 2.24 is of rank μ.

$$C^{g+1} \longleftarrow (1 - c_{cov}) + c_{cov} \sum_{i=1}^{\mu} w_i y_{i:\lambda}^{g+1} \left(y_{i:\lambda}^{g+1}\right)^T \tag{2.24}$$

Learning rate c_{cov} ($c_{cov} \leq 1$) controls the changing rate of the covariance matrix C explicitly. The learning rate parameter is the inverted 'backward time horizon'. It considers how much to the past we were looking.

However, this was not the final estimation of the C. The adaptation can be enhanced by exploiting dependencies between consecutive steps applying cumulation, as described below.

Adapting the step-size σ

With a constant step-size σ, the algorithm is likely to stagnate in the neighborhood of a global minimum. In order to prevent this, the step-size was adaptable. Adapting the step-size means adapting the scale of the search and sampling isotropically at each step. Thus, the adaptation represents the trade-off between exploration and exploitation. Adaptation of σ was guided by the theory of progress rate. σ was updated in logarithmic scale because it is unbiased at the scale.

$$\ln \sigma^{g+1} = \ln \sigma^{g} + \frac{c_\sigma}{d_\sigma \mathbb{E}\left\|\mathcal{N}(0, I)\right\|} \left(\left\|p_\sigma^{g+1}\right\| - \mathbb{E}\left\|\mathcal{N}(0, I)\right\|\right) \tag{2.25}$$

where c_σ is the learning rate parameter for the cumulation for the step-size control, $c_\sigma < 1$, d_σ is the damping parameter which scales the magnitude of $\ln \sigma^{g+1}$, $d_\sigma \approx 1$.

Expression in the brackets compares the size of the evolution path p with a desired size that is the expected size of a vector sampled from the identity.

The evolution path p is the sum of the successive consecutive steps (Equation 2.26). This cumulation path exploits the correlations between consecutive steps and it is used for estimation of the new covariance matrix.

$$p_{\sigma}^{g+1} = (1 - c_{\sigma})\, p_{\sigma}^{g} + \sqrt{c_{\sigma}\,(2 - c_{\sigma})\,\mu_{eff}}\, C^{-\frac{1}{2}} \frac{m^{g+1} - m^{g}}{\sigma^{g}} \tag{2.26}$$

where μ_{eff} (Equation 2.27) is 'variance effective selection mass'.

$$\mu_{eff} = \left(\sum_{i=1}^{\mu} w_i^2 \right)^{-1} \tag{2.27}$$

The larger is μ_{eff} the smaller is the change rate of m.

It is worth noting that applying '$C^{-\frac{1}{2}}$' rescales the ellipsoid function to the sphere and makes the expected length of the oriented vector of the evolution path independent.

Rank 1 update

Rank 1 update adds the matrix of a rank one to the covariance matrix. For this purpose, we used the evolution path (Equation 2.26) but without applying rescaling because we intended to deform the shape of C in the direction of the evolution path (Equation 2.28).

$$p_{c}^{g+1} = (1 - c_{c})\, p_{c}^{g} + \sqrt{c_{c}\,(2 - c_{c})\,\mu_{eff}}\, \frac{m^{g+1} - m^{g}}{\sigma^{g}} \tag{2.28}$$

where c_c is the learning rate for cumulation for the rank one update, $c_c \leq 1$.

Then the update is as in Equation 2.29.

$$C^{g+1} \longleftarrow (1 - c_{cov}) + c_{cov} \left(p_{c}^{g+1}\right) \left(p_{c}^{g+1}\right)^{T} \tag{2.29}$$

Final estimation of the covariance matrix

By combining rank 1 update, Equation 2.29, with rank μ update, Equation 2.24, we obtained the final estimation of the C as in Equation 2.30.

$$C^{g+1} = (1 - c_{cov}) + \frac{c_{cov}}{\mu_{cov}} \left(p_{c}^{g+1}\right) \left(p_{c}^{g+1}\right)^{T} + \tag{2.30}$$

$$+ c_{cov} \left(1 - \frac{1}{\mu_{cov}}\right) \sum_{i=1}^{\mu} w_i y_{i:\lambda}^{g+1} \left(y_{i:\lambda}^{g+1}\right)^{T}$$

where μ_{cov} parameter characterizes the trade-off between rank *one* and rank μ update.

2.7 Monitoring Termination Conditions

The overall termination of the evaluation of population of embryos is guided by CMAES optimizer which is monitoring several conditions such as the stability of the mean value, range of the best fitness values, condition number of the covariance matrix, standard deviation of the distribution, the step-size, etc.

The termination of one individual evaluation (MLP Controller's update function) was guided explicitly by a maximum number of the function runs. The number of performed iterations was present in the calculation of the fitness function also and the solutions with lower number of iterations were considered to have better quality than the solutions with higher number of iterations.
Implicitly, the stopping signal was guided by **Energy Monitor**. The Energy Monitor monitored the standard deviation of the 'energies' over the number of consecutive steps (a time window). The 'energy' of a cell was a sum over squares of all state variables of the cell. The energy of the embryo was the sum of the energy values of all its cells as in Equation 2.31. If the energy deviation was lower than a given threshold than we believed that we have reached the steady-state and our stopping criteria were met.

$$E_t = \sum_{i=1}^{cells} \left(\sum_{j=1}^{chemicals} chemical_j^2 + \sum_{k=1}^{states} state_k^2 \right) \tag{2.31}$$

The standard deviation was then calculated over the time window (Equation 2.32). Author of the algorithm is D.Knuth [4].

$$m_e = \frac{\sum_{t=1}^{flags} E_t}{flags} \tag{2.32}$$

$$where\ m_e\ is\ the\ estimated\ mean$$

$$m_{te} = \sqrt{\frac{\sum_{t=1}^{flags} \delta_t \left(E_{te} - m_t\right)}{flags - 1}} \tag{2.33}$$

$$where\ E_{te} = E_t - m_e;$$

$$\delta_t = E_{te} - m_e - m_t$$

$$m_t = \sum_{t=1}^{flags} \frac{\delta_t}{flags - 1}$$

Alternatively, during the optimization of the structure, the information about growth or division was stored. If no growing or dividing occurred in the embryo over the time window, the update process stopped.

The fitness scoring values achieved by the energy stopping criterion showed better or at least as good results as by using just the fixed maximum number

of iterations, see [5]. However, such fixed maximum number of iterations was used as the upper bound. If the energy deviation did not sink below the threshold during the given number of maximum steps, the individual was not likely to be able to reach the steady-state and thus its updating stopped and the individual received the worst fitness reward.

Chapter 3

Implementation

This chapter describes the implementation of the cellular embryogenic representation described in the Chapter 2. The source code can be found on the attached CD as well as on the embryo project homepage [37].

The program is implemented in C++ under Linux. It is open-source distributed under The GNU General Public License. Besides the standard C++ libraries, it uses 'Cairo' and 'GNU Scientific Library' also. The implementation consists of three main parts:

1. embryo
2. optimizer
3. viewer

There are three version of the framework:

1. embryo - optimization of color pattern
2. embryo2 - optimization of multiple color patterns
3. embryo3 - optimization of structure and color pattern

Further on, the implementation of the embryo3 version will be discussed because the embryo3 is the latest version and it is built upon the previous embryo versions.

The structure of the 'embryo' and the relationships between the 'embryo' classes are shown as a UML (Unified Modeling Language) class diagram in Figure 3.1. A library called 'libembryo' is build up from the 'embryo' classes and used by both, the optimizer and the viewer.

3.1 Embryo

The main class of the program is obviously called 'Embryo'. Embryo has several important attributes objects (instances of classes):

- 'mCellContainer' of type 'CellContainer'. The 'CellContainer' is an alias ('typedef') of 'std::list<Cell* >', thus the 'mCellContainer' is a container of 'Cell' objects. The 'Cell' class has four such container (for each neighboring side): 'mUpperCell', 'mLowerCell', 'mLeftCell', and 'mRightCell'. Besides, each 'Cell' object holds also 'mStateVector', 'mPrevStateVector', 'mChemicalVector', and 'mPrevChemicalVector'. These are of type 'double*', thus an array of double precision floating point numbers. Initially, such array of all cells as real values was a property of the embryo. However, such implementation was sufficient unless we wanted to optimize the structure thus changing the dimensions of the cells. Dynamic grid structure required more complex implementation of a cell as an object.

- 'mMonitor' of type 'Monitor'. This is the energy monitor of the embryo.

- 'mController' of type 'Controller'. This is the MLP Controller of the embryo.

- 'mTargetPic' of type 'Picture*' and 'mCandidatePic' of type 'Picture'. 'mCandidatePic' is the phenotype of the embryo and 'mTargetPic' is an array of target patterns used by fitness function computation.

- 'mBorderVector' of type 'double*', what is an array of border cells value, which is used in optimization of multiple color patterns.

3.1.1 Update Function

The embryo update function is the most important method in the embryo class or library. It calls the following important functions:

- 'diffuseChemicals()', for each cell 'blurFirstPass()' is called, then for each cell 'blurSecondPass()' is called. This is the two pass diffusion, (Section 2.2).
  For each cell:

  - 'cell→update()', update of the cell, the function returns the actual values of old and new state variables as well as the chemicals of the neighbors.

  - 'controller→update()', update of the embryo MLP controller, the response to the diffusion, (Section 2.2).
  - 'controller→structure()', the controller is asked for the action value if we are optimizing the structure. Then 'cell→setDividing()' or 'cell→setGrowing()' is called; these functions set the cell for dividing or growing action respectively, if the controller forces it (Section 2.3).

- 'updateStructure()', the function is called if optimizing the structure and guides growing and dividing of the cells. Inside the function, following functions are called for each cell:
  - 'cell→wantsGrow()', the function returns a true value if the cell would like to grow and following functions are called after that:
    - 'cell→getGrowingDirections()', the function returns true value, merging directions of the cell, for which the function has been called, and pointers to neighbors intended for merging. If the merging is not possible, the function return false value.
    - 'changeNeighborhood()', if the previous function returns true value, thereby it must return also the cells that want to merge together (see merging conditions in Section 2.3), the current function changes the neighbors of the new cell as well as the neighbors of the neighbors of the old cells (an object 'Cell' contains 4 vectors of pointers to the neighbor cells, see the diagram Figure 3.1).
    - 'cell→removeMeGetState()', the function is called for each cell intended for merging and modifies the resulting state vector, which will be then assigned to the new cell as its own state vector.
  - 'cell→wantsDivide()', the function returns a true value if the cell would like to divide and the following function is called after that:
    - 'cell→divide()', if the size of the cell is bigger than one, then the function divides the cell into four identical square cells with state variables values identical to the parent cell.

- 'monitor→next()', the function calls 'embryo→energy()', which returns an energy of the embryo and stores the energy into the time window (Section 2.7)

The 'embryo→update()' finally returns a boolean value of function 'monitor→hasNext()'. The value is false when standard deviation of energies over the time window is too low ($\leq 1e-10$), and in that case, the

update process would stop. Otherwise the value is true and the update is repeated again.

3.2 Optimizing the Embryo

3.2.1 User Interface

A Linux executable program of the embryo optimization is called 'embryo-optim'. Executing 'embryo-optim –help' provides usage information. The 'embryo-optim' has the following arguments on the input:

- -c, CMAES configuration file, e.g. '-c cmaes.conf'
- -p, experiment folder, e.g. '-p ./'. The path must contain embryo configuration file 'embryo.conf' and target picture file(s) (must be in PGM format). The target pictures names as well as the parameters of the embryo (dimensions, monitor window size, number of hidden neurons of the MLP controller, number of states and chemicals) are specified inside the embryo configuration file.
- -s, optional seed value for the initialization of CMAES

During the optimization process, there is always an actual information about the number of generation and the generation's best fitness value on the standard output. In older versions, the fitness value of each individual with the number of iterations to meet the stopping criterion of the development is printed also. After updating all individuals of one generation, the best fitness value of the generation with the number of all evaluations up to this point appears.

The decreasing fitness value and iterations/generations number under the limit value refer to the convergence of the system.

When the optimization is over, the resulting optimized MLP weights will be stored in the folder path specified by the argument. When optimizing only the color, one set of weights is stored in 'best.params'. When optimizing the structure also, the second set of weights is stored in 'best.cells.params' in the same location.

3.2.2 Inside the Code

The embryo is instantiated by the function 'Embryo::load()' according to the parameters specified by the configuration file. The optimization starts with the CMAES initialization 'cmaesInit()' that sets up the CMAES parameters

according to specification in the other configuration file and prepares an array of n fitness variables of double type, where n is a size of population thus the size of the MLP Controller parameters. Until CMAES does not terminate the optimization of the population, the following process is repeated in a loop:

- New population is sampled (Section 2.6.1).
- For each individual the 'embryoEvaluate()' function is called and returns the fitness value for the individual.
  'embryoEvaluate()' sets up the MLP Controller parameters and calls the following embryo functions for each of the target flags:
  - 'embryo→initCell()' during structure optimization or 'embryo→init()' during color optimization; 'initCell()' initializes the structure and both eventually initialize the cells state variables.
  - 'embryo→resetMonitor()', monitor is reset.
  - 'embryo→update()', the update of the embryo in a loop until the monitor does not stop the update or the evaluation number reaches the maximum number of evaluations.
  - 'embryo→getSimilarity()', the function obtains a similarity value between the embryo and the target pattern, then adds the penalty to the similarity and returns the fitness value (Section 2.5).
- If a better (lower) fitness is located in the array of all fitness values, the fitness is remembered as the best ever achieved and the parameters are stored in an external file.
- The distribution is updated by CMAES function 'cmaes_UpdateDistribution()' (Section 2.6.2).

If we are optimizing not only the color but the structure as well, the process is repeated in two rounds, first time for the structure optimization and second time for the color optimization. The two resulting sets of MLP controller parameters are stored in two distinct external files.

3.3 Watching the Evaluation and Its Graphical Result

After the successful evaluation, the last line of the output should read a generation number smaller than the maximum (1000) and a fitness close to zero. Then one would like to visualize the evaluation of the embryo.

Graphical viewer ‘embryo-view’ (executable under Linux in terminal) can visualize the results recorded in the file ‘best.params’ and ‘best.cells.params’. The output is a window with the embryo color state and the structure state guided by the MLP Controller. In the previous versions, graphical output for each of the states (chemicals, states, color) is displayed in the window.

The ‘embryo-view’ has the following arguments on the input:

- -e, the same embryo configuration file, with which the ‘embryo-optim’ has been initialized before, e.g. ‘-e embryo.conf’
- -p, the file path with the best color parameters, e.g. ‘-p best.params’
- -s, the file path with the best structure parameters, e.g. ‘-p best.cells.params’
- -t, optional target file, e.g. ‘-t twobands90.pgm’ (the actual fitness will be displayed in the top of the window if this parameter is specified)

If the development was successful the required target is produced at the end. Its steady-state can be verified by typing the ‘N’ key thus adding some Gaussian noise into the cells’ states.

Press ‘Esc’ to quit the viewer.

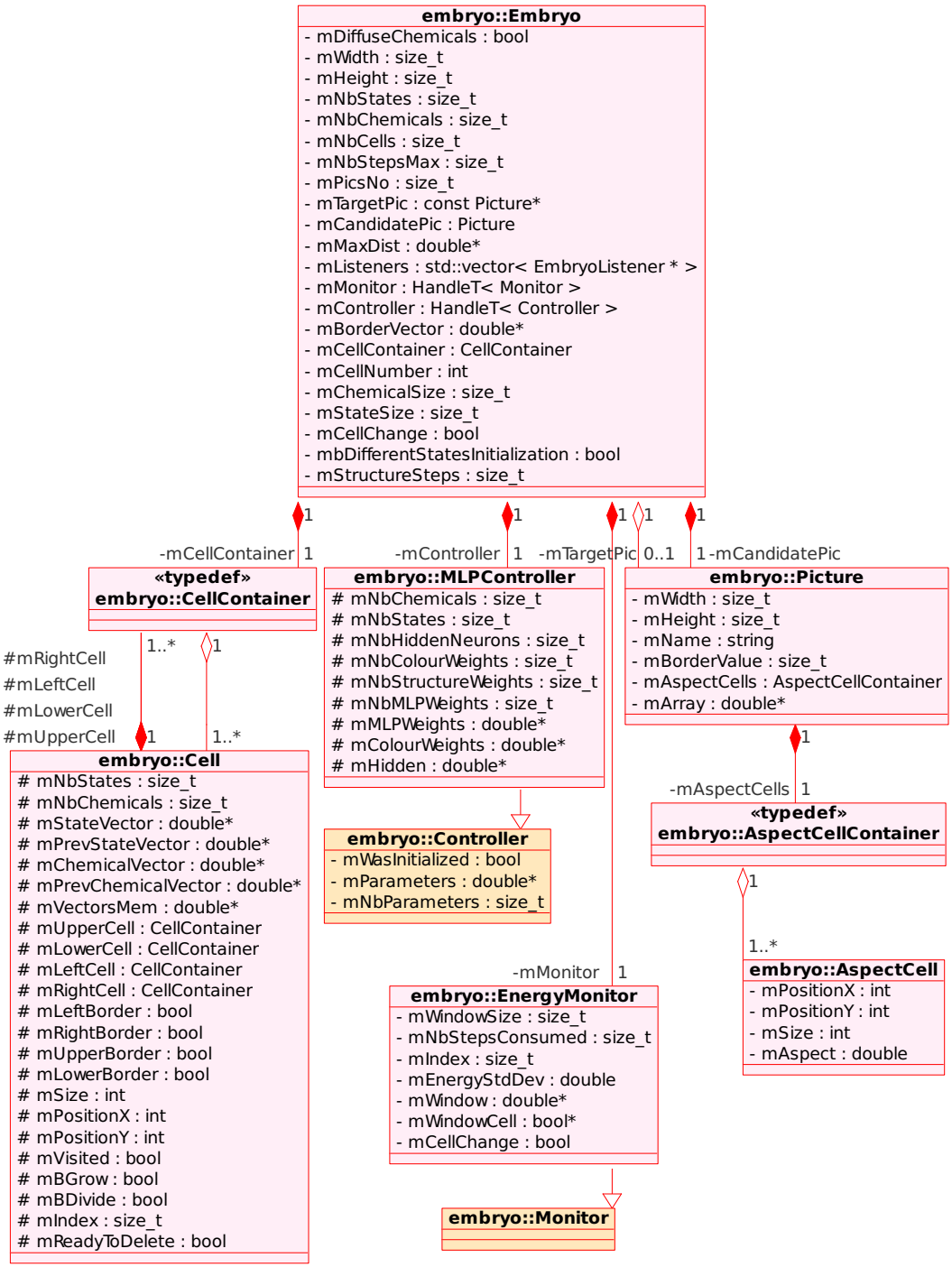


Figure 3.1: UML diagram of selected 'embryo' classes with their attributes.

Chapter 4

Experiments

This chapter describes the experiments and their conditions that we carried out by using the model described in Chapter 2.

All the input parameters of embryogenic flag and CMAES parameters are specified by configuration files. We used the default values for most parameters. These were tuned previously by the authors of CMAES and the work [5], which we were building upon.

4.1 Experimental Conditions

Unless otherwise stated, the experiments presented further on were performed under the following conditions:

Static grid experiments. Sixteen replicate experiments were performed for every setting with standard default (9,18) CMAES evolution with population size 18. All runs were allowed at most 1000 generations. The experiments ran in foursomes on several quad-core processor computers in the cluster network of LRI[1], each CPU running at 1808 MHz and 2GB memory or 1995 MHz and 3GB memory. The development of every evolutionary run took several hours, sometimes more than one day.

Dynamic grid experiments. 32 replicate experiments were performed for every setting with standard (50,100) CMAES evolution with explicitly specified population size 100. All runs were allowed at most 1000 generations. The experiments ran on one dual-core processor authors laptop with Intel CPU U7300 running at 1.30GHz and 4GB memory, and on multiple computers of

[1]Laboratoire de Recherche en Informatique in Paris

Shared Robotics Laboratory of FMFI UK[2] and FEI STU[3]: one dual-core Intel Core2 Duo CPU E4700 running at 2.60GHz and 2GB memory, one quad-core AMD Athlon II X4 620 running at 800MHz and 1.5 GHz memory, one dual-core Intel CPU 2140 running at 1.60GHz and 1GB memory, and one simple-core Intel Pentium 4 CPU running at 3.00GHz and 1GB memory. The development of every evolutionary run took at most several hours.

4.2 Initialization

After reading two configuration files, the CMAES optimizer and the embryo with its controller and Monitor are initialized.

4.2.1 Initialization of the Embryogenic Cells

The height and width of the embryo and also optimized flag was specified by a configuration file. We used 32x32 pixels for experiments with static grid and 33x33 pixels for experiments with dynamic grid. The number of states and chemicals for every cell of an embryogenic flag was specified as one state and two chemicals. Number of hidden neurons is assigned from the configuration file also and fixed to 8 for our purpose.

The number of parameters of the controller (the number of weights in the neural network) was assigned to 111 for the color or the structure optimization experiments and to 115 for both, the color and the structure optimization experiments (107 for the update function, 4 for the color expression function, and 4 for the growing action expression function (Equation 2.1).

All state variables were real values initialized to zero. However, the experiments with initial values either one or random values in the range (0,1) were also performed to see whether different initial conditions would influence the final result. The various initializations showed no significant difference in results.

The Monitor energy deviation was initialized to 1 and energy threshold to 1^{-10}. The maximum number of update runs as well as the time window was read from the configuration file. We fixed 1024 (32^2) as the maximum number of updates and we considered the history energy of 8 flags backwards. If the Energy Monitor did not stop the update after 1024 runs, the update function ended and assigned fitness was of value 1. The maximum number of evaluations 1024, as the square of the target picture side size, meant that the

[2]Faculty of Mathematics, Physics and Informatics of Comenius University in Bratislava

[3]Faculty of Electrical Engineering and Information Technology Slovak University of Technology in Bratislava

information from one corner of the grid comes to the other corner of the grid at most 32 times (if the grid was static and the cells did not change their sizes). After such a number of state variables exchanges, the diffusion was complete and the state variables of all cells were identical. Further diffusion of the state variables would be redundant thus the update stopped.

4.2.2 Initialization of the CMAES

CMAES optimizer has internal and external (strategy) parameters $\lambda, \mu, w_{i=1..\mu}, c_\sigma, d_\sigma, c_c, \mu_{cov}, c_{cov}$. Parameters can be changed by configuration file. The initialization of all CMAES optimizer's property values remained the same and default in all experiments, unless otherwise stated.

The genotype consisted of the search space dimension N equaled to the number of parameters of the controller thus $N = 111$ for the color or the structure optimization experiments and $N = 115$ for both, the color and the structure optimization experiments (Equation 2.1). Then the size of the population λ was set to 18 (Equation 2.20) and parent number chosen for selection μ to 9 (Equation 2.21). The weights of parent individuals were assigned in a logarithmic scale (Equation 2.23) and therefore the variance effective selection mass μ_{eff} to 5.6 (Equation 2.27).

4.3 Multiple Target Flags vs. Border

Four different target flags (Figure 4.1), were used for the experiments. They were grayscale 32x32 pixels anti-aliased pictures in format *pgm*. The color values were coded in the range $(0, 1)$, 0 for black, 1 for white.

The 'two bands 90' target with one half white and the other half black was the simplest flag pattern to learn [5]. Other flags were created by rotating this base flag by $\angle$ 30 (*two bands 30*), $\angle$ 45 (*two bands 45*) or inverting the colors - 'two bands inverted' target flag.

The values of border cells' states were dependent on the number of target flags. They were assigned as in Figure 2.7c. However, the experiments with initial values either one or random values in the range (0,1) were also performed to see whether different border value conditions would improve the final result.

4.3.1 Fitness Function Tuning

In the experiments optimizing multiple color patterns, the overall fitness value is the average of all the particular fitness values, thus the average of the sum of the fitness values for each optimized pattern. However, the experiments

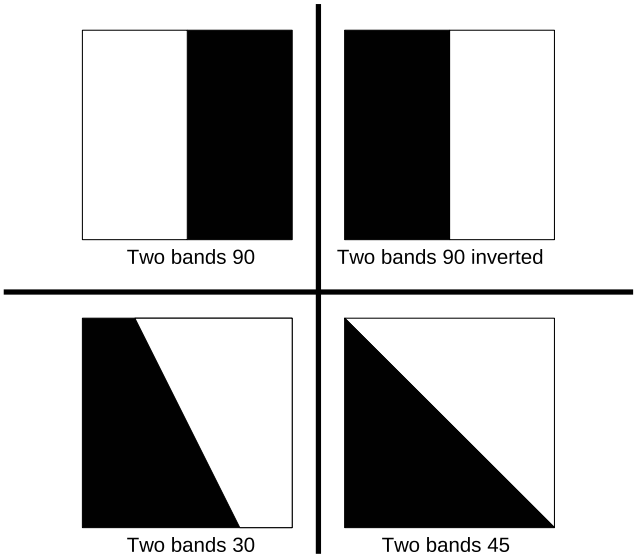


Figure 4.1: Target flags used for the experiments.

with other fitness assignments (e.g. fitness value as the worst of all particular fitness values) were performed to find out whether different fitness functions could improve the optimization of multiple flags. Of course, the goal remained the same, namely to obtain a system able to restore all the optimized flags as good as possible.

With the 'averaging' fitness, the optimization function was linear (Figure 4.2). To obtain better results with the sequential multiobjective optimization, the experimental heuristic fitness function, which counts with mutual relationships between the fitness values of the optimized patterns, was used. Thereby, we were not neglecting the other fitness values while optimized one objective. We were looking at the best fitness value and the worst fitness value at least. Because we wanted to optimize all flags simultaneously, we defined a penalty as a difference between the single fitness values that we tried to minimize to the zero. Another penalty counted for the ratio of fitnesses where the effort was to maximize the ratio to value one. Finally the focus was on minimizing the worst fitness. By minimizing the worst fitness all other fitnesses were minimized also because they could not be higher than the worst fitness value.

The penalty values can be scaled by weights. The weights were sine and cosine functions. These functions change the behavior of the fitness function and make it steeper descending with the decreasing fitness values (Figure 4.3). The result was that a shape of the fitness function was similar to a funnel

where all the solutions were likely to fall into the funnel's neck. Thus fitness values for all the flags were decreased somehow simultaneously.

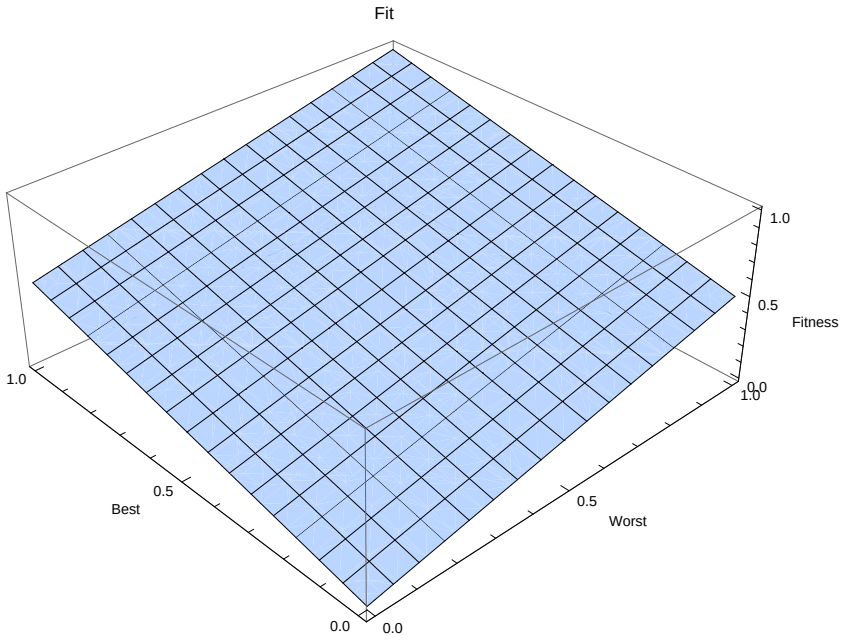


Figure 4.2: Simple averaging fitness landscape.

4.4 Variable Cells Grid

The first set of experiments was optimizing only the structure while the color was neglected. Four different target flags were used in the experiments: 64x64 pixels 'Chess' pattern shown in Figure 4.4, the 'two bands 90', modified 'two bands 90', and modified 'two bands 45' patterns from Figure 4.1. The difference between the modified patterns and the previously used patterns was that the two modified flags were of size 33x33 pixels and the 'two bands 90' consisted of 18 px wide black stripe and 15 px wide white stripe.

The second set of experiments was sequentially optimizing both - the color and the structure pattern. We used the two 33x33 pixels target patterns described above, and Equation 2.12 was used for the color fitness computation.

After the evaluation of the structure, the system loaded the best achieved MLP Controller weights optimized for the structure. With these weights, the embryo evaluated itself, then fixed the structure and switched to the color optimization.

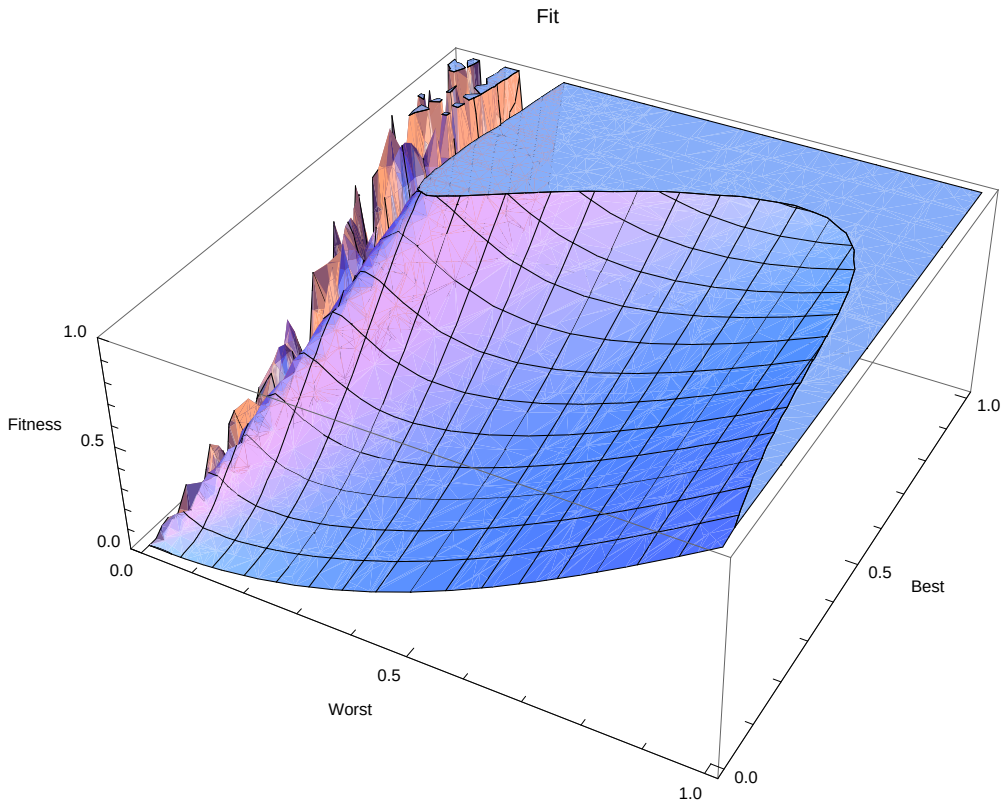


Figure 4.3: Heuristic fitness landscape.

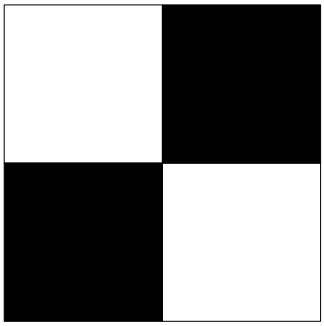


Figure 4.4: 64x64 pixels ‘Chess’ target flag.

Chapter 5

Results and Discussion

The previous work of A.Devert [5] showed the system's capability to optimize any given flag pattern and to reach the stable attractor. Even with different relative proportions the system preserves the pattern and restores the flag.

This work focused on changing the border of the flag as the holder of the impulse for specific target flag and thus optimize multiple flags. Another focus was on changing the static grid structure to the dynamic grid in order to optimize the structure pattern.

5.1 Optimizing 1 Flag

5.1.1 Two Bands Flag Evaluation - Comparison

Figure 5.1 shows the evaluation (the average of best achieved fitness values) of the three flags - 'two bands 90' and its two rotated versions - 'two bands 30' and 'two bands 45'. In the previous work [5] 'two bands 90' flag has appeared to be the simplest color pattern to optimize. The experiments with rotated versions showed that these flags are not so easy to learn as the original 'two bands 90'. The difficulty of learning was probably caused by increasing the number of black-white crossings. The anti-alias filter smudged the picture thus putting some grayscale pixels around these crossings. These parts were harder to learn because of the transfer function - hyperbolic tangent.

The embryo with static cellular grid showed good and stable results for the optimization of one flag. After the development, the cellular flag has been capable of recovering from significant amount of damage.

However, all of the given flags could be learnt by automata and after the optimization ended, resulting automata could restore the flags quite reliably after approximately 400 cycles of the MLP updates (Figure 5.2).

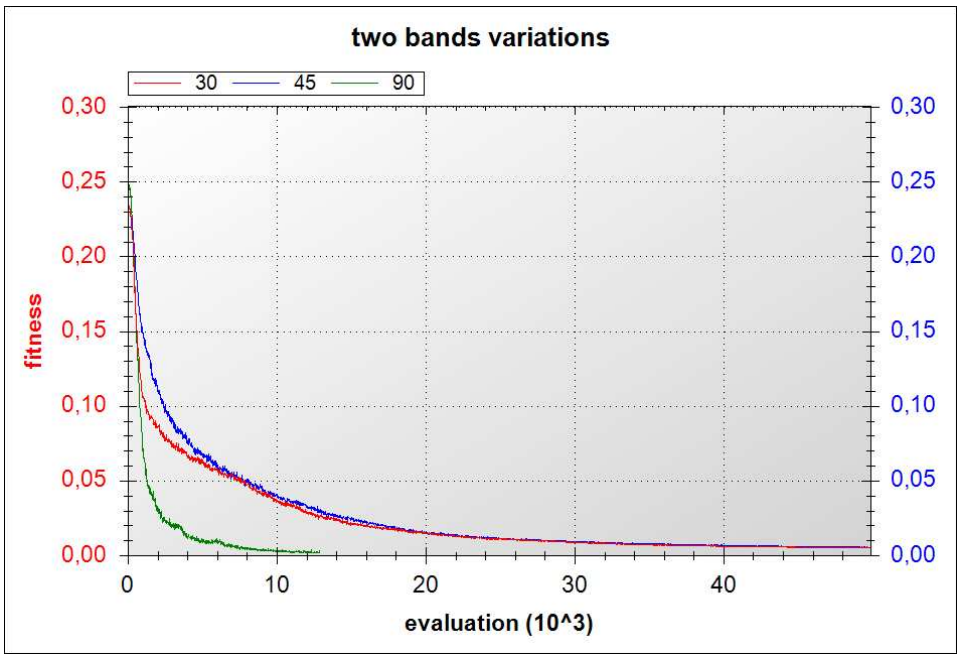


Figure 5.1: Comparison of the evaluation of average best fitness values of two bands flags.

Figure 5.2: Two bands flags restored after 400 cycles of the MLP updates.

5.2 Optimizing 2 Flags

To achieve multiple flags optimization, the method involving the border as the holder of specific information for each flag was used. Results show that the task to optimize two targets is solvable by this method but it is not sufficient for more than two target flags.

In the experiments with two flags, the border values were initialized to zero values for the first flag and to values of one for the second flag. The fitness function was computed as an average of a sum over the two single fitnesses - one fitness value for the first flag optimization and the other one for the second flag optimization.

5.2.1 Two Bands 90 versus Two Bands Inverted

These experiments optimized 'two bands 90' flag and 'two bands inverted' flag. The results showed that only approximately half of the experiments converged to a fixed point able to restore both pictures (Figure 5.4a). The average best fitness value of the half of the experiments (8 out of 16), in which both fitness values converged is shown in Figure 5.3. Some of the embryos (7 out of 16)

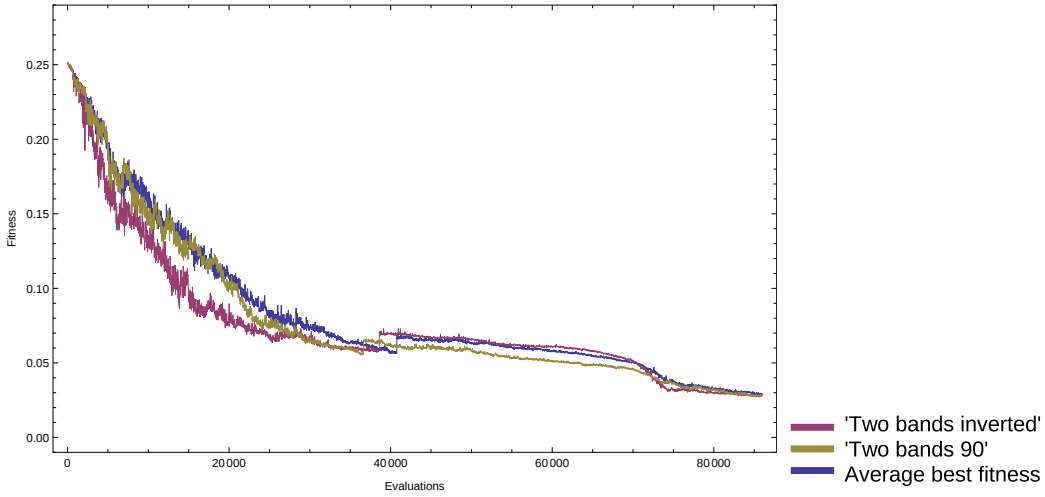


Figure 5.3: Evaluation of an average best fitness for 'two bands 90' and 'two bands inverted' optimization; convergence of both fitness values

were able to restore just one flag (Figure 5.4b), namely five of the embryos were able to restore only the 'two bands 90' and the remaining two of the embryos were able to restore only the 'two bands inverted' flag. One of the optimized embryos could restore neither one (Figure 5.4c). Such automata ended in a loop. The energy deviation over the searched history computed by the monitor (Equation 2.32) went down to the minimum thus the update function ended prematurely. These automata never achieved a good fitness and development process was stopped after the maximum number of evaluations was reached.

5.2.2 Two Bands 90 versus Two Bands 45

These experiments optimized two color patterns - 'two bands 90' flag and 'two bands 45' flag. The results were similar to the results of experiment described above. Approximately half of the experiments (7 out of 16 experiments) converged to the fixed point able to restore both flags (Figure 5.6a). Some of them were able to restore just one flag (7 out of 16) (Figure 5.6b), namely six of the embryos were able to restore only the 'two bands 45' and the remaining one

of the embryos was able to restore only the ‘two bands 90’ flag. 2 out of 16 optimized embryos were not able to restore any of the two flags (Figure 5.6c).

The average best fitness value of the 7 optimized embryos, in which both fitness values converged, is shown in Figure 5.5.

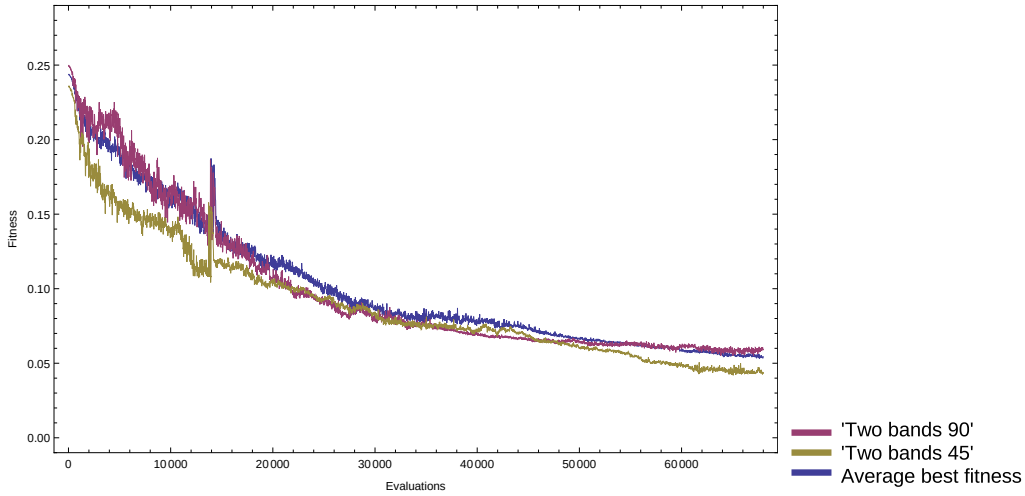


Figure 5.5: Evaluation of an average best fitness for ‘two bands 90’ and ‘two bands 45’ optimization; convergence of both fitness values.

5.3 Optimizing 4 flags

None of the experiments with four flags was successful. Not even a single flag could be restored.

The use of border as the only information specifying the target was not sufficient. Even the increasing number of states and chemicals or different border assignment (Figure 2.7) did not help.

Probably a different method must be used to address the optimization of multiple patterns.

5.3.1 Changing Fitness Function

The experiments with various fitness functions (Section 4.3.1) were performed. We tried tuning the objective function by heuristics and obtain more promising landscape. The development process of the automata could be accelerated by using the heuristics functions with the flow as in Figure 4.3. However, the fitness function was not steep over the whole scope and becomes flat around zero. Because of that, a flag similar to the target pattern was obtained quickly,

but the detailed tuning of the flag did not happen (Figure 5.8). Thus despite the fact, that the heuristic functions showed increased speed in optimization process, they did not show any improvement in achieved fitness values. The reason could be, that the initial function was not really "hard" to optimize, it was not multimodal but linear.

5.4 Variable Cells Grid

Experiments with dynamic cell structure were performed to find out whether our representation is capable of structure optimization and further capable of both the color and structure optimization. As in the previous experiments, we used the same CMAES evolution strategy for the perceptron weights optimization.

5.4.1 Structure Optimization

The structure experiments started with the well known flag 'two bands 90'. The ideal final state for the optimization of the 32x32px 'two bands 90' flag were four squares 8x8 px. This state was easily achieved after few evaluations, the resulting graphical output is shown in Figure 5.9a. Because we wanted to try to optimize more difficult structures, we modified the 'two bands 90'. The irregularity to the target pattern (different stripe sizes in the 'two bands 90' pattern) was made in order to force the embryo to create cells of different sizes. Basically, the cells were allowed to merge only into square cells, thus a range of cell sizes that could evolve using our target flags was quite limited: 1x1 px (initially), 2x2 px, 4x4 px, 8x8 px, 16x16 px and 32x32 px. The biggest cell able to evolve was a 64x64 px for the big pattern 'chess'. An example of an evaluated embryo for the 'chess' pattern is shown in Figure 5.9b.

The experiments to optimize the structure showed the capability of the embryo to adapt its structure to the structure of the target. Additional examples of resulting optimized structures of the 33x33 px flags are shown in Figure 5.10a and 5.10b.

Some interesting emergent properties can be seen on the 'two bands 45' result. Besides the expected diagonal, which stretches from the upper left corner to the lower right corner (as on the target pattern), another diagonal emerged on the resulting 'two bands 45' flags (notice that in the upper flag in Figure 5.10b shows the whole second diagonal and in the lower flag in Figure 5.10b approximately one half of the second diagonal was formed).

5.4.2 Structure and Color Optimization

The successes in structure optimization were a necessary condition for the optimization of both the structure and the color. Evaluations of average best fitness values of experiments performed to optimize the structure of 'two bands 90' (30 experiments) and 'two bands 45 ' (20 experiments) targets are compared in Figure 5.11. Evaluations of average best fitness values of experiments performed to optimize the color of the targets are compared in Figure 5.12.

The experiments showed the capability of the variable embryogenic cellular representation to optimize the structure and the color patterns successfully thus optimize multiple objectives. Even more, by exchanging the static cells grid for the dynamic cells grid, the optimization process sped up from few days to few hours (data not shown). In the end, the color optimization of the embryo with previously evolved structure was faster and more effective than the color optimization with static cellular grid. However, evaluations of the average best fitness values in the graphs (Figures 5.1, 5.12) show, that the optimization of one color pattern (Figure 5.1) took approximately the same number of evaluations as the optimization of the structure and the color pattern (Figures 5.11, 5.12). Thus, if we count the time consumed by the update function of the embryo in the terms of cells' interactions, the update of the embryo with bigger number of cells must take longer time than the update of the embryo with smaller number of cells. Therefore, the update of an individual with representation consisting of dynamic cells grid was faster than the update of an individual with static cells grid. Consequently, the evaluation time of an individual ((number of evaluations of the update function) * (time consumption of the update function)) should decrease under the assumption that the number of evaluations remains approximately the same.

Two bands 90

The resulting plot (Figure 5.11 and 5.12) shows that the structure was optimized faster and more efficiently than the color. Unfortunately, the color optimization was not as successful as in experiments optimizing only the color. While the fitness value in the color optimizing experiments sank to zero (Figure 5.2), the average best fitness sank at best to 0.005 (Figure 5.12). Also, the quality of solutions in terms of the self-repair was not as good as in the experiments with static grid. In some cases, only a small amount of noise distorted the flag pattern. However, the repeating noise addition helped to restore the required flag again. Such instability was probably caused by the irregular grid structure with a smaller number of cells than in the static regular grid with as many cells as pixels in the target picture. While addition of the noise in static

grid affects only a random number of cells that are of size one (corresponding one pixel), addition of noise in dynamic grid affects the same random number of cells that can be of size one or larger. Thus under the assumption that the size of the cells is larger than the initial size, a bigger number of pixels of the resulting picture is affected by color noise and consequently, the repair is more difficult.

Examples of the evolved graphical results are shown in Figure 5.13.

Two bands 45

The results of the optimization of the structure of 'two bands 45' target are similar to the results of the structure optimization of 'two bands 90' target (Figure 5.11). However, the evaluation of the average best color fitness for two bands targets (Figure 5.12) show that the color optimization of 'two bands 45' flag remained harder than the color optimization of 'two bands 90' flag (Figure 5.1). Examples of graphical results are shown in Figure 5.14. It is interesting that the ranking of the results according to the fitness value (Figure 5.14) was different for the two used fittness functions (Equation 2.11, Equation 2.12). Such difference is caused by the assumption that the structure was previously well optimized, and the cells are aligned with the color components of the picture (Section 2.5.1). However, the colors were assigned well within this evolved structure, because if two random black pixels are selected, one being covered by the upper left corner cell and the other covered by the lower right corner cell, the color distance between the two cells is zero - thus perfect.

A comparison of different resulting pictures and different fitness value assignments of the embryos optimized by different fitness functions is shown in Figure 5.15. Figure 5.15 shows that the result in Figure 5.15b is better than the result in Figure 5.15a from the view of one fitness function (Equation 2.11) ($0.117807 < 0.217229$) but not from the view of the other fitness function (Equation 2.12) ($0.0136053 > 0.00052863$).

The experiments with variable cellular grid showed the capability of the representation to optimize both the structure and the color. We incorporated two different fitness functions that computed the color fitness values. The first fitness function was used in the experiments optimizing the color pattern(s). The second fitness function was used only in the sequential optimization of the structure and the color and worked under condition that the structure was previously well optimized. Otherwise, the function would assigned a better fitness value to a worse result and thus the color optimization would not be as sufficient as if the first fitness function was used. However, under favor-

able conditions, the second function is useful and faster than the first fitness function. In any case, the second fitness function calculation would be not efficient in the simultaneous optimization of the structure and the color. We started to do experiments with simultaneous optimization by using the first fitness function. However, such optimization is still under investigation and conclusive results are not available yet.

5.5 Future Directions

There are several possibilities that could be examined in order to obtain better results in multiobjective optimization. The possibilities are in the areas of the neural network, cells merging possibilities, cells topology, and fitness function.

One approach could be to add an additional input to the perceptron. Namely, to add an additional input to the perceptron (a constant bias) related to specific picture similarly as the border is specifying the picture now. However, this special input would be ubiquitous (it would be visible for each cell during its update). Now, only the cells directly next to the border receive the information about the specific target. Because of the diffusion, they send the information further, but nevertheless, the information would never be as strong near the middle as it would be when the specific input was ubiquitously present. Another solution could be to use a Multi Objective Optimization approach, e.g. Pareto front approximation. Using such method means that we would use the same fitness functions for a single target as we used before but we would keep a set off all these fitness values per an evaluation step. The fitness remains the same but a method for the approximation of the Pareto front, the method which would be able to differentiate which set of fitness values is better than other, must be implemented. For the beginning, some standard algorithmic approaches, e.g. Non-dominated Sorting Genetic Algorithm-II (NSGA-II), could be examined.

Setting less strict constraints on cell merging could lead to improvements in dynamic structure optimization. Currently, the four neighboring cells can merge only into a square cell. Future attempts include changing the merging possibilities into rectangles, later to polygons, and furthermore allow to merge any number of neighbors simultaneously. Moving beyond simple polygons, we could further evaluate completely irregular shapes and try such representations as Voronoi diagrams, fractal representation, or custom irregular representation models.

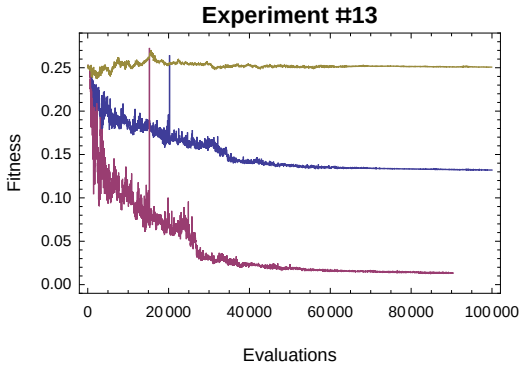
Structure versus color optimization is also a nice example where the multi objective optimization approach could bring fruitful results. The Pareto front would be approximated by an algorithm considering the coupled solutions -

one solution would consist of the fitness value and NN weights after structure optimization, its couple would consist of the color fitness values and corresponding NN weights. The single fitness computation might remain the same as before.

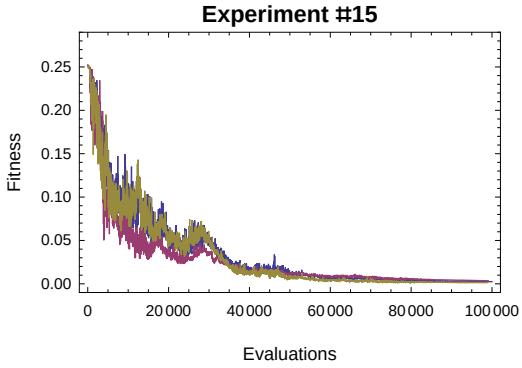
In addition, future work will consider the change of embryo parameters as increasing the size of monitor history window, population size, neural network controller parameters (more hidden layers with more neurons) and other parameter changes. Probably, we would also continue tuning the objective functions by heuristics thus obtain better results faster.

It could be possible to investigate various ranges of interaction by using different shapes of neighborhood as well as by changing the radius of the local neighborhood. This property might be particularly important in modeling complex engineering systems where local and highly nonlinear interactions among design elements are impossible to describe using traditional mathematical formulas.

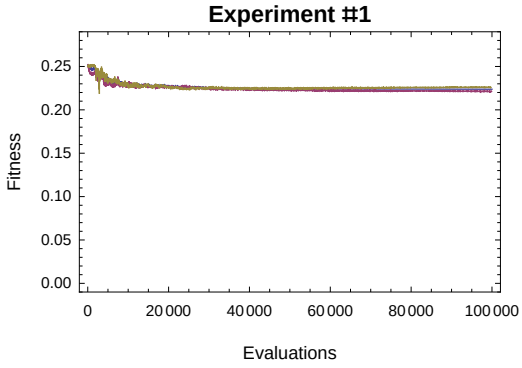
Another attempt would be to use a parametric representation of the target picture for the optimization. This would be a step towards the parametric representation of three dimensional (3D) objects and 3D optimization of structure and possibly also other objective(s). Such optimization is close to real application optimization problems and could build the bridge to practice.



(a) convergence of a single fitness value

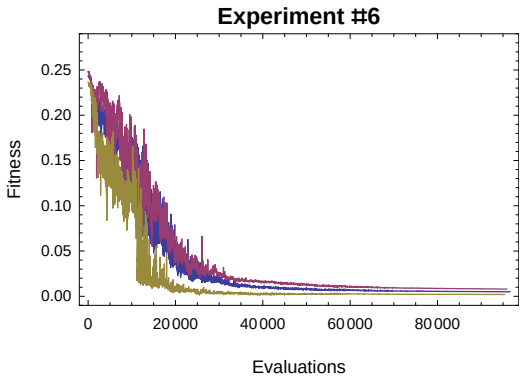


(b) convergence of both fitness values

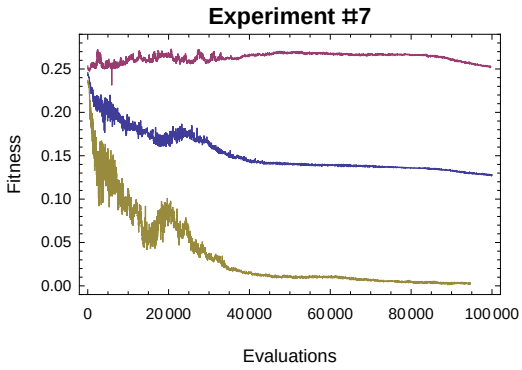


(c) divergence of both fitness values

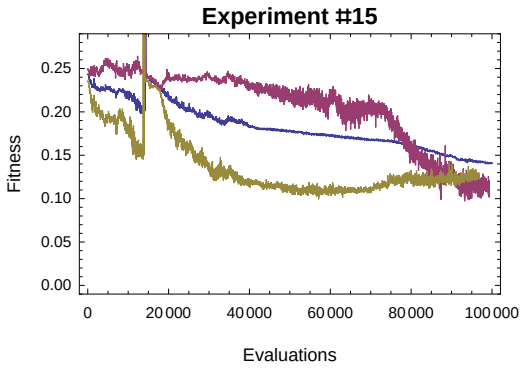
Figure 5.4: Selected examples of the best fitness evaluation for 'two bands 90' vs. 'two bands inverted' and the graphics output after 130 updates of the embryo.



(a) convergence of both fitness values



(b) convergence of a single fitness value



(c) divergence of both fitness values

Figure 5.6: Selected examples of the best fitness evaluation of 'two bands 90' vs. 'two bands 45' and the graphics output after 130 updates of the embryo.

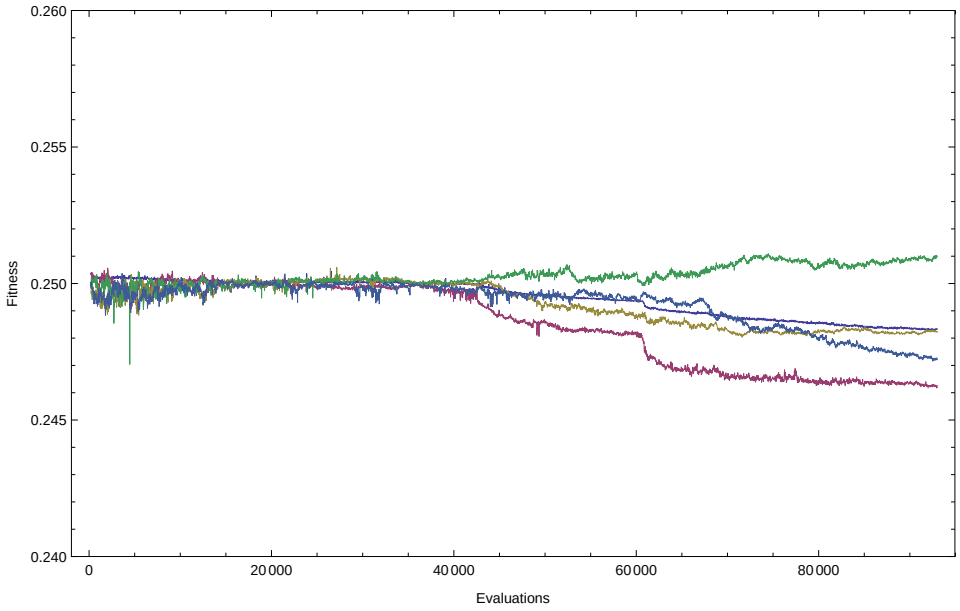
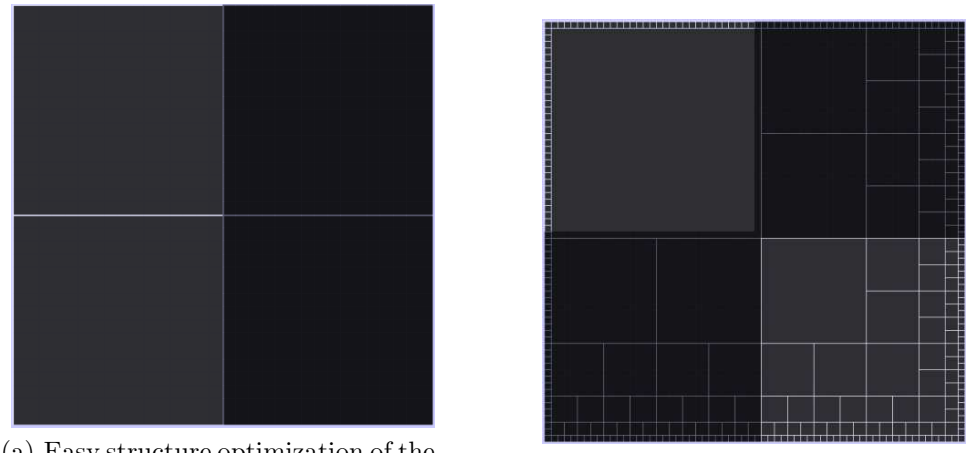


Figure 5.7: Optimizing 4 flags: no successful result, the average best fitness. Each color represents a fitness value of different a target.

(a) after 20 updates

(b) after 200 updates

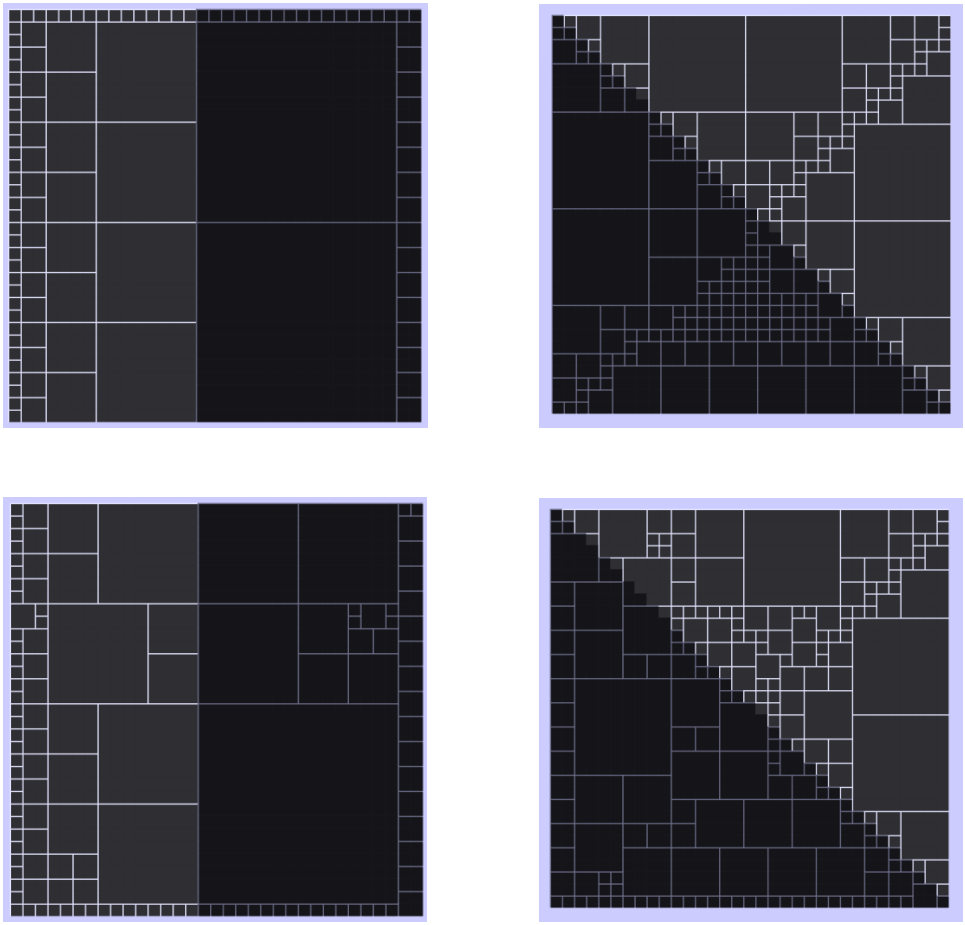
Figure 5.8: Early converge by using the heuristic fitness function.



(a) Easy structure optimization of the pattern 'two bands 90'

(b) optimizing the 'chess' pattern

Figure 5.9: Examples of resulting optimized flags.



(a) ‘two bands 90’ (b) ‘two bands 45’

Figure 5.10: Examples of resulting optimized flags.

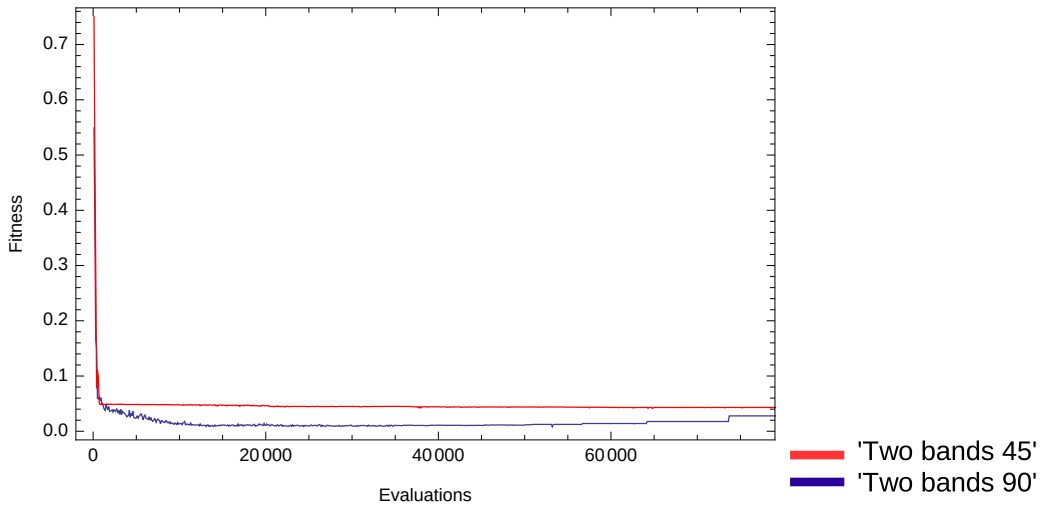


Figure 5.11: The average best fitness values achieved by optimization of the structure of modified two bands flags.

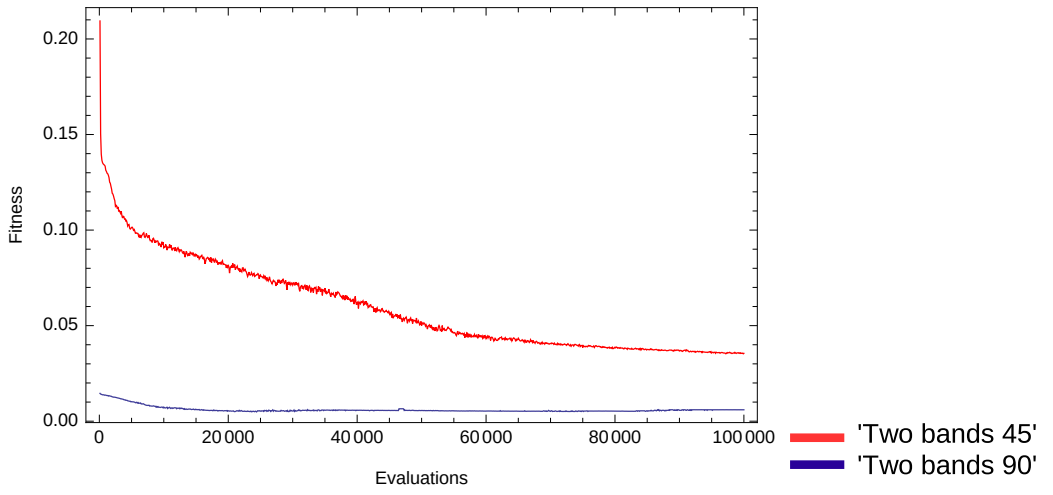
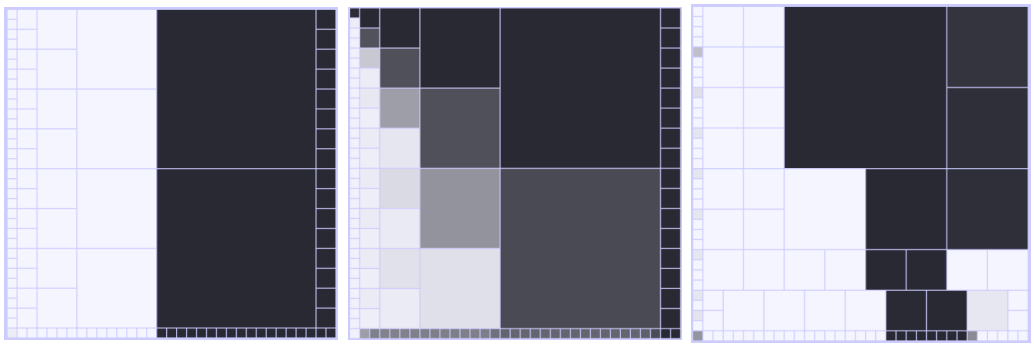


Figure 5.12: The average best fitness values achieved by optimization of the color of modified two bands flags.



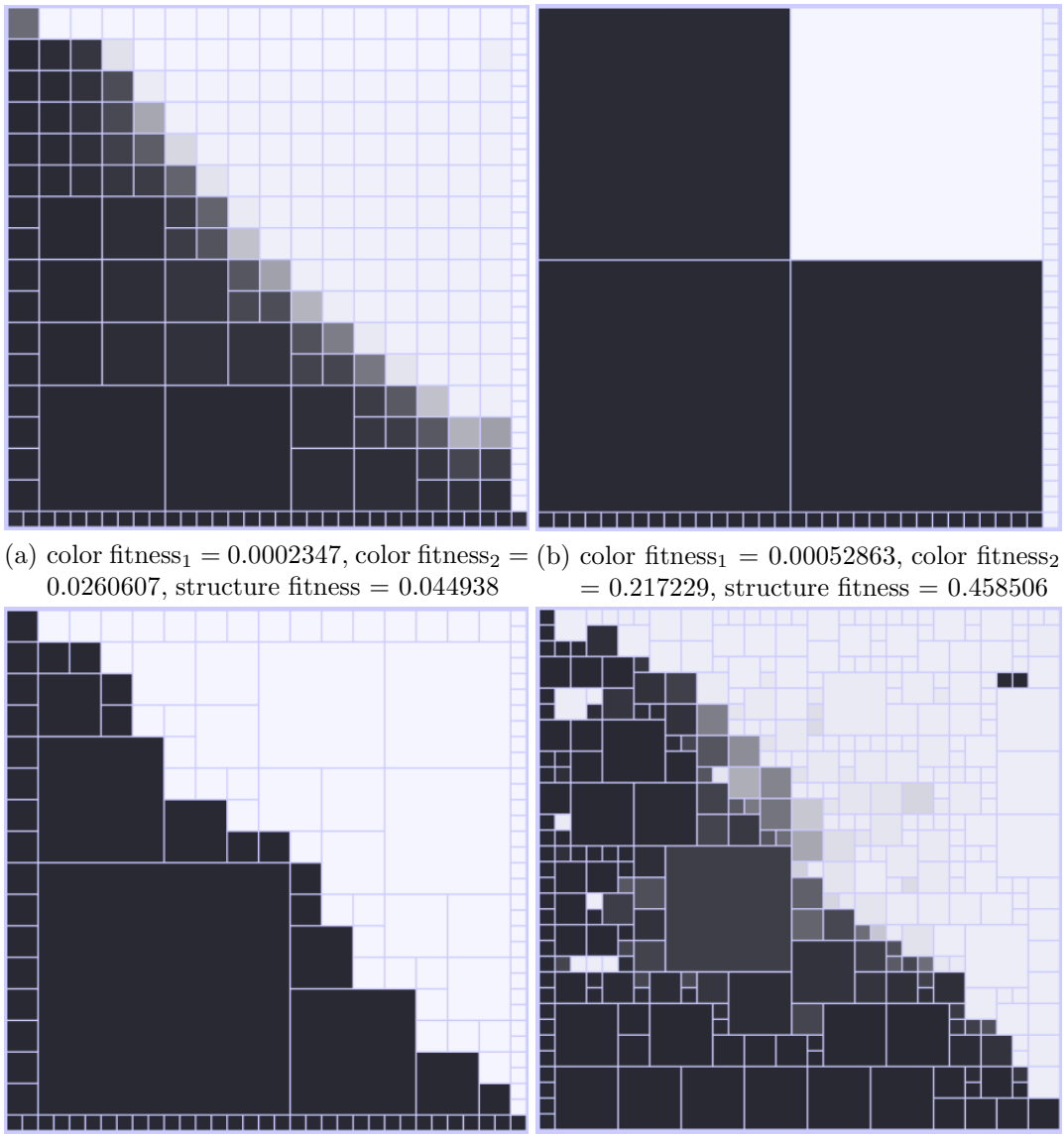
(a) color fitness$_1$ = 0.0002347, color fitness$_2$ = 0.0000581, structure fitness = 0.00679522

(b) color fitness$_1$ = 0.0123063, color fitness$_2$ = 0.208296, structure fitness = 0.00679522

(c) color fitness$_1$ = (0.0039, 0.0041), color fitness$_2$ = (0.1893, 0.1904), structure fitness = 0.248669

Figure 5.13: Visualization of results of optimization of the structure and the color of 'two bands 90' target. The color fitness$_1$ was the fitness value of the evolved embryo (Equation 2.12), the color fitness$_2$ would be a fitness value assigned by the other fitness function (Equation 2.11)[a].

[a]fitness value in an interval means that the embryo was unstable.



(a) color fitness$_1$ = 0.0002347, color fitness$_2$ = 0.0260607, structure fitness = 0.044938

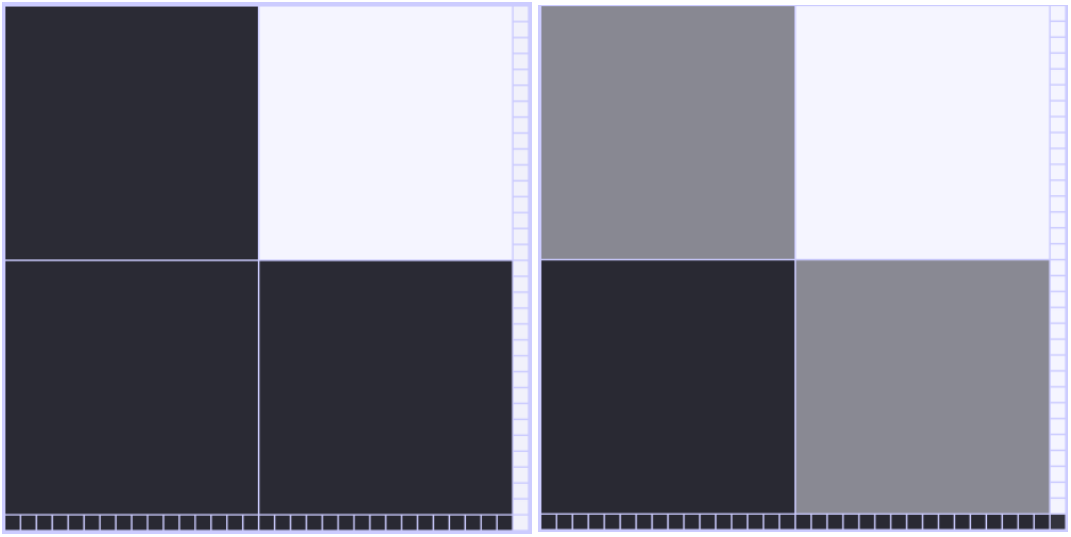
(b) color fitness$_1$ = 0.00052863, color fitness$_2$ = 0.217229, structure fitness = 0.458506

(c) color fitness$_1$ = (0.0009, 0.0015), color fitness$_2$ = (0.0356, 0.0477), structure fitness = 0.0385675

(d) color fitness$_1$ = (0.0053, 0.0055), color fitness$_2$ = (0.0554, 0.0599), structure fitness = 0.277204

Figure 5.14: Visualization of results of optimization of the structure and the color of 'two bands 90' target. The color fitness$_1$ was the fitness value of the evolved embryo (Equation 2.12), the color fitness$_2$ would be a fitness value assigned by the other fitness function (Equation 2.11)[a].

[a] fitness value in an interval means that the embryo was unstable.



(a) Optimized by fitness function as in Equation 2.12, color fitness = 0.00052863, potential color fitness assigned by the other fitness function (Equation 2.11) would be 0.217229.

(b) Optimized by fitness function as in Equation 2.11, color fitness = 0.117807, potential color fitness assigned by the other fitness function (Equation 2.12) would be 0.0136053.

Figure 5.15: Results of a previously optimized structure of 'two bands 45' target optimized by different color fitness functions.

Chapter 6

Conclusion

We tested the scalability of a developmental strategy by a multiobjective optimization task under various initial conditions. The developmental strategy combines cellular representation (embryo) with artificial neural networks and evolution strategies.

The embryo with static cells grid showed good and stable results for the optimization of one and two color patterns. The cellular automaton (embryo) was also capable of recovering from significant amount of damage. The embryo with dynamic cells grid showed good results for the optimization of the structure of a pattern and for the optimization of the structure and the color of a pattern. However, these automata were not sufficiently robust and stable as the automata optimized only for the color.

Cellular automata as representations that combine advantages of implicit embryogeny of the genotype with direct topology mapping to the phenotype are a novel approach in the domain of evolutionary design. As far as we know, there has not been any earlier attempt to perform multiobjective optimization by using cellular representations with variable topology. Concepts developed as part of this work can be applied in modern areas of evolutionary design where evolutionary methods are used to optimize parameters of complex shapes, structures, constructions, configurations, etc. The sound of future is to design a system capable of automated, practical, or even innovative design by simulation of evolution.

Bibliography

[1] *Flag problem*,
http://www.csse.monash.edu.au/~lloyd/tildeAlgDS/Sort/Flag,
on-line, accessed 05/07/2010

[2] A. Eiben and J. Smith, *Introduction to Evolutionary Computing*,
Springer Verlag, 2003.

[3] *Conway*,
http://en.wikipedia.org/wiki/Conway'_Game_of_Life,
on-line, accessed 05/07/2010

[4] Donald E. Knuth, *The Art of Computer Programming volume 2: Seminumerical Algorithms*,
3rd edn., p. 232. Boston: Addison-Wesley.

[5] Alexandre Devert, *Building processes optimization : Toward an artificial ontogeny based approach*,
Universit Paris-Sud, France, 2009, Ph. D. Dissertation.

[6] Nikolaus Hansen, *The CMA Evolution Strategy: A Tutorial*,
2008.

[7] *Nikos page*,
http://www.lri.fr/~hansen/cmaesintro.html,
on-line, accessed 06/30/2009

[8] *Gaussian blur*,
http://en.wikipedia.org/wiki/Gaussian_blur,
on-line, accessed 06/30/2009

[9] A. M. Turing, *The chemical basis of morphogenesis from Philosophical Transactions of the Royal Society of London*,
1952.

[10] *Turing Digital Archive*,
`http://www.turingarchive.org/browse.php/B/22`,
on-line, accessed 06/30/2009

[11] Julian F. Miller, *Evolving a self-repairing, self-regulating, French flag organism*,
2004.

[12] C. Kane and M. Schoenauer, *Topological Optimum Design using Genetic Algorithms*,
Control and Cybernetics, 1996

[13] S. Y. Wang and K. Tai, *A Constraint Handling Strategy for Bit-Array Representation GA in Structural Topology Optimization*,
2004

[14] edited by P. J. Bentley and D. W. Corne, *Creative evolutionary systems*,
2002

[15] edited by P. J. Bentley, *Evolutionary design by computers*,
1999

[16] J. Romero, P. Muchado, *The Art of Artificial Evolution*,
2008

[17] *Iterated Function Systems - Chaos & Fractals*,
`http://www.stsci.edu/ lbradley/seminar/ifs.html`,
on-line, accessed 05/08/2010

[18] H. Hamda, O. Roudenko and M. Schoenauer, *Multi-Objective Evolutionary Topological Optimum Design*,
Adaptive computing in design and manufacture, ACDM 2002

[19] Kicinger, Arciszewski, Dejong, *Morphogenic Evolutionary Design: Cellular Automata Representations in Topological Structural Design*,
Evolutionary Design and Manufacture, ACDM 2004

[20] H. Hamda, O. Roudenko and M. Schoenauer, *Application of a Multi-Objective Evolutionary Algorithm to Topological Optimum Design*,

[21] S. Wolfram, *Application of a Multi-Objective Evolutionary Algorithm to Topological Optimum Design*,
Wolfram Media, 2002

[22] S. Wolfram, *Cellular automata and complexity: collected papers*, Addison-Wesley, 1994

[23] Gregory S. Hornby, *Generative representations for evolutionary design automation*, Brandeis University, Waltham, MA, 2003, Ph.D. Dissertation.

[24] P. J. Bentley and S. Kumar, *Three Ways to Grow Designs: A Comparison of Embryogenies for an Evolutionary Design Problem*, Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'99), 1999

[25] H. Hamda, F. Jouve, E. Lutton, M. Schoenauer and M. Sebag, *Compact unstructured representations for evolutionary topological optimum design*, Applied Intelligence, Volume 16, 2002

[26] R. Kicinger, *Cellular automata in structural design*, Proceedings of the Second International Conference on the New Kind of Science (NKS 2004), Boston, MA, 2004

[27] A. Devert, N. Bredeche and M. Schoenauer, *Robust Multi-Cellular Developmental Design*, Proceedings of the 9th annual conference on Genetic and evolutionary computation (GECCO), London, England, 2007

[28] S. Wolfram, *Cellular automata as models of complexity*, Nature 1984; 311:419-24.

[29] Alistair N. Boettiger and George Oster, *Emergent complexity in simple neural systems*, Communicative & Integrative Biology 2:6, 1-4; 2009

[30] *Mathworld by S. Wolfram*, `http://mathworld.wolfram.com/ElementaryCellularAutomaton.html`, on-line, accessed 05/01/2010

[31] A. Ortega, A. A. Dalhoum and M. Alfonseca, *Grammatical evolution to design fractal curves with a given dimension*, IBM Journal of Research and Development, Volume 47, Issue 4; 2003

[32] Faming Lianga and Wing Hung Wongb, *Evolutionary Monte Carlo for protein folding simulations*, Journal of chemical physics, Volume 115, Number 7; 2001

[33] A. Boettiger, B. Ermentrout and G. Oster, *The neural origins of shell structure and pattern in aquatic mollusks*,
2008

[34] A. Boettiger, B. Ermentrout and G. Oster, *Introduction to neural networks*,
2008

[35] *The picture of a wooden chair*,
`http://brainaudit.com/blog/wp-content/uploads/2008/08/chair.jpg`,
on-line, accessed 05/01/2010

[36] *The picture of a design wooden chair*,
`http://files.myopera.com/kamiloly/albums/785245/Spring_Chair_600.jpg`,
on-line, accessed 05/01/2010

[37] *The homepage of the project of cellular embryogenic representations*,
`http://virtuallab.kar.elf.stuba.sk/~jana`,
on-line, accessed 05/16/2010

[38] I. Kroo, *Aeronautical applications of evolutionary design*,
2004

Prepared in LaTeX 2_ε by Jana Hlavacikova

Declaration

I hereby declare that the research and results presented in this thesis were conducted by myself using the mentioned literature.

Bratislava, May 12, 2010

Jana Hlavačiková