István Fehérvári
Matr.-Nr. 0861367

On Evolving Self-organizing Technical Systems

DISSERTATION

zur Erlangung des akademischen Grades
Doktor der technischen Wissenschaften

Alpen-Adria-Universität Klagenfurt
Fakultät für Technische Wissenschaften

Begutachter:

Univ.-Prof. Dr.-Ing. Wilfried Elmenreich
Institut für Vernetzte und Eingebettete Systeme
Alpen-Adria-Universität Klagenfurt

Univ.-Prof. Dr. Ágoston E. Eiben
Department of Computer Science
Vrije Universiteit Amsterdam

November 2013

Declaration of honor

I hereby confirm on my honor that I personally prepared the present academic work and carried out myself the activities directly involved with it. I also confirm that I have used no resources other than those declared. All formulations and concepts adopted literally or in their essential content from printed, unprinted or Internet sources have been cited according to the rules for academic work and identified by means of footnotes or other precise indications of source.

The support provided during the work, including significant assistance from my supervisor has been indicated in full.

The academic work has not been submitted to any other examination authority. The work is submitted in printed and electronic form. I confirm that the content of the digital version is completely identical to that of the printed version.

I am aware that a false declaration will have legal consequences.

(Signature)	(Place, date)

On Evolving Self-organizing Technical Systems

The trend toward pervasive computing and networked systems has lead to increased complexity and dynamics of today's technical systems. Thus, future systems are expected to be even more complex requiring novel ways to handle such complex networked systems. One approach to solve this problem is to increase the level of self-organization in those systems. Self-organizing systems offer numerous advantages over traditional ones like robustness against a failure of a component and scalability but due to the distributed structure there is no straightforward way to design a self-organizing system.

This thesis investigates how evolutionary computation can be applied to find the appropriate micro-level rules of a self-organizing system that provide the desired emergent global behavior for a given system. In particular, we propose a design methodology built on meta-heuristic search that guides the designer throughout the whole engineering process.

Additionally, we investigate the evolvability of self-organizing technical systems via several case studies focusing on the effects of certain design decisions explained in the proposed methodology. First, a self-organizing cellular automata model is described that is evolved to present a desired 2D structure. Using this example the connection between problem complexity and evolvability is discussed.

Two further studies focus on evolutionary swarm robotics. In the first one, we discuss the effects of various interaction interfaces and their effects on the quality of the evolved solutions. We find that seemingly identical interfaces can produce significantly different group behavior.

The second experiment investigates how a self-organizing team of soccer robots can be evolved. Here, we study the effects of different agent controller structures and interface interpretation models.

We also describe a novel evolutionary software framework that supports the proposed design methodology and aids engineers and researchers working with self-organizing systems.

Evolution von Selbstorganisierenden Systemen

Der Trend bezüglich Pervasive Computing und vernetzter Systeme hat zur erhöhten Komplexität und Dynamik heutiger technischer Systeme geführt. Künftige Systeme werden voraussichtlich noch komplexer sein, was neue Möglichkeiten erfordert, um solche komplexe vernetzte Systeme zu verarbeiten. Ein möglicher Ansatz um dieses Problem zu lösen besteht in der Erhöhung des Niveaus der Selbstorganisation in diesen Systemen. Selbstorganisierende Systeme bieten zahlreiche Vorteile gegenüber traditionellen Systemen wie Robustheit gegen den Ausfall einer Komponente und Skalierbarkeit, aber aufgrund der verteilten Struktur gibt es keine einfache Möglichkeit ein selbstorganisierendes System zu entwerfen.

Diese Arbeit untersucht wie evolutionäre Algorithmen angewendet werden können, um die entsprechenden Mikro-Ebene Regeln eines selbstorganisierendes Systemes zu finden, welche das gewünschte emergent globale Verhalten bietet. Dazu wird eine Design-Methodik für meta-heuristische Suche vorgeschlagen, welche den Designer durch den gesamten Engineering-Prozess führt.

Mit Hilfe verschiedener Fallstudien werden die Auswirkungen bestimmter Design-Entscheidungen auf die Entwicklungsfähigkeit von selbstorganisierenden technischen Systemen untersucht. Zuerst wird ein Modell für selbstorganisierende Zellularautomaten beschrieben, das entwickelt wird um eine vorgegebene Struktur zu präsentieren. Mit diesem Beispiel wird die Verbindung zwischen Problemkomplexität und Entwicklungsfähigkeit diskutiert.

Zwei weitere Studien konzentrieren sich auf evolutionäre Schwarmrobotik. In der ersten Studie diskutieren wir die Auswirkungen von verschiedenen Interaktionsschnittstellen und ihre Auswirkungen auf die Qualität der entwickelten Lösungen. Die Ergebnisse zeigen dass beinahe identische Schnittstellen ein deutlich unterschiedliches Gruppenverhalten produzieren können.

Das zweite Experiment untersucht, wie ein selbstorganisierendes Team von Fußball-Roboter evolviert werden kann. Hier untersuchen wir die Effekte verschiedener Steuerungsstrukturen von Agenten und verschiedene Interpretationsmodellen von Schnittstellen.

Schlussendlich beschreiben wir ein neues Software-Framework, das die vorgeschlagene Entwurfsmethodik implementiert und Ingenieuren und Forschern dabei helfen kann sich mit selbsorganisierenden Systemen zu beschäftigen.

Acknowledgements

I would like to express the deepest appreciation to my main supervisor Professor Wilfried Elmenreich, who has the attitude and the substance of a genius: he continually and convincingly conveyed a spirit of adventure in regard to research and scholarship, and an excitement in regard to teaching. Without his guidance and persistent help this dissertation would not have been possible. In addition, a thank you to Professor A. E. Eiben, who dedicated his time to review this thesis. His work in the field had a significant influence on my professional life.

I would also like to extend my thanks to Dr. Vito Trianni, whose fruitful discussions and constructive criticism kept me on the right path. I am grateful to him for holding me to a high research standard and enforcing strict validations for each research result, and thus teaching me how to do research.

I am indebted to my many colleagues who helped me in fixing software bugs, joined in for discussing ideas or proofreading my drafts.

Finally, I wish to thank my friends and loved ones, who have supported me throughout the entire process, both by keeping me harmonious and helping me putting the pieces together.

Contents

List of Figures

List of Tables

CHAPTER 1 Introduction

Chaos was the law of nature; Order was the dream of man.

– Henry Adams, *The Education of Henry Adams*

Due to the technological advancement in the recent decades we see a huge increment of pervasive, networked systems around us. This is explained by the fact that computing devices are getting faster, smaller and cheaper, thus even in our private life we are getting more and more surrounded with large collections of these devices. This opens up many new application areas of such networked intelligent autonomous machines. Imagine cars that warn each other of possible traffic scenarios, autonomous flying robots that collaboratively survey an area, or swimming robots that clear oil patches from the ocean's surface. However, with the increased complexity one has to face new design and reliability issues. The challenge arises from the fact that these systems cannot be managed in every little detail or to anticipate every possible configuration. Therefore, there is a high desire for flexibility and self-adaptation in order for them to be able to handle uncertain and dynamic environments while maintaining their planned functionality.

Since nature provides probably the most complex systems, a new field called organic computing [MS04] has risen recently with the vision of bringing these natural processes into technical systems in order to tackle complexity issues. One of the key concepts within this field is self-organization that allows a system to maintain its structure and function via internal dynamic processes without any external control. Moreover, it brings many new advantageous properties to the table that traditional, centrally controlled systems do not offer, such as robustness, scalability, and adaptability.

Despite of all these advantages, designing a self-organizing system is not straightforward. The problem is that traditional top-down design approaches fail to build systems with emergent functions since those are results of bottom-

up processes. Practically, any pure top-down attempt is useless since one cannot simply predict the effect of a change at the micro level on the macro level.

Though in the recent years systems with self-* properties have been thoroughly investigated, there were only a handful of proposals on the design and engineering of self-organizing systems [Ger07, AWdM08, BS08]. The main issue is that one cannot simply establish a set of internal rules for a given system that leads to the desired emergent behavior. Furthermore, the highly dynamic characteristics of these systems requires one to find the right set of rules, meaning it provides a guaranteed stable operation even under unexpected circumstances.

One idea would be to obtain knowledge on natural self-organizing systems and apply the acquired insights in the technical domain. For example, an interesting emergent effect has been found while studying a special breed of fireflies in South-East Asia. These insects are famous for synchronously emitting light flashes to attract mating partners [Buc88]. This phenomenon inspired the model of pulse-coupled oscillators where each unit exhibits an activation state that increases over time and can influence neighbors only via short pulses. This firing event occurs when the activation level reaches a certain threshold after which it is set back to zero. Similarly, units observing this firing event adjust their activation by a small amount. As a result almost always a common state emerges in which every unit fires at the exact same point in time [MS90]. A useful application of this phenomena is distributed clock synchronization in wireless, ad-hoc sensor networks [TAB07, WATP$^+$05, LE09], however an interesting application of this algorithm was also proposed by Christensen *et al.* [COD09] to detect non-operational units in a multi-robot system. Another very prominent example of a natural self-organization is the pheromone trail laying behavior of ants that creates a stigmergic [MO08] relation among them. Thus, via local interactions with the environment these insects are capable of coordinating the whole population in order to map the shortest route between food sources and nests or to build amazing complex structures. This behavior inspired the well-known Ant Colony Optimization algorithm [DMC96], network traffic routing [DCD98] and decentralized control of swarm robotic systems [WKF$^+$10].

As we can see, learning from nature can help finding appropriate templates that can be used to design self-organizing system. However, the drawback of this approach is that in many cases there exists simply no such suitable natural pattern. Therefore, one is left with the option to manually tweak the micro-level interactions of a system following some kind of trial-and-error process in order to find the desired emergent behavior. Due to the inherent unpredictable nature of self-organizing systems, this means that after each change the whole system has to be tested and evaluated in simulation or in real hardware until all the requirements are met. Now it is easy to imagine that at this level of

complexity a pure bottom-up search of all possible configurations is simply not feasible, thus there is a need for more sophisticated search techniques.

Due to the development of available computational power in the last decade, we can bring this idea one step forward by mimicking the same process that created those natural systems: evolution. Its digital counterpart, evolutionary computation has already been applied in many fields of science as a general tool for numerical optimization. Indeed, evolutionary algorithms are considered to be robust enough to be used in many application areas (especially where the search space is large), however even with additional techniques they can get stuck at sub-optimal results. These properties nominate evolution the *second best* approach for *any* given problem [Gol89].

The idea of evolving simple rules to obtain complex emergent behavior has been discussed first by Koza [Koz92] where he presents two examples of evolving collective intelligence by means of genetic programming (GP). Since this approach requires the user to define a limited set of mostly problem-specific actions (e.g. move, climb), one would get the idea that the whole engineering process boils down to a simple task of "take a set of possible actions and let it evolve". However, in most cases it is not possible to simplify the micro-level behavior to a set of simple distinct actions. Later on, many new examples of the application of evolutionary computation came up mostly replacing the limited genotype-to-phenotype mapping of GP to artificial neural networks or finite state machines. For instance, in the field of swarm robotics it has been applied to evolve autonomous controllers for simple robots [Tri08], or in the field of cellular automata, where an edge detection algorithm was presented [BMA06].

However, related works indicate an overuse of empiricism while building those experiments, meaning that many different choices of the experimental setup are arbitrarily performed, without relying on a well-assessed methodology. Fortunately, the robustness and flexibility of evolution sometimes counterbalances ill-conceived setups, but this cannot be *a priori* guaranteed. This all points towards the fact that the engineer has to face important design questions, which if not handled correctly, could lead to sub-optimal, or in some cases, unusable outcomes at the cost of not only lost hours of human work, but measurable computational time.

1.1 Motivation

In contrast with traditional top-down design approaches for distributed systems, evolutionary methods require very little *a priori* knowledge of the solution of a given problem. In particular, they relieve the designer from finding both the local interaction rules of the components and the required internal

mechanisms that drive the them towards this behavior. Therefore, they can be extremely useful to synthesize self-organizing behaviors and to obtain the desired emergent macro-level behavior. However, there is no free lunch: the experimenter still has the burden of designing the setup of the evolutionary experiment. While the design effort for the experiment is in principle simpler than designing the solution itself, the lack of systematic methodologies may keep the experimenter from optimally exploiting the evolutionary method or might constrain the outcome to the intuition of the designer. Recently, in the field of evolutionary swarm robotics, Trianni and Nolfi already recognized the need for such an engineering methodology that guides the experimenter through the relevant choices that must be made when setting up an evolutionary experiment [TN11].

The aim of this thesis is to follow this line of thought by extending their work to provide a general design method for self-organized networked technical systems. Thus, the ultimate goal is to provide a set of guidelines that can help the engineer to make the right choices by enabling him to assess various critical decisions in a meaningful way. In particular, we aim at:

- Performing a comprehensive analysis on the evolutionary design of self-organizing systems in order to identify core methods, processes and interfaces.
- Formulating a set of guidelines that can be used as basis for a design architecture, along with useful examples.
- Reducing the design complexity by introducing a software tool that hides many details of the evolutionary approach by decomposing it into smaller parts. Thus, the designer can focus on the relevant part of the experiment.
- Assessing particular decision points in the previously formulated guide within case studies in order to increase the understanding of the underlying processes. Used cases will come from the two most prominent domains of applied self-organization: cellular automata and swarm robotics.

1.2 Approach

Evolving parameters of various multi-agent systems is a very complex topic. On top of that, if we are dealing with something that is so powerful, yet often counter-intuitive, like self-organization, then we find ourselves facing a very difficult task if want to use traditional mathematical analysis. Therefore, we look at new design paradigms by learning from the many existing examples to

obtain insights on how such systems behave when their micro-level interaction rules are exposed to evolution. Since evolution is practically an automated trial-and-error process, each candidate solution has to be evaluated, either in simulation or on real hardware. Certainly, the latter approach would yield better evolved solutions in terms of quality and accuracy, however it would also increase the overall time and complexity required for each experiment, especially the ones the require a higher number of robots to be programmed and operated simultaneously. In order to provide a sufficiently high number of evaluations with controllable and repeatable experiments with high flexibility, this thesis follows the modeling and simulation approach.

In order to be able to conduct the necessary studies we are going to need an appropriate set of tools. While there are many powerful software tools available that can help to model complex multi-agent systems, there are currently no solutions out there which support direct studies on the evolution of self-organizing systems. Consequently, we introduce a novel software framework that provides the necessary abstractions and models in a modular way helping our questing within this field.

Afterwards, we address each identified property within the proposed architecture using a systematic analysis of many simulation runs with increasingly complex problems. We first introduce a case study based on a simple problem in the domain of cellular automata. This example provides means to assess the evolvability of a problem with respect to its complexity. We analyze the evolvability of increasingly complex problems, and study the effects of different neural network structures.

Afterwards, we continue the same idea but within the frame of swarm robotics. Namely, we study the effect of the interaction interface that operates among the system's components by comparing the performance of the evolved teams on a bi-objective scale. Therefore, we extended this work with an analysis of the pareto front of the evolved behavior to obtain an idea how a clear comparison can be made.

The goal of our last case study is to see how the system's evolvability is changed within adversary environments. Thus, we turn towards today's probably most popular collaborative robotics example: soccer. We also investigate the effects of competing populations and propose an efficient way to evolve them.

Each scenario is then followed by a discussion section with the aim of increasing knowledge on the specific decision point within the previously proposed evolutionary design architecture.

1.3 Outline

The outline of this thesis is the following: after this short introduction, Chapter 2 discusses the concept of self-organization, beginning with a short summary of its history. A practical notion in the technical domain is given that will be used for the rest of the thesis and an overview of related design approaches is presented.

In Chapter 3 a system architecture for evolving self-organizing technical systems is proposed as the core part of this thesis. Along the architecture, the major cornerstones and decisions points that the designer has to face are drawn along with examples, thus providing a practical set of guidelines. To illustrate this three case studies are discussed in Chapters 5, 6 and 7 each revealing important information about various design decisions.

Chapter 4 presents a software framework for evolving and evaluating self-organizing systems (FREVO) that is built on top of the methodology discussed earlier. The aim of this framework is to provide a tool for engineers and researchers that helps to focus on specific aspects of evolving such systems removing the burden of being an expert in the field of other segments of evolutionary computations. A simple example in the domain of self-organizing smart energy grids is also shown.

The first case study is presented in Chapter 5 where the methodology is applied to design a complex self-organizing multicellular system based on Cellular Automaton. The aim of this project is to give an intuitive idea of how evolving a self-organizing system towards a desired pattern or structure should take place. Within this example, the complexity of the problem can be quantitatively assessed and thus the performance of various evolutionary strategies can be easily compared using problems of increasing difficulty. The results show that simple regular structures can be achieved relatively easy, but for complex patterns additional search techniques are required.

In Chapter 6 we turn our attention towards evolutionary swarm robotics. The presented project highlights the importance of the interaction interface of the system's components by studying the effect of different robot configurations on the quality of the evolved solutions. In this domain, the choice of a good configuration is crucial as even small parameter differences can lead to completely different group behaviors. The system's performance is evaluated on a bi-objective scale that also demonstrates how such experiments can be qualitatively evaluated. The results confirm that different configurations not only have a strong impact on the performance, but they also correspond to entirely different group behaviors.

The last case study in Chapter 7 focuses on the evolution of complex adaptive behaviors with a team of self-organizing soccer robots. The presented ex-

periment demonstrates how such complex adaptive team behaviors can emerge among components with very limited sensing capabilities. Furthermore, the question of *what complexity is required on the component level?* is addressed by comparing different decision unit models. This chapter also gives a brief analysis and hints on how to efficiently co-evolve competing populations of candidates.

Finally, the last chapter summarizes everything that is been done in this thesis along with conclusive remarks and an outlook on what can be expected from future research in this area.

CHAPTER 2

Self-organization

It turns out that an eerie type of chaos can lurk just behind a facade of order - and yet, deep inside the chaos lurks an even eerier type of order.

– Douglas R. Hofstadter, *Metamagical Themas: Questing For The Essence Of Mind And Pattern*

When in the early 1950's a Russian scientist called Boris Belousov was working on his personal project he came across something that he just could not explain. His work at the time was focused on mimicking one part of the process of glucose absorption in the body. In one of his experiments he mixed various chemicals expecting to obtain a stable solution. Instead, the color of the solution started to oscillate between colored and clear. At that time, this reaction seemed to violate the known laws of physics, which even resulted in a total rejection of all his publication attempts [Win84]. Unfortunately, these events crushed him so hard that he eventually quit science.

While this example is most certainly not the only case when a great discovery gets lost from those times, it demonstrates the lack of understanding and acceptance of the idea that explains Belousov's incredible results: self-organization. In short, something is called self-organizing if it tends to become more organized when left alone. This is indeed a rather counter-intuitive property; in general, we expect things to become more messy or unorganized without external intervention. Yet, as more and more examples came to light, the existence of a self-driving process of order out of disorder could not be denied anymore.

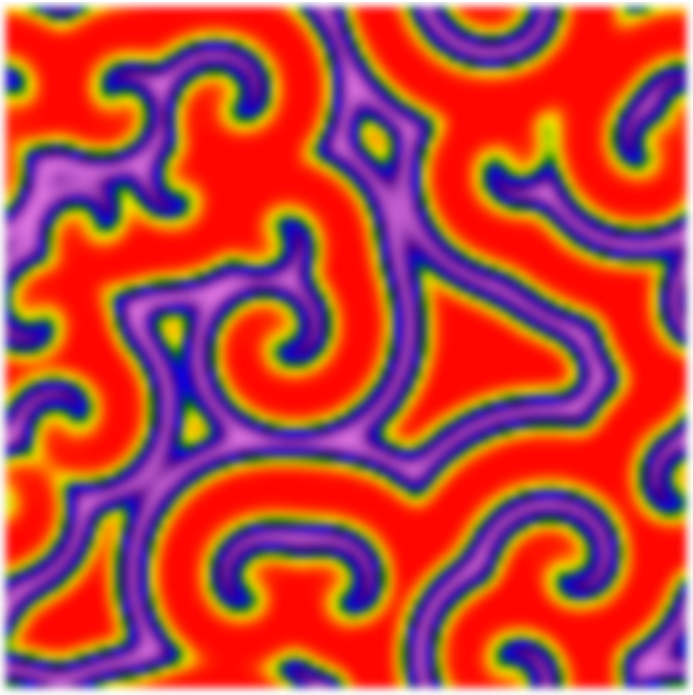

Figure 2.1: Pattern formation observed while simulating the Belousov-Zhabotinsky reaction occurring in a Petri dish. The actual calculations are based on Turing's equations [Tur52].

2.1 A Brief History of Self-organization

The idea that a system can become more organized only via internal processes is not new at all. Probably one of the first documents that mentions it is by René Descartes, in the fifth part of his *Discourse on Method* where he writes the following [Des37, Part V]:

> *[Consider] what would happen in a new world, if God were now to create somewhere in the imaginary spaces matter sufficient to compose one, and were to agitate variously and confusedly the different parts of this matter, so that there resulted a chaos as disordered as the poets ever feigned, and after that did nothing more than lend his ordinary concurrence to nature, and allow her to act in accordance with the laws which he had established.*

What he basically meant is that ordinary laws of nature tend to produce organization and expresses it as something that God could have arranged to have happen, if He hadn't wanted to create everything Himself. Later on Descartes elaborated the idea in much more detail in his book called *Le Monde*, which he did not to publish in his life to avoid conflict with the Church[Gau04].

The actual term of "self-organization" was introduced much later by W. Ross Ashby in 1947 [Ash47], while the first formal definition came by Farley and Clark of the Lincoln Laboratory [FC54] in 1954 as a "system which changes its basic structure as a function of its experience and environment". Besides the philosophical question of what "self" is, the term inherently suffers from the problem of defining what an organization of a system is. According to Ashby it can be described as a functional dependence of a system's future state on its current state and any external inputs. Mathematically put, the organization of a system is the function $f : S \times I \to S$ where S is the state space and I is the input state. His idea was that a system is self-organizing if it changes its own organization without an external driving force. But how is this then possible? Ashby's answer was that it is not, since structure is invariant. Nevertheless, it is possible then for one to find functions like g and h well approximating f in two different regions. Now if the actual dynamics drive the system from this first region to the second then one can observe an apparent change in the organization from g to h, though it is a result of the same underlying dynamics. This argument is elaborated in more details in [Ash62].

Later on, Foerster argued [vF60] that an organism cannot organize itself independent of its environment, thus the environment of such systems is a *conditio sine qua non*. As a consequence, in 1960 Ashby redefined the term as a system that consists of the organism and its environment essentially binding the two things together.

While Ashby's definition of a self-organizing system provides a good basis, there are certain problems with it. The main issue is that it does not give any qualitative information on the changes that happen within the system. This means that one cannot distinguish changes that lead to more organization from ones that are either neutral or lead to less organization. Probably this missing qualitative evaluation caused his definitions to remain mostly neglected by following researchers.

Nevertheless, this gave rise to novel definitions and applications in many different disciplines of science. In physics it was extensively applied from the 1970s onwards to pattern formation and spontaneous symmetry breaking, in non-linear thermodynamics [NP77], and to explain cooperative phenomena [Hak78]. For example, Eigen and Schuster *et al.* describe a set of perquisites for a system in order to be able to self-organize [ES77]. In the 1980s when self-organization became an important part of the models and techniques of the "sciences of complexity" [Pag89], it quickly spread around in nearly every field of science[TR02]. Indeed, we can find studies from chemistry[Leh90], biology[CFS+01], mathematics and from the 90's in the domain of socio-technical sciences as well [Kel96, Ves12].

Despite the vast amount of studies on self-organization across many fields, a commonly-accepted interdisciplinary definition is still missing. This often can lead to misinterpretations [Ort94]. An attempt to resolve this issue was taken by Gershenson [GH03] claiming that self-organizing systems do not make a separate class, but it is rather a way of observing systems. Nevertheless, we can state that all definitions converge towards the following three observations [EdM08]:

- There is a continuous interplay of positive and negative feedback between components.
- The structure and function of the system is an emergent property.
- There is no external directing influence.

In this work we will not attempt to propose yet another definition, but rather to see how these observations can help us to apply self-organization in the technical domain.

2.2 Self-organization in Networked Technical Systems

Even though self-organization has already been known in many fields of science, the idea to apply it in the technical domain has gained interest only recently. The reason for this lies in the apparent paradigm shift from monolithic or systems with a small number of components to large networked systems. This paradigm shift can be explained by the technological advancement and the appearance of pervasive information systems integrating into everyday objects and activities. A typical example would be to replace a fieldbus network with accurate but expensive sensors with one that contains a much larger number of inexpensive sensors connected by a wireless ad-hoc connection. The advantage of such a system is obviously the massively distributed view of the desired target that leads to a more robust and accurate observation. However, such an approach demands a paradigm shift on the control side as well to cope with the increased complexity. Here comes the principle of self-organization into the picture where both the control and the controlled system are decentralized.

At this point we have to come up with some kind of definition of what a self-organizing technical system is. From the practical point of view, a self-organizing system is fully defined by its components and their *local* interactions. This means that there are no external driving forces, i.e. the global level function or behavior of the system is achieved via the components following a set of

rules based only on local information at any given time. While these conditions might sound too restrictive, such systems are capable of exhibiting significantly more complex behaviors than the simple sum of the individual actions. Thus, we can say that a *global* behavior emerges from simple *local* interactions. For a more rigorous mathematical model see Holzer *et al.* [HdMB08].

It is important to mention that although the emergent property of the system is often seen as a clear indicator for self-organization, there is no common agreement if its existence is a requirement for a system to be called self-organized or is only a side-effect. Furthermore, Elmenreich and de Meer [EdM08] argue that the emergent pattern should not be seen as a primary property since in many self-organizing systems it is hidden from the observer.

Besides the capability to tackle complexity issues, self-organization offers many advantages over traditional hierarchic or decentralized systems. Systems built this way are inherently scalable, adaptive and robust against single-point failures. Imagine a school of fish that has no leader, yet by each fish following a simple set of rules we can observe a much larger entity coming alive. This system is not only beneficial to scare away or avoid predators (adaptivity), but can be composed of any number of fish (scalability) even if some of them get caught or leave the school (robustness) [PB05]. Additionally, most emergent behaviors can be achieved by using relatively simple components. For example, to build a system that exhibits swarming behavior one requires units having only a distance and a bearing sensor along with a reactive decision unit that follows three simple rules [Rey87]. This means making a transition to self-organization, we can probably reduce the cost of the hardware and the complexity of the onboard software.

Typically, a design procedure of such systems would be to elaborate the local, micro-level interactions that will result in the desired global behavior. Unfortunately, there is no straightforward way for the design of these rules yet so that the overall system will show the desired macro-level properties. The problem lies in the fact that emergent behaviors are very hard, or even impossible to predict or even to be boiled down into simpler interactions of objects. Moreover, even experts have a strong tendency to falsely predict the effect of a parameter change in a complex system (see for example the slime mold behavior experiment described by Mitch Resnick [Res97]). Furthermore, due to the high interplay and dependency among the components of the system, a slight change of one parameter could cause unexpected and even counter-intuitive results. Thus, finding a set of rules that causes the overall system to exhibit the desired properties presents a great challenge to the system designers.

2.3 Approaches to Design Self-organizing Systems

Traditional design methods are usually organized hierarchically and follow a top-down approach. This means that one starts with a high level specification and through many iterations and refinements arrives at a model that contains the description of all the components and interactions of the system. However, this approach conflicts with the emergent behavior of self-organizing systems which is a result of bottom-up processes. Thus, there is a need for a different kind of methodology that considers the unpredictable bottom-up micro-macro direction in self-organizing systems.

Today, there exists no common formal methodology to find the appropriate micro level rules that result in the desired macro level behavior, although some steps have already been taken in this direction. Many proposed techniques are from the field of multi-agent systems and agent-oriented software engineering [GYWA05][GCGC08][SBP+09]. However, as explained above, a formal design methodology must contain some kind of process that revisits and iteratively refines the micro level behaviors in order to create a bridge between local and global level behaviors [EB04].

One way to do this is to imitate and adopt an already working solution. In the recent years many natural and artificial systems with emergent effects have been studied. One of the most successful ways to reproduce such systems is to use agent-based simulations. A prominent example of this approach was presented by Reynolds [Rey87] who simulated the swarming behavior of birds. Each bird in his computational model followed the following three rules: *separation*, *cohesion* and *alignment*. The first rule is basically a short-range repulsion among the entities ensuring a collision free group, while the second one drives the birds towards the average position of their neighbors (long range attraction). These two rules represent the positive and negative feedback acting within the system. Finally, the last rule guarantees a common group direction by steering each bird towards the average heading of its neighbors.

Another very often used technique to model emergent behavior is cellular automata (CA). CAs are spatially and temporally distributed discrete dynamical systems composed of a lattice of simple elements (cells). Cells are only dependent on their neighbors, thus their state can be fully described by the states of the neighboring cells. During a simulation, each cell's state is recomputed simultaneously, which gives rise to complex spatial and temporal patterns [Sym08]. This model can be used for example to model self-organizing traffic lights [GA05] or biological systems [EEK93]. In chapter5 we will also present a study using cellular automata.

Most work in this field however describes mechanisms, but does not give answers as to how to apply those mechanisms to achieve an intended technical effect. Yet, one can collect these experiences as a pool of design patterns in order to adopt them to engineer the desired system. Such collections have already been reported, for example by De Wolf and Holvoet *et al.* [WH07], or by Sudeikat and Renz *et al.* [SR08].

In recent years, there have been several proposals for a general design method for self-organizing systems. Gershenson *et al.* introduced the notion of "friction" between two components as a utility to design the overall system using trial and error [Ger07]. Besides being tedious, the problem with such trial-and-error methods is that even if they are improved by certain notions, they often suffer from counter-intuitive interrelationships between local rules and emergent behavior. Another interesting approach was proposed by Auer *et al.* [AWdM08], where the behavior of a hypothetical omniscient "perfect" agent is analyzed and mimicked in order to derive the local rules. An example would be a poker player who can peek into the hands of the others players. His strategy would then be used as a basis to derive a tactic that has only the regular information available to a player. The problem here and in all methods based on imitation is the limitation to cases where an appropriate example model is available.

When designing large, parameter-heavy complex systems, manual trial-and-error solutions are often not efficient or sometimes not feasible. In this case, one can apply an automated process that systematically tests micro-level rules driven by some evaluation function based on the desired global behavior. Since the search space expands heavily with the number of possible local states and interaction rules, a full search is simply not possible. One possible solution for automated simulation-based search and design of self-organizing systems would be to use evolutionary methods. The core advantage of such algorithms is that they can be applied to any problem where the quality of a candidate solution can be directly measured, thus there is no need for any internal knowledge about the simulated system. Moreover, such algorithms are easily parallelizable and can cope with the uncertainty of stochastic simulation models. There are several examples of evolving the local rules of a self-organizing system. The usefulness of evolutionary algorithms to evolve cooperative behavior is demonstrated by Quinn et al. [QSMH02] by evolving teamwork strategies among a set of robots. In the field of swarm robotics, Nelson describes the evolution of cooperative systems with Artificial Neural Network controllers for a team that plays "Capture the flag" against an opponent team of robots [NGH04]. Trianni presents several experiments where the controllers of a group of robots were evolved to achieve a certain team behavior [Tri08]. The examples are not limited to robotic applications; Arteconi [Art08] applies evolutionary meth-

ods for designing self-organizing cooperation in peer-to-peer networks. In the field of cellular automata, cell update rules were evolved to search for glider guns [SB08].

As seen above, evolutionary methods can produce systems with emergent properties via executing a huge combination of local rules within a simulated environment. The drawback of this approach is that it is hard, or even impossible to model all possible influences that can act on the evolved system. Therefore, the engineer has a hard time guaranteeing the behavior of the developed self-organizing system. A possible solution could be to have the system search for the local behavior during run time instead of during the design phase. Such self-adaptive systems that can reason about their state and environment have already been proposed by Cheng *et al.* [CLG+09] within the field of software engineering. Their core mechanism for adaptation is the usage of feedback loops that enables them to deal with unforeseen system states [BMSG+09]. An early architectural implementation of such a system was provided by IBM Corporation [Cor04] which was mainly developed for monitoring and adapting enterprise server applications.

Based on the idea above, a more general approach called the generic observer/controller architecture has been proposed by Schöler and Richter *et al.* [RMB+06][SMS05]. The idea here is to achieve a controlled self-organized behavior by influencing the micro level rules of the system during run time. This way, it is possible to handle such emergent states that have not been considered during the design phase. This architecture consists of the following three elements: the observer, the controller, and the system that is being observed and controlled which is assumed to be composed of many components with a high level of interdependent connections. The observer collects information of specific attributes on the micro and macro level components of the system. Based on this information and the expected behavior, the controller steers the system in a more desired direction of operation. There exist several applications of this model. For example, Prothmann *et al.* reports of an organic traffic controller [PRT+08]; Tomforde *et al.* applies this model to self-adapting networked systems within dynamics environments [TSHMS09]. The disadvantage of online trial-and-error processes is that in order to be able to assess different control decisions the system must be allowed to go into sub-optimal, or even close to critical states. In a running system this behavior if often not wanted.

CHAPTER

3 Design by Evolution

Man's longing for perfection finds expression in the theory of optimization. It studies how to describe and attain what is Best, once one knows how to measure and alter what is Good and Bad.

– Charles S. Beightler, *Foundations of optimization*

Most of our artificial systems that show self-* properties are built on some basis inspired by nature. Indeed, natural systems are inherently distributed and self-organizing which makes them very appealing to be used in the technical domain. Unfortunately, traditional top-down design approaches usually cannot be followed to create a self-organizing system, since emergence is a bottom-up phenomenon. A promising way would be to mimic the same process that created many of those systems: evolution.

In the last decades, several optimization method have been designed based on the principles proposed by Darwin in the 19th century[Dar59]. They combine the idea of the survival of the fittest (natural selection) with a structured yet randomized information exchange (natural genetics) to create a search algorithm. Even though they contain a fair share of randomization, evolutionary algorithms are not random walks. Instead, by retaining a pool of possible candidates they can exploit historical information to find new search points with higher expected performance. A big advantage of such algorithms is that they are universal, i.e. one does not need any *a priori* knowledge on the solution, thus they can be applied to a very wide set of problems. On the other hand, every evolutionary algorithm requires a function that can quantitatively assess the performance of any candidate solution in order to find the best ones to be used by the reproduction operators. A good introduction to evolutionary computing has been given by Eiben and Smith in [ES03].

While algorithms inspired by evolution can be a promising way to design self-organizing systems, they have several drawbacks as well. In many problem domains, especially with multi-modal fitness landscapes, such algorithms

tend to converge towards local optimum rather than the global optimum of the problem. This means that the designer of the algorithm either has to fine-tune the balance between exploitation and exploration (which is often unfeasible with large multimodal search spaces), or to use additional techniques, for example, using different fitness functions, on-the-fly adaptation of the rate of the reproduction operators, or by maintaining a high diversity in the population. Considering the many design parameters a self-organizing system already has, an engineer can easily find him or herself facing the problem of building the right experimental setup instead of designing the solution itself. While in theory this effort is smaller, without a systematic methodology, one might be restrained to fully exploit the benefits of the evolutionary approach, or obtain sub-optimal solutions constrained by the intuition of the experimenter.

In this chapter, we review how Darwinian principles can be applied to design self-organizing technical systems. In particular, we investigate how and why is it different from the traditional evolutionary design of systems. We start by reviewing how artificial evolution can be used to find the optimum solution for a given problem. Here we reveal the basic advantages and disadvantages along with the related techniques in the field. Then we move onto the domain of self-organization, we carefully analyze each decision/design step an engineer has to face when aiming for an evolutionary approach. In the respective sections, we also indicate techniques and methods that possibly help one to mitigate design errors, thus providing a more reliable methodology. Parts of this chapter have been published and presented at the 7th International Workshop on Self-organizing Systems [FE13].

3.1 Evolution in Engineering

Evolution described initially by Darwin is based on the following three basic principles: heredity, variation, and selection. In nature these simple rules ensure the adaptation of living organisms to their environment in order to preserve their species. Inspired by the elegant yet effective characteristic of this process, John Holland transferred this knowledge to the field of search and optimization [Hol75]. He described the Genetic Algorithm (GA) that operates on a (mostly fixed sized) population of candidate solutions and applies the principles of evolution in order to gradually improve them and ultimately reach a given goal.

In essence, heredity means that every new individual is similar to at least one of the old ones. This is typically achieved through the offspring inheriting genetic information from its parent(s). Variation implies that the process of inheritance is not perfect, i.e., the offspring is not going to be completely

identical to its parents, but some of its genetic material will be intentionally changed during the process. In order to keep the size of the population stable, a control mechanism must be implemented that defines the direction of evolution by selecting which individuals can survive to the next generation and which are removed from the population. Without such an operator the population would still change over time, however it would eventually just randomly drift. In the natural world selection is happening in the form of natural differential survival and mating capacities of entities of various species, while in artificial evolution it is replaced by the objective (fitness) function which reflects the desired goal from the system under evolution.

During an evolutionary run new generations are created by randomly mutating single genes or by recombining the genes of two or more individuals. Under proper selection the algorithm efficiently explores the solution space looking for better solutions. This space can be visualized as a multi-dimensional landscape where neighboring points represent similar genotypes and their heights correspond to their fitness. Such a way, distinct hills and valleys can be considered as local maximums/minimums whereas the extremes are indicated as the global maximum/minimum. Therefore, we can consider evolutionary algorithms as search techniques that try to find the highest/lowest peak within this landscape.

In the field of optimization, evolutionary algorithms fall into the category of meta-heuristic search, thus they are always characterized by their balance between intensification and diversification, or exploitation and exploration [Yan08]. Exploration means to generate diverse solutions in order to provide a good overall "coverage" of the fitness landscape. Exploitation on the other hand means the effort of searching for a better solution in a local context provided by an already found good solution. While selection of the best ensures a monotone convergence towards better solutions, diversification with randomization helps to avoid being trapped at local optima and at the same time it increases the diversity in the population. Therefore, a healthy balance of these two components results into a convergence towards the global optima.

Unfortunately, evolutionary algorithms are prone to get stuck at suboptimal solutions when applied for complex problems. Even though increasing the mutation rate and/or the population size could improve the exploration of the algorithm and prevent getting stuck at a local optima, it would do so for the price of reduced exploitation of good solutions. A well-known approach to overcome this issue is to use some method that can maintain a sufficiently high diversity within the population. For example, *fitness sharing* [GR87] is a niching method that adjusts the fitness of individuals based on some distance metric defined on the genotype level. Typically, it lowers the fitness of candidates that are similar to each other which spreads the population out

over multiple niches in the landscape; thus preserves a higher diversity. A similar approach is *clearing* [Pet96] which, instead of degrading the fitness of similar individuals, removes the least-fit candidates within a similarity radius from the population. An approach called *deterministic crowding* was introduced by Mahfoud *et al.*, which changes the crossover operation in a way that every offspring that is fitter than its parents will replace the parent that is more similar to the offspring [Mah95]. In fact it resembles to *fitness sharing*, however there is no requirement to define the similarity radius. Spatially structured evolutionary algorithms are also applied to keep a constant high diversity within the population. Here every individual is constrained to a limited set of other candidates it can mate with, thus the population is structured into a number of demes. Probably, the most popular version of this approach is an n-dimensional grid-like structure that ultimately defines the neighborhood relations of every individual. For an in-depth summary of diversity maintaining methods see [Dic05, Mah95].

3.1.1 Evolutionary Techniques

Practically, any system that conforms to the requirements of heredity, variation and selection will result in evolution. Therefore, several variants of EAs have been developed in the recent decades. Differences exist mostly on the genotype to phenotype mapping level i.e., how the candidate solutions are represented during evolution. The following (certainly not exhaustive) list presents the most popular techniques:

Genetic Algorithm (GA): In this type of EA, solutions are formed as strings of numbers. Originally binary numbers were used but it has been shown that representations that reflect something about the problem generally perform better, such as real-valued numbers [JM91]. Genetic algorithms are often used in numerical optimization problems.

Genetic Programming (GP): Unlike in GAs, solutions are encoded as individual computer programs that are evaluated based on their ability to solve the given problem. Since source code of typical programming languages cannot be easily changed in a way that they remain syntactically correct, candidates are represented in a tree structure, where every node has an operator function and every terminal node has an operand. This structure enables mathematical expressions to be represented in an evolvable way.

Evolutionary Programming (EP): Very similar to GP, however the structure of the candidates is fixed and only the numerical parameters are

being evolved. It has been originally introduced by Lawrence J. Fogel to evolve finite state machines [FOW66].

Gene expression programming (GEP): Similar to GP, however GEP also explores the genotype-phenotype system, where the solutions are encoded in fixed-length linear chromosomes leaving the interpretation of the genes to evolution.

Evolution strategy (ES): This technique uses variable-sized vectors of real values as genotypes and applies self-adaptive evolutionary operators (e.g mutation)[Rec94].

Memetic algorithm (MA): MAs are hybrid techniques that are viewed as a population based global search coupled with individual learning procedures capable of performing local refinements. At the time MAs are subject to intense research to obtain knowledge on finding the balance between exploration and exploitation, i.e., how to efficiently combine individual learning with population based strategies.

Differential evolution (DE): Introduced by Storn and Price [SP97], DE is used to find the optimum of multidimensional real-valued functions. It is operating on vectors of real values, but unlike ES, it is applying simple operators based on vector differences.

Neuroevolution (NE): Similar to GP, but the genomes are interpreted as artificial neural networks by describing their connection weights and/or their structure. Encoding can be direct or indirect. Direct encoding means that every neuron and connection is specified directly and explicitly in the genotype, while indirect encoding encapsulates a "plan" which is used to generate the network.

There are also several bio-inspired meta-heuristic search techniques that are considered to be related, yet they are not classified as evolutionary. Examples are particle swarm optimization that is based on the ideas of animal flocking behavior [KE95], or cuckoo search [YD09], which was inspired by the brooding parasitism of the cuckoo species. These approaches are similar to artificial evolution since they use an iterative search over a population of candidate solutions, however their way of exploration and exploitation is based on other principles. Nevertheless, these techniques have been proved to be effective in many combinatorial optimization problems, but will not be discussed within this thesis.

3.1.2 Applying Evolution to Solve Problems

Although the concept of evolutionary design is fairly simple, when it comes to application, it starts to become increasingly complex. In general, the whole engineering process can be decomposed into three major steps: modeling the system; an iterative search that explores new solutions; and a final validation phase (see Figure 3.1). The core element of this workflow is to find a suitable algorithm that leads the designer towards the optimal solution.

In the field of evolutionary computation, the "heart" of every experiment is how one represents a candidate solution in an *evolvable* way. Let it be a string of bits, a vector of real values, or a tree structure, the genotype-to-phenotype mapping ultimately has a huge impact on the complexity of the search algorithm (i.e., number of parameters, possible reproduction operators), thus on the outcome of the experiment. A bad choice of representation might introduce unnecessary parameter-tuning from the experimenter, or create a too large search space for the evolutionary algorithm. While the general rule of thumb is to aim for the simplest possible representation to reduce the search space, the designer has to decide what assumptions can be inserted into the system and what are left to be found out by evolution. This problem is primarily present when one has to design how the component under evolution is going to be interfaced with another system, or environment. For example in evolutionary robotics, when designing the genotype-to-phenotype mapping for a robot controller sensor inputs are usually pre-processed (e.g., extracting information from the camera, or fusing sensor data) in order to reduce the complexity of the optimization problem. This step ensures that the search process will not be busy with solving an unrelated problem.

As discussed earlier, every meta-heuristic algorithm is a combination of exploration and exploitation. However, according to the *no free lunch theorem*[WM97], there is hardly any single algorithm that is the best for every problem domain. Unfortunately, there is no simple method that tells what the right balance is between exploration and exploitation. Typically, one relies on previous experiences or applies a more tedious trial-and-error process to reach an acceptable deal. Another approach is to map and analyze the fitness landscape in order to obtain knowledge on the number and possible location of local and global minima. This is however not always feasible, especially if the search space is too large, and/or has a high dimensionality. Fortunately, the robustness of the evolutionary approach does not always require one to find the optimal parameters of the algorithm; in most cases a trade-off would lead only to increased computation time, and not necessarily worse results.

Obtaining the best results after the first run happens quite rarely. The usual approach when using stochastic based search is to repeat the process

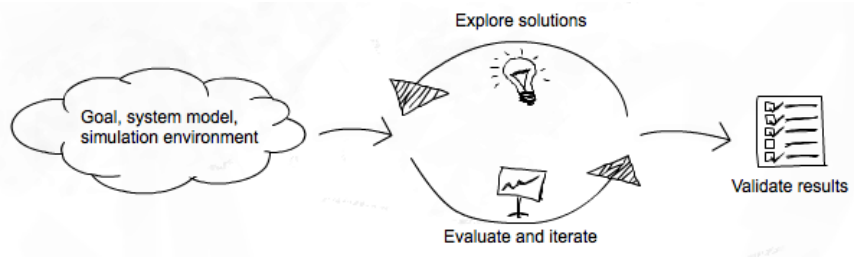

Figure 3.1: Basic flowchart of the evolutionary design

several times to reduce the noise of this technique. This step might also reveal deficiencies introduced by wrong parameters of the algorithm (e.g. too high/low population size, mutation rates), which can be easily corrected. Furthermore, one can obtain knowledge on the possible shape of the fitness landscape and could apply advanced techniques if the algorithm seems to be stuck at a sub-optimal solution.

3.2 Evolving Self-organizing Systems

As discussed in Chapter 2, designing self-organizing systems is a challenging task. This is mostly due to the fact that the emergent behavior is a result of a dynamic bottom-up process that cannot be tackled with tradition top-down design approaches. On top of that, such systems can be very sensitive to parameter changes i.e., a small tweak in the system might have completely unexpected results. A viable approach would be to exploit the robustness and flexibility of artificial evolution in order to engineer self-organizing technical systems.

In theory, when going for the evolutionary design, the same approach as for traditional systems can be used. However, the counter-intuitive nature of many self-organizing systems [Res97] makes the experimental setup of the evolutionary run a critical point in design. In other words, instead of finding the optimal local rules for the desired emergent behavior, one has to find the optimal setup of a method that will eventually find these rules. This shift of focus might seem superfluous or redundant; however the design effort for the experiment is in principle simpler than designing the solution itself. Unfortunately, the lack of systematic methodologies may prevent the designer from optimally exploiting the evolutionary method or might constrain the outcome to the intuition of the designer.

From an engineering point of view, establishing the right models for evolution has the upmost priority, where crucial decisions have to be made in order to obtain useful results. Among many things this includes selecting the right

computation model, the way of interactions and the whole evolutionary model (genotype to phenotype mapping, search algorithm, etc.). The next step is to let the whole system run on its own, thus gradually developing a viable solution with no or very little user interaction. The last phase for any design process is to verify the obtained results. Usually, this means a set of white or black box testing, but in the case of complex networked systems, novel approaches are necessary. Fortunately, there is a lot of ongoing research on how to evaluate such systems [RPS12].

The first idea to establish a set of guidelines for designers has been proposed by Trianni and Nolfi in the field of evolutionary swarm robotics [TN11]. They define four major categories where the most important decisions take place: the ecology, the sensory-motor system, the genotype-to-phenotype mapping, and the fitness function. Ecology refers to the environment in which the robots are evolved, which introduces ecological selective pressure. The sensory-motor system means the choices the engineer has to make in order to separate fixed and evolvable parameters of the robots. The representations of the evolved features are decided in the frames of the genotype-to-phenotype mapping, while the design of the fitness function is responsible for evaluating single solutions. Though their categorization works very well in the domain of swarm robotics, it might be unclear how this can be generalized to arbitrary self-organizing technical systems. Furthermore, they do not discuss the effects of particular design questions nor provide any hints on how one can systematically asses such problems. Thus, our aim in the following is to extend their idea and describe a more generic design methodology that works for any self-organizing system.

3.2.1 Development or evolution?

Self-organizing systems are considered to be inherently dynamic, since their properties on the global level will arise due to the dynamic interactions on the local level. As expressed by Holzer, de Meer, and Bettstetter [HdMB08], the level of emergence of a system is highly dependent on the actual time. In most literature, these continuous phase transitions are often noted as the evolution of the system. Though it is correct, in the scope of this paper we are dealing with the evolution of a system design instead of the evolution of the system during operation i.e., the evolution of the local rules. Therefore, to avoid confusion we recommend to call the former phenomena as *development*, that is a different process to evolution. Chapter 5 will describe a study presenting an easy to access example of this idea.

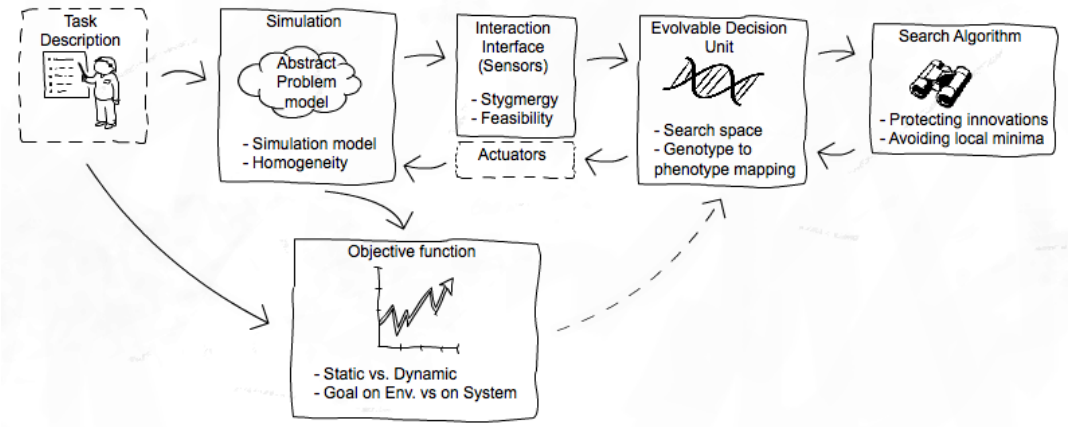

Figure 3.2: Proposed design methodology

3.3 Design Methodology

We propose a design methodology that covers the most important decision points the experimenter has to face. As can be seen in Figure 3.2, we distinguish 5 major components originally rooting from the task description [FE13]:

Task description: Set of requirements that the solution has to meet.
Simulation setup: Describes the simulation model and the relations between the system's components and their environment.
Evolvable decision unit: The evolvable representation of the local rules.
Interaction interface: Describes how the decision unit interacts with its environment.
Search algorithm: A meta-heuristic algorithm responsible of finding novel and better solutions.
Objective function: The cost function that guides the search algorithm.

Typically, every engineering problem starts with a *task description* that properly identifies the objectives and constraints in detail. In other words, this can be seen as a contract that describes the expectations of the desired system on a high abstraction level. Consequently, the task description has a high influence on the applied simulation model and naturally on the objective function.

The next step is then to specify a system model based on the task description that is not only efficient, but accurately represents the important aspects of the system to be modeled while abstracting unnecessary factors. However, this step should not include how the components are represented or the way they interact with each other, since this would anticipate several further important decisions. For this purpose we separate them into an interaction interface unit and to an

evolvable decision unit. In the former, one should plan the way the components of the system can interact with each other and their environment, thus defining the communication possibilities (i.e. sensors) and their underlying interfaces (i.e. protocols).

The latter is focused on the actual representation of the components of the system. This might include one or more types of models depending on the homogeneity of the system. The reason why this part is separated from the system model is that the evolutionary method requires *evolvable* representations. This means that the components have to be able to generate adaptive genetic diversity, for example by defining mutation or crossover operators [Alt94]. While there are a good number of such representation, their often very different properties require a careful choice from the designer.

In order for the evolutionary process to be operational there must be a valid and efficient search algorithm that iteratively optimizes the candidate solutions. Though in theory, the search algorithm and the component representation can be modeled independently, certain representations might require special techniques from the optimization side in order to work optimally. For example, evolving neural network structures often requires some mechanism that prevents premature innovations to die out to soon [SM02]. Therefore, the designer has to find the search algorithm that is possible the most efficient with the chosen component representation. Furthermore, as explained earlier in this chapter choosing the right set of parameters of the algorithm has a significant overall effect on the convergence speed and on the quality of the results.

The last aspect is the design of an objective function that will guide the evolutionary process through the problem environment to an optimal solution. Since the effectiveness of this function is directly related to the effectiveness of the evolutionary approach, a good objective function has a key role in the process. The design of such an effective function is often difficult, even for experts of the domain [JB05]. Even though there are attempts at creating guides for non-experts based on taxonomy [WT11], unfortunately, there exist no general guidelines available as to how to make a good function. Within Section 3.3.5 we try address this issue by reviewing what factors are important, especially in the self-organization domain.

3.3.1 Simulation Model

Foerster argued [vF60], that an organism cannot organize itself independently of its environment, modeling the environment of SO systems is a *conditio sine qua non*. Therefore, not only good component models are important, but there is an equal emphasis on the definition of the system's environment as well. As

will be explained in Section 3.3.5, this has a great impact on the quality of the objective function.

A further question when modeling a self-organizing system is homogeneity. While it might be a matter of choice, it is typically determined by the task description. From the evolutionary perspective, a system composed of homogeneous components usually requires a smaller set of parameters to be optimized (smaller genotype, simpler interfaces), thus reducing the search space. On the other hand, heterogeneous systems can potentially offer more sophisticated behaviors that might be required for specific tasks to be solved, such as artificial predator-prey societies [FK10a]. In this case, all roles must be somehow encoded into the genotype. Probably the simplest solution is to define all roles *a priori* and encode all parameters in a single genotype. Note, that such approach would greatly increase the search space for the evolutionary algorithm. A possible approach that prevents this problem to obtain heterogeneous teams is to use parallel populations dedicated for different roles. This way the system under evaluation is formed by drawing from these populations with a certain strategy and the best candidate of each population composes the best solution for the problem. Additionally, heterogeneity can be extended to the morphological level, where not only the behavior of a component is being evolved, but its representation in its environment as well [BBZ05] in order to enable a higher level of adaptivity.

From a practical point of view the type of computational model is a general modeling question, since it defines the depth of the simulation and/or its complexity and possible required computational resources. A viable modeling approach is to use cellular automata (CA). The concept, originally discovered in the 1940s at Los Alamos National Laboratory received much attention with the release of Conway's Game of Life [Gar70]. CA have been widely used in physics, biology, social sciences, mathematics and computer science, since they are very efficient in modeling dynamic spatial phenomena. Essentially, they model cells on a grid that change their states according to their previous own and neighboring cells' states, thus it can model any universe, where space can be represented as a uniform grid. The local rules are computed when each cell's state is being updated and the time advances in discrete steps.

For more complex situations, where these limitations are not enough, the dominant approach is currently agent-based modeling (also called multi-agent systems). As with CA, there are certain rules that describe the possible interactions within the system; however an agent is generally defined as an autonomous, intelligent entity that may interact with other similar entities or its environment. The main benefit of using agent-based modeling is the agents tend to capture reality more effectively, since they offer a higher freedom of representing dynamic events, without assumptions such as monotonicity. Fur-

thermore, they can easily model numerous noise, behavior and effect models to fit the designer's needs. As a rule of thumb for greater efficiency, event-based simulations should be preferred over discrete time ones if the events happen less regular.

3.3.2 Interaction Interface

The *interaction interface* describes how the decision unit interacts with the environment. This primarily involves the selection of type and number of sensors and actuators and also their placement and the representation of transducer data to the decision unit. The problem of defining an appropriate interaction interface that allows a decision unit to interact with the environment in an abstract way is typical for multi-sensor embedded systems [Elm07].

Designing an appropriate sensor system is constrained by the available sensors, their energy consumption, coverage of sensor area, sensor accuracy, the available communication bandwidth and finally their financial cost. Imagine a mobile robot that should navigate autonomously. The task could be solved by having a single expensive camera, two more cost-efficient cameras, or by installing multiple ultrasonic distance sensors. The two-camera solution might be the one that delivers the most information about the robot's environment, but this solution will come with the highest bandwidth and energy requirements. The ultrasonic sensors deliver less information, but work independently of lighting conditions. Furthermore, they have the highest cabling effort. While these are known tradeoffs in the design of sensor systems, in the context of designing self-organizing systems using an evolutionary approach, the question *"Which sensor systems offers the best configuration to solve the problem?"* translates to *"Which sensor systems offers the best configuration for which a solution of the problem can be evolved?"*. While we cannot answer this question analytically, previous experiments have shown that the performance of an evolutionary algorithm is affected by the interface. In contrast to interface complexity as perceived by humans, an evolved system differs in some notable aspects. The evolutionary approach is in general agnostic to non-conventional implementations of the sensor interface. This mostly applies to switching inputs or changing sensor ranges. Certain models like artificial neural networks can also cope with different scales or non-linear measurements as well. Some tasks appear to be simple for a human, because the task relates to abilities provided by human legacy. A prominent example are all kinds of image processing tasks which can often be done by humans in an intuitive simple way, while the necessary functions are hard to provide by a computer system. Thus, a sensor system featuring several independent simple sensors is expected to provide a better basis for evolving an appropriate decision unit than a complex image sensor. It

is, however, feasible to reduce complexity by applying smart cameras [RW08] that preprocess and reduce the information to an amount that is feasible for being processed by an evolved decision unit.

Apart from the hardware sensor configuration, the representation of data also plays a role for the feasibility in an evolved system. Experiments with self-organized soccer robots have shown that the same sensor configuration yields different performance for different representations of the same data [FE10] Chapter 7 will discuss this issue in more detail. A similar issue exists on the actuator side. For a given set of actuators the way how these are interfaced influences the quality of the solution that can be evolved.

Unfortunately, there exists no single algorithm for planning sensors and actuators for a system so that it can be evolved well for a given purpose. However, we identified the following principles to guide the system design:

Feasibility: Ensure that the problem can be solved given the available combination of sensors and actuators and their respective interfaces. A system that is missing an essential sensor input or a necessary degree of freedom would never evolve to a solution. For example, studies show that swarm behavior requires the individuals to be able to infer on the relative heading of other members of the group [Rey87]. This criterion is necessary, but not sufficient.

Simplicity: Avoid overly complex interfaces. This on the one hand means reduction by avoiding unnecessary sensors or actuators and on the other hand avoids interfaces that can change their meaning based on some implicit or explicit system state. For example, in a Cartesian coordinate system relative to the actor a positive value on one axis always refers to a point ahead of the robot's direction of movement. In contrast, in a polar coordinate system, the meaning of a coordinate depends on the value of the other one, which makes it more complex for particular tasks.

Continuity: Evolutionary algorithms apply hill climbing algorithms using small steps in a high-dimensional fitness landscape. Therefore, if two values in the sensor/actuator interface are close to each other, this should indicate that the respective physical values are also close. In other words, it is preferred that the sensors values satisfy the Lipschitz condition [SS01] for better evolvability. This is a soft requirement, but in general interface systems with continuous behavior (at least around the working point) tend to evolve faster.

3.3.3 Evolvable Decision Unit

As described earlier in Section 3.1, the main idea in evolutionary design is to find the right set of *local rules* that drive the system towards the desired global behavior. Essentially, these rules form the logic, of the "brain" or controller of the individual components within the system. While it might seem that these components have to be intelligent alone, this is nonessential. Imagine a complex living being (e.g. a human), that is the result of the emergent behavior of a very large number of non-intelligent cells. The idea is to encapsulate the necessary actions and responses of a unit into something, so it will result in a fine interplay of positive and negative feedback in the self-organizing system. Therefore, in order to apply any iterative optimization method on them, they must support some kind of genetic operators, such as mutation or reproduction, in other words they have to be evolvable.

In the past decades many suitable representation models have been proposed, but here we will discuss only the most frequently used ones. Before that, the following list will enumerate the most important properties that should be taken into account when selecting the right decision unit model.

Search space: The smaller, the better. It is tempting to reject a simple model in the mistaken belief that the solution lies in greater complexity. Remember that most self-organizing systems are composed of components following relatively simple rules and that evolutionary methods perform way better on smaller search spaces.

State: Ensure that the problem can be solved with the amount of memory provided by the decision unit model. In most cases, the ability to involve previous states of the system into the decision making is not only beneficial but already required. The idea is that the component becomes more aware of the ongoing dynamical processes in the system. On the other hand, involving too much memory would mean unnecessarily complex representations, thus larger search space.

White/Black box models: Currently there is no solid theory on how to evaluate self-organizing systems, or even to ensure that their behavior will always be within controlled limits. Therefore, an obvious step would be to reduce the obfuscation and prefer white box models over black box ones. Unfortunately, practice shows that general evolvability of a model is inversely proportional of its understandability.

In the field of optimization and artificial intelligence many models have already been proposed that at least partially fit the aforementioned criteria.

One of the simple models is decision trees, that are tree-like graphs of decision points and their possible consequences. They also include chance events, resource costs, and utility in order to provide a full representation of an algorithm. There have been many successful implementations of evolving decision trees from data mining [FaHY04] to robot control [Gre12] using different optimization algorithms[BK11][PK00]. The advantage of using decision trees is their relatively simple white-box model, which provides easy access to the resultant behavior being useful for later evaluation. On the other hand, it also forces the designer to predefine all possible outcomes of a decision, thus the outcome is bounded by the designer's knowledge of the target system. Furthermore, they posses very limited, if any, internal states, which is important to solve certain tasks [Pet02].

Another approach is to use finite state machines (FSMs), that are, broadly speaking, simple machines being in one (and only one) of a finite number of previously defined states. They can change from one state to another as a result of a triggering effect called transition. Their biggest advantage over other representations is their white box model, which can be described by a regular grammar. Although simple FSMs have very limited memory (depending on the number of states), more complex models like the Mealy Machine [Mea55] can sustain more states. Unfortunately all types of FSMs suffer from the problem of the limits set by the discretization of their possible inputs and outputs. In other words, fine-tuning a specific action would probably require a huge number of states and transitions that would defeat the purpose of the whole model. This increase of states also enlarges the search space presented by the FSM model, which makes the optimization task more difficult. Nevertheless, they also have a growing application field and can be evolved by various metaheuristic algorithms [SG10].

One of the most popular and oldest technique mentioned in conjunction with evolution are artificial neural networks (ANNs). These are simplified representations of biological neurons and synapses that have been applied in classification and many machine learning tasks in the last 50 years. Since ANNs can theoretically approximate any non-linear function with enough neurons, they exhibit excellent capabilities, such as noise tolerance and generalization. Some versions of them also have some consistent internal states. Furthermore, these benefits can be further combined with other representations such as fuzzy logic or decision trees [LH05]. The price one has to pay for these properties is that neural networks are black box models, meaning that it is an extremely difficult task to reverse-engineer and understand how a given ANN works. They are very parameter intensive, increasing the search space rapidly as new neurons are introduced.

An interesting and widely-used approach is genetic programming (GP) [Koz92]. The idea, first introduced in 1985 [Cra85] is that the representation is a complete computer program mostly represented in a tree structure, where the nodes are either operators or operands. Thus, programming languages that are naturally based on tree structures are favored (e.g., Lisp). The advantage of GP is that it does not impose any fixed length on the solution, thus the algorithm can find the size necessary for a solution itself. Since GP also provides a complete program, requirements such as special functions or memory can be easily added to it. However, the main disadvantage of this approach is the immense search space usually associated with it.

3.3.4 Search Algorithm

The task of the search algorithm is to optimize the decision unit models of the components according to the objective function that is typically defined on the global behavior of the system. Mathematically we can define a design vector

$$\mathbf{x} = (x_1, x_2, \ldots, x_n)^T \tag{3.1}$$

where the components x_i of $\mathbf{x}$ are the decision variables. The objective function

$$f_i(\mathbf{x}),\ (i = 1, 2, \ldots, N) \tag{3.2}$$

describes $i = 1, 2, \ldots, N$ single objectives. The space spanned by the decision variables is called the search space, while the space formed by the objective function values is called the solution space.

When the optimization problem is formulated correctly, the main task is to find the optimal solutions by some iterative mathematical solution. As figuratively expressed by Rechenberg [Rec94], it is like finding the tallest hill in an unknown landscape. Initially we look at random places and then move to another plausible location and so on, until the we reach the termination criteria, which is either the maximum number of allowed steps or another indicator that the solution cannot be further improved.

Some algorithms such as simulated annealing use single point exploration, much like a trajectory-based search that is in general very effective in simple, mostly unimodal landscapes. Unfortunately, the solution space of evolved self-organizing systems is rarely that simple, and thus requires more sophisticated search algorithms. Such algorithms usually employ many parallel exploration agents sharing information, thus applying the principles of the so-called swarm intelligence. A well-known example of such an algorithm would be particle swarm optimization [KE95]. Additionally most modern algorithms use techniques to improve the search agents during the process. They use only the best

solutions (or agents) and apply random-driven operators on them to replace the worst ones, while evaluating each individual's fitness in combination with the system history.

When the search space is extremely large, the search process might take a very long time. Theoretically, if there is no time limit and every point of the search space is accessible by the algorithm, it is possible to find the global solution for the problem. Due to the fact it is not practical to evaluate every point in the solution space, the aim is to find good feasible solutions in an acceptable timescale. Unfortunately, there is no guarantee that the best solution will be found, but the idea is to use an efficient algorithm that most of the time produces good quality solutions.

As stated above, meta-heuristic algorithms try to compromise between local search and randomization. These two components are also known as intensification and diversification, or exploitation and exploration. The former tries to refine the actual best solutions in order that the solutions converge towards optimality. On the other hand, exploration via randomization helps to avoid getting stuck at a local optima, and at the same time increases the diversity of the solutions.

There have been many such efficient algorithms proposed for evolving systems. However, there exists so far no single algorithm or a guideline that tells the designer which one is best suited for a given problem. In this thesis we address the evolvability of self-organizing technical systems from various perspectives. Therefore, we restrain ourselves to use only several well-known evolutionary algorithms. A complete demonstration and comparison of available approaches is thus outside the scope of this work. Nevertheless, a good overview of available meta-heuristic algorithms and their application fields can be seen in [Yan08].

3.3.5 Engineering the objective function

The objective function is essentially a mathematical representation of the force that drives the optimization algorithm towards good solutions. In literature, it has many names, such as fitness function, cost function, or utility function and can be the target for maximization or minimization depending on the problem.

As written in Equation 3.1, it either has one or many objectives resulting in a single or multi-objective optimization problem. Multi-objective functions can also be partially substituted with a single-objective function which encapsulates all the objectives in a priority sequence such as a weighted sum. The advantage here is that the performance of a candidate can be represented in a single numerical value instead of a vector which could make analysis much

easier. However, finding the correct utility values (i.e., weights) is generally not a straightforward task. Additionally, multi-objective functions offer the possibility to analyze the solution's behavior with respect to all defined objectives. Thus, they provide a deeper insight on the driving forces of the component (see Chapter 6).

The fitness function that rewards the desired emergent behavior is usually highly problem-dependent, although there are studies available in evolutionary robotics on possible generic methods [NBD09]. In order to classify each of them, Floreano and Urzelai *et al.* proposed a three-dimensional *fitness space* [FU00], which can be generalized for evolving self-organizing systems with the following dimensions:

Functional vs. behavioral: A functional fitness is based on components that directly measure the way in which the system functions. A behavioral one however, rewards the system for displaying a given behavior. This can be also seen from the perspective of instantaneous or temporal fitness.

Global vs. local: Global fitness rewards the system based on information that is available to an external observer, while the local one is restricted to information available to a single component. Usually, it is easier to use global fitness, however it carries the risk of introducing some bias from the experimenter.

Explicit vs. implicit: An explicit function rewards *the way in which* a certain goal is achieved (e.g., a trajectory), while implicit fitness is focused on *how much* the goal is reached (e.g., a distance). Implicit functions are also extensively used in search algorithms operating the behavioral space [LS11].

In certain cases, analyzing the results of different evolutionary experiments might turn out to be difficult, if for example, a clear best approach cannot be determined. This mostly happens due to the poor expressiveness of the fitness function. Under this we mean the following properties:

Comparable: This property indicates if a clear relation between any two values can be determined. If so, the function is fully comparable, otherwise a dominance relation will exist. Typically, non-dominance happens when using multi-dimensional fitness functions, for example between values on the Pareto-front.

Transitivity: An objective function is transitive if the following property applies to it:

$$f(a) > f(b) \wedge f(b) > f(c) \Rightarrow f(a) > f(c), \text{ where } \{a, b, c\} \in S$$

Selection based on non-transitive functions is difficult to define. A possible but more resource-consuming approach is to perform a full set of pariwise comparisons resulting in $^{n(n-1)}/_{2}$ comparisons. If this is not possible, approximate ranking can be used for a cost of accuracy [EIF09].

Normalized: Indicates if the function returns values between given bounds (typically 0 and 1). This property can be useful for particular EAs, for example when using roulette-wheel selection in evolutionary algorithms.

Evolutionary design regularly receives criticism saying that if the designer knows a good objective function, then the task is already solved. However, in the field of evolving complex systems, where the level of interconnectivity of the components is very high, it is extremely difficult to directly relate to the system's behavior even if the local rules are clearly understood. However, this criticism also reveals the importance of the objective function in the design process. In fact, it is the heart of the whole evolutionary approach, since the resulting system can only account for the behavior that is expressed by this function.

While it is almost impossible to give a general guide on objective function design, the following set of principles can guide the designer:

Type: Consider the expectations from the system: is the aim to bring the system in a specific state under given conditions (e.g., distributed synchronization), or the focus is on the changes of the environment as an effect of the system (e.g., foraging)? Knowing the right focus will improve the quality of the objective function.

Dynamics: Self-organizing systems are almost never static. In fact, their emergent behavior is a result of the underlying dynamics of their components. Therefore, it is advised to formulate a function that considers these dynamics. In other words, the objective function should be defined on a dimensionless timescale, rather than in a specific point in time (e.g., end of the simulation run).

Knowledge: It is often beneficial to add as much information to the function as possible. While a possible solution to this might appear too complex to the human designer, there could exist a simple solution discovered by

evolution. However, care should be taken not to introduce too much bias by the experimenter, which would lead to sub-optimal results. Evaluation of the initial results can help to reveal such problems.

Fail-safe: As above, states that should be avoided should also go into the function to ensure fail-safe operation.

Guidance: Multiple objectives can form a multi-objective function, but in general it is more advised to find hierarchical levels between objectives in order to guide evolution. If an objective proves to be too difficult for the system, it might help to decompose it into simpler sub-objectives with lower utility value, for example, to evolve robots playing soccer it is a good approach to reward kicking since it directly correlates to the number of goals, and consequently to the fitness of the solution.

CHAPTER

4 FREVO: A Tool to Design Self-organizing Systems

If you want to teach people a new way of thinking, don't bother trying to teach them. Instead, give them a tool, the use of which will lead to new ways of thinking.

– Richard Buckminster Fuller

Traditionally, systems are built like a jigsaw puzzle – each system component has to fit in order to get a correct working system. When building complex systems, this approach is very difficult to maintain. One drawback is that the designer of a self-organizing system has to give up "direct" control. Instead, the intended goal is achieved indirectly by defining the micro-level behavior. This is typically a very difficult task, since the global behavior of a system of interacting agents can hardly be predicted for a given set of local rules. In some cases, the emergent behavior appears to be the opposite of what is expected [Res97].

In order to apply the earlier proposed design methodology, we need a proper software tool that fully supports this approach. While there are already many powerful frameworks for agent-based simulations and evolutionary methods, here is a need for a generalization of design and implementation of such self-organizing systems to reduce the set-up time for a problem and improve its evaluation possibilities. In this chapter we introduce FREVO (FRamework for EVOlutionary design), a framework that reflects the component-wise thinking required for the evolutionary approach. FREVO splits the design of a system into a problem, representation, and optimization allowing for exchange of different parts seamlessly. Appropriate interfaces simplify the design process, and it supports statistics and graph generation in order to help the evaluation of such self-organizing controllers. Furthermore, it is possible to validate and evaluate the obtained results on a large scale of parameters within the tool. FREVO has been successfully applied to various problems, from cooperative robotics

to economics, pattern generation and wireless sensor networks. Parts of this chapter have been published and presented at the 1st International Workshop on Evaluation for Self-adaptive and Self-organizing Systems [SFE12].

4.1 State-of-the-Art

Agent-based modeling is one of the most frequently used approaches in the domain of design and analysis of complex systems. Consequently, there is a vast number of software tools offering features for building large-scale distributed systems. For example, JADE [BR01] supports the development of agent applications in compliance with the FIPA[1] specifications. The Repast suite [MN09] integrates many existing modeling and simulation tools from the last 14 years. MASON [LCRP+05] focuses on simulation and visualization of multi-agent systems. The problem with these tools is that even though they are very powerful in code re-usability and flexibility, they often do not include advanced machine learning or optimization features, or they lack any bridging between them. In these cases the user is left to re-implement the required features which often need deep programming knowledge or expertise in algorithms.

In the field of optimization and artificial intelligence, more and more researchers tend to make their implementation public. However, these packages focus only on one specific optimization method (e.g., genetic algorithms [Mef12]) or on one target representation (e.g., neural networks [SBW+10]). On the other hand, general-purpose libraries provide a larger set of methods and/or representations to the user [IGHM08, Abe09], but they either lack the possibilities for the user to easily exchange methods and representations within the same problem.

4.2 FREVO Architecture

Our proposed framework FREVO, supports engineers in evolutionary design and evaluation of self-organizing systems. It provides the necessary tools to make an agent-based model of a desired self-organizing system and to search for the required local interaction rules using iterative heuristic search. In addition, FREVO supports the evaluation of evolved solutions under predefined conditions.

The main feature of FREVO is the component-wise separation of the key building blocks that conforms to the evolutionary design methodology. In short,

[1] http://www.fipa.org

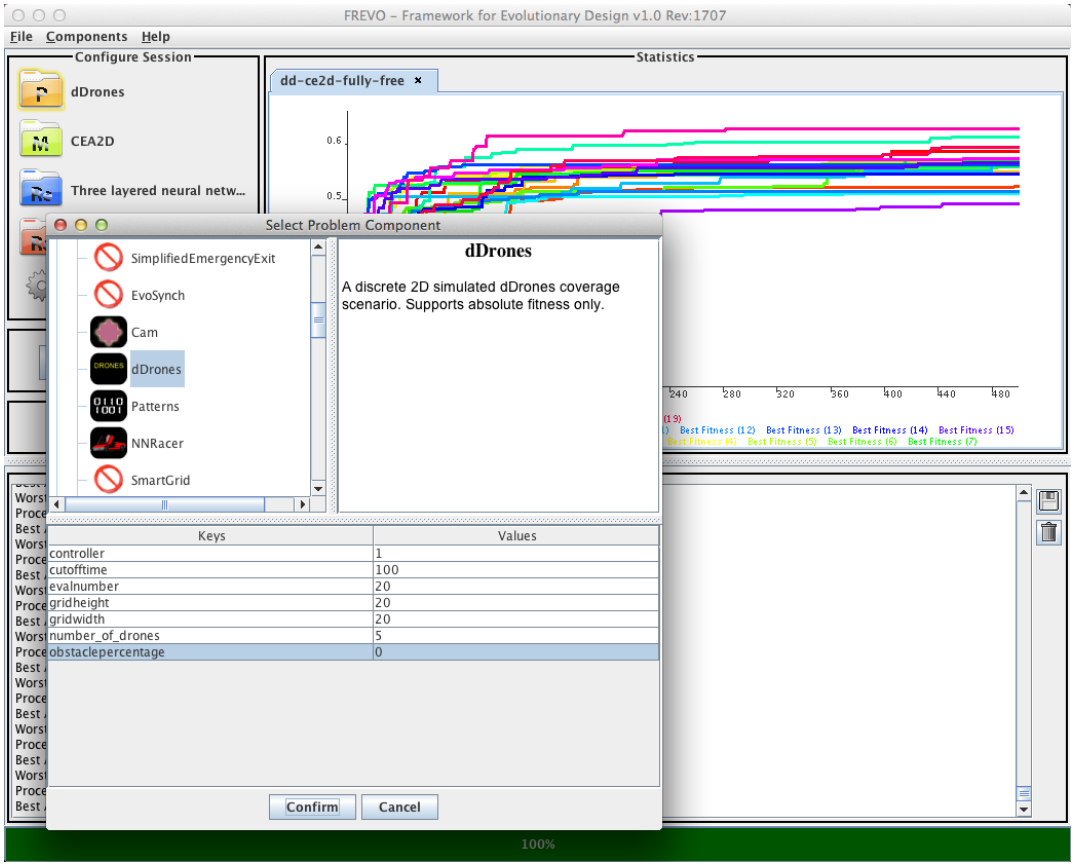

Figure 4.1: Overview of FREVO's graphical interface while selecting a problem component

the *problem* component models the ecology of the agents along with the goals of the simulation. The models of the system's components are defined in the *representation* part and they are optimized with the algorithm defined in the *method*. This structure enables the components to be designed separately allowing the user to easily change and evaluate different configurations, methods, and representations. FREVO is written in Java and makes use of Java's object-oriented model by defining interfaces to generic parent components. When a new component type needs to be added, the user is required to write the code for the component. In this task, the user is assisted by a built-in component generator tool and guided by the methods required by Java's interfaces. Problems can be easily developed within the Java language, but FREVO also provides helper classes to connect to an external simulation tool. Such architecture also allows the exchange of research ideas or engineering solutions in a flexible way.

For easier access, FREVO comes with a graphical user interface allowing the engineer to pick the particular components for a project (see Fig-

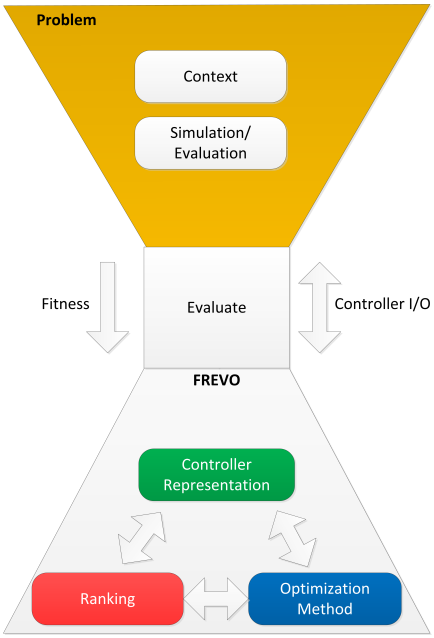

Figure 4.2: FREVO components: (problem, optimization Method (blue), representation and ranking)

ure 4.1). This component concept also supports fast evaluation of different configurations, for example, in order to see which controller representation (e.g., neural network vs. finite state machines) works better for solving a given problem. FREVO is released as open source under GPL v3.0 (available at: `http://http://frevo.sourceforge.net/`).

In the following sections, we define the four component types within FREVO. As depicted in Figure 4.2, the architecture introduces a waistline interface between the problem (top) and the other parts of FREVO. This eases modeling of a new problem – only a few methods have to be provided: an interface for connecting the agent's I/O to the agent controllers, a method for evaluating the problem (typically by a simulation run) and a fitness value that is given as feedback from an evaluation. A single composition of a problem, method, representation and ranking defines a so-called FREVO session containing details of parameters set for the experiment.

In order to store simulation data for later experiments and evaluations, FREVO saves individual sessions and results in a compact XML format that can be easily accessed, displayed and archived using tools available within the program.

MyProblem.java

```
1  package utils;
2
3  import core.AbstractRepresentation;
4  import core.AbstractSingleProblem;
5
6  public class MyProblem extends AbstractSingleProblem {
7
8      @Override
9      protected double evaluateCandidate(AbstractRepresentation candidate) {
10         // Your fitness function comes here
11         return 0;
12     }
13
14     @Override
15     public double getMaximumFitness() {
16         // Returns the fitness value corresponding to the best possible solution
17         return 0;
18     }
19
20 }
21
```

Figure 4.3: Implementing a new problem class in FREVO. The user is required to implement the objective function and a simple getter function that returns the theoretically maximum value (if available)

4.2.1 Problem definition

In the problem definition, the evaluation context of the agent's behavior has to be implemented. In other words, this component is responsible for the evaluation of the candidate representations. From the perspective of the design architecture, this part is responsible for defining the ecology of the agents i.e., how they interact with each other and their environment. One has to also implement the objective function here that drives the heuristic optimization process. Currently, FREVO supports normal fitness functions in the genotype space, as well as and functions that measure behavioral diversity in the phenotype space.

Due to the simplified nature of interfaces within FREVO, implementing a new model takes little effort. It is also possible to connect FREVO with external simulation tools such as JProwler [SVML03] or ArGoS2 [PTO+12].

There are two types of problem components: The first variant is to implement a subclass of `AbstractSingleProblem` (see Figure 4.3), it connects a controller into a simulation and returns a fitness value as the result. In a multi-agent system, each controller would have the same behavior. An example of this problem type would be a group of robots in a cooperative search mission.

The other variant, implemented as a subclass from `AbstractMultiProblem`, evaluates multiple candidates relative to each other. An example would be a simulation of two soccer teams playing each other. The result of such a simulation only gives a relative ranking and requires to run a tournament algorithm

to get a ranking of a pool of candidates. Details on the actual implementation of problem components will be elaborated in more detail in later chapters.

4.2.2 Candidate Representation

The candidate representation describes the structure and model of the agent's controller, i.e., the representation of a possible solution. In principle, FREVO does not operate on candidates that are simple data structures, instead they have to contain a generic structure that encodes a reactive behavior as described in Chapter 3. For example, artificial neural networks (ANNs), decision trees, genetic programs, or finite state machines. Furthermore, this implementation must extend from the `AbstractRepresentation` class that is agnostic about the selected problem or the optimization method. Reproduction operators (e.g., mutation, crossover) have to be defined within this component. Additionally, these components can optionally support different output formats for later processing or analysis. For example, the ANN representation provides an export of the network structure in the Pajek [dNMB02] format.

FREVO also support problems that evaluate a set of candidates instead of a single one in the form of bulk representations. Such heterogeneous systems usually require a mix of different types of agents. A good example would be a wireless sensor network where some nodes are mostly responsible for relaying information while other nodes do some onboard processing as well. The application of heterogeneous swarms of robots is also focus of many research activities.

The base release version of FREVO comes with the following representations:

- *Fully-meshed net*: A time-discrete, recurrent ANN with one hidden layer where each neuron is connected to every other neuron and itself. During evolution, the biases of each neuron, plus the connection weights are taken into account. This representation support adaptive mutations.

- *Three-layered net*: A similar ANN to the previous one, but with a feed-forward structure instead of the fully-meshed one. Provides a lower search-space for less complex problems. This representation is suitable for problems of lower complexity. For the same number of neurons, the three-layered net comes with a significant smaller search space than the fully-meshed net.

- *NEAT net*: An ANN whose connectivity structure is also taken into account for selection during evolution. This representation was imple-

mented based on the Neuroevolution of Augmenting Topologies (NEAT) model described in [SM02].

- *HebbNet*: A fully connected, recurrent artificial neural network with hebbian learning. This is an unsupervised learning method which changes the ANN connection weights during evolution. Each synapse weight is assigned a plasticity which defines its ability to learn. Plasticity and initial weights are evolved.
- *MealyFSM*: A Mealy Machine [Mea55] whose structure and transition probabilities are evolved.
- *SimpleBulkRepresentation*: Encapsulates a composition of various other representations.

4.2.3 Optimization Method

The optimization method (`AbstractMethod`) is used to maximize the fitness returned from the problem definition. Typically, an optimization method creates a pool of possible candidates from the solution representations, evaluates them using the problem definition and gradually obtains candidates with better performance. FREVO support evaluations in the genotype level with traditional fitness measures, as well as evaluation functions defined on the behavioral space as initially proposed by Lehman and Stanley *et al.* [LS11]. Methods never implement reproduction or randomization functions that are executed directly on the genotype. Instead, they can call for certain types of operators that are implemented within the provided representation class.

Currently, we have the following two optimization methods provided in the release version of FREVO:

- *RandomSearch*: A very simple optimization algorithm that replaces candidates with low fitness values with completely random ones. This method is meant to be a base for comparison only.
- *NNGA*: An evolutionary algorithm that supports multiple populations, different selection schemes (e.g., roulette wheel selection) while trying to maximize population diversity [EK07]. The name NNGA stands for *Neural Network Genetic Algorithm* but it works for other representations as well.
- *GASpecies*: An evolutionary algorithm that sorts candidates into species based on a similarity function defined on their genotypes (similar structures are put into the same species). In order to prevent immature solu-

tions from dying out too early candidates share their fitness with others in the same species. For more details of this technique see [DG89].

- *CEA2D*: A cellular evolutionary algorithm that arranges candidates in a two-dimensional map and performs genetic operations such as selection, mutation and recombination in a local context, i.e., in the respective Moore neighborhood of a candidate. This algorithm has a slower convergence rate, but better population diversity than a standard genetic algorithm.

- *NoveltySearch*: A genetic algorithm that rewards behavioral diversity instead of normal fitness. Code is based on Ken Stanley' rtNEAT implementation (UT Research Licence) [Sta08].

- *NoveltySpecies*: Similar to the one above with speciation included to support complex structures like *NEAT*.

4.2.4 Ranking

The ranking module evaluates all candidates and returns a descending sorted list based on their performance. For problems that give an absolute value of fitness (e.g., the control problem of an inverted pendulum), this means evaluating each candidate and then sorting them according to their fitness. In case of problems derived from `AbstractMultiProblem]`, solutions can only be compared relative to each other (e.g., a soccer team). Thus, it is necessary to infer the ranking from pairwise comparisons. For this, FREVO allows one to choose between several *ranking mechanisms*, defining how to pair the solutions in order to find a dependable ranking with a low number of comparisons (i.e. simulation runs). This concept will be elaborated in more detail in Chapter 7. Additionally, ranking components are responsible for parallelizing evaluations in order to decrease the overall simulation time.

At this time, FREVO contains the following ranking components:

- *AbsoluteRanking*: A basic component that sorts candidates based their fitness values returned directly from the problem component. Supports multi-threading to improve evaluation speed.

- *FullTournament*: Ranking designed for problems where an absolute fitness cannot be determined. Sorts the candidates based on a round-robin style tournament that requires $n(n-1)/2$ number of pairings for populations of n candidates.

- *SwissSystem*: Similar as above, but is based on a swiss-style tournament that uses only $n\lceil \log_2 n \rceil$ number of pairings.
- *MultiSort*: A sorting algorithm which ranks by always evaluating a group of randomly selected entities where each entity will be selected with equal times.
- *MultiSwiss*: This ranking method is a combination of the two components above. It is capable of evaluating problems with any number of players.
- *NoveltyRanking*: Ranks candidates based on their novelty in the behavior space instead of fitness. Can be applied in a similar way as *AbsoluteRanking*.

For an overview of component class hierarchies see Figure 4.4.

4.3 FREVO Example Problem

FREVO has been used for the evaluation of many case studies, involving evolutionary robotics [FE09b, FE09a, FE10, PBSE12], pattern generation [EF11], and wireless sensor networks [CLNR12].

In this section, we show the capabilities of FREVO by way of a simple scenario that investigates different control algorithms for a set of self-organized unmanned aerial vehicles in order to cooperatively achieve maximum coverage of a partially obstructed area under certain constraints like limited time (minimum time-to-complete) and minimum energy (with minimum number of turns) [Cho01]. The area to be covered is modeled as a time-discrete and space-discrete lattice of obstacles and free space. We evolve and test different controllers based on artificial neural networks and compare them with naive algorithms like random walk and random direction. We will not go into deep analysis on the obtained results, since the aim here is to demonstrate what is possible with FREVO.

Using FREVO, we define this scenario as a problem component. The provided component creator helps to set up the necessary class with the needed methods and the required XML configuration file. Since an agent's performance can be directly assessed we chose our problem to be a child of the `AbstractSingleProblem` class. The design of the problem requires the definition of the environment, the input values and output types for the candidate representation being evaluated (which we do not have to implement). Additionally, we need to define a fitness function that will be used by the optimization method. In this experiment we study only homogeneous sets, thus every agent

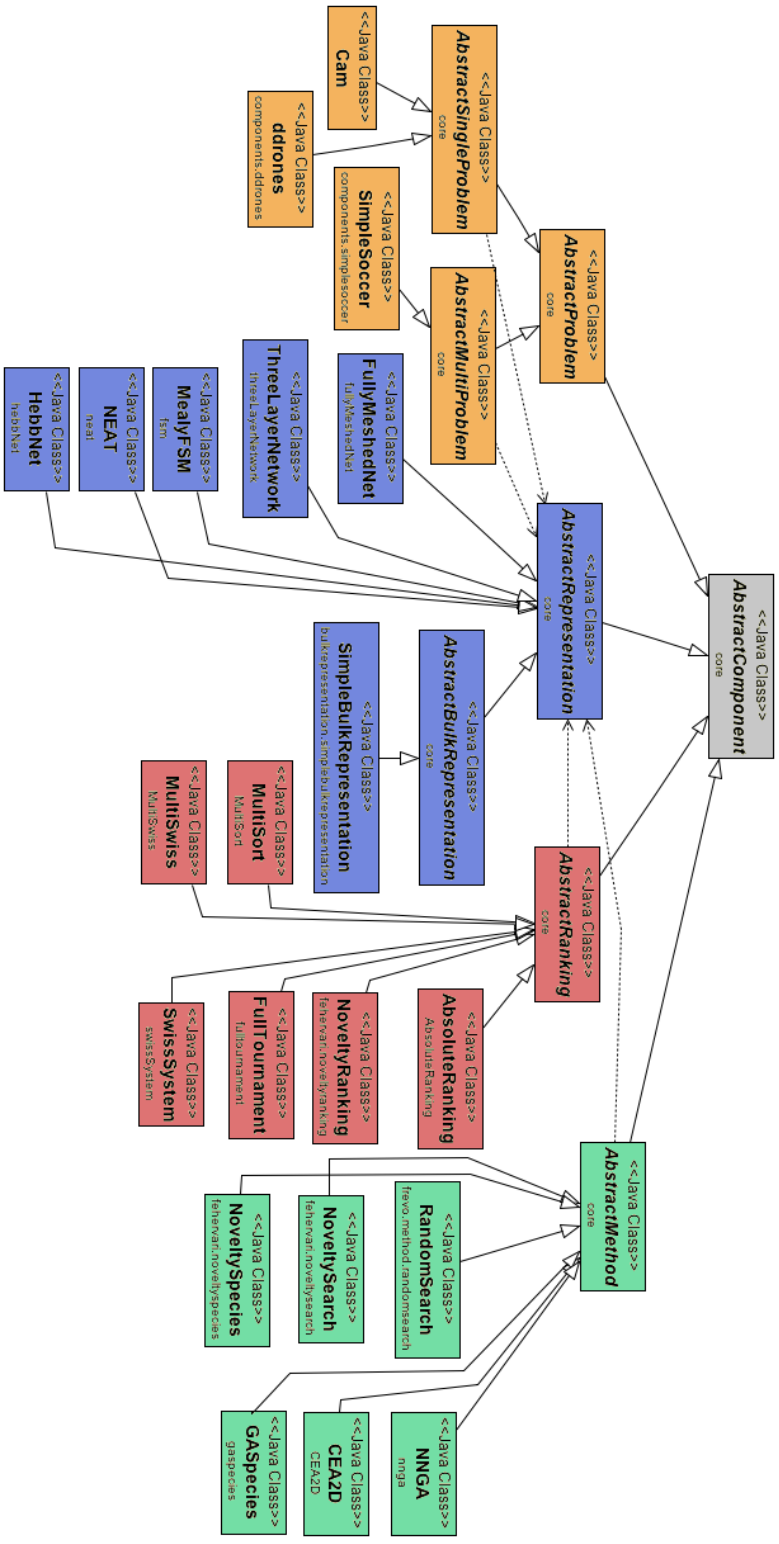

Figure 4.4: FREVO's components with class generalizations and dependencies

will be initially created as an identical copy of the same candidate representation that is being evaluated.

We model the environment as a 2-dimensional grid, whose size can be adjusted with 2 parameters (width/height). During simulation, a cell can be either free, blocked, or occupied by a single agent. We will run all experiments both without any obstacles, as well as with predefined amount of grid cells blocked. In each time step of the simulation, every agent receives sensory information about their direct von Neumann neighborhood regarding blocked cells and other agents. Afterwards, they can decide to move in any of the neighboring cells that are free. In order to simplify the simulation, we use only binary inputs with 2 sensors indicating the presence of obstacles/other agents, such that we end up with 2×4 number of inputs indicating if the related cell is blocked/occupied or not. A single output of the agent's controller is mapped to the four possible states (up, down, left, right).

In order to obtain a fair comparison between various control strategies and evolutionary techniques we measure the fitness as the average coverage rate achieved over all runs. We define a cell covered if it has been visited at least once by one of the agents. Therefore, multiple visits do not improve the fitness any further.

4.3.1 Experimental setup

For our scenario, we select our problem together with an existing representation, optimization and ranking method. In the following paragraphs, we will explore which combination of optimization method and candidate representation allows for the most stable results. For the optimization method, we make a session with *NNGA* and another one using the *CEA2D* method. The representation can be either a three-layered neural network, or a fully-meshed one with 2 hidden neurons. We chose empirically to evolve 500 generations, because the fitness values stabilized. Our problem is configured for 5 agents with initial random positions on a 20×20 grid with either 0% or 15% of blocked cells. The length of a single run is set to 100 steps. The fitness is calculated based on the system state after these 100 steps. To account for random effects, we repeat every single evaluation and each complete evolution run 20 times with different random seeds and present the fitness development as box plots per generation (FREVO generates automatically the correct output for generating boxplots of the fitness with R). We configured the evolutionary methods with a population size of 50, 40% mutation, 30% crossover rate and 15% elite selection. Table 4.3.1 summarizes the tested experiment configurations.

Obstacle %	Method	Representation
0	*NNGA*	*Three-layered net*
0	*NNGA*	*Fully-meshed net*
0	*CEA2D*	*Three-layered net*
0	*CEA2D*	*Fully-meshed net*
15	*NNGA*	*Three-layered net*
15	*NNGA*	*Fully-meshed net*
15	*CEA2D*	*Three-layered net*
15	*CEA2D*	*Fully-meshed net*

Table 4.1: Configurations used for the case study.

4.3.2 Results

Let us first investigate the effect of the different neural network controllers. As seen on Figure 4.5, the average area covered by the three-layered network is around 10%, which is much less compared to its fully-meshed counterpart with 36%. This effect can be explained by the fact that feedforward neural networks do not possess any feedback mechanisms that could serve as some memory. Such simple controllers cannot even perform a regular sweeping behavior. However, fully-meshed networks are able to cope with situations where they require at least the information of which direction they came from, thus can exhibit more complex behavior.

Figure 4.6 shows the comparison of different evolutionary algorithms working on the fully-meshed network with 15% blocked cells. The results indicate a higher overall performance with a maximum of approximately 56% coverage when the cellular evolution is being used. This is most likely accounted to the higher diversity kept during evolution, which is shown on Figure 4.7. One can also observe a lower fitness variation indicating a more stable run. Our results indicate that fully-meshed net performs significantly better than the three-layered one. This problem is easier to be solved with agents that memorize an internal state during evolution.

It is easy to define different scenarios to be compared using FREVO. One can decide what optimization method or candidate representation results in the most stable and best performing simulation/system for a given problem. The next step is to validate and evaluate the resulting candidate representation. FREVO generates result files that can be loaded after evolving the candidates. The result file contains all settings and a complete state of every representation in the population of the last evolved generation, ranked by fitness. One can also configure in the optimization settings that the results files are periodically written every n^{th} generation.

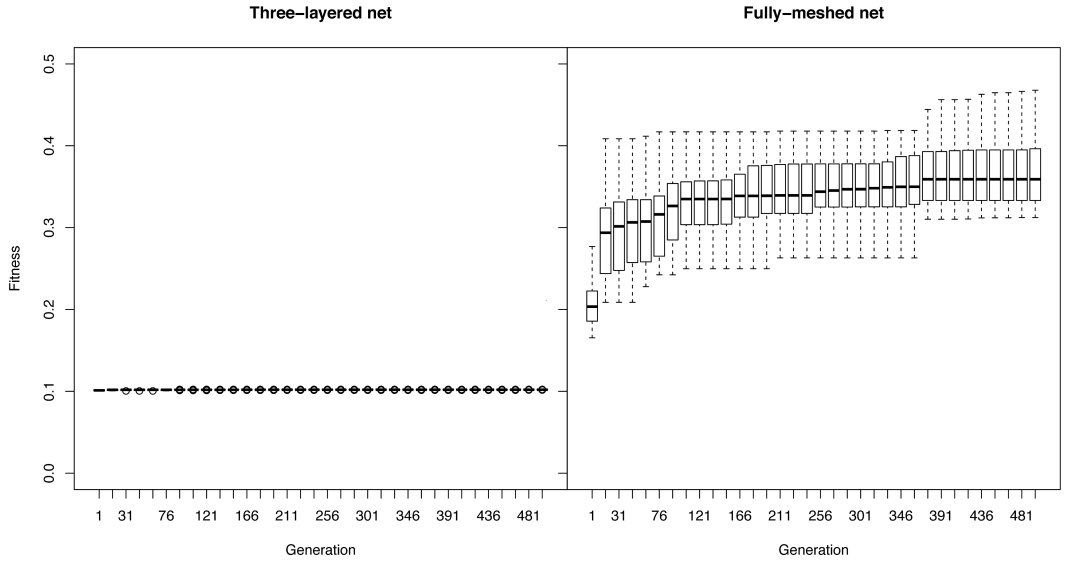

Figure 4.5: Fitness development vs. number of generations with different neural networks. Optimization method: *NNGA*, obstacle percentage: 0%. The *Fully-meshed net* clearly outperforms the *Three-layered net* in every generation.

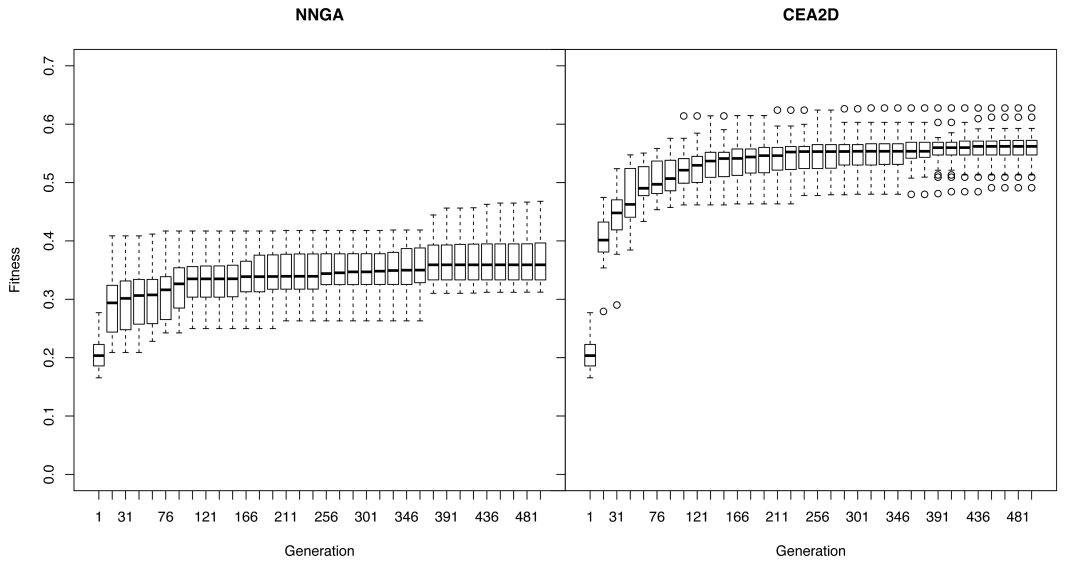

Figure 4.6: Fitness development vs. number of generations with different optimization methods. Representation: *Fully-meshed net*, obstacle percentage: 15%. *CEA2D* outperforms *NNGA* in almost every generation.

An overview is given in Figure 4.8, where, on the left-hand side, the candidates are listed. By selecting one candidate, it can be replayed with the same settings as used during the evaluation. In order to test with different settings, the parameters can be adjusted in the window to the right. For example, the

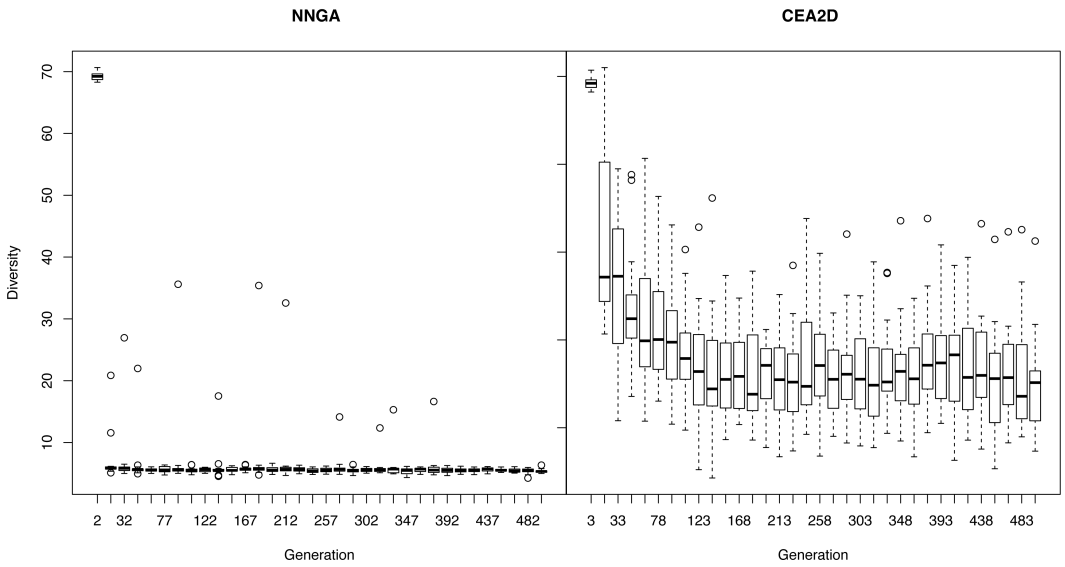

Figure 4.7: Diversity development vs. number of generations, optimization method: *NNGA* (left), *CEA2D* (right) representation: *Fully-meshed net*, obstacle percentage: 15%.

scalability of the evolved candidate can be evaluated by adjusting the number of drones, or we can test simple, non-evolved controllers, like random walk (drone moves in a random direction each time step), or random direction (drone moves in a direction that is changed randomly if an obstacle is encountered). FREVO gives us the average fitness of 100 runs by quickly running 100 simulations with different seeds. In our scenario, random walk provides a coverage of approximately 41.2%, while random direction 47.1%.

An additional feature of FREVO is the possibility to implement graphical evaluations, to learn and validate the evolved behavior. For our scenario, we implemented a simple display class, that is based on the `JGridMap`[2] open-source grid visualization engine. A sample of the drones exploring a partially obstructed are can be seen on Figure 4.9.

4.4 Summary

We introduced FREVO, a tool for designing and evaluating self-organizing systems. FREVO concentrates on evolutionary methods for agent controllers, as often applied in autonomous robots, but extends this principle to arbitrary applications. With the principle of reusable, independent components, FREVO

[2]http://sourceforge.net/projects/jgridmap

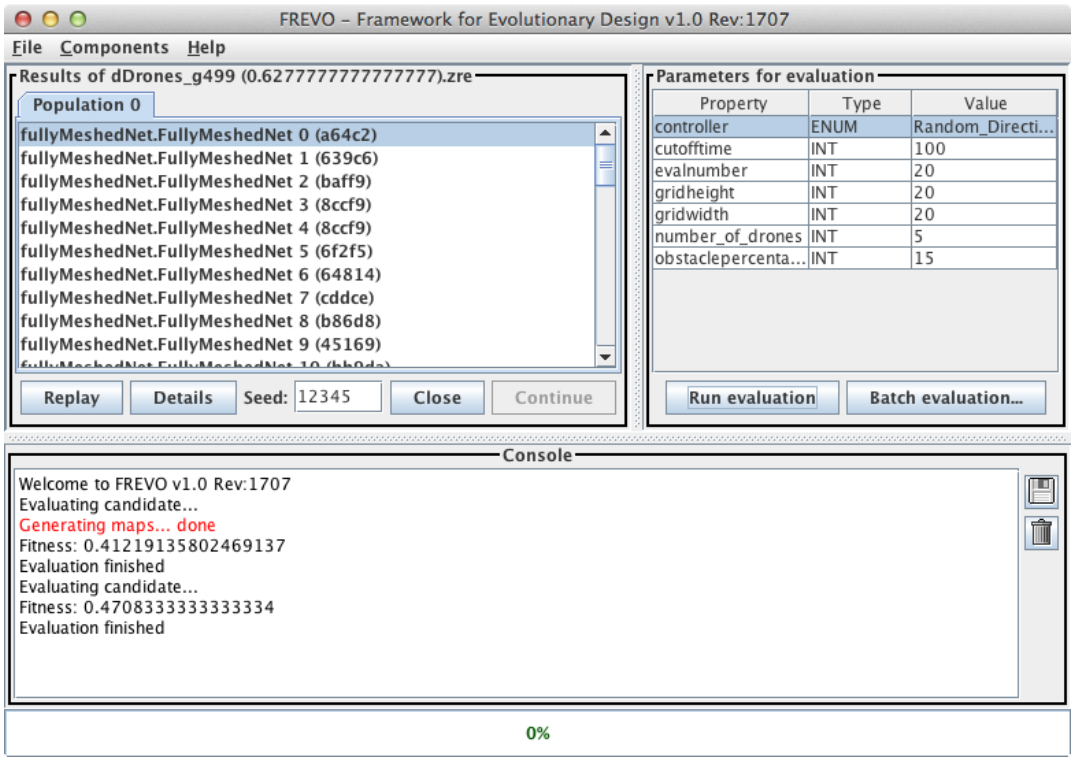

Figure 4.8: FREVO in evaluation mode. Left: list of candidates, right: configurable parameters

allows for easily exchanging different implementations of evolvable agent controllers, optimization methods and ranking methods. A simple example can be implemented with a few lines of code. The implementation effort is thus reduced to defining the context, the fitness function and define input and output of the agent. After evolving the agent controllers, the simulations with the resulting candidates can be replayed either with the same settings or with different parameters for evaluation purposes. As an example, we implemented a system of mobile agents with the task of covering a partially obstructed area defined on a simple grid. We compered two different neural network controllers and two different evolutionary algorithms. We also showed how simple algorithms perform in this task.

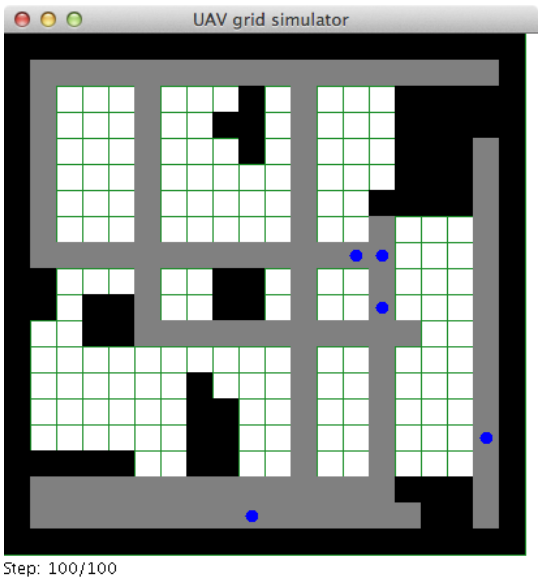

Figure 4.9: UAV example project visualization with 5 drones and 15% obstacles. Black cells are blocked, while grey ones are covered. The blue dots indicate the position of the drones.

CHAPTER

5 Evolving Spatial Patterns via Self-organized Agents

The phenomenon of emergence takes place at critical points of instability that arise from fluctuations in the environment, amplified by feedback loops. Emergence results in the creation of novelty, and this novelty is often qualitatively different from the phenomenon out of which it emerged.

– Fritjof Capra, *The Hidden Connections*

This chapter depicts and evaluates the evolutionary design process for generating a complex self-organizing multicellular system based on cellular automata (CA). In the presented model the cell behavior is controlled by an artificial neural network according to the cell's internal state. The idea is to evolve the cell rules in a way that a previously defined reference pattern emerges by interaction of the cells. Unlike similar approaches that operate with well-defined morphogens, in our study we try to minimize the designer's bias by including the communication protocol in the genotype as well. This problem is easily understandable and provides an accessible way to explore the domain of evolving self-organizing CAs. We test reference patterns of different complexity, for example flags, natural patterns, and paintings in order to map their evolvability with the corresponding complexity. Furthermore, we analyze the performance of different fixed neural network structures to understand what level of complexity is required for a nontrivial structure. We then compare this result to an approach that tries to find the optimal neural network structure for a given problem.

We find that generating simple regular structures such as flags or shapes can be learned relatively easy, but for complex patterns for example paintings or photographs the output is only a rough approximation of the overall mean color scheme. Thus, an increase of complexity dramatically lowers the quality of the evolved solutions, while the number of steps required to produce a given image seems to be unaffected by the difficulty of the problem. Our results also

indicate that complex images present a very difficult problem which cannot be solved with the current approach even with neural networks with more complex structures.

The application of a genotypical template for all cells in the automaton greatly reduces the search space for the evolutionary algorithm, which makes the presented morphogenetic approach a promising and innovative method for overcoming the complexity limits of evolutionary design approaches. Parts of this chapter have been published and presented at the Fifth International Workshop on Self-Organizing Systems (IWSOS 2011) [EF11].

5.1 Introduction

As shown by many examples in nature, simple interaction rules between subsystems can lead to quite complex emergent behavior [CFS+01]. Pattern formation studies visible and orderly results of such self-organizing systems. Many naturally occurring complex systems produce visible patterns, for example cell development and differentiation, chemical reaction-diffusion systems, and cloud formations. A typical purpose of patterns in living systems is visual mimicry, i.e., as the similarity in the appearance of one species to another (e.g., the yellow coloring and stripe patterns on several species of flies that mimic wasps). A special form of mimicry, called camouflage, occurs when a species appears similar to some abiotic environment. Some forms of mimicry (e.g., color change in some species of chameleons) are able to function at short time scales such that the organism can quickly induce a different pattern based on its environment.

Pattern formation in general is studied in developmental biology in terms of cell fate control through a morphogen gradient. A morphogen has been conceptually defined in the 1960s by Lewis Wolpert using the French flag model [Wol69] where the colors of the French flag represent the effects of the morphgen on cell differentiation. Herman and Liu [HL73] have solved this French flag problem through the simulation of linear iterative arrays of cells.

Patterns resembling natural ones can be reproduced by cellular automata (CA) models running simple algorithms. It is however, difficult to find the correct algorithm for an intended pattern. One possibility would be to evolve the rules using an evolutionary algorithm, but when using a model of elementary CA rules [Wei10], small changes in the ruleset can lead to large variations in the result. Nevertheless, there are notable examples of using evolution to obtain a desired pattern in a CA. Miller [Mil04] has created a simulated differentiated multicellular organism that resembles the structure and coloring of the French flag. He used a feed-forward Boolean circuit implementing a cell

program. The cell programs are evolved using a specialized genetic programming system. Chavoya and Duthen in [CD06] have used a genetic algorithm to evolve cellular automata that produce 2D and 3D shapes such as squares, diamonds, triangles, and circles. Furthermore, in [CD07], an artificial regulatory network (ARN) for cell pattern generation, producing the French flag pattern is evolved. Fontana generated predefined arbitrarily shaped 2D arrays of cells through an evolutionary-developmental technique [Fon08]. He was able to successfully generate complex patterns such as dolphin, hand, horse, foot, frog, and the French flag. Fontana has extended this work in [Fon09] to evolve complex 3D forms.

A typical problem in this domain is that as the system to be evolved becomes more complex, the evolutionary approach suffers from problems such as disruption of inheritance, premature convergence and failure to find a satisfying solution [BK99]. Yet, natural evolution has managed to create the very complex design of life. A main difference between natural and artificial evolution is, in many cases, the genotype-phenotype mapping. Natural organisms grow from a single cell into a complex system, while in many applications of artificial evolution, there is a one-to-one mapping from genotype to phenotype, leading to poor scalability. Therefore, there is a strong need for introducing generic genotype descriptions that can emerge into arbitrarily complex systems.

In this chapter we describe such an approach by the model of a cellular automaton where the state-transition logic of each cell is an instance of the same genotypical controller. Therefore, the control algorithm of every cell is implemented by a small artificial neural network that is evolved to reproduce a given pattern on the cellular automaton. The combination of neural networks and cellular automata which grow under evolutionary control is present in the China-Brain project [dGTH$^+$08], an effort to create an artificial brain of interacting small (typically 12-20 neurons) neural networks. This work, having several aspects in common, differs from the work presented here in two aspects: China-Brain is intended as a controller (e.g., to a robot) while in our problem the structure and form of the cells are the output, and second, the interconnections between the modules are designed explicitly by so-called brain architects, while they are an output of the evolutionary process for our model. Thus, we do not define any morphogens in our model *a priori*. Note, that this approach is not a simple classification task, where the cells need to learn what is the right output for a given context, since now direct spatial information is given. Instead, the cells have to first learn how to communicate efficiently, in order to be able to infer on their own position.

We tested the ability of the resulting pattern formation model for replicating several naturally occurring patterns (such as animal skin patterns) as well as man-made patterns (such as flags and paintings) of different complex-

ity. The remaining parts of this paper are structured as follows: The following section 5.2 introduces the model, i.e., the CA structure and the properties and interconnections of the ANN. We also discuss our complexity measurement approach. The evolutionary programming method is based on the FREVO framework (see Chapter 4), its application for the given problem is described in Section 5.3. Experiments including evolving several patterns and discussion of the results are elaborated in Section 5.4. A study on the optimal neural network structure is described in Section 5.5. Finally, Section 5.6 concludes the chapter and sketches possible further research and applications based on the presented approach.

5.2 Cellular Automaton Model

The used model consists of a regular rectangular grid matching exactly the resolution and proportion of the reference image (reference images are very small, typically 100-500 overall pixels). The colors of the reference image are converted into a scale, where neighboring colors resemble each other. This color transformation from RGB has been achieved by assembling a binary number by arranging the most significant bits of the channels R, G, and B, followed by the second most significant bits of those channels, and so on until the least significant bits, finally yielding a 24-bit color code.

The colors which are present in the reference image are then sorted according to the new scale and assigned to the possible output spectrum of the neural networks. Each color gets an equal proportion of the output space, which is continuous between 0 to 1.

Each cell is controlled by an ANN, which is modeled as a time-discrete, recurrent artificial neural network. Each neuron is connected to every other neuron and itself via several input connectors. Each connection is assigned a weight and each neuron is assigned a bias value. At each step, each neuron i builds the sum over its bias b_i and its incoming connection weights $w_j i$ multiplied by the current outputs of the neurons $j = 1, 2, ..., n$ feeding the connections. Weights can be any real number, thus have either an excitatory or inhibitory effect. The output of the neuron for step $k + 1$ is calculated by applying an activation function F:

$$o_i(k+1) = F(\sum_{j=0}^{n} w_{ji} o_j(k) + b_i)$$

where F is a simple linear threshold function

$$F(x) = \begin{cases} 0.0 & \text{if } x \leq 0.0 \\ x & \text{if } 0.0 < x < 1.0 \\ 1.0 & \text{if } x \geq 1.0 \end{cases}$$

In total each ANN consists of 9 input neurons, 5 output neurons and 6 hidden neurons. The 6 hidden neurons have been selected based on the recommendations by Boger and Guterman [BG97] and turned out to be an applicable number in order to balance between capability of the ANN and reduction of the search space. Nevertheless, in Section 5.5 we study the effects of different number of hidden neurons in the network. The number of neurons and structure of the ANNs was fixed at runtime, since the problem statement has no dynamic aspects and networks with fixed structure have shorter evolution times [FDM08]. One outgoing connection is used to define the cell's color and four pairs of incoming and outgoing neurons connect the ANN with cells of its von Neumann neighborhood. Furthermore, an ANN can sense the colors of these neighboring cells. This implementation does not impose an *a priori* strongly defined communication scheme on the cells i.e., it is not fixed *what information* the cells have to communicate with each other. Instead, evolution is expected to come up with a solution that allows the cells to efficiently exchange crucial data required for the desired emergent behavior.

The ANN of a cell does not get any direct information about its position in the grid. In order to allow a cell at a border or in corner to be able to infer about its position, these cells will receive a -1.0 input from the respective "missing" cells. During simulation, this information can propagate to the cells in the center via the inter-cell connections. Without this information or any external initiator mechanism emergence would not be possible.

Figure 5.1 depicts the interconnections between the ANNs via neighboring cells in the CA model. The light bulb, which is controlled by the ANN in the corresponding cell, indicates the current color state of a cell. Each cell gets an instantiation of the same ANN. When the CA is iterated, its ANNs can however acquire a different internal state that can be kept via self-holding loops and that can lead to differentiation in the ANN's behavior.

5.2.1 Measuring Complexity

In order to explore the limits of the evolutionary approach and to be able to compare different methods we need a measure to assess the difficulty of a particular problem. One solution could be to infer on the complexity of the desired emergent pattern i.e., the reference image to be evolved.

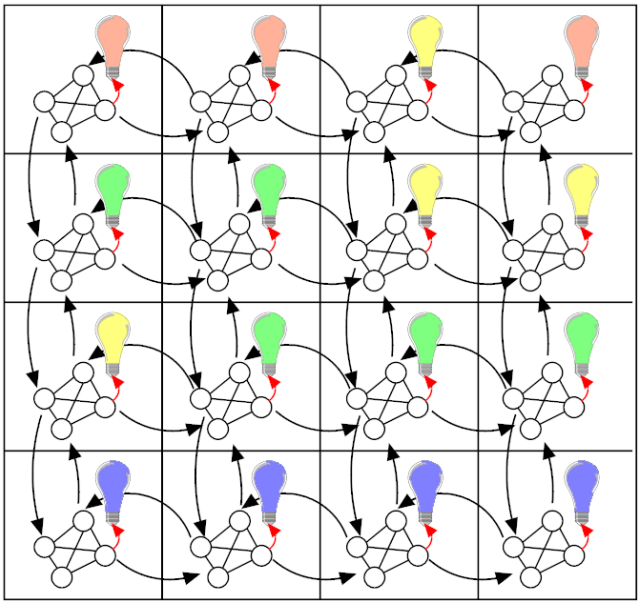

Figure 5.1: Interconnections of ANNs in neighboring cells

The complexity of an object is defined by Kolmogorov as the length of the shortest binary computer program that describes it [Kol63, CT06]. Unfortunately, Kolmogorov's complexity is not computable, thus in the image processing domain it is frequently approximated by how well it can be compressed by various algorithms [YW13]. Another approach used is Shannon's information theory, which measures the amount of information present in a set of symbols [Sha48, CT06]. However, it is often calculated without considering spatial relations between these symbols. To overcome this issue we will use the following equation to calculate the entropy of an image:

$$H = -\sum_{i=1}^{n} p(x_i) \log_2 p(x_i)$$

where x_i represents a possible cell configuration considering its own and its neighborhood's color. $p(x_i)$ is the probability of a particular configuration to be present in the image. Cells on the borders are not considered in this calculation in order to reduce bias.

5.3 Evolutionary Setup

In order to evolve the weights of the artificial neural network, we used the FREVO framework (see Chapter 4). The modular concept of FREVO allowed

Table 5.1: Parameters used for the evolutionary algorithm

Population size	100
Elite selection	11%
Mutated networks	59%
Recombination	30%
Mutation rate	3%
Grid Size	10x10

to reuse the model for the artificial neural network and the optimization algorithm from previous projects, thus reducing our task to formulate and implement the problem.

For our experiments we choose the spatially structured evolutionary algorithm (*CEA2D*) as the optimizer. This algorithm works as follows:

```
 1: create n randomly initialized candidate networks in a population
 2: place all candidates on a regular 2D grid
 3: for generations
 4:     for x=0 to WIDTH
 5:       for y=0 to HEIGHT
 6:         neighbor_list = Collect_neighbors(position(x,y))
 7:         selected_inds = Perform_selection(neighbor_list)
 8:         temp_pop = Apply_reproduction_operators(selected_inds)
 9:       end for
10:     end for
11:     Replace_population(temp_pop)
12:     Evaluate_Population()
13: end for
```

Algorithm 1: Cellular EA used as optimization method

The selection criteria are based on the rank (according to the candidate's fitness) and, in case of the random selection, also on diversity. Diversity means that candidates which are more different to the already selected pool of candidates have a higher chance to be chosen. Functions for mutation, recombination and the difference between candidates are provided by the specific candidate implementation. In our case, the ANNs applied mutation and recombination on the weights and biases of the artificial neurons. The difference between the candidates is the sum of the squared differences of all weights and biases. The full set of parameters used for the evolutionary algorithm is listed in Table 5.1.

5.3.1 Fitness function

A problem consists typically of a simulation with a generic interface to the control system. The simulation returns a fitness value which is used by the optimization algorithm for evolving the system. We implemented a CA simulator that is controlled by a representation. The first version of our fitness function was implemented as a sum of the squared color index differences between the image after a number of iterations and the reference image. This formula can evaluate an intermediate solution well, however as explained previously in Chapter 3.3.5, a fitness function that includes some expert knowledge is in general more preferred at the initial stages of evolution. Therefore, the summands have been weighted by a function giving higher values for pixels having pixels of different color in their neighborhood (based on the reference image). With this attempt we reward cells that lie on the edges of possible shapes within the image. Formally, the fitness is calculated as follows:

$$F(E) = -\sum_{x}^{W}\sum_{y}^{H}(T_{xy} - E_{xy})^2 \hat{w}_{xy} \tag{5.1}$$

where T_{xy} is the color of a pixel at position (x, y) of the reference image, E is the image being evaluated with W width and H height. $\hat{w_{xy}}$ is the normalized weight which is calculated with the following equations:

$$w_{xy} = \frac{\sum_{p}^{P}(T_p - E_p)^2}{(N_c - 1)^2} \tag{5.2}$$

$$\hat{w}_{xy} = \frac{w_{xy}}{\sum w} \tag{5.3}$$

where P refers to the pixels in the Von Neumann neighborhood and N_c is the number of colors present in the reference image. However, this fitness value does not allow a direct comparison between reference images of different dimensions and color depth. A possible solution would be to normalize the fitness values, which requires to define the worst-case solution to divide with. Theoretically, this would mean a maximum color index distance at each pixel. For example, in case of two colors this would mean the negative of the reference image. Since we calculate squared errors, this would lead to an apparent very high fitness even for images that are completely filled with the same color with a low index.

Therefore, we propose a method that calculates the possible average squared error as a factor for the worst-case scenario for N_c different colors. We can approximate it with the following equation:

$$e = \frac{\sum_{x=0}^{N_c}(x - C_m)^2}{N_c} \tag{5.4}$$

where C_m is the index of the color that lies in the middle of the color spectrum. For images with even number of colors, the average of the two middle values is taken into account. Such a way the final normalized fitness is calculated as

$$F_n(E) = 1 - \frac{-F(E)}{e(C-1)^2 \cdot W \cdot H} \tag{5.5}$$

Another important factor that we have to consider is the length of the simulation, i.e., the number of discrete steps that we calculate. Certainly, with longer simulations the chances that the desired structure emerges will probably increase, it also greatly increases the overall simulation time. However, too low number of steps might not be enough for the cells to communicate, infer on their own position and decide on the proper color. The optimal solution would be to find the minimum number of steps needed for a given reference image and run the simulation only for that long. Unfortunately, it is very difficult (if not impossible) to analytically derive this number. One possible solution would be to run a trial-and-error process to find a suitable step number for every image. The issue with this is that by setting a fixed minimum number the system has to exhibit the desired pattern at a fixed time, which is probably a more difficult (and somewhat different) task.

Instead, we decided to change the fitness function in a way that the fitness score is calculated in each time step, and at the end of the simulation we return the maximum of these fitness values. This way we do not limit evolution to a particular solution, but we allow it to explore and find an appropriate number of steps.

5.4 Experiments and Results

In order to be able to sketch the performance of the evolutionary approach as a function of the image complexity, we tested various reference images ranging from very simple regular structures to more complex ones. Besides low complexity images like a single dot or a regular checkerboard we found flags to be particularly interesting for this problem. They are not only very simple structures, but their complexity is in line with the French flag problem mentioned earlier. Initially we tried to replicate the Austrian flag that has three equal horizontal bands of red, white, and red. As next step we attempted the Hungarian flag, which features three equal horizontal bands of red (top), white, and

Image	Name	Dimension	Number of Colors	Complexity
	Single Point	10 x 10	2	0.35
	Checker board	10 x 10	2	1.0
	Austrian flag	10 x 10	1	2.5
	Hungarian flag	10 x 10	3	2.75
	Maple symbol	15 x 16	2	3.33
	Mona Lisa	20 x 29	8	7.8

Table 5.2: Reference images used for the experiments.

green. These images are scaled to 10×10 pixels with the middle band enlarged by 1 pixel.

Since our preliminary results were promising we tried to evolve even more complex structures such as a downsized version of the maple symbol from the Canadian flag. Theoretically, this is a very difficult task for a system that considers only von Neumann neighborhoods, due to the fine details. Similarly, the famous Mona Lisa painting in downscaled resolution has been tested to get an idea how this approach would perform if applied to real images. For a complete list of reference images and their complexity see Table 5.2.

5.4.1 Simple structures

We managed to get perfect solutions for the simple reference images: the single dot, the checkerboard, the Austrian and Hungarian flag in most experimental runs. Figure 5.3 depicts the evolution of a Hungarian flag over the generations. Note, that the proceedings in the quality were highly non-linear over the number of generations. For example, the evolutionary algorithm got sometimes stuck in local cost minima after about 100 generations. Thus, improvements past these generations happen only very infrequently.

The mechanism to recreate an image over several iterations of the CA can be observed by the example of an Austrian flag. The Austrian flag contains only red and white color and was therefore easier to evolve than the three-colored Hungarian one. We achieved a perfect image after running the evolutionary

Figure 5.2: CA steps for Austrian flag; best solution after 200 generations

algorithm for approximately 200 generations. Figure 5.2 shows how the result unfolds over several CA iterations into the intended image.

5.4.2 Complex structures

The limits of the approach can be observed when going to more complex images. Figure 5.6 depicts the results of trying to reproduce a small image of the Mona Lisa painting (left image). The middle image shows the downsized reference image. The best achievable result after over 1000 generations is depicted in the right image. The overall background color scheme is present, although, unfortunately, Mona is missing. The main reason for this result lies in the increased size of the image – while the flags were evolved on a raster of 10x10, the Mona Lisa image is 20x29. Note that initially only cells at the corner and the borders can detect their position in the image - the inner cells must rely on propagated information. For a larger image, the ratio between border and inner cells is more extreme. In order to check if more steps could improve the results, we ran the experiments with a maximum of 50 steps, however this did not yield any better solutions. This probably indicates that evolution gets stuck to a local optima much earlier.

The low performance on more complex images can also possibly be explained by the limitations of the artificial neural network. In theory, more hidden neurons could yield better performance, but our attempts with 6, 8, or 10 hidden neurons did not show any improvement. The cause of this is the increased search space the evolutionary process has to cope with. We will investigate this problem in the following section.

Figure 5.4 shows a boxplot of the fitness obtained throughout multiple runs. As it can be seen, with higher complexity the average fitness decreases, which is expected. We also plotted the number of steps that yielded the best fitness on Figure 5.5. Here, we cannot observe any direct relation between complexity and required steps.

5.4.3 Natural patterns

Another question of interest was how well natural patterns can be evolved. Several patterns resembling natural ones can be reproduced by CA executing

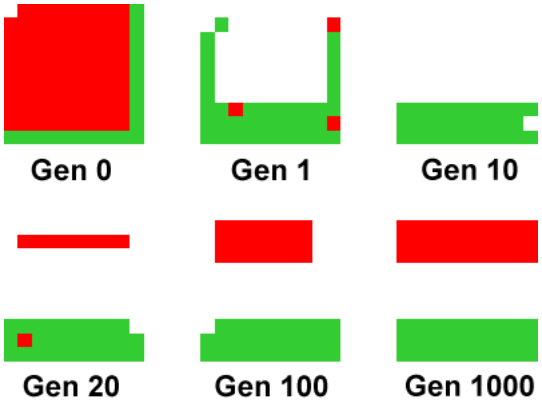

Figure 5.3: Evolution of recreating a Hungarian flag. Best solutions over generations.

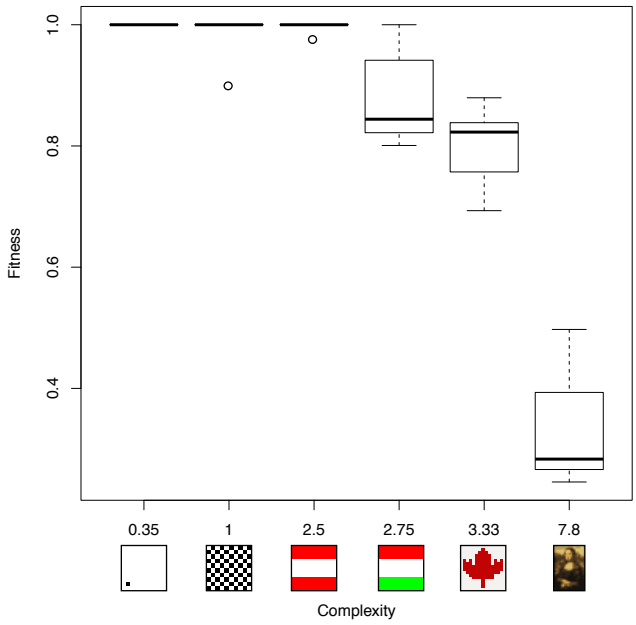

Figure 5.4: Fitness versus complexity. Higher complexity results in typically lower fitness values.

simple state-transition rules of positive and negative feedback [BY99]. We tested two images of 15×15 pixels based on the skin patterns of a cow and a zebra. The respective complexity values of the images are 3.1 and 3.0.

Interestingly, our evolutionary algorithm did not come up with a feasible solution. This is likely because the fitness function was inappropriate for that task, since it compared the potential solution pixel by pixel to a reference image. Thus, a *similar pattern* is not considered as solution, although a human observer might perceive a similar pattern as being closer to the reference than

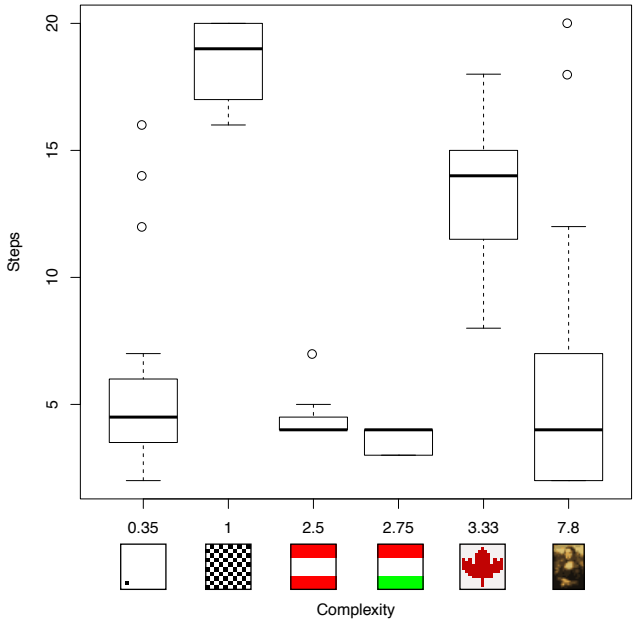

Figure 5.5: Number of steps required for the best solution. Complexity of the structure plays no apparent role.

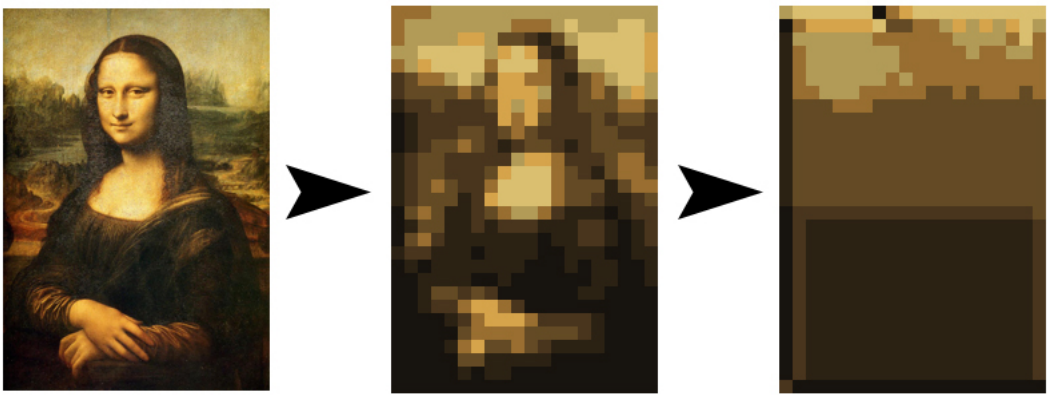

Figure 5.6: Attempt to reproduce the work of Leonardo da Vinci

an image that partially reproduces the original layout of objects in the image. Figure 5.7 shows that although the created image resembles the reference one on a pixel-by-pixel basis, the quality and type of the reference pattern is not matched.

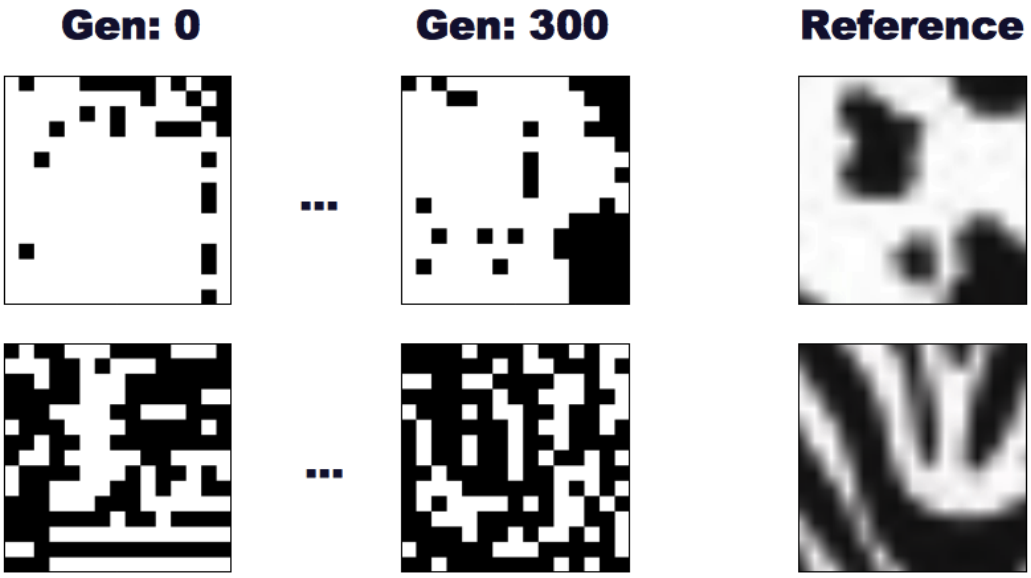

Figure 5.7: Evolving a reproduction of animal skin patterns

5.5 Optimizing the Network Structure

An interesting question is how the complexity of the agents' controllers plays a role in the evolvability of this problem. In theory, an ANN with more hidden neurons can perform better by being able to approximate more complex functions. However, more neurons mean more connection weights (and biases) that radically enlarge the search space. Therefore, it is important to find the optimal number of nodes that offers the highest fitness with a search space that is still manageable by the search algorithm. Unfortunately, there is no way to determine the optimal number of hidden neurons analytically. The optimal number depends on the complexity of the function to be approximated, and, therefore, indirectly on the number of input and output nodes. Besides a trial and error approach, there are some empirically-derived rules-of-thumb, of these, the most commonly relied on is 'the optimal size of the hidden layer is usually between the size of the input and size of the output layers' [Blu92]. Swingler [Swi96] and Berry [BL04] propose a maximum of two times the number of input nodes for the hidden nodes. Boger and Guterman [BG97] suggest that the number of hidden nodes should be 70%-90% of the number of input nodes. Caudill and Butler [CB92] recommend that the number of hidden nodes should be two third of the sum of input and output nodes.

In order to obtain some knowledge on the effect of different fixed network sizes we re-ran our experiments 30 times with different seeds. We tested ANN controllers with various numbers of hidden neurons ranging from 0 to 10. We

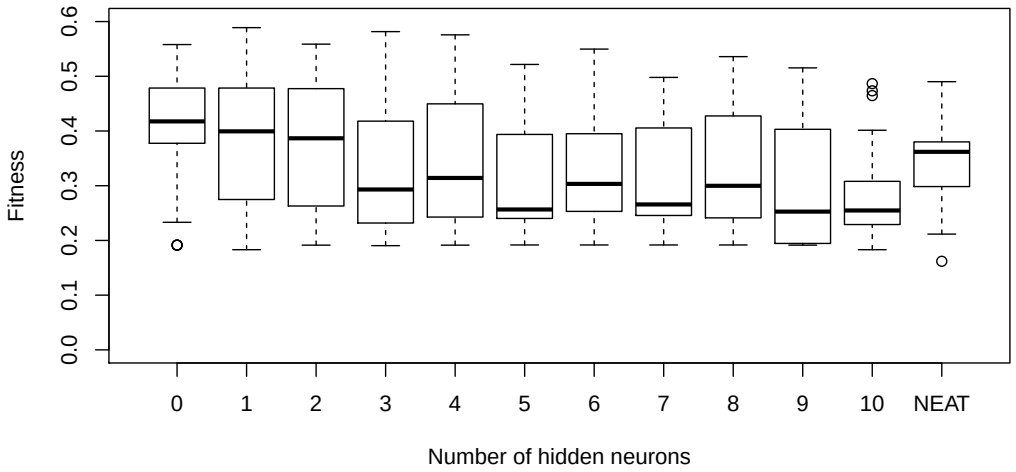

Figure 5.8: Comparison of best evolved solutions for the Mona Lisa problem with different number of hidden neurons. Results indicate a decrease of performance with higher number of hidden neurons. The NEAT algorithm also cannot outperform fixed neural networks with very low number of hidden neurons.

used the Mona Lisa problem, as it turned out to be the most difficult one for the evolutionary approach. Every other parameter has been retained from the previous experiments.

As depicted on Figure 5.8 the results indicate decreasing performance when using more hidden neurons. Thus, the best performance is achieved when using no hidden neurons at all. A possible explanations for this phenomena is that the problem is already too difficult, even for more complex neural networks, so that the simplest configuration can be evolved best. Note, that cells have to propagate information and remember on the passed information as well. It also shows that having more neurons does not necessarily helps the process.

Another interesting approach to find the optimal neural network structure is to use a complexifying algorithm such as NEAT [SM02]. This algorithm starts from the simplest structure having no hidden layer and its structure gradually evolves along with the respective weights and biases. NEAT uses speciation and fitness sharing to protect candidates with novel structures to be selected out early. FREVO implements NEAT as a candidate representation component, therefore making a direct comparison to the previous results possible. We ran the same experiments and found that NEAT also cannot achieve better performance than the fixed-sized networks with low number of hidden neurons

(See last column on Figure 5.8). On average, neural networks evolved by NEAT have 3.5 hidden neurons which is similar to the performance of the fixed neural networks with similar structures. This result also supports the idea that the task is too difficult to be solved with the current approach, even with higher number of gradually introduced neurons and synapses.

5.6 Summary

We depicted and evaluated the evolutionary design process generating a multicellular system out of a genotypic description for a single cell. The mechanisms have been realized via the open source FREVO framework for evolutionary design (see previous chapter). At the beginning of each simulation, all cells had the same state and commenced their operation at the same time - this is comparable with a number of people cooperatively drawing an image in the dark. This differentiates our problem from the ones in the literature, where usually a zygote cell is given, from where the other cells grow. Still, the evolutionary process evolved a solution where also some eminent cell (typically a particular corner cell) serves as a zygote.

The main contribution of this work is not presenting an algorithm for "drawing images in the dark" but rather presenting a proof-of-concept on integrating ANNs into a CA in order to initiate a morphogenetic process. Furthermore, it servers as a good case study for evolving self-organizing systems due to its simplicity and easily understandable results. We also introduced a complexity metric based on spatial entropy that can be used to investigate the evolvability of reference images with increasingly complex patterns.

The best results have been achieved when evolving simple structures with large areas of a single color as they are present for example in flags. For more complex images, the current setup causes the evolutionary algorithm to get stuck at a suboptimal stage. Our findings also show that this task can be very difficult even for more complex neural networks. Our attempts using NEAT - an algorithm that optimizes the neural network structure - performed similarly as fixed neural networks with similar number of hidden neurons.

CHAPTER

6 Study on the Interaction Interface Design

Behavior is the mirror in which everyone shows their image.

– Johann Wolfgang von Goethe

In the field of evolutionary swarm robotics, there has been a lot of attention on designing the fitness function or the genotype-to-phenotype mapping (i.e., evolvable decision unit). However, the configuration of the agent regarding its sensory interfaces is mostly left to the naïveté of the experimenter. Yet, ill-defined configuration—in terms of the selected subset of the sensory-motor system, or in the pre-processing of the raw sensor data—may be decisive in determining the failure of the evolutionary process.

In this chapter, we turn towards swarm robotics in order to study the effect of different robot configurations on the ability to evolve efficient behaviors for a self-organized robotics system. In particular, we would like to emphasize the importance of the choice of a good configuration being fundamental as even small details can lead to large differences in the group behavior. To demonstrate and analyze this effect, we test different alternatives and measure the group performance on a bi-objective scale. We find that different configurations not only have a strong effect on performance, but they also correspond to behaviors with radically different features concerning the organization of the group. Major content of this chapter has been published and presented at the IEEE Congress on Evolutionary Computation 2013 [FTE13].

6.1 Introduction

Evolutionary Robotics (ER) can be a powerful method for the automatic synthesis of robotic systems, as demonstrated in the research carried out in the last two decades [NF00, FK10b, PB07, HDW+13]. However, despite the potential

advantages of such an automatic methodology, the success/failure of an ER experiment is often left to the expertise/naïveté of the experimenter. Indeed, it is often the case that many different choices of the experimental setup are arbitrarily performed, without relying on a well-assessed methodology. The robustness and flexibility of the evolutionary method sometimes counterbalances ill-conceived setups, but this cannot be *a priori* guaranteed. The situation is possibly worsened for collective and swarm robotics, due to the greater variability and dynamism characterizing these systems. Trianni and Nolfi [TN11] recognized the need for an engineering methodology in ER, pointing to four different aspects that must be taken into account when designing an experiment: the sensory-motor system of the robots, the genotype-to-phenotype mapping, the fitness function and the ecology. They analyzed the challenges posed by the application of ER methods to swarm robotics and showed that a small difference in the communication protocol exploited by the robots had a huge impact on the scalability of the evolved solutions [TN11].

In Chapter 3, we described an extension of this methodology to self-organized technical systems pointing out the interface design as an issue that is often disregarded when building self-organizing systems. In ER, the term *robot configuration* is generally used to refer to the choice of the sensory-motor system of the robots which includes the set of available sensors and actuators that are used during the experiment, along with the respective pre- and post-processing mechanisms. In the collective/swarm robotics case, the robot configuration also includes the communication devices and protocols to allow the exchange of information between robots. In short, the robot configuration corresponds to everything that is at the *interface* between the control system of a robot and the robot's environment.

Even though there are attempts to co-evolve the brain and body of a single robot [LP00], strictly speaking its *configuration* is typically not modified by an evolutionary process. Instead, it is usually chosen through intuition or experience, often relying on the smallest set of sensors and actuators, minimizing pre- and post-processing in order to limit the intervention of the experimenter and then letting the evolutionary machinery optimize the system within the given constraints. We believe that this approach is ill-posed for evolving complex robotic systems, such as self-organized robotics ones. Indeed, these systems are very sensitive even to minor changes in their configuration, which might easily lead to unpredictable or unwanted system behavior [TN11].

Therefore, even if the selection of the robot configuration may seem a trivial issue, a proper choice can hardly be done without *a priori* information on the effects it has on the evolutionary dynamics. In this chapter, we corroborate this claim with a well-grounded experimental setup, and we point to the need of methodological tools that support the choice of the best robot configuration.

To demonstrate our claims, we have chosen to evolve a coordinated motion behavior ("flocking") for a swarm of autonomous wheeled robots. This is a common benchmark for swarm robotics, and many different studies have dealt with it to demonstrate its feasibility in different contexts [HDT02, TcGc08, HLV+11]. This is mainly due to the large body of knowledge available on both the biological systems displaying coordinated motion [HW92b, CKJ+02, CCG+10] and the underlying self-organizing process, which has been extensively discussed and understood [Rey87, JLM03, VZ12]. On this basis, we can design our evolutionary experiment to provide the robots with the relevant information needed to display flocking behavior.

Generally speaking, flocking can be obtained by three basic individual rules: collision avoidance, flock centering, and velocity matching. Collision avoidance ensures that the individuals in the group do not get too close to each other, while flock centering prevents them to get too far as well. Velocity matching requires the individuals to steer towards the average heading of their neighbors. The first two rules serve to achieve *cohesion* in the group, while the third one leads to the alignment of the individuals in a coherent direction, which is necessary for group *motion*. Collision avoidance and flock centering can be easily executed relying on the *distance* and the *bearing* of close neighbors. Instead, velocity matching usually requires the knowledge of their *relative heading*. Though it is a rather complex task, some studies attempted to extract this information from simple sensors [HDT02, MSC10]. In order to evolve flocking in a group of robots, we exploit group cohesion and motion as the objectives to be maximized by our evolutionary algorithm. We test different robot configurations, which are designed to provide distance, bearing and relative heading information. Finally, we compare the features of the evolved behaviors for their capability to display efficient coordinated motion in the group.

In this study, we evolve robotic controllers exploiting a multi-objective evolutionary algorithm [BNE07]. Multi-objective evolution leads to a wide exploration of the search space at the cost of a more complex analysis of the solutions that constitute the obtained Pareto front. In contrast to optimizing for a single aggregated measure, a multi-objective approach explores all the possible trade-offs between possibly conflicting objectives such as group cohesion and motion, and therefore the evolutionary process can produce a more diverse set of candidate solutions. By investigating the evolved behavior at different points on the Pareto front, we can explore the effect of a given robot configuration at large, without constraints from an averaging function.

The chapter is structured as follows: Section 6.2 presents a step-by-step guide on the setup of our evolutionary experiments and the choice of the robot configurations to test. A first comparison between the different configurations based on performance only is presented and discussed in Section 6.3, and a

deeper analysis of the evolved behavior belonging to the Pareto fronts is provided in Section 6.4. Section 6.5 summarizes the results and gives some final remarks.

6.2 Evolving Flocking Behavior

In this section, we describe the design choices made for the evolution of coordinated motion behaviors in a group of robots. Evolutionary experiments are performed in simulation using ARGoS, a simulator tailored for swarm robotics, which provides high-speed and accurate simulations [PTO+12]. The simulated robots model the marXbot platform [BLM+10]. These robots are equipped with a belt of evenly distributed RGB LEDs that allow signaling with different colors, and that can be perceived by the onboard omni-directional camera. We exploit the marXbot LEDs to define several robot configurations that can provide the required information for coordinated motion, as described in Section 6.2.1. We complete the description about the evolutionary setup presenting the full sensory-motor configuration of the robots (Section 6.2.2), the controller and the genotype-to-phenotype mapping (Section 6.2.3), and the evolutionary algorithm along with the fitness function used (Section 6.2.4).

6.2.1 LED configuration

According to the models of self-organized flocking [Rey87], the relevant information needed by an individual to decide its action consists of the distance, the bearing and the heading of close neighbors. To convey such information we use the RGB LEDs around the robots, which allow displaying various colored patterns. In our experiments, the chosen pattern will be always displayed to allow neighboring robots to detect each other, and possibly to obtain all the information required for moving in a coordinated way. To comply with our goals, the configuration of these light sources is kept static during every evolutionary run.

To investigate the influence of a robot configuration on the evolved behavior, we test different LED configurations. In total there are 12 LEDs all around the robot circular body, each configurable to display any color in an 8-bit RGB spectrum (see the figure in Table 6.2.1). We decided to restrict the number of colors to red and blue, because each color corresponds to additional inputs to the robot controllers, as detailed below. In the whole, for each of the 12 LEDs, we have 3 possible states—red (`R`), blue (`B`) and off (`0`)—that correspond to 3^{12} possible configurations. The situation of the LEDs around the robot's turret can be seen on Figure 6.1.

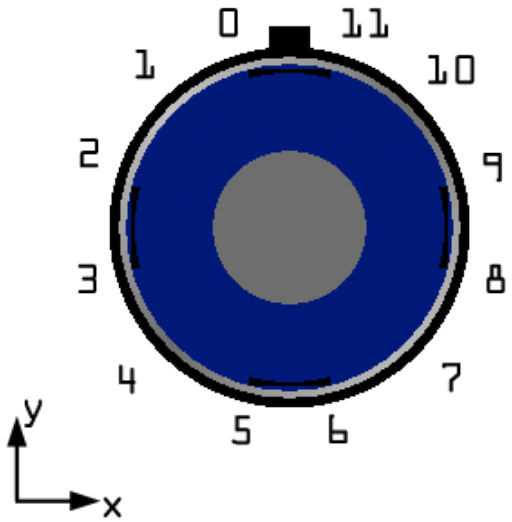

Figure 6.1: LED arrangement on the robot's body,as seen from top

We selected a subset of configurations that intuitively convey information about the robot heading, which is crucial for effective coordinated motion. We defined both left-right and front-rear colored patterns, and we vary the number of LEDs used. Additionally, we also run two control experiments with "naive" LED configurations: the one with all LEDs off, and the other with all LEDs in the same state (`B`), therefore not conveying any heading information. As shown in Table 6.2.1, in total we have 10 different configurations that can be grouped in three categories: the naive (1-2), the left-right (3-6) and the front-rear (7-10). The first configuration is only meant for testing if adding LEDs truly have an effect on performance, while the second one should determine if adding heading information is beneficial. The other two categories include setups with different number of LEDs turned on (respectively 1, 2, 4 and 6 LEDs). For the sake of simplicity, we considered only symmetric configurations.

6.2.2 Sensory-motor System

Each robot is equipped with a minimal set of sensors and actuators, which is considered sufficient for displaying a flocking behavior. The selected LEDs in the robot configuration are always on, and can be perceived by the omnidirectional camera up to the distance of 1 meter. The image is processed to extract a list of $i = 1, \ldots, N$ red/blue color blobs $c \in \{r, b\}$ with their distance $\rho_{c,i}$ and angle $\theta_{c,i}$, resulting in a vector $\mathbf{v}_{c,i}(\rho_{c,i}, \theta_{c,i})$ for each detected blob (in polar coordinates). Additionally, each robot is equipped with 24 infra-red proximity sensors evenly distributed along the circumference of the robot's body. The typical range of these sensors is around 4-5 cm. Each sensor i provides a scalar

No.	Configuration	Description
1	000000000000	All LEDs are turned off
2	BBBBBBBBBBBB	All LEDs are turned blue
3	00B00000R000	1-6 LEDs indicating left and right side
4	00BB0000RR00	
5	0BBBB00RRRR0	
6	BBBBBBRRRRRR	
7	R00000B00000	1-6 LEDs indicating front and rear side
8	R0000BB0000R	
9	RR00BBBB00RR	
10	RRRBBBBBBRRR	

Table 6.1: We tested 10 different LED configurations, each defined by a character array in which the i^{th} element refers to the state of LED i, positions are as indicated on Figure 6.1

value P_i inversely proportional to the distance of the object. For each sensor, we build a 2-dimensional vector $\mathbf{v}_{p,i}(\rho_i, \theta_i)$ in polar coordinates, where $\rho_i = P_i$ and θ_i corresponds to the sensor bearing. Measurement uncertainty is modeled by uniform noise within 5% of the input range.

Directly feeding the values obtained by the sensors would be straightforward, but would also present a huge search space for our evolutionary approach (24 proximity values + N colored blob values). Thus, some form of input preprocessing is necessary. To this purpose, we compute a single resultant vector for red blobs, for blue blobs and for the proximity sensors:

$$\mathbf{V}_k = \sum_i \mathbf{v}_{k,i}, \qquad k = r, b, p. \tag{6.1}$$

Then, we rescale the vector length to be within the range $[0, 1]$ by exploiting a sigmoid normalization:

$$\hat{\mathbf{V}}_k = \frac{\mathbf{V}_k}{|\mathbf{V}_k|} \frac{2}{1 + e^{-\beta|\mathbf{V}_k|}} - 1, \tag{6.2}$$

where $\beta = 2$ is a normalization parameter. Finally, we consider the projection along $M = 6$ equally distributed axes, by computing the scalar product:

$$I_{k,m} = \hat{\mathbf{V}}_k \cdot \mathbf{V}_m, \qquad m = 1, \ldots, M, \tag{6.3}$$

where $\mathbf{V}_m$ is the versor in the direction $(2m - 1)\pi/M$.

In this way, we reduce the total number of scalar inputs to a more manageable size without significant loss of information. These values are then going

to be fed to the robot's control software.[1] On the actuators side, given that LEDs are always kept in their state, the controller only commands the motors of the left and right wheels, which can linearly vary in the range $[-\omega_M, \omega_M]$, where ω_M is the maximum angular speed of the wheels ($\omega_M \approx 4.5\text{s}^{-1}$).

6.2.3 Genotype-to-phenotype mapping

All robots are completely identical both in body and control software (we use a homogeneous group). Therefore, we map the genotype to one single control structure that is cloned and instantiated separately for each robot. We employ a fully-connected feed-forward neural network without hidden layers. The neural network has 18 sensory inputs and 2 motor outputs. At each step the sensory neurons act as simple relays, while the output of the motor neurons is calculated as follows:

$$O_j = \sigma\left(\sum_i w_{ij} I_i + \beta_j\right), \quad \sigma(z) = \frac{1}{1+e^{-z}} \tag{6.4}$$

where I_i is the activation of the i^{th} input unit, β_j is the bias term, O_j is the activation of the j^{th} output unit, w_{ij} is the weight of the connection between the input neuron i and the output neuron j, and $\sigma(z)$ is the sigmoid function. The first 6 input neurons receive the input from proximity sensors: $I_m = I_{p,m}$. Neurons in the range $[7, 12]$ and $[13, 18]$ receive the data corresponding to the red and blue vectors: $I_{m+6} = I_{r,m}$ and $I_{m+12} = I_{b,m}$. Finally, the output of the two motor neurons is scaled onto the range $[-\omega_M, +\omega_M]$ and used to control the speed of the wheels. The bias terms and the connection weights of the network are genetically encoded parameters. Therefore, we have a direct encoding, meaning there exists a bijective function that relates the genotype to the phenotype.

6.2.4 Evolutionary algorithm and fitness function

In our experiments, we used a simple multi-objective evolutionary algorithm that operates on a population of 100 randomly generated genotypes. Each genotype contains the parameters of the control software of each robot as a vector of floating-point genes varying in the range $[-5, 5]$. After the evaluation of the performance of each individual of the population, the new population is created using a combination of elitism and mutation. In particular, the population is ranked according to the hypervolume metric [BNE07] and the best

[1] Note that the heading information encoded in the LEDs color pattern (if any) is implicit. The available information is only given by the color vectors, and no assumption is made on how the neural controller should make use of it.

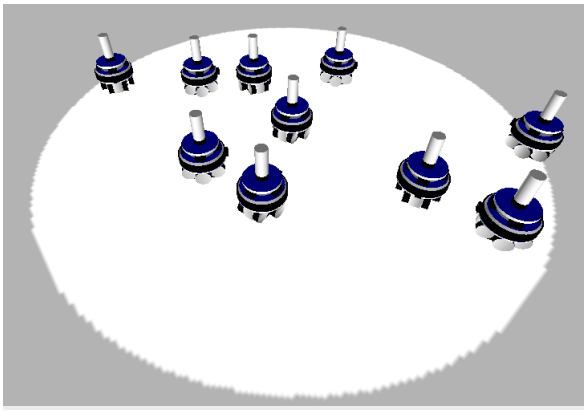

Figure 6.2: A possible initial setup of the 10 robots.

25 individuals are selected for reproduction: all individuals are retained unchanged in the next generation, while the rest of the population is generated by applying a mutation operator to copies of the elite individuals. Mutation is applied by adding to each gene a random value drawn from a normal distribution N(0,1), and trimming the value to keep it within the range $[-5, 5]$. The algorithm runs for a total of 200 generations.

Due to the fact that random initial conditions have an effect on the immediate performance of a candidate, each genotype is evaluated in 10 trials and the average performance over these trials is used to assess its fitness values. Each trial lasts $T = 120$ seconds corresponding to 1200 simulation steps. Initially, all robots are placed randomly within a circle with a diameter of 2 meters. An example of the initial setup can be seen on Figure 6.2.

Robots are rewarded for displaying coordinated motion, that is, they have to move as far as possible from the initial position while maintaining group cohesion. As a consequence, we defined a bi-objective function based on the following criteria: *cohesion* and *motion*. Cohesion is maximized when the average distance between the robots and the geometric center of the group is minimized:

$$C = \max\left(0, 1 - \frac{1}{N}\sum_{i}\frac{|\mathbf{X}_i(T) - \hat{\mathbf{X}}(T)|}{d_m}\right), \tag{6.5}$$

where N is the number of robots, $\mathbf{X}_i$ is the position of robot i, $\hat{\mathbf{X}}$ the one of the group center of mass, and d_m is a normalization factor. Motion is computed as the total distance covered by the geometric center of the group:

$$M = \frac{|\hat{\mathbf{X}}(T) - \hat{\mathbf{X}}(0)|}{D_m(T)} \tag{6.6}$$

where $D_m(T)$ is the maximum distance a single robot can travel in T seconds.

It is important to mention that by measuring the movement of the geometric center of the group at the end of the simulation we can effectively filter out solutions that exhibit random oscillations around the center point. This way cohesive groups with high motion are the best rewarded regardless on the direction the group decides to take.

6.3 Evaluating LED configuration

We performed 20 independent evolutionary runs for each configuration, each starting with different randomly generated populations. At the end of each evolutionary run, a post-evaluation procedure is employed where all candidates in the last generation are evaluated 300 times. To ensure a fair comparison between the different configurations, we use the same set of random seeds to initialize the evolutionary runs and to perform the post-evaluation.

We exploit the Pareto-optimality relations to compare the results obtained with different configurations. In the simplest case, when one experimental condition is always dominated by another one (according to the $\lhd$-relation [ZTL$^+$03]), we can say that the latter gives a better approximation to the Pareto-optimal set. If such clear advantage cannot be determined, we can make use of attainment functions to extract information on the quality of different configurations [dFFH01]. The attainment function is associated to a given experimental condition, and it indicates the probability of a given point attaining (i.e., dominating or being equal) in the objective space. It thus characterizes statistically the output of a given experimental condition. Since this function is practically unknown, we approximate it using the simulation results (in particular, the obtained Pareto fronts of the 20 evolutionary runs we performed for each setup) obtaining the empirical attainment function (EAF) [LIPS10]. Once obtained the EAF for each condition, we can compute the difference between conditions to compare the relative quality of their output (see Figure 6.3 for an example). On this basis, we can comparatively analyze the effects of different robot configurations.

6.3.1 Naïve configurations

The naïve control conditions have been designed to make sure that robots can actually learn to profit from the extra information obtained by the use of LEDs and camera. Indeed, we could not rule out *a priori* the possibility that coordinated motion could be effectively performed exploiting the IR sensors only, or without heading information. Therefore, we initially test the `000000000000` and the `BBBBBBBBBBBB` configurations, which provide similar information about

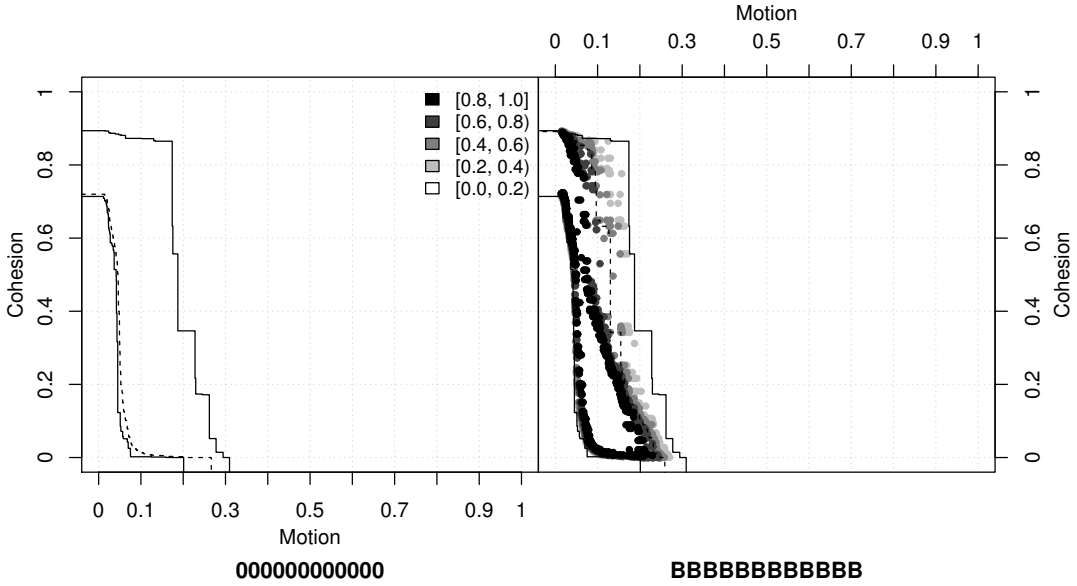

Figure 6.3: Comparing naive configurations through empirical attainment functions (EAF). Left-side and right-side indicate the differential advantage of the 000000000000 and the BBBBBBBBBBBB configurations, respectively. Gray levels indicate the magnitude of the difference between the two setups: the darker the color, the larger the difference. For instance, a black point on the right indicates that the right-side condition attains that point in at least 80% more runs. Solid lines indicate the best and the worst surfaces, while the dashed line indicates the median. In this case, we observe a large advantage for the BBBBBBBBBBBB configuration.

distance and bearing of close neighbors, the latter featuring a longer range thanks to the visual information from the camera. Naturally, one expects that the long-range camera has a beneficial effect on the outcomes of the experiment, but this is not trivial. The reason is that extra sensors require extra input neurons (with the additional connections) that might enlarge the search space to a limit where no learning can take place.

As expected, the comparison of the obtained results for the two conditions shows that robots exploiting long-range visual information significantly perform better (see Figure 6.3), by systematically attaining higher values in both motion and cohesion. After investigating the evolved behaviors, we observe that no true coordinated motion has been achieved, as also suggested by the low motion performance of the attained points in the objective space. This can be partially read from concave shape of the Pareto front of *BBBBBBBBBBBB* and the low motion values. The increased performance with respect to the 000000000000 configuration can be definitely credited to the increased sensing range of the camera although it only helps the group to maintain coherence, which is not

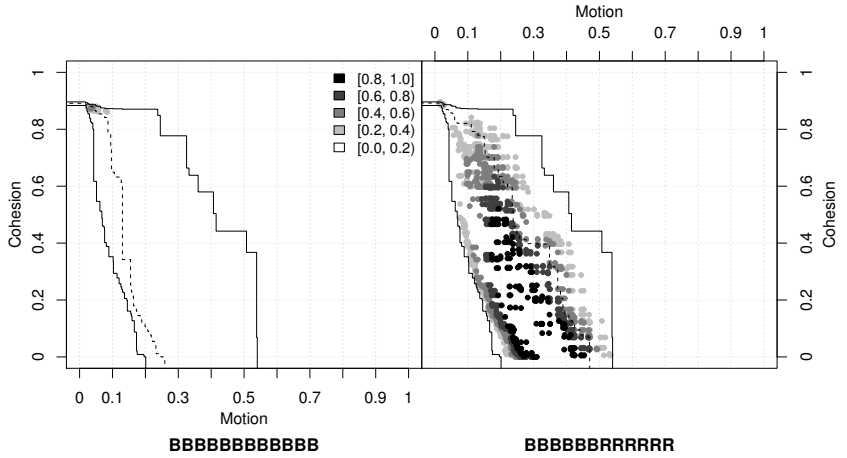

Figure 6.4: Comparison between the uniform configuration (BBBBBBBBBBBB) and a left-right configuration featuring an equivalent number of LEDs on (BBBBBBRRRRRR)

easily feasible on the basis of IR information only. We therefore obtain a confirmation that a suitable LED configuration is necessary for coordinated motion, and also that it must provide some information about the heading of the robots in order to obtain a good performance.

6.3.2 Left-right configurations

Left-right configurations are probably the most natural way of indicating the direction of a moving object, as this type of signaling is quite frequent in technical systems (e.g., visual signs of boats). However, choosing the right configuration for efficient coordinated motion with the marXbot robots remains a question. To find out how many LEDs should be turned on, and in what position, we ran tests with 1, 2, 4 and 6 LEDs, with different color on each side of the robot. Figure 6.5 shows a comparison of the tested configurations.

First, we compare the results of the uniform configuration (BBBBBBBBBBBB) with a left-right configuration exploiting the same number of LEDs (BBBBBBRRRRRR). The heading information provided by the latter can be actually exploited for better coordinated motion, as the difference between the EAFs indicates (see Figure 6.4). If we consider cohesion only, the two setups are equivalent, but the left-right configuration definitely outperforms the uniform one as soon as motion is taken into account.

The comparison between different left-right configurations is performed for decreasing number of LEDs. We find that there is a inverse relationship between the number of LEDs used and the performance (see the second, third and fourth graph in Figure 6.5). Plus, this advantage appears at every part of

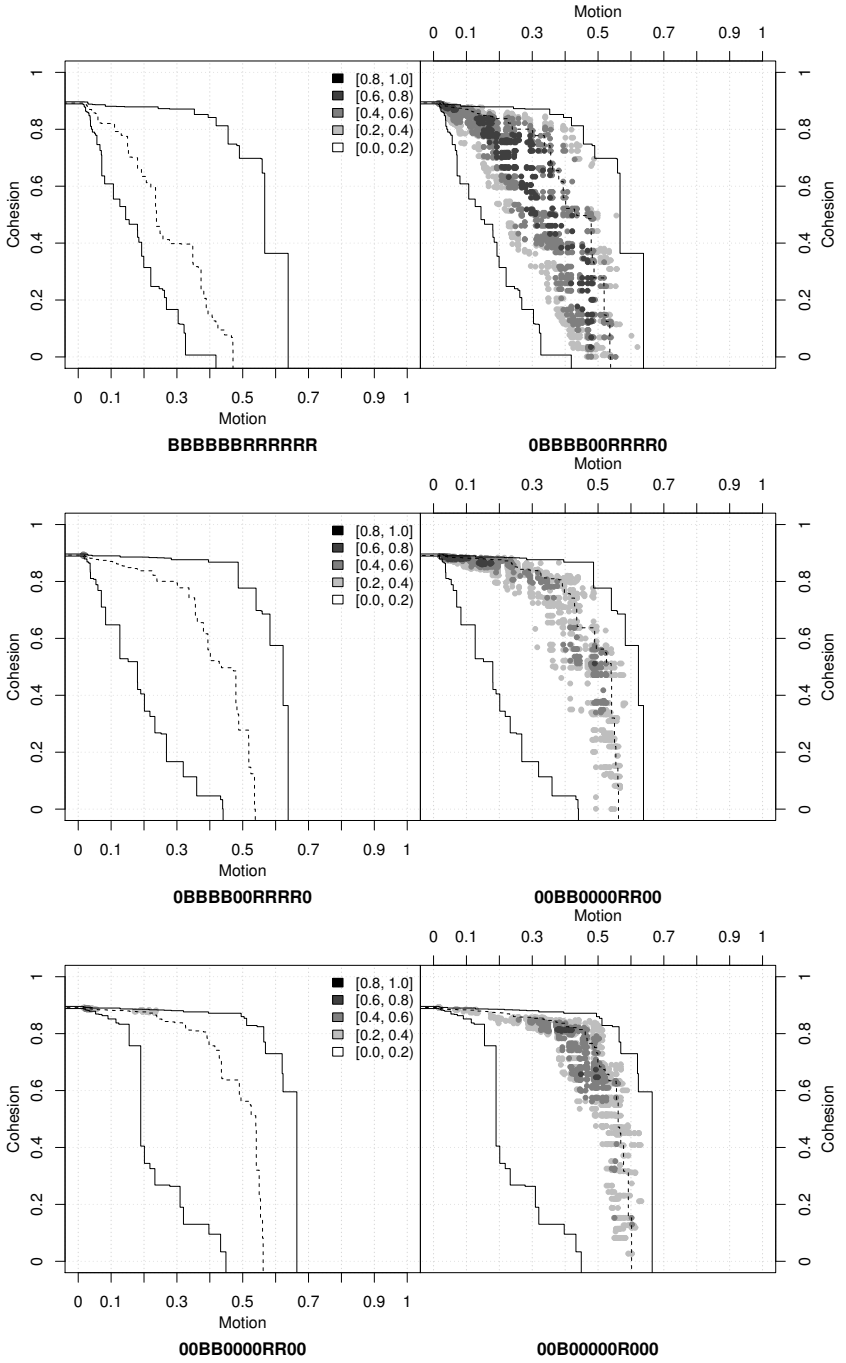

Figure 6.5: Comparison among left-right configurations with decreasing number of LEDs. Setups with less LEDs perform better (see text for details).

the objective space, excluding only a small portion in which solely cohesion is maximized. We also note that the lower the number of LEDs, the more the difference fades. A possible explanation for the advantage of less LEDs lies

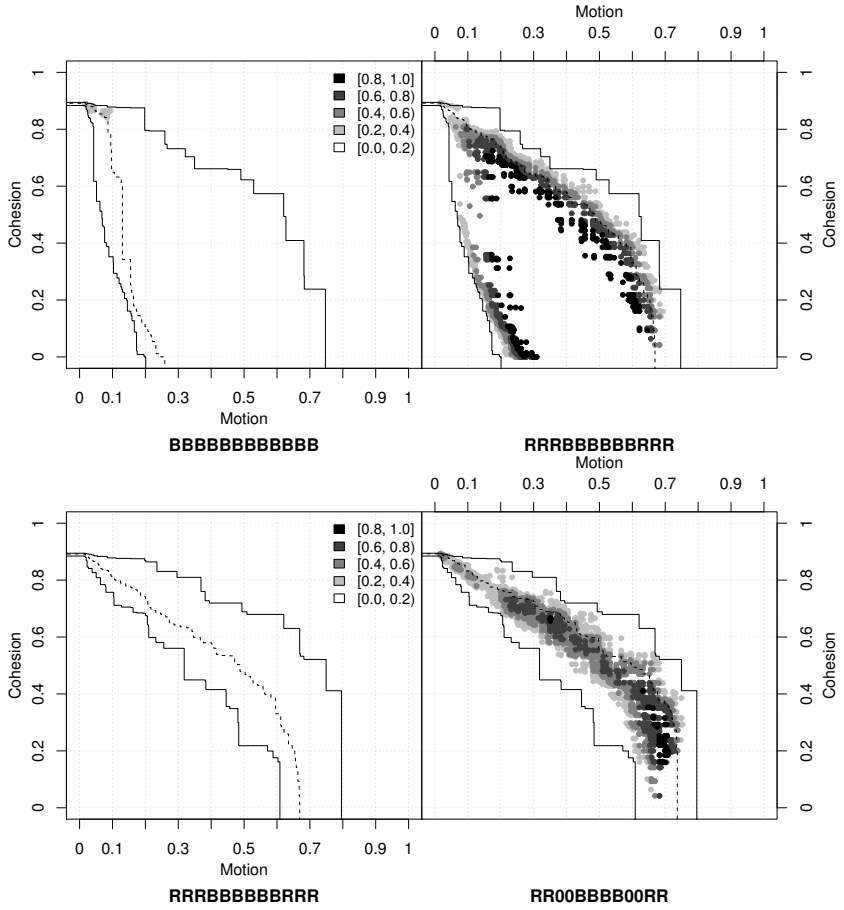

Figure 6.6: Comparison among front-rear configurations (see text for details).

in our encoding of the visual data. The encoding presented in Section 6.2.2 might give less distinguishable information when two detected robots are very close. This is a counter-intuitive result as one would expect that the more information is available through LED signals, the better the quality of the coordinated motion behavior. Instead, minimal configurations seem to provide a selective advantage over the entire objective space. Apart from this, the results demonstrate that there is a strong influence of the configuration on the performance of the system.

6.3.3 Front-rear configurations

It is also possible to deduct the heading information from front-rear markers, as it is done for cars. For example, most road vehicles have white headlights and red taillights to indicate the direction of their movement. Indeed, evolved strategies may exploit front-rear markers to obtain a coherent orientation of the

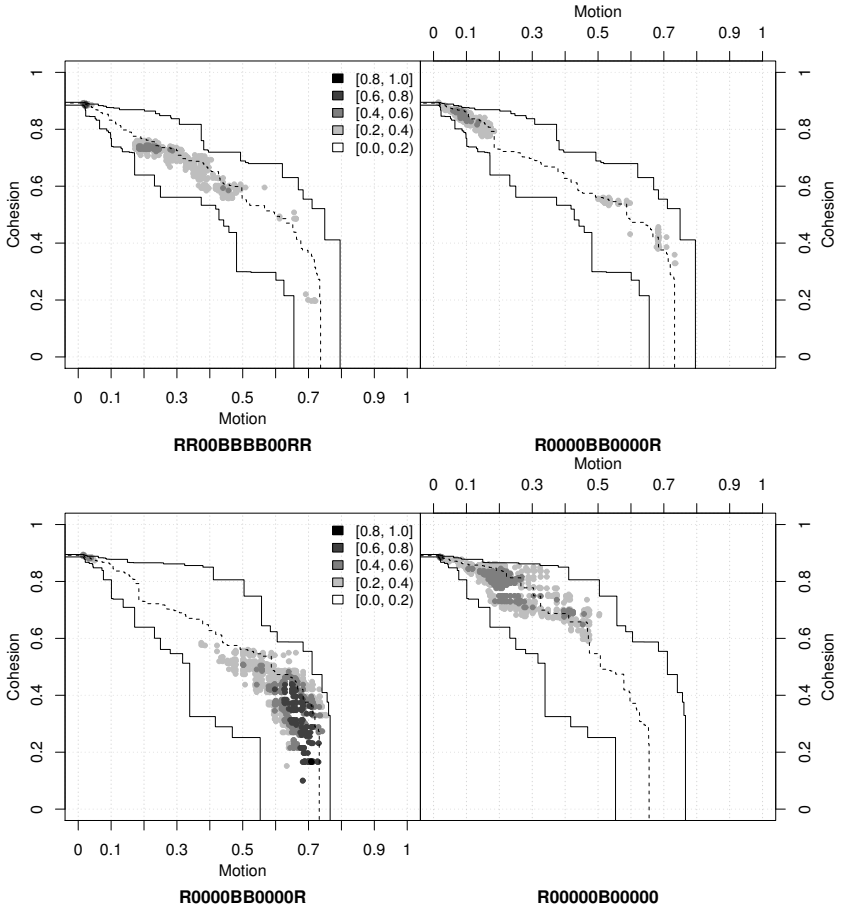

Figure 6.7: Comparison among front-rear configurations with decreasing number of LEDs (see text for details).

group, which is the basics for coordinated motion. Therefore, we performed the same simulations as presented above with different number of LEDs used to signal the front and the rear of a robot. The comparison among the different configurations can be seen in Figure 6.6 and 6.7.

Also in this case, the comparison with the uniform configuration shows that front-rear markers can be actually exploited for coordinated motion. When comparing among different front-rear configurations, we note that it is not possible to draw a strong order of performance among them. While it is clear that the setup with 6 LEDs is strictly dominated by configurations with less LEDs, we note no clear difference between configuration with 4 and 2 LEDs and a differential advantage in separate zones of the objective space between configuration with 2 and 1 LEDs. Therefore, differently from the left-right configurations, front-rear markers appears to be exploited in a similar way despite their number.

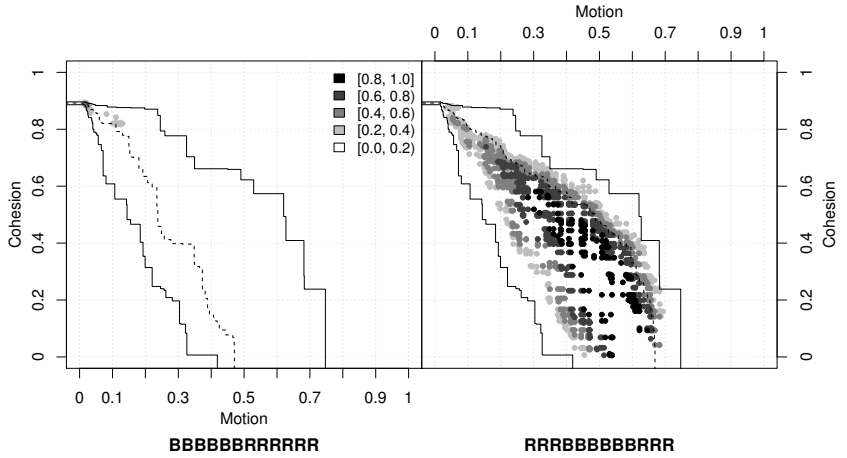

Figure 6.8: Comparison of the left-right and front-rear configurations (see text for details).

6.3.4 Comparison between different configuration categories

We finally propose a comparison between left-right and front-rear configurations, performed among setups featuring the same number of used LEDs (e.g., `00BB0000RR00` vs. `R0000BB0000R`). In the case with 6 LEDs, the front-rear configuration demonstrates better performance in covering the objective space (see Figure 6.8). As we decrease the number of used LEDs, the situation changes in favor of the left-right configuration, which starts to dominate in the top part of the objective space, where more and more solutions featuring high cohesion and good motion are found. As shown by the various graphs in Figure 6.9, the front-rear configurations keep an advantage in the areas of the objective space characterized by high motion but low cohesion. Thus, without further analysis it is impossible to determine the best configuration.

6.4 Classification of the obtained solutions

Despite the usage of relatively simple configurations, we were unable to conclusively determine the best setup. Indeed, the average values of motion and cohesion alone do not indicate what kind of behavior the group is displaying. For this reason, it is necessary to observe the evolved behaviors and identify their characteristics in relationship to the area occupied in the objective space. As discussed, certain configurations yield better performance in different parts of the Pareto front. However, it is very likely that not every non-dominated solution corresponds to flocking. The idea to solve the comparison problem would

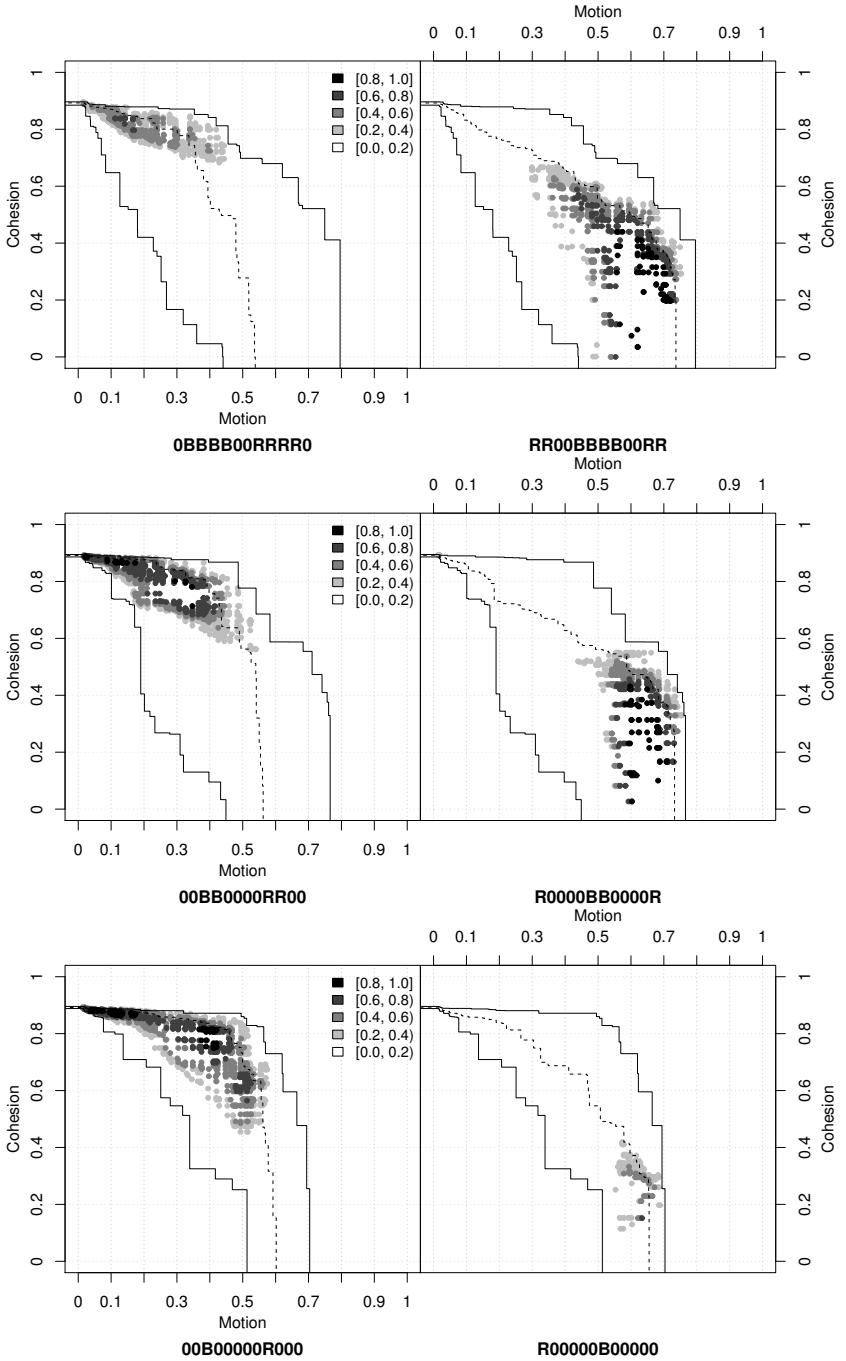

Figure 6.9: Comparison of the left-right and front-rear configurations with decreasing number of LEDs (see text for details).

be to find clusters of basic group behaviors on the motion-cohesion diagram in order to find where most flocking solutions exits. For instance, solutions with high cohesion and low motion, despite belonging to the Pareto front, do not

correspond to coordinated motion. Similarly, high motion and low cohesion correspond to the group splitting in various subgroups. By thorough analysis of the evolved behaviors on the entire Pareto front, we observed the following prototypes of group motion[2]:

Stationary: Robots stay close to each other, showing minimal or no group motion.

Disperse: Robots spread around with almost no cohesion.

Wavefront: Robots move together forming a single-width arc.

Train: Robots align and follow each other one by one like wagons of a train.

Flocking: Robots show a healthy mix of cohesion and motion by displaying true flocking.

To objectively classify each solution into one of the above listed behaviors we defined a set of suitable metrics and a classification procedure. Besides motion and cohesion measures, we compute the following metrics at the end of each trial.

- The number of connected components K in a graph where each node corresponds to a position of a robot and an edge exists if and only if the distance between the robots is less than the maximum visual range. This measure serves to understand if the group splits in sub-groups moving in different directions.
- The angle Θ between the average direction of motion of the group and the main axis of the group given by the position of the two robots that are farthest away from each other. This allows to distinguish between *trains* and *wavefronts*.

We performed 50 trials for every non-dominated solution of each configuration to obtain a reliable classification, and on the basis of this sample, we ran the classification procedure shown in algorithm 2. We first classify as "disperse" those behaviors in which the group splits into two or more smaller groups more than 25% of the time. This is measured as $Q3(K) > 1$, where $Q3(K)$ is the third quartile of the number of connected components.[3] Then, we identify the solutions belonging to the class "stationary", which exhibit very low motion

[2]A video of the observed behaviors is available at `http://www.youtube.com/watch?v=DMTajtJasTs`

[3]The rest of the classification is performed discarding those trials in which the group splits (which are anyway a minority), in order to avoid incorrect classifications.

if $Q3(K) > 1$ **then**
  return *Disperse*
else if $Q_3(M) \leq D_a$ **then**
  return *Aggregation*
else if $Q_1(C) > D_c$ **then**
  return *Flocking*
else if $Q_2(\Theta) \leq \frac{\pi}{4}$ **then**
  return *Train*
else
  return *Wavefront*
end if

Algorithm 2: Classification of different behaviors (see text for details).

Configuration		Stationary		Disperse		Train		Wavefront		Flocking	
		#	%	#	%	#	%	#	%	#	%
3	00B00000R000	128	47,8%	45	16,8%	0	0,0%	1	0,4%	94	35,1%
4	00BB0000RR00	135	50,4%	50	18,7%	0	0,0%	4	1,5%	64	23,9%
5	0BBBB00RRRR0	155	57,8%	70	26,1%	0	0,0%	11	4,1%	41	15,3%
6	BBBBBBRRRRRR	211	78.7%	116	43,3%	0	0,0%	5	1,9%	11	4,1%
7	R00000B00000	200	74,6%	60	22,4%	41	15,3%	0	0,0%	32	11,9%
8	R0000BB0000R	180	67,2%	44	16,4%	90	33,6%	0	0,0%	20	7,5%
9	RR00BBBB00RR	168	62,7%	50	18,7%	122	45,5%	0	0,0%	15	5,6%
10	RRRBBBBBBRRR	185	69,0%	85	31,7%	82	30,6%	0	0,0%	8	3,0%

Table 6.2: Detailed classification results for each configuration.

values. We classify as "stationary" those behaviors that have $Q_3(M) \leq D_a$, where $Q_3(M)$ is the first quartile of the motion samples and $D_a = 0.25$ is an empirically selected threshold below which no significant motion of the group is observable. By using the third quartile we encompass in the class those behaviors that aggregate at least in 75% of the trials.

At this point, we have removed all cases that do not present a good coordinated motion behavior. We then classify as "flocking" those behaviors that do not present an elongated shape, determined as $Q_1(C) > D_c$, where $Q1(C)$ is the first quartile of the cohesion samples and $D_c = 0.8$ is an empirically determined threshold. This class encompasses behaviors in which the group presents high values of cohesion in at least 75% of the trials. Finally, we distinguish between "train" and "wavefront" on the basis of the angle Θ, which indicates whether the main axis of the group is close to the average direction of motion of the group (train) or orthogonal to it (wavefront). We therefore look at the median value $Q_2(\Theta)$ and classify the behavior as "train" in case it is lower than $\pi/4$, and as "wavefront" otherwise.

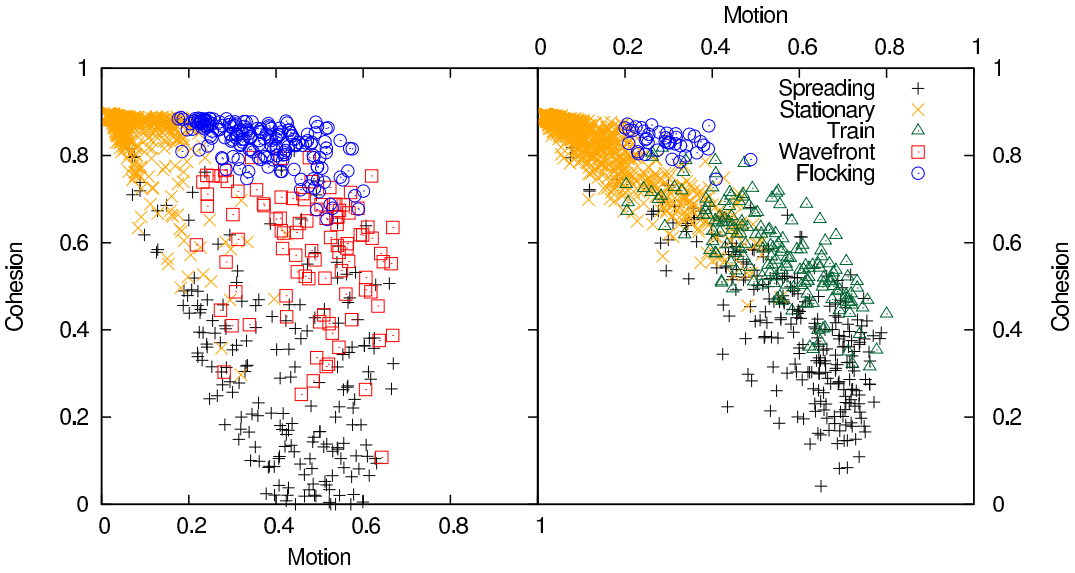

Figure 6.10: Classification of left-right (left) and front-rear (right) configurations.

Table 6.4 shows the behavior classification of every setup. We notice that left-right configurations do not produce "trains", and conversely front-rear configurations do not produce "wavefronts". Therefore, the LEDs configuration strongly determines the quality of behaviors that can be evolved. Moreover, we also notice that left-right configurations present an higher percentage of "flocking" behaviors, the lower the number of LEDs, the higher the probability to observe flocking. Therefore, the `00B00000R000` configuration seems to be the best one if flocking is the desired outcome.

Finally, we mapped the classification of behaviors over the corresponding points in the objective space, and we aggregate all solutions respectively for left-right and front-rear configurations (see Figure 6.10). It is possible to observe that different classes are rather separated on the objective space, with flocking behaviors occupying the area with high cohesion and variable motion. We also observe that "train" and "wavefronts" are specialized solutions, and that they are less clustered in the objective space, indicating a wide range of possibilities of displaying these behaviors in the different configurations. If we compare Figure 6.10 with Figure 6.9, we can understand how the ability to produce more flocking behaviors gives a performance advantage to the left-right configuration, while the front-rear configurations dominate mainly thanks to behavior in the "train" or "disperse" categories, which correspond to larger motion values. Again, from this analysis the left-right configuration results in better performance.

6.5 Summary

We have shown the importance of the robot configuration on the ability to evolve efficient coordinated motion behaviors in a swarm of robots. We evolved artificial neural networks controlling these robots to exhibit flocking behavior. The robots had different fixed LED configurations in order to test the effect of various setups. In particular, we compared 2-color front-rear and left-right setups that are seemingly identical in terms of information content. Yet, the evolved swarm behavior was qualitatively different, i.e., we obtained different variations of line following instead of flocking behavior. Thus, our results indicate that the selection of the robot configuration can determine the success or the failure of the evolutionary experiment. This also supports the need for a well-assessed methodology to guide the experimenter in the choice of the robot configuration.

CHAPTER

7 Self-organized Behavior in Adversarial Environment

Soccer is simple, but it is difficult to play simple.

– Johan Cruijff

Nature provides many examples of self-organized societies, such as foraging ants, or bird flocks. This has inspired many researchers to replicate similar behaviors with autonomous robots using different machine learning techniques. However, these experiments mostly assume a fixed environment to which the system has to adapt to, which limits the solutions to a particular experimental setting. Thus, an interesting question is if evolution is capable of finding a set of local rules that are robust enough to cope with an ever-changing, adversarial environment.

In this chapter we will study how a self-organizing system can be evolved that is exposed to an adversarial environment. In particular, we evolve a competing population of candidate solutions based on artificial neural networks that play a simplified version of the popular soccer game among each other. Unlike in real soccer games with attackers, defenders, and midfielders, in our implementation of the game no dedicated roles are assigned *a prior*. Thus the team has to self-organize and dynamically assign these roles in order to be successful and achieve victory. We analyze the evolved exhibited gameplay to see if this hypothesis is correct. Furthermore the effects of different neural network structures and interfaces between simulation and controllers are also investigated to increase the understanding of the required complexity of the underlying controllers for this type of emergent behavior, and to identify possible design issues.

While it is very hard to quantitatively assess the performance of a soccer team, our results indicated, that the robots managed to exhibit a high level of team play and succeeded in the game of soccer by dynamically assigning roles

to members of the team. We also find that the design of the interface between neural network and sensors/actuators plays a major role on the performance of the evolved solutions. In our experiments, a simple Cartesian representation of the sensors and the intended robot movements provided the lowest "cognitive complexity" for the artificial neural networks. This experiment and the corresponding results have been published at the Journal of Robotics [FE10].

7.1 Introduction

The concept of systems consisting of multiple autonomous mobile robots is attractive for several reasons [UFK97]: Multiple cooperative robots might be able to achieve a task with better performance or with lower cost. Moreover, loosely coupled distributed systems tend to be more robust, and more flexible than a single, more powerful robot performing the same task. A benefit from the collaborative interaction of mobile robots can be an emergent service, i.e., a progressive result that is more than the sum of the individual efforts [Joh02]. A swarm of robots can thus build a self-organizing system [FN00].

The continuous technical development in robotics in the last decades has provided us with the hardware for swarms of small cheap autonomous devices [Nov03, BP07, RSZF07]. However, designing the behavior and interactions between the robots remains a very complex task. Using a standard top-down design approach with fixed task decompositions and allocation typically leads to systems working only for a small set of parameters. On the other hand, effects like changing environments or breakdowns and faults of hardware require a robust and flexible solution that provides a useful service for many possible system states.

An alternative to the classical design approach is to organize the robots as a self-organizing system performing the intended task. Thus, the robots achieve a global system behavior via simple local interactions without centralized control [EdM08]. As shown by many examples in nature, simple rules for the interactions can emerge to quite complex behavior while being scalable and robust against disturbances and failures. This would allow for simple control systems like for example having a small ANN on the particular robots.

There are many examples of evolving swarms of autonomous robots (see Chapter 6.1), however most of these experiments assume a fixed environment. As explained in Chapter 3, a disadvantage of the evolutionary approach is that the solutions are limited to the conditions taken into account when the experiment was designed. Therefore, if the environment changes in an unexpected way, the whole evolutionary process has to be started over. A possible solution would be to coevolve the environment as well, i.e., let evolution find challenging

experimental setups in order to overcome the limitations of a human designer. The advantages of coevolution in evolutionary robotics have been discussed by Nolfi and Floreano *et al.* [NF98], where they coevolved predator and prey robots. However, in their study, dedicated roles are already assumed, meaning the personal goals of each individual are fixed, thus the effects of the environment are limited. This makes their results difficult to be extended to a situation considering self-organized teams of identical robots.

An idea would be to model the adversarial environment as a competing team with identical conditions racing for the same resources. Ideally, this would produce solutions that can adapt to conditions not considered by the experimenter, thus applicable to a wider range of environmental effects.

In this chapter we analyze how this goal can be achieved by evolving a competing population of ANN-based control systems for homogeneous teams of self-organizing robots. The robots are evolved to play a simplified version of the popular soccer game. In order to preserve the competitiveness of the game, no absolute fitness metric is used. Instead, selection is based on different tournament ranking algorithms. In our analysis we tackle the effect of different neural controller types and their interface to the robot and elaborate on the particular influence of fitness function parameters on the evolved behavior.

The chapter is structured as follows: The next section describes the details of the evolutionary experiment based on robot soccer, while Section 7.3 shows and explains the acquired results. This chapter is concluded in Section 7.4.

7.2 Experimental Setup

As a case study, we have defined the problem of teaching soccer to a team of autonomous robots in a 2D environment similar to the official RoboCup Simulation League. This problem provides a rich testbed domain for the study of control, coordination and cooperation issues described in [Kum03, BR00]. We used the *Simulated Robot Soccer* component of FREVO, which is based on the official RoboCup soccer simulator[1] with the same physics engine. However, in contrast to the official one, this implementation does not include the roles of a referee or goalkeeper and there is a simulated boundary around the field, which avoids situations where the ball goes out of bounds. A further change with respect to our approach is that our simulation runs as a discrete event simulation with maximum computation speed, or in real-time mode to allow watching a game. This greatly reduces the actual time for performing the evolution.

[1] http://sourceforge.net/projects/sserver

Population size	100
Number of generations	500
Percentage of elite selection	15
Percentage of mutations	40
Percentage of crossover	30
Percentage of randomly created offsprings	5

Table 7.1: Parameters of the evolutionary algorithms

A simulation run consists of initially placing a configurable number of soccer players (typically teams of 11 players each) on the respective side of the field and to simulate a game where each player can accelerate with a desired strength in a desired direction and, if being close enough to the ball, can kick the ball with the desired strength towards a given direction. One game lasts for 300 steps corresponding to 60 real-time seconds. This is a sufficient amount of time for gameplay and to score goals. Goals can be scored if the ball ends up in the opponent's goal area, after which the position of all players and the ball is reset. However, we introduced a border around the field to prevent the ball from leaving the designated area. Furthermore, we removed the offside rule for the sake of simplicity.

7.2.1 Evolutionary Algorithm

We used the *NNGA* evolutionary algorithm component of FREVO to evolve the controllers of the soccer players. The implementation of the optimization method is based on the one presented by Elmenreich and Klingler in [EK07]. The size of the population was set to 100 and the length of the simulation was fixed at 500 generations. The parameters of the genetic operators can be seen in Table 7.1.

7.2.2 Candidate Representations

The candidates for the evolutionary algorithm were realized as artificial neural networks. Training has been applied to optimize the weights and biases of the neural network. We tested two different candidate representations in our case study, namely a fully-connected and a feedforward ANN, both of which are standard components of FREVO.

The fully-connected network is a time-discrete, recurrent artificial neural network. Each neuron is connected to every other neuron and itself via several input connectors. Each connection is assigned a weight that is a floating point

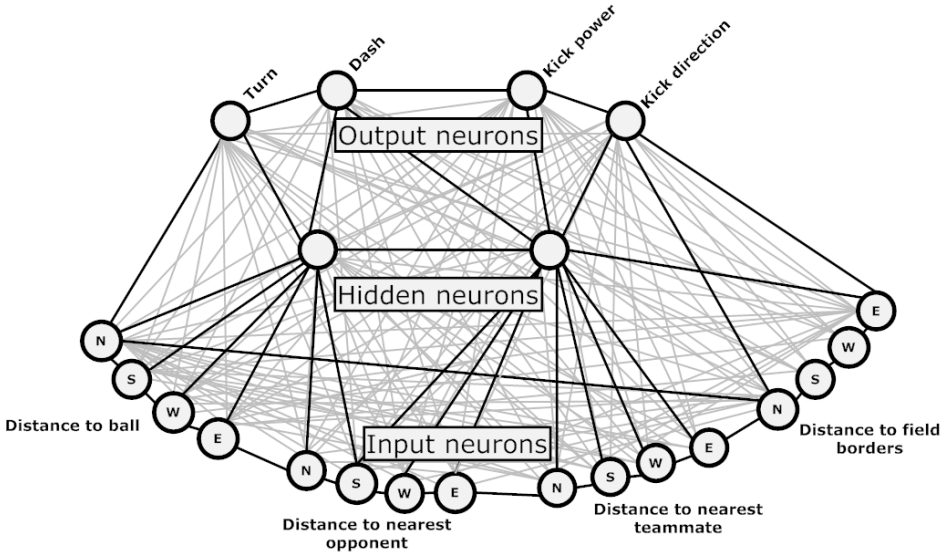

Figure 7.1: A possible wiring of the neural network showing the groups of inputs, outputs and hidden neurons. Connections with stronger weight are indicated with bold lines while ones with lower weight are colored with grey.

value and each neuron is assigned a bias. The problem requires 16 inputs and 4 outputs. Additional "hidden" neurons are added in order to increase the expressiveness and the number of representable states. The feedforward network only provides forward connections from the input nodes (the input layer) to the nodes in the hidden layer and forward connections from the hidden layer to the output layer.

In most cases, feedforward networks are employed for ANN applications, since they can be programmed via back propagation, by using a set of input-output pairs. However, our evolutionary setup provides only belated rewards, that is the feedback after a simulation involving many different actions of the ANN controller. Thus, it is not possible to know the right set of outputs for each upcoming set of inputs of a network during simulation.

The modular interface of FREVO allows an easy change between both representations. The implementation of both types is almost identical, the only difference is that the feedforward network only features a subset of connections per neuron. At each step, each neuron i builds the sum over its bias b_i and its incoming connection weights $w_j i$ multiplied by the current outputs of the neurons $j = 1, 2, ..., n$ feeding the connections. Weights can be any real number, thus have either an excitatory or inhibitory effect. The output of the neuron for step $k+1$ is calculated by applying a activation function F:

$$o_i(k+1) = F(\sum_{j=0}^{n} w_{ji} o_j(k) + b_i)$$

where F is a simple linear threshold function

$$F(x) = \begin{cases} 0.0 & \text{if } x \leq 0.0 \\ x & \text{if } 0.0 < x < 1.0 \\ 1.0 & \text{if } x \geq 1.0 \end{cases}$$

The linear threshold function has been chosen over a sigmoid one to improve computation speed and our tests did not show any significant difference in terms of performance. As described below in detail, the output of the output neurons is further scaled to match the input range of the actuators.

7.2.3 I/O interface between Simulation and Controllers

In the case study, the information passed to the simulation is predefined; it consists of the strength and direction for the players' moves and, in case a player can kick the ball, the strength and direction of the kick. The information provided by the simulation is also determined; it consists of the position of the ball, a list of visible teammates, a list of visible opponent players, and information about the distance to the field's (upper, lower, left, right) border. The visibility model was adopted from the official Robocup soccer simulator. The position of the goal is given indirectly by combining the distance to the borders with the knowledge that the goal resides in the middle of the side-borders.

However, there are different ways to pass this information into the controller. In general, the ANN will have a set of so-called input neurons, which activate their output according to a given input from external sources. Respectively, a number of output neurons is used to export information from the network. Finally, some unspecified or hidden neurons are added. All neurons are interconnected by directed weights, which are evolved in the framework (See Figure 7.1).

Since the number of neurons defines the search space, the number of neurons should not become too large. On the other hand, input neurons should be defined in a way that their result is easily interpretable by the ANN. For example in [EK07], a distance sensor was modeled that is periodically changing its orientation via several input neurons, each representing the input for a particular orientation. In the current example, the inputs are arranged in groups of four neurons. Each group is responsible for communicating the detection of

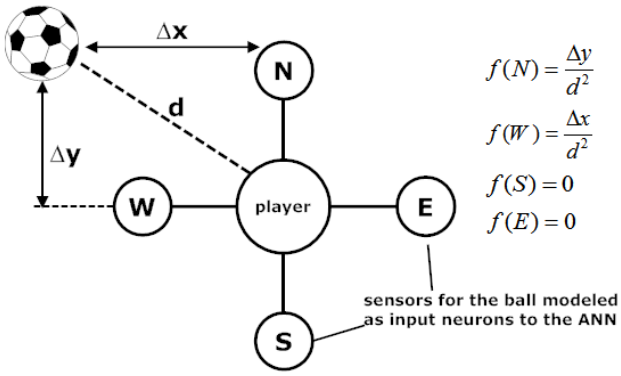

Figure 7.2: A group of input neurons detecting the ball

a particular object class (ball, teammate, opponent, border) and consists of a *north*, *south*, *east* and *west* neuron. This notation of directions translates to up, down, right, left for players of the team defending on the left side of the field and to down, up, left, right for the team defending on the right side of the field. This mirroring is necessary so that teams only have to learn playing on one side, thus the same behavior can be executed regardless of the starting conditions.

If the nearest object of that class is in a particular quadrant, lets say north-west, then the north neuron and the west neuron are activated inversely proportionally to the components of the vector to the object. So, if in our example the ball is towards the north-north-west, the north neuron gets a high activation and the west neuron a moderate one (See Figure 7.2).

For the output neurons, we tested two setups: in the first setup, the outputs are scaled to $[-100\%, 100\%]$ and $[-180, +180]$, respectively and interpreted as polar coordinates for the move and kick vectors. The second setup interprets the neurons as being the x and y components of a vector in Cartesian coordinates. In general, both approaches are expected to work, since they transport exactly the same information and the ANN will be evolved to adjust to the given representation.

7.2.4 Fitness Function

Typically, when the task is more complex the definition of the fitness function is not trivial. In the case of a soccer game the primary aim is to train teams scoring the most goals during the given time interval. However, this measure is far too large scale to be used for teams, initially composed of random ANN controllers, to expect improvement over generations just by rewarding them by the final number of goals they score in each game. The idea here is to decompose the overall goal into smaller achievements (so-called guidelines) and

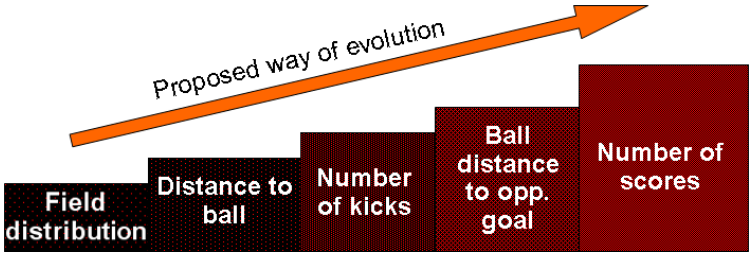

Figure 7.3: Weighted fitness

let the teams fulfill them one after the other. This method tries to ensure a smooth learning process assuming some preliminary knowledge or ideas about the solution. The guidelines are assigned a weight to setup a hierarchical order. It means a task with smaller weight is less important, but will most likely be accomplished before another tasks with higher value. This is because the second one is too complex to be achieved without learning the first one. Figure 7.3 shows the applied tasks in their respective order in our simulation.

At the beginning of the training we wanted the teams to learn that a good distribution on the field might lead to good overall play. Therefore, we introduced the first guideline (field distribution). It was implemented by defining 64 evenly distributed checkpoints on the field and counting the number of controlled points every 5 seconds for both teams. A point is controlled by a team if it has the nearest player to this point. The accumulated points are added to the final fitness value. The second guideline was an advice for the teams to move their players closer to the ball. The distance of the nearest player for both teams to the ball is measured and compared every 4 seconds. The team having a player closer to the ball earns one point. At the end of the game this point is weighted and also added to the final fitness. The number of kicks is also counted with a weight however only the first 10 kicks are taken into account to create an upper bound and to prevent dead team strategies where they only pass the ball back and forth. Concerning the kicking direction the ball distance to the opponent's goal is also measured and calculated every 2 game seconds in the same manner as guideline two. The highest weight is assigned to the number of scores, being more significant than the other fitness components. Therefore, we define the fitness function as the following equation:

$$F = w_p p_p + w_{bd} p_{bd} + (w_k p_k - w_{fk} p_{fk}) + w_{bg} p_{bg} + w_s p_s$$

where W and P stand for the weights and the points respectively. Table 7.2 explains the corresponding indexes and values.

i	P_i	W_i
p	field distribution	10^0
bd	distance to the ball	10^3
k	number of kicks	$2 \cdot 10^4$
fk	number of false kicks (ball is kicked out of bounds)	10^4
bg	ball distance to the opponent's goal	10^5
s	number of scores	$4 \cdot 10^6$

Table 7.2: Parameters of the fitness function

7.2.5 Optimized Tournament Ranking

Evolving competitive team behavior is a good example where one cannot assign a simple absolute fitness value. To rank the teams one solution is to play a tournament among the candidates in each generation (assuming one population with n candidates). A full tournament would mean $n(n-1)/2$ number of pairings when n is the number of entities in the population. In case a simulation run takes too much time or a high number of generations is needed, this approach can be very ineffective. For example, a population of 50 individuals would require 1225 runs for each generation. The proposed solution tries to minimize the number of necessary pairing using Swiss System style tournament [fid88]. It reduces the required number to $\lceil \log_2 n \rceil \frac{n}{2}$ which is in the mentioned case only 150 games, instead of 1225 (see Figure 7.4). Inspired by the official FIDE[2] rules for chess tournaments we established the following system:

In each game the winner gets two points, loser gets zero, in case of a draw both get one point. After the first round players are placed in groups according to their score (winners in group "2", those who drew in group "1", and losers in group "0"). The aim is to ensure that players with the same score play against each other. Since the number of perfect scores is cut in half each round, it does not take long until there is only one player left with a perfect score. The actual number of rounds needed is $\log_2 n$ to be able to handle n teams. In chess tournaments there are usually many draws, so more players can be handled (a 5-round event can usually determine a clear winner for a section of at least 40 players, possibly more), although in our simulation a draw is very unlikely. To avoid early games between elite selected entities, the first round is not randomized but cut into two halves where the first half, consisting of teams which have performed well so far, is playing against the second half.

The drawback of the Swiss system is that it is only designed to determine a clear winner in just a few rounds. Regarding other players, we have little

[2]http://www.fide.com

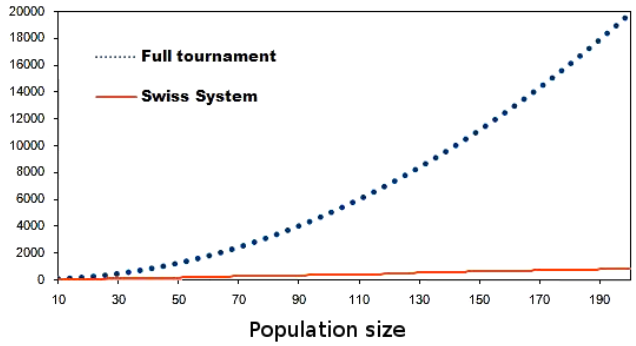

Figure 7.4: Total number of games in full tournament and Swiss System

information about their correct ranking. For example, there could be many players with 3-2 scores and it is hard to say which player is better than the other, or whether a player with 3.5 points is better than a player with 3 points. To help determine the order of finish, a tiebreak method has been implemented. In order to decide on the ranking for players having the same score, we used a method developed by Bruno Buchholz [HW92a]. There, the score of the players' opponents is summed up thus favoring those who have confronted better opponents. In case it is still undecided the sum is extended by the points of those opponents who have lost against the player. This uncertainty in the ranking could cause problems in the evolutionary algorithm when selecting entities for survival to the next generation. In our case elite selection was 15% while the Swiss System ensures only the first and last position to be ranked correctly, thus the position of all other players carries also some obscurity. After observing this effect in our simulation we came to the conclusion that having a somewhat imprecise selection among the top players slows down the process just a little or not at all.

To select entities for survival we used a roulette wheel selection where the probability being selected is directly proportional to the fitness, in our case the ranking of the Swiss System. Since this approach already carries some randomization some more uncertainty did not make a crucial impact.

7.3 Simulation Results

We ran several simulations evolving soccer teams. In particular, we varied

- the type of representation (fully connected or feedforward ANN),
- the number of hidden nodes (2, 4, or 6),
- the type of the interface between simulation and controllers.

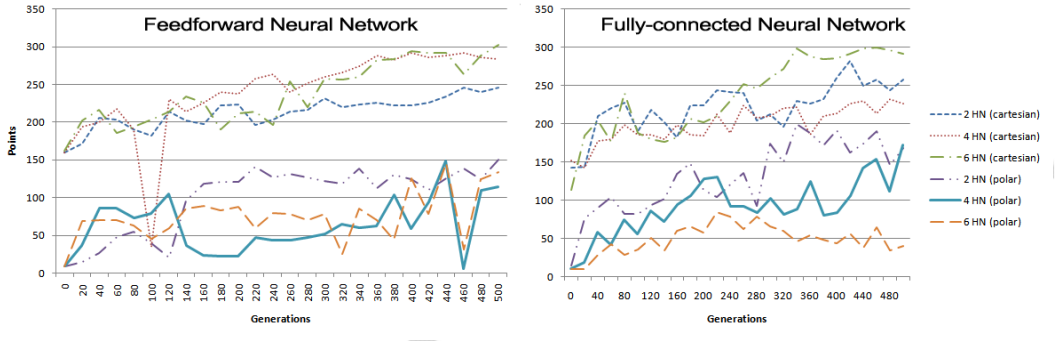

Figure 7.5: Tournament results of ANN with different I/O interfaces

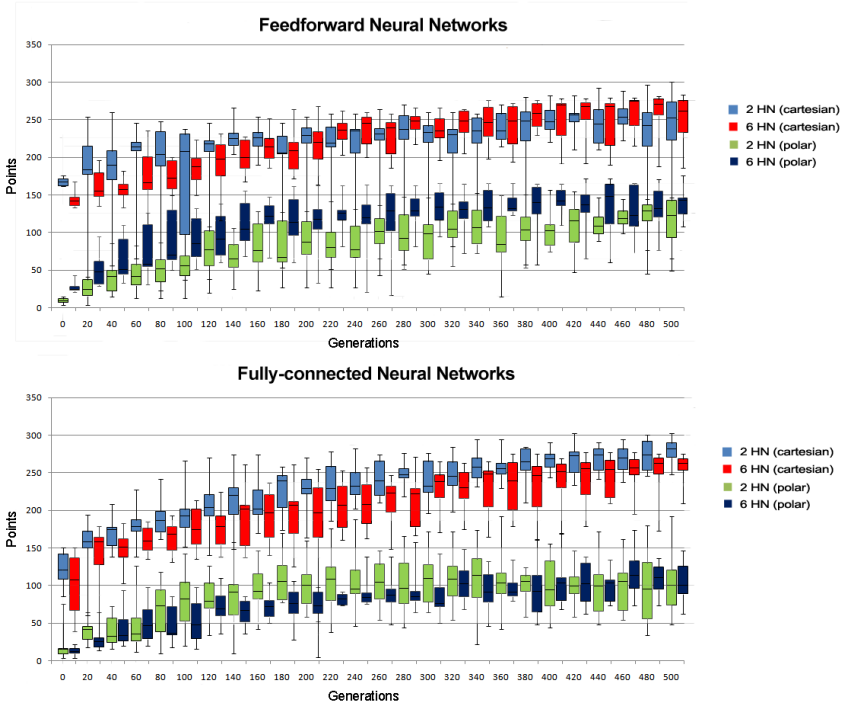

Figure 7.6: Box-and-whisker diagram of the repeated evaluation of different I/O models and different number of hidden neurons for feedforward and fully connected ANNs

We evolved each setting up to 500 generations. Unfortunately, there is no absolute fitness value for depicting the quality of an evolved result. Only relative comparisons of teams by matching them in a simulation are possible. For our evaluation, we picked the best result of every 20th generation. These "champions" have than been matched in a round-robin tournament against each other in order to determine if there is a constant evolution towards better gameplay and if one setting is performing better than the other.

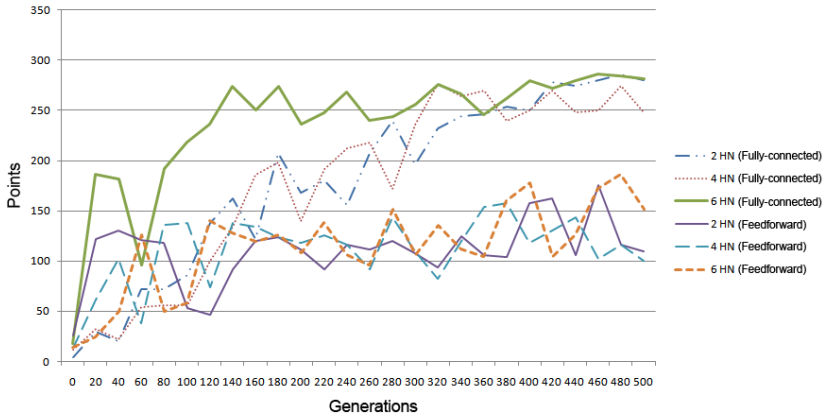

Figure 7.7: Tournament results of fully connected vs. feedforward ANN with Cartesian interface

We found out that the design of the interface between simulation and controllers is of major importance to the success of the evolutionary algorithm. The results showed that the selection of the interface between simulation and controllers has a significant influence on the speed of convergence and quality of the evolved solution. When the output neurons were interpreted as polar coordinates, the ANN controller needed to learn the coherent semantics of polar coordinates, and, probably, learn to emulate trigonometric functions. Figure 7.5 visualizes this observation for both, feedforward and fully connected ANN. The figure depicts the results of a tournament of the above mentioned champions for various settings. Most curves are increasing over generations which depicts that the gameplay of the teams has been improved by the evolutionary algorithm.

As can be seen in the graphs, the systems using polar coordinates, even after several hundreds generations, are ranked lower than almost every other systems using Cartesian coordinates. So, in this case, yielding output in polar coordinates posed a higher "cognitive complexity" for the system than yielding output in Cartesian coordinates. Figure 7.6 shows a boxplot covering 20 different iterations of the evolutionary algorithms and also confirms that the Cartesian coordinate I/O model is significantly superior to the polar coordinate model. This, however, may be specific for the chosen problem and may be different for other problems.

There was no significant effect of the number of hidden neurons on the overall performance or speed of convergence. This could be explained by the fact that a less complex ANN is already sufficient to learn the local interactions producing a competitive behavior.

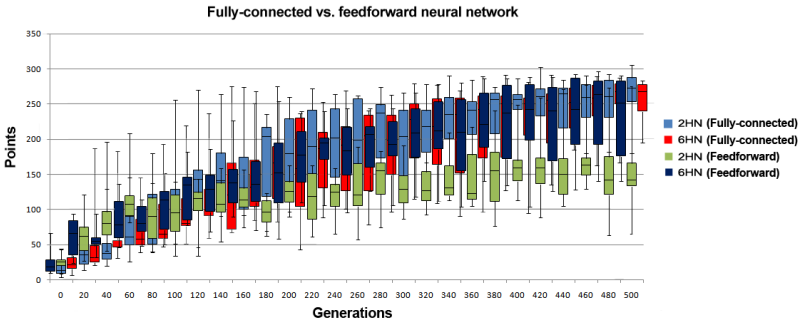

Figure 7.8: Box-and-whisker diagram of the repeated evaluation of fully connected vs. feedforward ANN with Cartesian interface

Figure 7.7 compares the different versions using polar coordinates with each other and depicts that the fully connected ANN have evolved faster and to a better gameplay than the feedforward networks. The box plot diagram in Figure 7.8 gives a statistic over 20 different runs of the evolutionary algorithm. While it confirms that all fully-connected ANNs are typically better than the feedforward ANNs with 2 hidden neurons, it also shows that a feedforward ANN with enough neurons (that is 6 in that case) is in many iterations able to compensate for the lower number of connections.

Thus, a fully connected network with 6 hidden neurons and an I/O interface based on Cartesian coordinates evolved 400 generations or more performed best according to the ability to win over others. Unfortunately, the quality and elegance of the result cannot be measured in this terms. By watching several games we observed the following behavior[3]:

- only the player nearest to the ball runs directly to the ball
- other players (of the same team) in the vicinity of the ball also follow the ball, but they do usually not converge to the same spot; instead they keep spread out
- the player at the ball kicks it to a direction bringing it nearer to the opponent's goal
- players far from the ball spread out and build a defense mesh in front of their own goal
- players sometimes tend to stick to opponent players (man-marking)

[3]A video of the evolution of the gameplay can be found at `http://mobile.uni-klu.ac.at/demesos`

Considering the relatively small size and complexity of the neural network controllers, the versatility of the emerging strategy is impressive. When both teams are well evolved, the ball is passed over several stations until the ball possession changes. Goals are scored roughly every few hundred simulation steps.

7.4 Summary

In this chapter, we studied how a self-organizing system can be evolved that is exposed to an adversarial environment that is coevolved. In particular, we have described an experiment where we evolved a competing population of artificial neural network controllers for a homogeneous team of cooperative robots. Given an overall goal function, we evolve the particular weights and biases of the neural network controllers using an evolutionary algorithm. Thus, the neural network learns to interpret the sensory inputs, to control the robots actuators and to behave according to a team strategy that is beneficial for the given task.

In our study, we have evolved control behavior for simulated soccer robots to cooperatively win soccer games. After a few hundred generations, the players of a team adopt a useful behavior. In contrast to related work, the players were not evolved to *a priori* defined roles, like defender, midfielder or striker, but all have an instance of the same neural network controller. Still, during a game, different behavior of the players emerge based on their situations. Thus, similar to biological systems, the entities specify to different roles in a self-organizing way. Since the entities are identical, the system has a high robustness against failure of some of the entities.

We have examined the influence of various factors to the results. The most important factor was the design of the interface between neural network and sensors/actuators. Although an ANN could theoretically adopt to different representations of sensor/actuator interfaces, it was necessary to find an interface with low "cognitive complexity" for the ANN, which was in our case a simple Cartesian representation of the sensors and intended robot movements. Furthermore, we analyzed the influence of using different sizes and types of ANN. While the number of neurons had the smallest effect on the performance, the type of representation favored the fully-connected network type.

CHAPTER 8 Conclusions

Increasing the level of self-organization within our networked technical systems presents the possibility to cope with the rising complexity system designers have to face. Advantages like robustness against single point failure, scalability, and adaptability are all desired when dealing with systems composed of a huge number of interdependent, networked components. The problem with designing self-organizing systems lies in the fact, that the phenomena of self-organization is essentially a bottom-up process, thus traditional top-down design approaches usually cannot be applied.

In this thesis we investigated how evolutionary algorithms can be used to design self-organizing technical systems, i.e., to find the appropriate micro-level rules of a system that drive it into the desired emergent behavior. Our survey of existing literature has revealed that despite the many reports on applying evolutionary approaches to generate self-organizing systems, there is a need for a generic description of the underlying system engineering process. In Chapter 3 we discussed the basic building blocks for applying evolutionary algorithms to the design of self-organizing systems for technical applications and proposed a methodology built on top of this that can potentially guide the system designer when engineering self-organizing technical systems. In this methodology we highlighted several decision points in each building block where careful consideration from the system designer is needed.

FREVO, a novel evolutionary software framework was also introduced with the aim to help system designers dealing with self-organizing systems. FREVO is a modular piece of software offering various components required to fully evolve and evaluate a self-organizing system. This includes popular optimization algorithms, genotype-to-phenotype mappings, and benchmark problems. This framework fully supports the presented design methodology.

Three case studies from the fields of cellular automata and swarm robotics have been described as examples for applying self-organization in a networked

technical system. Furthermore, we investigated several of the mentioned design decision points in order to increase the understanding of their effects on the evolved self-organizing system. In Chapter 5 a cellular automata (CA) model was described where each cell was controlled by an artificial neural network. These networks were evolved so that the cells display a previously defined reference pattern. In this study the relation between problem complexity and controller complexity has been analyzed. We found that CAs displaying simple structures can be evolved, but the performance rapidly decreases with patterns of higher complexity, even when using more complex neural network structures. Thus, the applied evolutionary algorithm could not cope with the large search space presented by the problem.

Swarm robotics is a widely used application field for self-organization. In Chapter 6 we investigated how different interaction interfaces effect the quality of the evolved behavior. We trained groups of robots to display flocking behavior defined by a multi-objective fitness function. Our results showed completely different evolved group behaviors even when using seemingly identical configurations. This points towards the need of automated interface configurations.

Self-organizing robot soccer teams were evolved in Chapter 7. Though the team is composed of identical robots, the evolved teams learned to dynamically assign roles in order to achieve victory. This study also shows how competing self-organizing systems can be successfully coevolved in order to continuously present challenging environments they need to adapt to. Furthermore, comparisons of different neural network controller structures indicated that the size of the networks have little impact on the performance. Thus, simple controller representations are enough to obtain complex adaptive self-organizing team behavior.

8.1 Contributions

A major contribution of this work is a proposal of such a methodology, which analyzes several vital cornerstones of the evolutionary design of self-organizing technical systems. It provides a set of guidelines for engineers and scholars working with such systems in order to help to identify and avoid possible bad design decisions leading to sub-optimal or non-working solutions. We address the major decision points by decomposing the overall effort into the following parts: the simulation model, the evolvable decision unit, the interaction interface, the search algorithm, and the objective function. The simulation model addresses issues concerning the relation between the system and its environment, the heterogeneity of the system, and the applied modelling technique. The evolvable decision unit is a representation of the components' controllers

that essentially forms the underlying logic, or the set of microscopic (local) rules in a self-organizing system. The interaction interface describes how this decision unit interacts with other components and its environment. The task of the search algorithm is to optimize the decision unit models of the components according to the objective function, which is a formal description of the desired goals. This approach may provide the basis for a qualified engineering process for self-organizing systems in technical applications.

The case-studies investigating the general evolvability of self-organizing systems in various perspectives are essential contributions of this thesis. In Chapter 5 we demonstrated the design approach by generating a complex self-organizing multicellular system based on cellular automata. In the presented model the cell behavior was controlled by an artificial neural network according to the cell's internal state. The idea was to evolve the cell rules in a way that a previously defined reference pattern emerges solely by the interactions of the cells. A definite contribution of this work was to present how a morphogenetic process can be initiated by integrating ANNs into a self-organizing CA model. We also introduced a quantitative measure for problem complexity based on spatial entropy in order to assess the difficulty of a particular reference image. This has been used to investigate the correlation between the performance of the evolved solutions and the difficulty of the problem. The best results have been achieved with simple structures consisting of large areas of a single color as they are present for example in flags. However, the presented setup could not solve images of high complexity (for example a simplified version of the Mona Lisa paiting) as the evolutionary process got stuck at a suboptimal stage. Thus, the results indicate a rapid decrease of fitness values with the increase of image complexity. Furthermore, our findings also show that this task can be very difficult even for more complex neural networks. Attempts using NEAT - an algorithm that optimizes the structure of the neural network - performed similarly as fixed neural networks with similar number of hidden neurons.

Two additional studies were presented in the domain of swarm robotics. The experiments described in Chapter 6 aim at increasing the understanding the effects of different interaction interfaces used for self-organizing swarms of robots. We evolved artificial neural network controllers for the robots to exhibit flocking behavior. The robots had different fixed LED configurations used for signalling their orientation, in order to test the effect of various setups. In particular, we compared 2-color front-rear and left-right setups that are seemingly identical in terms of information content. Yet, the evolved swarm behavior was qualitatively different, i.e., we obtained different variations of line following instead of flocking behavior. Our results indicate that the selection of the interaction interface can determine the success or the failure of the evolutionary

experiment. Thus a contribution of this work is to show the importance of the interface design on the ability to evolve efficient coordinated motion behaviors in a self-organizing team of robots.

The second case-study on evolutionary swarm robotics is described in Chapter 7, where we studied how a self-organizing system can be evolved that is exposed to a coevolved adversarial environment. In particular, we have described an experiment where we evolved a competing population of artificial neural network controllers for a homogeneous team of cooperative robots playing a simplified version of the RoboCup simulated soccer game. After a few hundred generations, the players of a team adopt a useful behavior. Unlike in real soccer games with attackers, defenders, and midfielders, in our implementation no specific roles were assigned *a priori*, but all robots were controlled by an own instance of the same neural network controller. Still, during a game, different behavior of the players emerge based on their situations. Thus, similar to biological systems, the entities specify to different roles in a self-organizing way. Since the entities are identical, the system has a high robustness against failure of some of the entities and a higher degree of scalability is ensured compared to teams with fixed configuration. We also examined the influence of the design of the interface between neural network and sensors/actuators. Although an ANN could theoretically adopt to different representations of sensor/actuator interfaces, it was necessary to find an interface with low "cognitive complexity" for the ANN, which was in our case a simple Cartesian representation of the sensors and intended robot movements. Furthermore, we analyzed the influence of using different sizes and types of fixed-structured ANNs. While the number of neurons had the smallest effect on the performance, the type of representation favored the fully-connected network type.

Furthermore, a novel software tool is described that aims to bring all this knowledge into a modular, easy-to-use software framework especially designed for evolving and evaluating self-organizing networked systems. With the principle of reusable, independent components, FREVO allows for easily exchanging different implementations of evolvable agent controllers, optimization methods and ranking methods. A simple example can be implemented with a few lines of code. The implementation effort is thus reduced to defining the context, the fitness function and define input and output of the agent. After evolving the agent controllers, the simulations with the resulting candidates can be replayed either with the same settings or with different parameters for evaluation purposes.

8.2 Future Work

There are many directions for future work due to the diverse nature of this thesis's content. Regarding the proposed design methodology, an important challenge is to tackle the integration of the presented approach with a system engineering approach that yields dependable and trustworthy self-organizing systems. Furthermore, deeper analysis of the components is necessary that involves novel algorithms and machine learning techniques. Additionally, the there is a vast set of meta-heuristic algorithms whose performance can be elaborated on self-organizing systems.

The presented findings of the case studies offer also many possibilities for future work. With respect to Chapter 5, there is a large space of possibilities for variations of the model which gives rise to future work. E. g., findings on well-suited or less well-suited model configurations could give insight to the understanding of such phenomena as morphogenesis and camouflage mechanisms in nature. Future experiments are planned to study the effects of increasing or decreasing the ability of ANNs to communicate with their neighbors. For evolving structures rather than replications of images, a new fitness function design is planned in a way to have the fitness based on the type of the emerging structure instead of a pixel-by-pixel comparison. Possible applications of this research could be the self-organized pattern formation in swarm robotics. In other words, given a desired pattern, how can robots acquire it? Another application could be smart painting (as indicated in [AAC+00]) that would decide on its color based on a morphogenetic process having only a few distinctive sensory inputs, thus not allowing for a zygote approach.

Insights presented in Chapter 6 raises the need of well-assessed methodologies to guide the experimenter in the choice of the robot configuration. In this respect, we envisage in future work the usage of automated methodologies. For instance, the robot configuration could be put under evolutionary pressure (i.e., as an additional objective in a multi-objective setup). However, suitable encoding must be devised to ensure the co-evolvability of configuration and behavior.

Our study on the evolution of self-organizing soccer robots open up interesting questions as well. One would be to investigate the effects of different evolutionary strategies, e.g., coevolutionary algorithms with multiple populations. Such approaches would allow very different strategies to emerge separately, thus providing solutions with even higher rates of adaptability. Furthermore, the flexibility of FREVO allows studies leading in many different directions, such as studying heterogeneous swarms, different controller types (instead of neural networks) or the effects of certain fitness functions.

FREVO is continuously being developed to offer more tools and algorithms for the users. Therefore, a definite future plan is to concentrate on further integrating different representation models, optimization methods, such as particle swarm optimization and ranking methods for instances of `AbstractMultiProblem`s.

Bibliography

[AAC+00] H. Abelson, D. Allen, D. Coore, C. Hanson, G. Homsy, T.F. Knight, R. Nagpal, E. Rauch, G.J. Sussman, and R. Weiss. Amorphous Computing. *Communications of the ACM*, 43(5):74–82, 2000.

[Abe09] T. Abeel. Java machine learning library, 2009.

[Alt94] L. Altenberg. Advances in genetic programming. In Kenneth E. Kinnear, Jr., editor, *The evolution of evolvability in genetic programming*, chapter 3, pages 47–74. MIT Press, Cambridge, MA, USA, 1994.

[Art08] S. Arteconi. Evolutionary methods for self-organizing cooperation in peer-to-peer networks. Technical Report UBLCS-2008-5, Department of Computer Science, University of Bologna, 2008.

[Ash47] W. R. Ashby. Principles of the self-organizing dynamic system. *Journal of General Psychology*, 37:125–128, 1947.

[Ash62] W. R. Ashby. Principles of the self-organizing system. In H. v. Foerster and G. W. Zopf, editors, *Principles of Self-Organization: Transactions of the University of Illinois Symposium*, pages 255–278. Pergamon, London, 1962.

[AWdM08] C. Auer, P. Wüchner, and H. de Meer. A method to derive local interaction strategies for improving cooperation in self-organizing systems. In K. A. Hummel and J. P.G. Sterbenz, editors, *Self-Organizing Systems*, volume 5343 of *Lecture Notes in Computer Science*, pages 170–181. Springer Berlin Heidelberg, 2008.

[BBZ05] G. Buason, N. Bergfeldt, and T. Ziemke. Brains, bodies, and beyond: Competitive co-evolution of robot controllers, morphologies and environments. *Genetic Programming and Evolvable Machines*, 6:25–51, 2005.

[BG97] Z. Boger and H. Guterman. Knowledge extraction from artificial neural network models. In *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics 1997*, volume 4, pages 3030–3035 vol.4, 1997.

[BK99] P. Bentley and S. Kumar. Three ways to grow designs: A comparison of embryogenies for an evolutionary design problem. In *In Proceedings of the Genetic and Evolutionary Computation Conference*, pages 35–43. Morgan Kaufmann, 1999.

[BK11] U. Boryczka and J. Kozak. New insights of cooperation among ants in ant colony decision trees. In *Nature and Biologically Inspired Computing (NaBIC), 2011 Third World Congress on*, pages 255 –260, oct. 2011.

[BL04] M.J. Berry and G.S. Linoff. *Data Mining Techniques: For Marketing, Sales, and Customer Relationship Management*. Wiley technology publication. Wiley, 2004.

[BLM+10] M. Bonani, V. Longchamp, S. Magnenat, P. Rétornaz, D. Burnier, G. Roulet, F. Vaussard, H. Bleuler, and F. Mondada. The marXbot, a miniature mobile robot opening new perspectives for the collective-robotic research. In *Processings of the 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4187–4193, 2010.

[Blu92] A. Blum. *Neural Networks in C++: An Object-Oriented Framework for Building Connectionist Systems*. Number v. 1 in Wiley professional computing. John Wiley & Sons, 1992.

[BMA06] M. Batouche, S. Meshoul, and A. Abbassene. On solving edge detection by emergence. In Moonis Ali and Richard Dapoigny, editors, *Advances in Applied Artificial Intelligence*, volume 4031 of *Lecture Notes in Computer Science*, pages 800–808. Springer Berlin Heidelberg, 2006.

[BMSG+09] Y. Brun, G. M. Serugendo, C. Gacek, H. Giese, H. Kienle, M. Litoiu, H. Mller, M. Pezz, and M. Shaw. Engineering self-adaptive systems through feedback loops. In Betty H.C. Cheng, R. Lemos, H. Giese, P. Inverardi, and J. Magee, editors, *Software Engineering for Self-Adaptive Systems*, volume 5525 of *Lecture Notes in Computer Science*, pages 48–70. Springer Berlin Heidelberg, 2009.

[BNE07] N. Beume, B. Naujoks, and M. Emmerich. SMS-EMOA: Multiobjective selection based on dominated hypervolume. *European Journal of Operational Research*, 181(3):1653–1669, 2007.

[BP07] S. Bergbreiter and K. S. J. Pister. Design of an autonomous jumping microrobot. In *IEEE International Conference on Robotics and Automation*, Rome, Italy, 2007.

[BR00] S. Buck and M. A. Riedmiller. Learning situation dependent success rates of actions in a robocup scenario. In *PRICAI*, page 809, 2000.

[BR01] F. Bellifemine and G. Rimassa. Developing multi-agent systems with a fipa-compliant agent framework. *Software Practice and Experience*, 31(2):103–128, February 2001.

[BS08] J. Branke and H. Schmeck. Evolutionary design of emergent behavior. In *Organic Computing*, Understanding Complex Systems, pages 123–140. Springer Berlin Heidelberg, 2008.

[Buc88] J. Buck. Synchronous Rhythmic Flashing of Fireflies. II. *The Quarterly Review of Biology*, 63(3):265–289, 1988.

[BY99] Y. Bar-Yam. *Dynamics of Complex Systems*. Perseus Books, 1999.

[CB92] M. Caudill and C. Butler. *Understanding Neural Networks; Computer Explorations*. MIT Press, Cambridge, MA, USA, 1992.

[CCG+10] A. Cavagna, A. Cimarelli, I. Giardina, G. Parisi, R. Santagati, F. Stefanini, and M. Viale. Scale-free correlations in starling flocks. *Proceedings of the National Academy of Sciences*, 107(26):11865, 2010.

[CD06] A. Chavoya and Y. Duthen. Using a genetic algorithm to evolve cellular automata for 2d/3d computational development. In *GECCO '06: Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pages 231–232, New York, NY, USA, 2006. ACM.

[CD07] A. Chavoya and Y. Duthen. Use of a genetic algorithm to evolve an extended artificial regulatory network for cell pattern generation. In *GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 1062–1062, New York, NY, USA, 2007. ACM.

[CFS+01] S. Camazine, N. R. Franks, J. Sneyd, E. Bonabeau, J.-L. Deneubourg, and G. Theraula. *Self-Organization in Biological Systems*. Princeton University Press, Princeton, NJ, USA, 2001.

[Cho01] H. Choset. Coverage for robotics a survey of recent results. *Annals of Mathematics and Artificial Intelligence*, 31(1-4):113–126, 2001.

[CKJ+02] I. D. Couzin, J. Krause, R. James, G. D. Ruxton, and N.R. Franks. Collective memory and spatial sorting in animal groups. *Journal of Theoretical Biology*, 218(1):1–11, 2002.

[CLG+09] B. H. Cheng, R. Lemos, H. Giese, P. Inverardi, J. Magee, J. Andersson, B. Becker, N. Bencomo, Y. Brun, B. Cukic, G. Marzo S., S. Dustdar, A. Finkelstein, C. Gacek, K. Geihs, V. Grassi, G. Karsai, H. M. Kienle, J. Kramer, M. Litoiu, S. Malek, R. Mirandola, H. A. Müller, S. Park, M. Shaw, M. Tichy, M. Tivoli, D. Weyns, and J. Whittle. Software engineering for self-adaptive systems. chapter Software Engineering for Self-Adaptive Systems: A Research Roadmap, pages 1–26. Springer-Verlag, Berlin, Heidelberg, 2009.

[CLNR12] C. Costanzo, V. Loscri, E. Natalizio, and T. Razafindralambo. Nodes self-deployment for coverage maximization in mobile robot networks using an evolving neural network. *Computer Communications*, 35(9):1047–1055, 2012.

[COD09] A.L. Christensen, R. O'Grady, and M. Dorigo. From fireflies to fault-tolerant swarms of robots. *IEEE Transactions on Evolutionary Computation*, 13(4):754–766, 2009.

[Cor04] IBM Corp. *An architectural blueprint for autonomic computing*. IBM Corp., USA, October 2004.

[Cra85] N. L. Cramer. A representation for the adaptive generation of simple sequential programs. In *Proceedings of the 1st International Conference on Genetic Algorithms*, pages 183–187, Hillsdale, NJ, USA, 1985. L. Erlbaum Associates Inc.

[CT06] T. M. Cover and J. A. Thomas. *Elements of Information Theory (Wiley Series in Telecommunications and Signal Processing)*. Wiley-Interscience, 2006.

[Dar59] C. Darwin. *On the Origin of the Species by Means of Natural Selection: Or, The Preservation of Favoured Races in the Struggle for Life*. John Murray, 1859.

[DCD98] G. Di Caro and M. Dorigo. Antnet: distributed stigmergetic control for communications networks. *J. Artif. Int. Res.*, 9(1):317–365, December 1998.

[Des37] R. Descartes. Discourse on method, 1637.

[dFFH01] Viviane Grunert da Fonseca, Carlos M. Fonseca, and Andreia O. Hall. Inferential performance assessment of stochastic optimisers and the attainment function. In *Lecture Notes in Computer Science*, pages 213–225. Springer, 2001.

[DG89] K. Deb and D. E. Goldberg. An investigation of niche and species formation in genetic function optimization. In *Proceedings of the 3rd International Conference on Genetic Algorithms*, pages 42–50, San Francisco, CA, USA, 1989. Morgan Kaufmann Publishers Inc.

[dGTH+08] Hugo de Garis, Jian Yu Tang, Zhiyong Huang, Lu Bai, Cong Chen, Shuo Chen, Junfei Guo, Xianjin Tan, Hao Tian, Xiaohan Tian, Xianjian Wu, Ye Xiong, Xiangqian Yu, and Di Huang. The china-brain project: Building china's artificial brain using an evolved neural net module approach. In *Proceeding of the 2008 Conference on Artificial General Intelligence 2008*, pages 107–121, Amsterdam, The Netherlands, 2008. IOS Press.

[Dic05] G. Dick. A comparison of localised and global niching methods. In *17th Annual Colloquium of the Sparial Information Research Centre (SIRC 2005: A Spatio-temporal Workshop)*, pages 91–101. Dunedin, New Zealand, 2005.

[DMC96] M. Dorigo, V. Maniezzo, and A. Colorni. Ant system: optimization by a colony of cooperating agents. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 26(1):29–41, 1996.

[dNMB02] W. de Nooy, A. Mrvar, and V. Batagelj. *Exploratory Social Network Analysis with Pajek*. Cambridge University Press, 2002.

[EB04] B. Edmonds and J. J. Bryson. The insufficiency of formal design methods - the necessity of an experimental approach for the understanding and control of complex MAS. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems - Volume 2*, AAMAS '04, pages 938–945, Washington, DC, USA, 2004. IEEE Computer Society.

[EdM08] W. Elmenreich and H. de Meer. Self-organizing networked systems for technical applications: A discussion on open issues. In K. A. Hummel and J. P. G. Sterbenz, editors, *Self-Organizing Systems*, volume 5343 of *Lecture Notes in Computer Science*, pages 1–9. Springer Berlin Heidelberg, 2008.

[EEK93] G.B. Ermentrout and L. Edelstein-Keshet. Cellular automata approaches to biological modeling. *Journal of Theoretical Biology*, 160(1):97 – 133, 1993.

[EF11] W. Elmenreich and I. Fehérvári. Evolving self-organizing cellular automata based on neural network genotypes. In *Proceedings of the Fifth International Workshop on Self-Organizing Systems*, volume LNCS 6557, pages 16–25. Springer Verlag, 2011.

[EIF09] W. Elmenreich, T. Ibounig, and I. Fehérvári. Robustness versus performance in sorting and tournament algorithms. *Acta Polytecnica*, 6(5):7–18, 2009.

[EK07] W. Elmenreich and G. Klingler. Genetic evolution of a neural network for the autonomous control of a four-wheeled robot. In *Sixth Mexican International Conference on Artificial Intelligence (MICAI'07)*, Aguascalientes, Mexico, nov 2007.

[Elm07] W. Elmenreich. A review on system architectures for sensor fusion applications. In *The 5th IFIP Workshop on Software Technologies for Future Embedded & Ubiquitous Systems*, 2007.

[ES77] M. Eigen and P. Schuster. A principle of natural self-organization. *Naturwissenschaften*, 64(11):541–565, 1977.

[ES03] Agoston E. Eiben and J. E. Smith. *Introduction to Evolutionary Computing*. Springer-Verlag Berlin, 2003.

[FaHY04] Wei F., Yi an H., and P. S. Yu. Decision tree evolution using limited number of labeled data items from drifting data streams. In *Fourth IEEE International Conference on Data Mining, 2004. ICDM '04.*, pages 379–382, 2004.

[FC54] B. Farley and W. Clark. Simulation of self-organizing systems by digital computer. *Information Theory, IRE Professional Group on*, 4(4):76 –84, September 1954.

[FDM08] D. Floreano, P. Drr, and C. Mattiussi. Neuroevolution: from architectures to learning. *Evolutionary Intelligence*, 1(1):47–62, 2008.

[FE09a] I. Fehérvári and W. Elmenreich. Evolutionary methods in self-organizing system design. In *Proceedings of the 2009 International Conference on Genetic and Evolutionary Methods*, pages 10–15, 2009.

[FE09b] I. Fehérvári and W. Elmenreich. Towards evolving cooperative behavior with neural controllers. In *IFIP Fourth International Workshop on Self-Organizing Systems*, 2009.

[FE10] I. Fehérvári and W. Elmenreich. Evolving neural network controllers for a team of self-organizing robots. *Journal of Robotics*, 2010.

[FE13] I. Fehérvári and W. Elmenreich. Evolution as a tool to design self-organizing systems. In *IFIP 7th International Workshop on Self-Organizing Systems*, 2013.

[fid88] *FIDE Swiss Rules: Instruction Manual*. Federation Internationale des Echecs, 1988.

[FK10a] D. Floreano and L. Keller. Evolution of adaptive behaviour in robots by means of Darwinian selection. *PLoS biology*, 8(1), January 2010.

[FK10b] D. Floreano and L. Keller. Evolution of adaptive behaviour in robots by means of darwinian selection. *Plos Biology*, 8(1):e1000292, 2010.

[FN00] D. Floreano and S. Nolfi. *The role of self-organization for the synthesis and the understanding of behavioral systems*, chapter 1, pages 1–18. MIT Press: Cambridge, 2000.

[Fon08] A. Fontana. Epigenetic tracking, a method to generate arbitrary shapes by using evolutionary-developmental techniques. May 2008.

[Fon09] A. Fontana. Epigenetic tracking: A possible solution for evo-devo morphogenesis? In *Proceedings of the 1st International Workshop on Morphogenetic Engineering*, 2009.

[FOW66] Michael J. Fogel, Lawrence J. Owens, and Alvin J. Walsh. *Artificial Intelligence through Simulated Evolution*. Wiley, Chichester, WS, UK, 1966.

[FTE13] I. Fehérvári, V. Trianni, and W. Elmenreich. On the effects of the robot configuration on evolving coordinated motion behaviors. In *2013 IEEE Congress on Evolutionary Computation (CEC)*, pages 1209–1216, 2013.

[FU00] D. Floreano and J. Urzelai. Evolutionary robots with on-line self-organization and behavioral fitness. *Neural Networks*, 13(45):431 – 443, 2000.

[GA05] C. Gershenson and CL. Apostel. Self-organizing traffic lights. *Complex Systems*, 2005.

[Gar70] M. Gardner. The fantastic combinations of John Conway's new solitaire game "life". *Scientific American*, 223:120–123, October 1970.

[Gau04] S. Gaukroger. *Descartes, an Intellectual Biography*. Oxford University Press, 2004.

[GCGC08] M.-P. Gleizes, V. Camps, J.-P. Georg, and D. Capera. Engineering systems which generate emergent functionalities. In D. Weyns, S. A. Brueckner, and Y. Demazeau, editors, *Engineering Environment-Mediated Multi-Agent Systems*, volume 5049 of *Lecture Notes in Computer Science*, pages 58–75. Springer Berlin Heidelberg, 2008.

[Ger07] C. Gershenson. *Design and Control of Self-organizing Systems*. PhD thesis, Vrije Universiteit Brussel, 2007.

[GH03] C. Gershenson and F. Heylighen. When can we call a system self-organizing? In W. Banzhaf, J. Ziegler, T. Christaller, P. Dittrich, and J. T. Kim, editors, *Advances in Artificial Life*, volume 2801 of *Lecture Notes in Computer Science*, pages 606–614. Springer Berlin Heidelberg, 2003.

[Gol89] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1989.

[GR87] D. E. Goldberg and J. Richardson. Genetic algorithms with sharing for multimodal function optimization. In *Proceedings of the Second International Conference on Genetic Algorithms on Genetic algorithms and their application*, pages 41–49, Hillsdale, NJ, USA, 1987. L. Erlbaum Associates Inc.

[Gre12] G. Greenfield. A platform for evolving controllers for simulated drawing robots. In *Evolutionary and Biologically Inspired Music, Sound, Art and Design*, volume 7247 of *Lecture Notes in Computer Science*, pages 108–116. Springer Berlin Heidelberg, 2012.

[GYWA05] J.M.E. Gabbai, Hujun Yin, W. A. Wright, and N.M. Allinson. Self-organization, emergence and multi-agent systems. In *International Conference on Neural Networks and Brain, 2005. ICNN B '05.*, volume 3, pages nil24–1863, 2005.

[Hak78] H. Haken. *Synergetics: An Introduction. Nonequilibrium Phase Transitions and Self- Organization in Physics, Chemistry and Biology (Springer Series in Synergetics)*. Springer, November 1978.

[HdMB08] R. Holzer, H. de Meer, and C. Bettstetter. On autonomy and emergence in self-organizing systems. In *Proceedings of the 3rd International Workshop on Self-Organizing Systems*, IWSOS '08, pages 157–169, Berlin, Heidelberg, 2008. Springer-Verlag.

[HDT02] A.T Hayes and P Dormiani-Tabatabaei. Self-organized flocking with agent failure: Off-line optimization and demonstration with real robots. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA '02)*, pages 3900–3905 vol.4. IEEE, 2002.

[HDW+13] I. Harvey, E. Di Paolo, R. Wood, M. Quinn, and E. Tuci. Evolutionary robotics: A new scientific tool for studying cognition. *Artificial Life*, 11(1-2):79–98, 2013.

[HL73] G. T. Herman and W. H. Liu. The daughter of celia, the french flag and the firing squad. In *WSC '73: Proceedings of the 6th conference on Winter simulation*, page 870, New York, NY, USA, 1973. ACM.

[HLV+11] S. Hauert, S. Leven, M. Varga, F. Ruini, A. Cangelosi, J.C. Zufferey, and D Floreano. Reynolds flocking in reality with fixed-wing robots: Communication range vs. maximum turning rate. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5015–5020. IEEE, 2011.

[Hol75] J. H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, MI, USA, 1975.

[HW92a] D. Hooper and K. Whyld. *The Oxford companion to chess*. Oxford University Press, 1992.

[HW92b] A. Huth and C. Wissel. The simulation of the movement of fish schools. *Journal of Theoretical Biology*, 156(3):365–385, 1992.

[IGHM08] C Igel, T. Glasmachers, and V. Heidrich-Meisner. Shark. *Journal of Machine Learning Research*, 9:993–996, 2008.

[JB05] Yaochu J. and J. Branke. Evolutionary optimization in uncertain environments-a survey. *IEEE Transactions on Evolutionary Computation*, 9(3):303–317, 2005.

[JLM03] A. Jadbabaie, Jie Lin, and A. S. Morse. Coordination of groups of mobile autonomous agents using nearest neighbor rules. *IEEE Transactions on Automatic Control*, 48(6):988–1001, June 2003.

[JM91] C. Z. Janikow and Z. Michalewicz. An Experimental Comparison of Binary and Floating Point Representations in Genetic Algorithms. In R. K. Belew and L. B. Booker, editors, *Proc. of the 4th International Conference on Genetic Algorithms*, pages 151–157. Morgan Kaufmann, 1991.

[Joh02] S. Johnson. *Emergence: The Connected Lives of Ants, Brains, Cities, and Software.* Scribner, 2002.

[KE95] J. Kennedy and R. Eberhart. Particle swarm optimization. In *Proceedings of the IEEE International Conference on Neural Networks, 1995*, volume 4, pages 1942–1948 vol.4, 1995.

[Kel96] K. Keller. Socio-technical systems and self-organization. *SIGOIS Bull.*, 17(1):6–7, April 1996.

[Kol63] A. N. Kolmogorov. On tables of random numbers. *Sankhyā: The Indian Journal of Statistics, Series A (1961-2002)*, 25(4):pp. 369–376, 1963.

[Koz92] J. R. Koza. *Genetic programming: on the programming of computers by means of natural selection.* MIT Press, Cambridge, MA, USA, 1992.

[Kum03] J. Kummeneje. *RoboCup as a Measure to Research, Education, and Dissemination.* PhD thesis, Stockholm University and the Royal Institute of Technology, Kista, Sweden, 2003.

[LCRP+05] S. Luke, C. Cioffi-Revilla, L. Panait, K. Sullivan, and G. Balan. Mason: A multiagent simulation environment. *Simulation*, 81(7):517–527, July 2005.

[LE09] R. Leidenfrost and W. Elmenreich. Firefly clock synchronization in an 802.15.4 wireless network. *EURASIP Journal on Embedded Systems*, pages 1–17, 2009.

[Leh90] J.-M. Lehn. Perspectives in supramolecular chemistryfrom molecular recognition towards molecular information processing and self-organization. *Angewandte Chemie International Edition in English*, 29(11):1304–1319, 1990.

[LH05] T. Larsen and S.T. Hansen. Evolving composite robot behaviour - a modular architecture. In *Proceedings of the Fifth International Workshop on Robot Motion and Control, 2005. RoMoCo '05*, pages 271 – 276, june 2005.

[LIPS10] M. López-Ibáñez, L. Paquete, and T. Stützle. Exploratory analysis of stochastic local search algorithms in biobjective optimization. In Thomas Bartz-Beielstein, Marco Chiarandini, Luís Paquete, and Mike Preuss, editors, *Experimental Methods for the Analysis of Optimization Algorithms*, pages 209–222. Springer, Berlin, Germany, 2010.

[LP00] H. Lipson and J.B. Pollack. Automatic design and manufacture of robotic lifeforms. *Nature*, 406(6799):974–978, 2000.

[LS11] J. Lehman and K. O. Stanley. Abandoning objectives: Evolution through the search for novelty alone. *Evolutionary Computation*, 19(2):189–223, June 2011.

[Mah95] S. W. Mahfoud. *Niching Methods for Genetic Algorithms*. PhD thesis, University of Illinois at Urbana-Champaign, 1995.

[Mea55] G. H. Mealy. A Method for Synthesizing Sequential Circuits. *Bell System Technical Journal*, 34(5):1045–1079, 1955.

[Mef12] K. Meffert. JGAP - Java genetic algorithms and genetic programming package, 2012.

[Mil04] J. F. Miller. Evolving a self-repairing, self-regulating, french flag organism. In *Proceedings of Genetic and Evolutionary Computation Conference (GECCO 2004)*, pages 129–139, 2004.

[MN09] C.M. Macal and M.J. North. Agent-based modeling and simulation. In *Proceedings of the 2009 Winter Simulation Conference (WSC)*, pages 86–98, 2009.

[MO08] L. Marsh and C. Onof. Stigmergic epistemology, stigmergic cognition. *Cognitive Systems Research*, 9(1-2):136–149, March 2008.

[MS90] R. Mirollo and S. Strogatz. Synchronization of pulse-coupled biological oscillators. *SIAM Journal on Applied Mathematics*, 50(6):1645–1662, 1990.

[MS04] C. Muller-Schloer. Organic computing - on the feasibility of controlled emergence. In *Proceedings of the International Conference on Hardware/Software Codesign and System Synthesis, 2004. CODES + ISSS 2004.*, pages 2–5, 2004.

[MSC10] C. Moeslinger, T. Schmickl, and K. Crailsheim. Emergent flocking with low-end swarm robots. In M. et al. Dorigo, editor, *ANTS'10: Proceedings of the 7th international conference on Swarm intelligence*, volume 6234 of *Lecture Notes in Computer Science*, pages 424–431. Springer, 2010.

[NBD09] A. L. Nelson, G. J. Barlow, and L. Doitsidis. Fitness functions in evolutionary robotics: A survey and analysis. *Robotics and Autonomous Systems*, 57(4):345–370, April 2009.

[NF98] S. Nolfi and D. Floreano. Coevolving predator and prey robots: Do "arms races" arise in artificial evolution? *Artif. Life*, 4(4):311–335, October 1998.

[NF00] S. Nolfi and D. Floreano. *Evolutionary Robotics: The Biology, Intelligence, and Technology of Self-Organizing Machines.* MIT Press/Bradford Books, Cambridge, MA, 2000.

[NGH04] A.L. Nelson, E. Grant, and T.C. Henderson. Evolution of neural controllers for competitive game playing with teams of mobile robots. *Robotics and Autonomous Systems*, 46(3):135 – 150, 2004.

[Nov03] G. Novak. Roboter soccer: An example for autonomous mobile cooperating robots. In *Proceedings of the First Workshop on Intelligent Solutions for Embedded Systems (WISES'03)*, pages 107–118, Vienna, Austria, 2003.

[NP77] G. Nicolis and I. Prigogine. *Self-Organization in Non-Equilibrium Systems.* Wiley, New York, May 1977.

[Ort94] P.J. Ortoleva. *Geochemical self-organization.* Oxford monographs on geology and geophysics. Oxford University Press, 1994.

[Pag89] H. R. Pagels. *The Dreams of Reason: The Computer and the Rise of the Sciences of Complexity*. Bantam, June 1989.

[PB05] C. Prehofer and C. Bettstetter. Self-organization in communication networks: principles and design paradigms. *Communications Magazine, IEEE*, 43(7):78–85, 2005.

[PB07] R. Pfeifer and J. Bongard. *How the body shapes the way we think: a new view of intelligence*. MIT Press/Bradford Books, Cambridge, MA, 2007.

[PBSE12] Á. Pintér-Bartha, A. Sobe, and W. Elmenreich. Towards the light – Comparing evolved neural network controllers and finite state machine controllers. In *Proceedings of the Tenth International Workshop on Intelligent Solutions in Embedded Systems*, pages 83–87, Klagenfurt, Austria, jul 2012.

[Pet96] A. Petrowski. A clearing procedure as a niching method for genetic algorithms. In *Proceedings of the IEEE International Conference on Evolutionary Computation, 1996*, pages 798–803, 1996.

[Pet02] P Peti. The concepts behind time, state, component, and interface - a literature survey. Research Report 53/2002, Technische Universität Wien, Institut für Technische Informatik, Treitlstr. 1-3/182-1, 1040 Vienna, Austria, 2002.

[PK00] A. Papagelis and D. Kalles. GA tree: genetically evolved decision trees. In *Proceedings of the 12th IEEE International Conference on Tools with Artificial Intelligence, 2000. ICTAI 2000*, pages 203 –206, 2000.

[PRT+08] H. Prothmann, F. Rochner, S. Tomforde, J. Branke, C. Müller-Schloer, and H. Schmeck. Organic control of traffic lights. In C. Rong, M. Jaatun, F. Sandnes, L.T. Yang, and J. Ma, editors, *Autonomic and Trusted Computing*, volume 5060 of *Lecture Notes in Computer Science*, pages 219–233. Springer Berlin Heidelberg, 2008.

[PTO+12] C. Pinciroli, V. Trianni, R. O'Grady, G. Pini, A. Brutschy, M. Brambilla, N. Mathews, E. Ferrante, G. Di Caro, F. Ducatelle, M. Birattari, L. M. Gambardella, and M. Dorigo. ARGoS: A modular, parallel, multi-engine simulator for multi-robot systems. *Swarm Intelligence*, 6(4):271–295, 2012.

[QSMH02] M. Quinn, L. Smith, G. Mayley, and P. Husb. Evolving teamwork and role allocation with real robots. In *Proceedings of the 8th International Conference on Artificial Life*, pages 302–311. MIT Press, 2002.

[Rec94] I. Rechenberg. *Evolutionsstrategie '94*. Frommann-Holzboog, Stuttgart, 1994.

[Res97] M. Resnick. *Turtles, Termites, and Traffic Jams: Explorations in Massively Parallel Microworlds (Complex Adaptive Systems)*. The MIT Press, 1997.

[Rey87] C. W. Reynolds. Flocks, herds and schools: A distributed behavioral model. In *Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '87, pages 25–34, New York, NY, USA, 1987. ACM.

[RMB+06] U. Richter, M. Mnif, J. Branke, C. Müller-Schloer, and H. Schmeck. Towards a generic observer/controller architecture for organic computing. In Christian Hochberger and Rüdiger Liskowsky, editors, *INFORMATIK 2006? Informatik für Menschen!*, volume P-93 of *LNI*, pages 112–119. Bonner Köllen Verlag, Oktober 2006.

[RPS12] W. Renz, T. Preisler, and J. Sudeikat. Mesoscopic stochastic models for validating self-organizing multi-agent systems. In *Proceedings of the 1st International Workshop on Evaluation for Self-Adaptive and Self-Organizing Systems*, Lyon, France, September 2012.

[RSZF07] J. Roberts, T. Stirling, J. Zufferey, and D. Floreano. Quadrotor using minimal sensing for autonomous indoor flight. In *European Micro Air Vehicle Conference and Flight Competition (EMAV'07)*, 2007.

[RW08] B. Rinner and W. Wolf. An introduction to distributed smart cameras. *Proceedings of the IEEE*, 96(10):1565–1575, 2008.

[SB08] E. Sapin and L. Bull. Searching for glider guns in cellular automata: Exploring evolutionary and other techniques. In N. Monmarch, E.-G. Talbi, P. Collet, M. Schoenauer, and E. Lutton, editors, *Artificial Evolution*, volume 4926 of *Lecture Notes in Computer Science*, pages 255–265. Springer Berlin Heidelberg, 2008.

[SBP+09] J. Sudeikat, L. Braubach, A. Pokahr, W. Renz, and W. Lamersdorf. Systematically engineering self-organizing systems: The sodekovs approach. *ECEASST*, 17:1–12, 2009.

[SBW+10] T. Schaul, J. Bayer, D. Wierstra, Y. Sun, M. Felder, F. Sehnke, T. Rückstieß, and J. Schmidhuber. PyBrain. *Journal of Machine Learning Research*, 11:743–746, 2010.

[SFE12] A. Sobe, I. Fehérvári, and W. Elmenreich. FREVO: A tool for evolving and evaluating self-organizing systems. In *Proceedings of the Sixth IEEE International Conference on Self-Adaptive and Self-Organizing Systems Workshops (SASOW), 2012*, pages 105–110, 2012.

[SG10] W. M. Spears and D. F. Gordon. Evolving finite-state machine strategies for protecting resources. In *Foundations of Intelligent Systems*, volume 1932 of *Lecture Notes in Computer Science*, pages 166–175. Springer Berlin Heidelberg, 2010.

[Sha48] C. E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27:379–423, 623–656, 1948.

[SM02] K. O. Stanley and R. Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary Computation*, 10(2):99–127, 2002.

[SMS05] T. Schöler and C. Müller-Schloer. An observer/controller architecture for adaptive reconfigurable stacks. In Michael Beigl and Paul Lukowicz, editors, *Systems Aspects in Organic and Pervasive Computing - ARCS 2005*, volume 3432 of *Lecture Notes in Computer Science*, pages 139–153. Springer Berlin Heidelberg, 2005.

[SP97] R. Storn and K. Price. Differential evolution a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11(4):341–359, 1997.

[SR08] J. Sudeikat and W. Renz. Toward systemic MAS development: Enforcing decentralized selforganization by composition and refinement of archetype dynamics. In D. Weyns, S. A. Brueckner, and Y. Demazeau, editors, *Engineering Environment-Mediated Multi-Agent Systems*, volume 5049 of *Lecture Notes in Computer Science*, pages 39–57. Springer Berlin Heidelberg, 2008.

[SS01] U. Schmid and K. Schossmaier. How to reconcile fault-tolerant interval intersection with the lipschitz condition. *Distributed Computing*, 14(2):101–111, 2001.

[Sta08] K. Stanley. Novelty search C++ - implemented as an extension to ken stanley's rtNEAT implementation., 2008.

[SVML03] G. Simon, P. Volgyesi, M. Maroti, and A. Ledeczi. Simulation-based optimization of communication protocols for large-scale wireless sensor networks. In *Aerospace Conference, 2003. Proceedings. 2003 IEEE*, volume 3, pages 1339–1346, 2003.

[Swi96] K. Swingler. *Applying Neural Networks: A Practical Guide*. Academic Press, 1996.

[Sym08] J. Symons. Computational models of emergent properties. *Minds and Machines*, 18(4):475–491, 2008.

[TAB07] A. Tyrrell, G. Auer, and C. Bettstetter. Biologically inspired synchronization for wireless networks. In F. Dressler and I. Carreras, editors, *Advances in Biologically Inspired Information Systems*, volume 69 of *Studies in Computational Intelligence*, pages 47–62. Springer Berlin Heidelberg, 2007.

[TcGc08] A. E. Turgut, H. Çelikkanat, F. Gökçe, and E. Şahin. Self-organized flocking in mobile robot swarms. *Swarm Intelligence*, 2(2-4):97–120, 2008.

[TN11] V. Trianni and S. Nolfi. Engineering the evolution of self-organizing behaviors in swarm robotics: A case study. *Artif. Life*, 17(3):183–202, August 2011.

[TR02] D. L. Turcotte and J. B. Rundle. Self-organized complexity in the physical, biological, and social sciences. *Proceedings of the National Academy of Sciences of the United States of America*, 99(Suppl 1):2463–2465, 2002.

[Tri08] V. Trianni. *Evolutionary Swarm Robotics - Evolving Self-Organising Behaviours in Groups of Autonomous Robots*, volume 108 of *Studies in Computational Intelligence*. Springer, 2008.

[TSHMS09] S. Tomforde, M. Steffen, J. Hähner, and C. Müller-Schloer. Towards an organic network control system. In Juan Gonzlez Nieto, Wolfgang Reif, Guojun Wang, and Jadwiga Indulska, editors, *Autonomic and Trusted Computing*, volume 5586 of *Lecture Notes in Computer Science*, pages 2–16. Springer Berlin Heidelberg, 2009.

[Tur52] A. Turing. The chemical basis of morphogenesis. *Philosophical Transactions of the Royal Society B*, 237:37–72, 1952.

[UFK97] Y. Uny Cao, A. S. Fukunaga, and A. B. Khang. Cooperative mobile robots: Antecedents and directions. *Autonomous Robots*, 4:1–23, 1997.

[Ves12] A. Vespignani. Modelling dynamical processes in complex socio-technical systems. *Nature Physics*, 8:3239, 2012.

[vF60] H. von Foerster. On self-organizing systems and their environments. *Self-Organizing Systems*, pages 31–50, 1960.

[VZ12] T. Vicsek and A. Zafeiris. Collective motion. *Physics Reports*, 2012.

[WATP+05] G. Werner-Allen, G. Tewari, A. Patel, M. Welsh, and R. Nagpal. Firefly-inspired sensor network synchronicity with realistic radio effects. In *Proceedings of The Third International Conference on Embedded Networked Sensor Systems*, pages 142–153, 2005.

[Wei10] E. W. Weisstein. Elementary cellular automaton. MathWorld - A Wolfram Web Resource: `http://mathworld.wolfram.com/ElementaryCellularAutomaton.html`, January 2010.

[WH07] T. Wolf and T. Holvoet. Design patterns for decentralised coordination in self-organising emergent systems. In S.A. Brueckner, S. Hassas, M. Jelasity, and D. Yamins, editors, *Engineering Self-Organising Systems*, volume 4335 of *Lecture Notes in Computer Science*, pages 28–49. Springer Berlin Heidelberg, 2007.

[Win84] A. T. Winfree. The prehistory of the Belousov-Zhabotinsky oscillator. *Journal of Chemical Education*, 61(8):661, 1984.

[WKF+10] D. Wang, N.-M. Kwok, G. Fang, Xiuping Jia, and F. Li. Ants based control of swarm robots for bushfire fighting. In *Proceedings of the International Conference on Artificial Intelligence and Computational Intelligence (AICI), 2010*, volume 1, pages 528–532, 2010.

[WM97] D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *Evolutionary Computation, IEEE Transactions on*, 1(1):67–82, 1997.

[Wol69] L. Wolpert. Positional information and the spatial pattern of cellular differentiation. *Journal of Theoretical Biology*, 25:1–47, 1969.

[WT11] J. L. Wilkerson and D. R. Tauritz. A guide for fitness function design. In Natalio Krasnogor and Pier Luca Lanzi, editors, *GECCO (Companion)*, pages 123–124. ACM, 2011.

[Yan08] X.-S. Yang. *Nature-Inspired Metaheuristic Algorithms*. Luniver Press, 2008.

[YD09] X.-S. Yang and S. Deb. Cuckoo search via lvy flights. In *World Congress on Nature & Biologically Inspired Computing, NaBIC 2009, 9-11 December 2009, Coimbatore, India*, pages 210–214. IEEE, 2009.

[YW13] Honghai Y. and S. Winkler. Image complexity and spatial information. In *Quality of Multimedia Experience (QoMEX), 2013 Fifth International Workshop on*, pages 12–17, 2013.

[ZTL+03] E. Zitzler, L. Thiele, M. Laumanns, C. M. Fonseca, and V. G. da Fonseca. Performance assessment of multiobjective optimizers: An analysis and review. *IEEE Transactions on Evolutionary Computation*, 7(2):117–132, April 2003.

List of Own Publications

Journal Publications

1. I. Fehérvári and W. Elmenreich. Evolving neural network controllers for a team of self-organizing robots. *Journal of Robotics*, 2010.

2. W. Elmenreich, T. Ibounig, and I. Fehérvári. Robustness versus performance in sorting and tournament algorithms. *Acta Polytechnica Hungarica*, 6(5):7, 2009.

Conference Publications

3. I. Fehérvári, V. Trianni and W. Elmenreich: On the Effects of the Robot Configuration on Evolving Coordinated Motion Behaviors. In Proceedings of the *IEEE Congress of Evolutionary Computation 2013 (CEC'13)*, Cancun, Mexico, Jun 20-23, 2013

4. I. Fehérvári, A. Sobe and W. Elmenreich: Biologically Sound Neural Networks for Embedded Systems Using OpenCL. In *Proceedings of the International Conference on Networked Systems (NETYS'13)*, Marrakech, Morocco, May 2-4, 2013

5. I. Fehérvári, W. Elmenreich: Evolution as a tool to design self-organizing systems. In *Proceedings of the 7th International Workshop on Self-Organizing Systems (IWSOS'13)*, Palma de Mallorca, Spain, 9-10th of May 2013

6. A. Sobe, I. Fehérvári and W. Elmenreich. FREVO: A tool for evolving and evaluating self-organizing systems. In *Proceedings of the IEEE Self-adaptive and Self-organizing Systems Workshop Eval4SASO*, Lyon, France, September 2012

7. I. Fehérvári, W. Elmenreich, and E. Yanmaz. Evolving a team of self-organizing UAVs to address spatial coverage problems. In R. M. Bichler,

S. Blachfellner, and W. Hofkirchner, editors, *European Meeting on Cybernetics and Systems Research Book of Abstracts*, pages 201-204, Vienna, Austria, April 2012

8. B. Lénárt, I. Fehérvári: Using an Adaptive Neuro-Fuzzy Inference System for Adaptive Inventory Control. In *Proceedings of the International Conference on Innovative Technologies (IN-TECH)*, 2011

9. W. Elmenreich, I. Fehérvári: Evolving self-organizing cellular automata based on neural network genotypes. In *Proceedings of the Fifth International Workshop on Self-Organizing Systems*, volume LNCS 6557, pages 16. Springer Verlag, 2011

10. I. Fehérvári, W. Elmenreich: Evolutionary Methods in Self-organizing System Design in *Proceedings of the 2009 International Conference on Genetic and Evolutionary Methods (GEM'09)*, part of World Congress in Computer Science, Computer Engineering, and Applied Computing (WORLDCOMP'09), Las Vegas, NV, USA

11. I. Fehérvári, W. Elmenreich, Towards Evolving Cooperative Behavior With Neural Controllers, in: *IFIP Fourth International Workshop on Self-Organizing Systems*, 2009

12. I. Fehérvári, W. Elmenreich: Design of Self-organizing Systems Using Evolutionary Methods. In: H. Kaiser, R. Kirner (Hrsg.): *Proceedings of the Junior Scientist Conference 2008*. Wien: Technische Universität Wien, 2008, pp. 53-54.

Poster Publications

13. W. Masood, J. Klinglmayr, I. Fehérvári, T. Watzl and Christian Bettstetter: Synchronization using Inhibitory and Excitatory Coupling: From Theory to Practice at *The 32nd IEEE International Conference on Computer Communications (INFOCOM2013)*, Turin, Italy, 2013

14. O. Maurhart, W. Elmenreich, I. Fehérvári and A. Bouchachia: Evaluation of Robustness and Performance of Environmental Influences on Evolutionary Algorithms compared to Ant Colony Systems, at the *European Conference on Complex Systems (ECCS'11)*, Vienna, Austria, 2011

15. I. Fehérvári, W. Elmenreich, O. Maurhart: FREVO: Framework for Evolutionary Design, at the *Fifth Internation Workshop on Self-Organizing Systems (IWSOS'11)*, Karlsruhe, Germany, 2011