Evolving Biologically Inspired Classifiers

Rinde R.S. van Lon
Man-Machine Interaction Group
Delft University of Technology

August 13, 2010

Evolving Biologically Inspired Classifiers

THESIS

submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER SCIENCE

by

Rinde R.S. van Lon
born in Delft, the Netherlands

Man-Machine Interaction Group
Department of Mediamatics
Faculty EEMCS, Delft University of Technology
Delft, the Netherlands
www.ewi.tudelft.nl

Evolving Biologically Inspired Classifiers

Author: Rinde R.S. van Lon
Student id: 1238434
Email: rindevanlon@gmail.com

Abstract

This thesis argues that natural complex systems can provide an inspiring example for creating software which incorporates emergent, self-organizing and adaptive properties. The advantages of complex systems are their natural resilience, redundancy and adaptivity. A generalization of neural networks and boolean networks called computational networks is presented as a model for complex systems. It is argued that this model satisfies the required properties for modeling complex systems. Furthermore, it is asserted that a computational network, being a network of mathematical functions, is appropriate for solving classification problems. For the design of computational networks an evolutionary design algorithm is constructed. Additionally, four extensions of this algorithm are presented. Each extension is inspired by natural evolution and theories from the evolutionary computing literature. An important component is a novel generative representation which can reuse substructures of computational networks. Experiments with this component have shown that it facilitates a higher level of complexity in the solution space, improving the computational network performance for more complex problems. Other components steer the evolutionary process towards a desired solution, either by introducing special stages during evolution, or by smoothing the fitness landscape. The experiments show that complex systems can be evolutionary designed to act as a classifier. The resulting computational network has a better performance on the Iris dataset compared to every classifier in the Weka classifier collection. Furthermore, an experiment was conducted using the TIMIT read speech dataset, the classifier was evolutionary designed using only 13 MFCC features, and a very small train set. Although the performance is not good enough to be of any practical use, the results are adequate given the limitations of the train data.

Thesis Committee:

Chair:	Prof. drs. dr. Léon J.M. Rothkrantz
University supervisor:	Dr. ir. Pascal Wiggers
Committee Member:	Dr. Tomas B. Klos

Preface

Throughout my bachelor at Delft University of Technology, I was already fascinated by the topics of this thesis. During my master I decided to learn more about them by following courses on evolutionary computing and self-organizing systems at Vrije Universiteit Amsterdam. These courses ignited my passion for these topics, and led to the idea of writing my thesis about evolution and biological inspired systems.

The particular idea of using biological inspired systems for automatic speech recognition came to me in different forms. I remember one of those 'visions' particularly well. One morning during a two day hiking trip in the Grand Canyon, Arizona, USA, I woke up envisioning ant-like computer programs cooperating as a collective to classify incoming audio signals. This simple initial idea, and eight months of hard work led to the thesis you are reading now. However, I'm very aware that without the help and support of my friends and relatives, I would never have come so far. I'd like to thank all people who supported me over the years. Not only during the writing of this thesis, but also during the completion of my master and bachelor years, which didn't always go easy.

First and foremost I would like to thank my thesis supervisor, Pascal, for the freedom of letting me pursue my intuitions and dreams. Your suggestions and encouragements really helped me to continue on the work at hand and steered my creativity in the right direction. I especially would like to thank you for expressing your trust at moments when I didn't see things clearly.

Furthermore I would like to thank Léon Rothkrantz for reading a draft version of this thesis and providing me with his honest comments, it really helped me to take it to the next level. Also I would like to thank Ruud and Bart for their efforts to keep the servers running while I was trying to exhaust them. And to all the people in the lab, thanks for the tips, help and coffee breaks!

On a more personal note, I would like to thank Jan Baltissen for his patience and time for explaining me all facets of algebra during so many nights in my first few bachelor years. And my friend Paul, I really enjoyed our cooperation during the master, I think we are a great team. My parents for their unconditional support and belief over the years. And last but certainly not least, I would like to thank Anouk for her patience, motivational talks, and compassion at moments I needed it the most.

Thank you.

Rinde R.S. van Lon
Delft, the Netherlands
August 13, 2010

Contents

List of Figures

List of Tables

1. Introduction

From the North Pole to the Sahara, from the deepest see floors to the highest mountain peaks, you cannot lift a rock without finding life underneath it. In fact the entire planet is so full of life that biologists even find it hard to agree on an estimate of the total number of living organisms. However, the exact number of living organisms isn't required to understand that these organisms are good at something, that is, they are good at living. In order to keep living, an organism has to survive, it needs enough time to produce offspring which are strong enough to survive on their own. The art of surviving can more specifically be described as the ability of adapting to the environment, but how does an organism adapt?

In nature many different adaptation strategies exist, the focus of this thesis lies on so called *complex systems*. Examples of complex systems are as diverse as ant colonies, immune systems, neurons or a market economy. A common feature of these systems is that they are comprised of autonomous interacting entities, it is hard to pinpoint the location of the adaptivity in such a system. Yet, their tremendous power is apparent when looking at them from an engineering point of view. An ant colony is very efficient at food foraging and also remarkably adaptive to changes in the local environment. In spite of their beneficial properties, these systems are still not fully understood:

> "*Economies, ecologies, immune systems, developing embryos, and the brain all exhibit complexities that block broadly based attempts at comprehension.* "
>
> — Holland, 1992

When looking at present day software, or computers in general, it is clear that they do not posses the adaptive properties which are so distinctive for natural systems. For example, if you flip a bit in the working memory of a running computer, it is likely that depending on the memory location of the bit flip, the system would behave incorrect or even crash completely. However, killing an ant would not significantly hinder an ant colony. Or consider a forest which has suffered from a forest-fire, within a few years it restores itself as though there never was fire.

Since the current prevailing software development paradigm produces such rigid systems, there is clearly a need for a new approach for developing adaptive software, a belief which is widely held according to Salehie and Tahvildari (2009). Since complex systems seem to contain this much needed adaptivity, they appear to be in a good position of becoming the center of a new paradigm for developing software. So why don't we create our own complex systems? One of the defining properties of complex systems is at the same time the biggest complication for designing one. It is the property of *emergence*, adaptivity emerges from the interacting entities of a complex system, you can't put adaptivity in, or remove it from the system, it is an inherent property. This makes designing such a system mystifying, and raises the following question; how did natural complex systems emerge?

The answer to this question comes from Charles Darwin, who formulated the theory of evolution through natural selection. Evolution is the continuous adaptation of a species to the environment, in short, it can be said that a property of a species doesn't exist if there isn't a demand for it from its environment. A trait is a solution to a problem which gives it a better fitness. This means that the environment, that is, our planet, somehow demands adaptivity from most species in order to survive. So what can be learned from these natural processes? Can we perhaps simulate them to achieve similar effects?

Luckily, a field of computer science, called evolutionary computing, already addresses a part of this question. Evolutionary computing is commonly seen as yet another optimization technique, why would evolution be the best optimization technique for this purpose? Important is that complex systems do not merely need to be optimized, it is not a problem of finding a few optimal parameter values, the problem is in finding an optimal *structure*, they need to be *designed*. Most optimization techniques have

no capabilities to introduce new structures in a solution, natural evolution does have this ability as can be seen by observing the variety in the world. As such it makes more sense to see evolution in the context of complex systems as a design tool, *evolution is the design algorithm of nature*. Since the process of evolution needs to be designed itself as well, the problem of creating a complex system is translated to the evolutionary domain. The evolutionary design perspective shifts the focus of attention from the solution towards the problem, how can the problem be represented such that it improves the evolutionary process. An advantage of this focus on the problem, is that in contrary to a solution, the problem already exists, and can as such be observed and analyzed. Instead of designing a complex system directly, a function needs to be designed which adequately measures the quality of any particular design. This function can than be used in the feedback loop of evolution, which rewards designs of higher quality, producing a new generation of better designs.

Many problems in the world are classification problems, in fact, humans are classifying things, sounds, objects, scents and feelings all the time. For example, when you see a four legged animal and you recognize it as a horse, or when you smell that your vegetables are burning in your pan. Most of these 'problems' are solved so naturally by humans, that it is easy to forget that in fact some of them are quite hard. That is, computers have not been able to solve some of these problems. The reason that some classification problems have not been solved until now is that some of them have a very complex nature. Here, evolutionary designed complex systems can play a role, complex systems are known to hold big complexities themselves and seem in principle to be able to solve a complex problem. Examples of classification problems where computers have legged behind are, facial recognition, emotion recognition, visual object recognition and speech recognition. Humans naturally possess these capabilities, what is remarkable of these problems is that they are perceived as being defining characteristics of humans, we identify ourselves with these capabilities. It is precisely these kind of classification problems for which complex systems seem to be most suited.

A feature that sets humans apart from most other species on earth is our capability of speech recognition. Humans can adapt their hearing to voices, can filter out background noise and can even focus on one voice when several people are talking at the same time, again *adaptation* is the key here. If you consider the effortless way in which humans can recognize speech, it is almost overlooked how difficult the problem actually is. For more than 40 years automatic speech recognition has been researched, although the field has known its successes, there is still a wide open gap for improvement. Consider for example the word error rate in conversational speech, for humans this rate lies around 4%, state of the art speech recognition systems come no further than an error rate of 20 to 40% (Wiggers, 2008).

The dominant approach in automatic speech recognition (ASR) is the so called statistical approach. Most research in speech recognition and related fields currently employ a statistical approach to the problem. One of the shortcomings of this statistical method is adaptation, an observation nicely summarized in the following quote:

> "*Adapting to changing environments remains the major obstacle to speech recognition in practical applications. Investigating innovative strategies has become essential to overcome the drawbacks of classical approaches.* „
>
> — Selouani and O'Shaughnessy, 2003

A common approach is to use probabilities of word occurrences to predict the next word. These probabilities are computed using large texts, by counting so-called n-grams. An example of an n-gram is a trigram, this counts the occurrences of three adjacent words. As can be expected, the probabilities are more accurate when bigger datasets are used. That's why over the years thanks to the increasing computing power, bigger datasets are used, several years ago Google even published a 1 trillion dataset[1].

The ever-increasing amount of available computing power together with the corresponding slightly improving speech recognition performance suggests that we can just wait until speech recognition eventually achieves a near human accurateness. From a scientific point of view this is not very satisfying since it would be a kind of brute force solution, while in computer science we attempt to find an *efficient* solution. Since humans seem to employ a different strategy which does not use any statistics or huge datasets, trying to mimic the human brain might be a viable alternative. This is what is done with evolving complex systems, the brain is a complex system itself, which has evolved over millions of years. To solve a speech recognition dataset using a complex system the evolution of (a part of) the human

[1] `http://googleresearch.blogspot.com/2006/08/all-our-n-gram-are-belong-to-you.html`

brain is simulated. It is assumed that human speech perception has been slowly evolving from a simple way to hear 'noises' to increasingly more complexity. Just as Torresen (1998), we expect that several measures for increasing the complexity of a complex system will be beneficial for solving the speech recognition problem.

In this thesis, evolutionary design of complex systems is proposed as a new method for approaching complex classification problems such as speech recognition. We perform speech recognition using our brain, which itself is a complex system. This suggests that a complex system is able to perform speech recognition, interesting is whether artificial evolution is capable of designing an artificial complex system which mimics the speech perception part of the brain. The problem with evolution is that it is a slow process, it is a black box which doesn't define any explicit knowledge. However, if evolution takes a long time to design a solution for a difficult problem, this isn't necessarily a problem. For example, if artificial evolution would take several months to successfully design a speech recognizer, it would still be significantly faster than all the human effort that has been put into it in the last 40 years. So this asks for a different perspective on computation time when considering evolutionary design. However, it is still very desirable to speed up and steer the evolutionary process.

In the next sections the objectives of this thesis are formalized in a way such that they can be objectively verified.

1.1 Research objectives

The main research objectives for this thesis are the following:

- Investigate natural complex systems.
- Research how complex systems can be designed using evolution.
- Create an evolutionary algorithm that can automatically construct classifiers.
- Test the algorithm on several problems including the problem of automatic speech recognition.

1.2 Research questions

To achieve the previously stated objectives, the following research questions will be investigated:

Q_1 What are the strengths and weaknesses of natural complex systems?

Q_2 Can complex systems model a classifier?

Q_3 Can artificial evolution be used to design a complex system classifier?

Q_4 Can artificial evolution be improved for the design of complex systems?

1.3 Hypotheses

To answer the research questions, several verifiable hypotheses are stated below:

H_1 Complex systems have properties that are desirable in computing.

H_2 A complex system can be modeled as a classifier.

H_3 Artificial evolution is a promising method for the design of complex systems.

H_4 The evolutionary design of complex systems can be improved.

$H_{4.1}$ Embryogenies enhance the evolutionary design of complex systems.

$H_{4.2}$ Boosting enhances the evolutionary design of complex systems.

$H_{4.3}$ Distance metrics enhance the evolutionary design of complex systems.

$H_{4.4}$ Increasing complexity enhances the evolutionary design of complex systems.

The hypothesis numbering corresponds to the numbers of the research questions in section 1.2.

1.4 Thesis organization

In chapter 2 a deeper understanding of complex systems is gained, it is argued that the best way to design a complex system is by use of artificial evolution. Chapter 3 introduces artificial evolution and describes several approaches for improving evolutionary design which are relevant for this thesis. In chapter 4 the theory presented in the previous chapters is transformed into a model for complex systems and an evolutionary algorithm. A more detailed description, including some implementation details, is discussed in chapter 5. To test the model and algorithm, the experimental design is presented in chapter 6, also the problem domains on which the tests are conducted are presented. Chapter 7 lists the raw results of the experiments, and chapter 8 conducts a detailed analysis of these results. Conclusions with regard to the research objectives and research questions are drawn in chapter 9 and recommendation for future work are presented in chapter 10.

Part I

Theory

2. Complex Systems

The process of adaptation is a continuous universal process, it happens on every part on and below Earth's surface, on a microscopic and macroscopic level, it is ubiquitous. Nothing escapes from adaptation as the entire universe is constantly changing. These adaptive processes can be called 'systems', and because of their complex nature the term *complex system* is chosen to denote them. For example, a (human) body, brain, weather system and economy can all be characterized as a complex system. Complex systems are of interest since they are powerful, adaptive and can as such deal with rapidly changing environments.

In order to design and use complex systems as classifiers, a deeper level of understanding is required, and their properties need to be known. For this reason several examples of natural complex systems are studied and related disciplines are discussed. Each discipline describes complex systems from a different perspective, these differences of opinion are useful for a full comprehension of the topic. Based on this theory we come to design guidelines for the creation of complex systems at the end of this chapter.

Section 2.1 describes examples of several natural complex systems, from these descriptions several common mechanisms already emerge. This is followed, in section 2.2, by a discussion of several research fields that are involved with complex systems. Based on these different research fields and the properties from natural complex systems a more precise definition of a complex system is presented in section 2.3. This definition is explained in detail in sections 2.4 and 2.5 about emergence and self-organization. This is followed by explaining the relation with adaptivity in section 2.6. In section 2.7 several methods for designing a complex system are presented. Also the most important design choices that have to be made are listed, and for each choice a recommended option is discussed. Finally in section 2.8 this chapter is concluded by discussing the first research hypothesis.

2.1 Natural complex systems

This section presents the inspiration for complex systems by discussing several examples from nature. The characteristic properties of these phenomena are discussed, which form the basis for a more precise definition of complex systems in section 2.3. The examples discussed here are not all the complex systems currently known, it is merely a representative subset.

2.1.1 Social insects and animal societies

Social insects such as ants and bees have developed sophisticated social structures which facilitate task division, communication, cooperation, etc. Remarkable is that individual ants have very limited cognitive capabilities but are, as a collective, capable of solving complex problems.

For example, ants in ant colonies are able to find the shortest path from a food source to their nest. Ants secrete a pheromone trail that is recognizable by other ants and which positively biases them to follow that trail. The stronger the pheromone trail the more likely it is for other ants to follow that trail. The collective behavior that emerges is a form of autocatalytic behavior, i.e. positive feedback, a reinforcing process that causes very rapid convergence (Dorigo et al., 1991).

The secreting and sensing of the pheromone trails is essentially a form of indirect communication, called stigmergy (Di Marzo Serugendo, 2003), which results in cooperation in the swarm. It is this interaction that gives rise to an organized society which is capable of exploring, finding food, and so on, without any individual alone knowing how to perform such a feat.

Another interesting property of ants is that from the colony perspective, they can be very adaptive to their environment. Experiments have shown that a colony forages efficiently even when the food sources change over time (Schut, 2007). However, the individual ants do not change their behavior (they do not

Swarming Behavior	Entities
Pattern Generation	Bacteria, Slime Mold
Path Formation	Ants
Nest Sorting	Ants
Cooperative Transport	Ants
Food Source Selection	Ants, Bees
Thermoregulation	Bees
Task Allocation	Wasps
Hive Construction	Bees, Wasps, Hornets, Termites
Synchronization	Fireflies
Feeding Aggregation	Bark Beetles
Web Construction	Spiders
Schooling	Fish
Flocking	Birds
Prey Surrounding	Wolves

Table 2.1: Some Examples of Swarming in Nature (adapted from Parunak and Brueckner (2004))

adapt individually) when they are for example foraging food. The adaptivity on the colony level is an emergent property of the interactions of the ants on a local level. These discoveries of ant behavior have lead to the development of the field of ant colony optimization (ACO) which started with Marco Dorigo and others (Dorigo and Socha, 2006).

Parunak and Brueckner (2004) have compiled an impressive list of examples of swarming behavior in nature, this list is shown in Table 2.1.

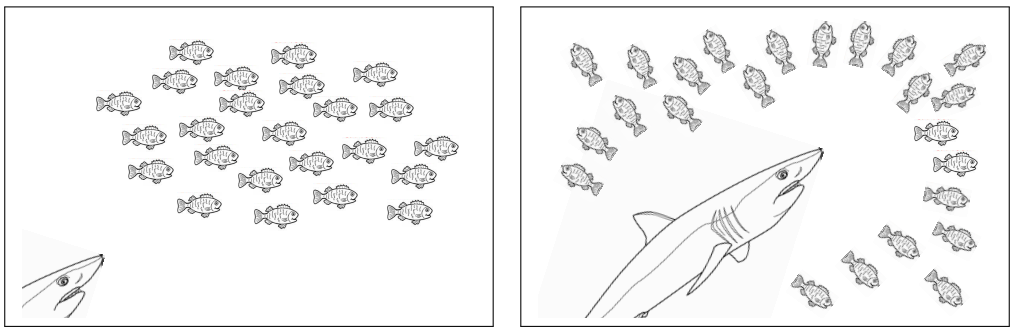
2.1.2 Ecology

Homer-Dixon (2009) states that the discipline that most effectively captures the core ideas of complex adaptive systems theory is ecology. Ecological systems deal with interacting populations of (different) species, examples are populations of predators and prey or a group of bacteria inside an organism. He states that ecological systems are characterized by 'causal openness', and that they tend to exhibit 'emergent properties', meaning that unexpected things happen even though the system's components are well known. Small events sometimes cause big changes, while big events sometimes cause hardly any change whatsoever. He gives the example of a clean pond that seemingly becomes clogged with algae in just one night. This capacity to flip suddenly from stability to turbulence, or from one equilibrium state to another is also referred to as 'non-linear behavior'.

2.1.3 Biology

The human immune system defends the body from invaders (antigens) such as bacteria or biochemicals, it is an example of a complex system (Holland, 1992; Di Marzo Serugendo, 2003). It consists of large numbers of mobile 'individuals', called antibodies. The task of some of these antibodies is to distinguish between 'self' and 'non-self' antigens. Holland states that the number of possible invaders is so large that there is simply not enough room to store all that information in the immune system. The immune system is nevertheless very successful because it can change or adapt its antibodies as new invaders appear. How this process of adaptation precisely works is not currently known, as it is hard to simulate or model. The research field of artificial immune systems (AIS) is actively researching immune systems to use them as learning systems (Bentley, 2002).

Camazine et al. (2003) describe an interesting example of a self organizing collective: schools of fish are capable of coordinating their movement in a cohesive manner as if they are part of a single organism. Each fish employs simple rules to avoid collisions, which can lead to complex coordinated maneuvers, an example maneuver called *flash expansion*, is used to escape from a predator. This is illustrated in Figure 2.1. Similar coordinated movements exists in many more species, such as birds (bird flocks), wildebeest, and flies in a swarm.



(a) normal situation (b) evasive maneuver

Figure 2.1: A fish school (a), showing an evasive maneuver called *flash expansion* (b).

2.1.4 Psychology

In his book 'The Society of Mind' Minsky (1988) presents a theory about the workings of the human mind. He envisions minds as consisting of many smaller processes which he calls agents. An agent is only capable of switching things on and off. However, when these agents combine into societies, Minsky argues that this leads to intelligence. He introduces a distinction between *agents* and *agencies*, where the latter can do all the things agents can do combined. This is analogous to the different levels of perspective of ants discussed in section 2.1.1. Individual ants perform simple actions and an ant colony is capable of complex behavior, similarly the agent level only performs basic operations while the agency level seems to *understand* things. According to his theory, the mind consists of an enormous hierarchy linking agencies together, where each level builds upon the previous. Agencies are groups of cooperating specialists (agents or agencies), which can themselves be part of an even higher level agency.

> "*What magical trick makes us intelligent? The trick is that there is no trick. The power of intelligence stems from our vast diversity, not from any single, perfect principle.* "
> — Minsky, 1988, The Society of Mind

This quote illustrates that Minsky believes that our intelligence is a result of a collective complex system. He furthermore states that to understand mind, all parts need to be understood as well as all interactions between the parts. This knowledge can then be combined to understand what the mind does when observed from the outside. He also states that 'intelligence' is our name for whichever of those mind processes we don't yet understand.

A remarkable property of the brain is its *plasticity*, meaning that functions in the brain can move to a different location. For example, if a person suffers from a cerebro-vascular accident (CVA), or stroke, a relatively small region of the brain (a *lesion*) is damaged irreparably (Best, 1999). Interestingly, the brain is capable of relearning the functions which were present in the lesion in a different location, this illustrates how resilient the brain is.

2.1.5 Physics

In a potentially magnetic material there are tiny magnets, called spins, which have a particular orientation corresponding to the direction of its magnetic field. Generally the spins will point in different directions, canceling each other out, this disorder is caused by the random movements of the molecules in the material. The higher the temperature, the more difficult it is for order to emerge. However, when the temperature decreases, the spins will align themselves, pointing in the same direction. This is because spins with different directions repel each other, and spins with the same direction attract each other. Heylighen states that magnetization is a clear case of self-organization (Heylighen, 1999c).

2.1.6 Economy

The concept of a free market as is commonly used in capitalism, is a perfect example of a collective system. It is a market with a minimum amount of centralized control, it works through a constantly changing

equilibrium between buyers and sellers. Buyers and sellers make decisions based on local information (e.g. their preferences), on a global level the effects of these decisions manifest for example through the stock market or through a surplus or scarcity in a certain good.

About the free market, Heylighen (1999c) states that the failure of the communistic system has shown that the market is much more effective at organizing the economy than a centrally controlled system. However, the recent economic crisis has shown that even the free market isn't perfect and that occasional oscillations can arise. Important about the free market is that there is no 'invisible hand' (Heylighen, 1999c; Kennedy et al., 2001), there is not one person that controls an economy, but it constantly evolves toward and maintains an equilibrium; it is a relatively stable stochastic process, not really predictable but dependable (Kennedy et al., 2001). The (in)stability of an economy can be seen as an emergent property, it follows from the interaction of the economies' participants.

2.2 Background

The phenomena described in the previous section have been studied in many fields, producing many different theories. Most of these theories have been used to analyze or explain the systems as they occur in nature. These theories come from people from different disciplines, resulting in a diverse and interdisciplinary body of knowledge. This section discusses the concepts which are closely related to complex systems, this should pave the way for the next section where a definition of complex systems is proposed.

2.2.1 Complex Adaptive Systems

Complex Adaptive Systems (CAS) is a term originally invented by John Holland (1992). He states that there are many different complex adaptive systems (as was discussed in section 2.1), all of them consist of great numbers of simultaneously interacting parts. They share three characteristics: *evolution*, *aggregate behavior*, and *anticipation*. In CAS, its the parts that evolve in a Darwinian fashion, each trying to survive in their interactions with the surrounding parts. This evolution can be called adaptation or learning. Aggregate behavior is the notion of globally observed behavior which can be said to emerge from the interactions of the parts. Anticipation can be thought of as the parts that are developing rules that anticipate the consequences of certain responses. Because the parts are conditioned in different ways and depend on the interactions between the parts, the consequential anticipation can cause major changes in aggregate behavior. John Holland notes that anticipation is the feature that we least understand but that is one of most importance (Holland, 1992).

The systems' basic components can be seen as sets of rules, they rely on three key mechanisms: *parallelism*, *competition*, and *recombination*. Parallelism allows the system to use individual rules as building blocks, it can activate sets of rules simultaneously. Competition provides the adaptation necessary to operate in realistic environments where the agent receives a stream of (mostly) irrelevant information. Using credit assignment for rules and rule discovery procedures, useful events are recognized from this stream of information and are used as building blocks in new, similar contexts. Recombination facilitates the discovery of new rules from parts of successful tested rules. These procedures give the system its characteristic "evolving structure", it continuously tries to improve it's own performance.

Typically research on CAS focuses more on the analysis of systems as there is currently not a complete mathematical model of it. This is contrary to the focus of this thesis which emphasizes the design and engineering of these systems. Related fields are that of chaos theory and complexity theory, these theories basically say that small local changes can yield big global changes, which results in unpredictable chaotic behavior.

2.2.2 Collective Intelligence

A similar concept is that of Collective Intelligence (CI), sometimes called Emergent Collective Intelligence (Eiben et al., 2005). In his paper about model design of collective intelligence Martijn Schut (2010) identifies two categories of system properties. The *enabling* properties (or the requirements), which indicate that if they are present, it is possible for CI to emerge, and the *defining* properties (or the success parameters, which indicate whether the system was correctly constructed), which if observed indicate that the system is a collective intelligence system.

The enabling properties Schut (2010) identified are listed in table 2.2 and the identifying properties are listed in table 2.3.

adaptivity	Meaning that the entire system is capable of adapting, changing it's structure to fit better in it's environment. This can, but doesn't necessarily have to, imply that an individual is capable of changing itself.
interaction	Interactions between the individuals are of importance, as a CI system cannot effectively be analyzed without them.
rules	Rules are the most suitable way of describing the behavior of an individual.

Table 2.2: Enabling properties of Collective Intelligence as identified by Schut (2010).

global-local	There is a distinction between the local (individual) and global (or aggregate) level of the system.
randomness	Randomness typically causes the system to show self-organized behavior. A system is usually on the so called 'edge of chaos' where on one side there is chaos and disorder, and on the other side there is structure and order. A system is capable of moving over the edge towards a new equilibrium.
emergence	The simple meaning of emergence is "the whole is more than the sum of its parts" (emergence is explained in more detail in section 2.4).
redundancy	The same information in the system is represented in different places.
robustness	The system is robust against malfunctioning.

Table 2.3: Identifying properties of Collective Intelligence as identified by Schut (2010).

A possible definition of collective intelligence is:

> "*Collective intelligence is defined as the ability of a group to solve more problems than its individual members.* "
>
> — Heylighen, 1999a

It is clear that this definition is closely linked to the concept of emergence described above. This definition applies to both artificially constructed and natural systems, however, it can also be viewed as a phenomenon or a perspective. As such it can also better be seen as a discovery instead of an invention.

In order to understand a CI system, a holistic perspective can be convenient. The following quote illustrates holism:

> "*In order to know something completely, it is necessary to know its relations, what it affects and what affects it.* "
>
> — Kennedy et al., 2001

This connects nicely with the definition of Heylighen stated above, in order to understand a group, the system has to be examined as a whole, it cannot be reduced.

2.2.3 Swarm Intelligence

The field of Swarm Intelligence (SI) is inspired by bird flocking, fish schooling, ant colonies, etc. As their natural counter parts, SI systems are comprised of a set of agents following simple rules. A nice definition for swarm intelligence is:

> "*Swarm Intelligence is the emergent collective intelligence of groups of simple agents.* "
>
> — Bonabeau et al. (as cited by Kennedy et al. (2001))

From this definition it is clear that SI is closely related to the fields discussed earlier in this section. Furthermore the usefulness of SI is noted:

> "*Swarm Intelligence provides a useful paradigm for implementing adaptive systems* "
>
> — Kennedy et al., 2001

In their book Kennedy et al. (2001) propose the 'particle swarm' as a useful computational intelligence, also ACO is commonly referred to as a swarm intelligence. SI is very similar to what Parunak and Brueckner (2004) call 'swarming' which they define as:

> "*useful self-organization of multiple entities through local interactions.* „
>
> — Parunak and Brueckner, 2004

Another very close related field is that of Multi-Agent Systems (MAS) (Di Marzo Serugendo et al., 2006), however, this approach is generally more focused on engineering systems than on biological inspirations of these systems.

2.2.4 Artificial Life

> "*A central theme of artificial life (a-life) is to construct artifacts that approach the complexity of biological systems.* „
>
> — Angeline and Pollack, 1994

Usually, these systems are only an abstract recreation of a biological system, a typical example is that of Cellular Automata (CA). CA's are usually represented as a two dimensional grid of cells, each cell is programmed with the same set of simple rules. Cells can either be turned on or off and can usually only observe their neighboring cells. From these simple settings, impressive patterns can emerge, these systems are excellent examples how simple rules can generate complex patterns. Furthermore Stephen Wolfram has shown that some types of cellular automata could function as universal computers (Kennedy et al., 2001).

One of the pioneers in artificial life, Christopher Langton, stated that CA's compute on the edge of chaos. Meaning that the behavior of a system at the edge of chaos is not predictable, and not random: it is *complex* (Kennedy et al., 2001). Also the work of Karl Sims (1994) is generally classified as artificial life, he created an evolutionary program that designed virtual robots. Interesting is that he evolved the robot's morphology as well as its brain (neural network) at the same time, this resulted in robots capable of walking, jumping, swimming, etc.

The a-life field is closely related to that of the field of Evolutionary Robotics (ER), where ER is actually an application of a-life using robotics. An example is the European funded SYMBRION[1] project which aims to develop principles of adaptation and evolution for symbiotic multi-robot organisms (Baele et al., 2009).

2.2.5 Organic Computing

Organic Computing (OC) is a field of computer science with the aim to design computing systems that are more lifelike. These systems need to possess so called self-x properties such as self-organization, self-configuration, self-repair, self-healing, self-optimizing and self-protecting.

OC is a very practical field, which aims to tackle the increasing complexity of technical systems by developing new design principles. Branke et al. (2006) present a generic architecture for organic systems. This architecture defines several modules, such as an observer, a controller and an observation model. Scheidler and Middendorf (2009) present a different approach utilizing Learning Classifier Systems (LCS).

From a software engineering perspective, these concepts are also sometimes called *self-adaptive software* (Salehie and Tahvildari, 2009).

2.3 Definition

The theories and examples presented in the previous sections have shown that there are many different view on complex systems. Based on the similarities in these theories a definition of complex systems is proposed:

> *Complex Systems are adaptive systems composed of emergent and self-organizing structures.*

[1] For more information see: `www.symbrion.eu`

This definition places *emergence* and *self-organization* as the most important properties of complex systems. These properties were chosen since all other relevant properties can be derived from these, in fact, it can be said that self-organization itself is often an emergent property. From section 2.1 and 2.2 it should be clear that these concepts are apparent in many natural complex systems as well as in many related theoretical fields, also Uny Cao et al. (1997) state that complex systems are often described using these terms.

This positions self-organization and emergence as the most important features for generating adaptive behavior. Because emergence and self-organization are itself complex concepts, they will be explained in detail in the next sections. A brief overview of their properties and their associations with each other are shown in Figure 2.2. The direction to which an arrow points determines the relation of the concepts, for example, self-organization is decentralized and usually contains some kind of randomness. So this image displays how adaptivity is generated from the definition of complex systems.

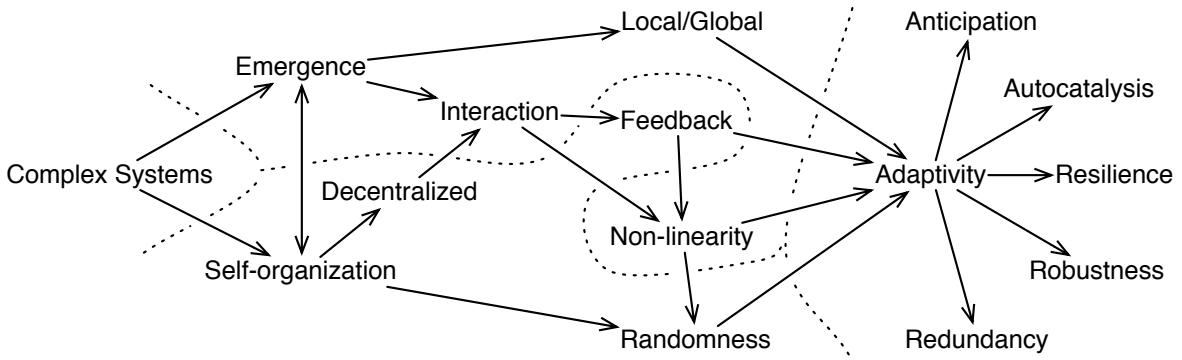


Figure 2.2: Conceptual overview of complex systems, showing the relations of the relevant concepts. The arrows leaving a concept indicate important properties for that concept, the dotted lines divides the concepts in four major topics.

It can be seen that the concepts of emergence and self-organization go 'hand in hand'. It is difficult to make a precise distinction for every property whether it belongs to emergence or self-organization. However, the next sections do make this distinction but it should be noted that this is by no means a definite categorization, it is meant as the most fitting organization for this thesis. The dotted lines in Figure 2.2 produce four areas for each of the following concepts: complex systems, emergence, self-organization and adaptivity. The latter three of these concepts are discussed in the next sections.

2.4 Emergence

In (Corning, 2002) it is stated that there is no universally acknowledged definition of emergence, however, he states that the most elaborate definition is

> "*Emergence is the arising of novel and coherent structures, patterns and properties during the process of self-organization in complex systems* „
>
> — Jeffrey Goldstein as cited by Corning (2002)

This definition positions emergence close to self-organization and complex systems, similar to the definition presented in the previous section. A more simple definition of emergence is given by Schut (2010) as "the whole is more than the sum of it's parts". The difficulty of finding a universal definition for emergence is illustrated by an expert on the topic of emergence:

> "*It is unlikely that a topic as complicated as emergence will submit meekly to a concise definition, and I have no such definition to offer.* „
>
> — John Holland, 1998

A result of this lack of a proper definition is that there is a lot of controversy how emergent properties are to be distinguished from non-emergent properties (Damper, 2000). Nevertheless, there have been made many attempts in the literature to define different types of emergence and to give a list of properties.

However, the purpose of the current investigation is to use emergence as a tool, emergence is seen as an advantage that can be exploited. In section 2.1 several examples of emergent systems were already given, this section lists three typical properties that can be incorporated in a system to yield emergence. As can be seen in Figure 2.2 emergence and self-organization are very interrelated, and can probably only be understood in conjunction. As such, this section should not be examined isolated from section 2.5.

2.4.1 Local and global

The distinction between a local and a global perspective or level is often associated with emergence (De Wolf and Holvoet, 2005). Local is the level of the individual which describes the behavior (or algorithm) of the individual. Global is the system wide perspective, which describes characteristics on a system level. The global behavior of an emergent system is not evident from the local behavior of its elements (Kennedy et al., 2001). This is similar to the notion of emergence of Schut (2010) as described in section 2.2.2.

This distinction can be generalized with the concepts of aggregates, for example, phenomena and natural laws at a given level emerge from the operation of laws at a more fundamental level (Damper, 2000). Another example of such a property is the gross domestic product of an economy, this is an aggregate of other properties of an economy. This process can be repeated several times, where each aggregate can be decomposed in entities called building blocks (Holland, 1995, page 34). Using mechanisms as building blocks it would be possible to create a system that exhibits emergent phenomena which would be difficult to predict from inspection of the mechanisms (Holland, 1998, page 44).

2.4.2 Interaction

Holland (1998) states that when parts interact in less simple ways, reduction does not work, the interaction as well as the parts have to be studied. For example, while there is an absence of central control in an economy it is a mostly stable process. This stability can be said to emerge from the interactions of the participants (e.g buyers, sellers) in this economy (Kennedy et al., 2001). In fact, there is an emergence of global organization from local interactions.

Holland (1995) describes *tagging* as a mechanism that allows selective interaction between parts. Tags are markings attached to each entity, the tag contains information about the entity (functionality, behavior, etc.) and can be observed by the other entities (Di Marzo Serugendo, 2003). These tags are similar to tags in the biological immune system, where 'wrong' cells are tagged to be cleaned up. Tags can further be used to delimit the number of interactions in a system, tags that mediate useful interactions are desired. This mechanism also facilitates the formation of aggregates.

The Gestalt effect is an excellent example of emergence from psychology, by positioning shapes in a certain way the mind can be made to think that it sees shapes which upon closer examination are not actually displayed. An example of this effect is shown in Figure 2.3, in Figure 2.3(a) a single shape is shown, Figure 2.3(b) shows the interaction effect that occurs with the six shapes. The positioning of the shapes yields the emergence of a white triangle in the middle of the image, each corner of the triangle is formed by two adjacent shapes. Although the white triangle is not 'real', it is an effect of the way the human mind works according to the Gestalt psychology[2], this is nevertheless a very good example of the interaction effect.

(a) basic shape (b) six shapes

Figure 2.3: A visual example of emergence, better known as the Gestalt effect.

[2] `http://en.wikipedia.org/wiki/Gestalt_psychology`

Another example of this interaction effect is the cooperation between multiple entities, for example in an ant colony, or cooperation between two companies, this is sometimes also called synergy.

2.4.3 Non-linearity

Non-linearity refers to the relation between cause and effect in a system. Usually in simple models this relation is linear, meaning that effects are proportional to their causes (Heylighen, 1999c). However this is not the case in emergent systems, this non-linearity can be explained by the (chaotic) *butterfly effect*[3] which says that small causes can have large effects, and large causes can have small effects. Holland (1998) poses this in a similar way by saying that small changes in local conditions can cause major changes in global long term behavior.

An example of non-linearity in a practical situation is given by the following thought experiment. Consider you are kicking a football, now you expect the ball to move a distance which is proportional to the amount of force used, now consider kicking the ball with twice as much power, you expect the ball to go about twice as far. Now think of what could happen when non-linearity would be involved, it could be that when kicking the ball twice as hard, the ball would not go twice as far but maybe ten, hundred or even thousand times as far! This illustrates the unpredictable nature of non-linear relations.

The occurrence of non-linearity in complex systems is best described by the notion of *feedback* (Heylighen, 1999c), feedback is discussed in section 2.5.2. The effects of non-linearity are sometimes also observed as randomness, this is explained in section 2.5.3.

2.5 Self-organization

As with the concept of emergence, there is no generally acknowledged definition for self-organization. However, this section tries to define the concept in a practical way that allows it to be used later in this thesis. An intuitive definition of self-organization is cited by De Wolf and Holvoet (2005): "Self-organization refers to exactly what is suggested: systems that appear to organize themselves without external direction, manipulation, or control."

The following statement ties self-organization and emergence very close together:

> "*Self-organizing systems are made of many interconnected parts whose local interactions, within a given environment, give rise to emergent properties, or behavior, observable at the level of the global system only.* „
>
> — Di Marzo Serugendo, 2003

Which is shared with a similar opinion:

> "*Self-organization is a process in which pattern at the global level of a system emerges solely from numerous interactions among the lower-level components of the system. Moreover, the rules specifying interactions among the system's components are executed using only local information, without reference to the global pattern.* „
>
> — Camazine et al., 2003

These are satisfactory definitions since the differences between emergence and self-organization are not clear-cut. The remainder of this section discusses the properties that are typically but not exclusively associated with self-organization.

2.5.1 Decentralized

This is one of the more obvious properties of self-organization, decentralization means that there is no central controller that supervises all the parts. In fact, all individuals are autonomous, meaning that they are self-governing, however, this doesn't mean that they don't influence each other, they interact as described in section 2.4.2. Typically this means that access to global information is limited by the locality of interactions, this is usually regulated by simple rules (Di Marzo Serugendo et al., 2006).

[3] An illustrative example of the butterfly effect is a movie of the same name: `http://www.imdb.com/title/tt0289879/`

2.5.2 Feedback

Each component in a system affects the other components, but these components in turn affect the first component. As a result there exists a circular cause-and-effect relation called feedback (Heylighen, 1999c). There are two kinds of feedback, positive (self-reinforcing) feedback allowing deviation to enlarge in an explosive way, and negative (self-counteracting) feedback which stabilizes the system.

This feedback results in the non-linear behavior described in section 2.4.3. These systems have in general several stable states (Heylighen, 1999c), the system can move from one stationary state (equilibrium) to another, the deviations are reinforced trough positive feedback while a negative feedback loop maintains the current equilibrium.

Examples of feedback can be found in the human body and in the ecology. In the human body the process of self-balancing is called *homeostasis*, for example the body temperature is controlled through such a negative feedback mechanism. Furthermore, the earlier discussed ACO algorithm consists entirely of feedback loops. An example of real ants using feedback is given by Camazine et al. (2003), consider an ant colony in a developing nest site. Ants are digging at the site, to increase it, while digging they add pheromone to the walls of the cavity. This increases the probability for other ants to start digging at the same place, this is a positive feedback loop. After a while, the nest site has grown substantially, which causes the density of ants to decrease. Due to this lower density of ants, the pheromones have more time to evaporate through the air. This lowers the concentration of pheromones on digging sites, which slows down the digging process. Finally, the digging will stop when the nest site's size is appropriate for the number of ants that live there.

The concept of feedback has an obvious connection with adaptivity which is discussed in section 2.6.

2.5.3 Randomness

According to Kennedy et al. (2001) adaptation in nature is almost always a *stochastic* process, meaning that it contains randomness. They state two reasons for this, one is that it is an expression of uncertainty, if you don't know where to go, a random direction is as good as any. Second, by trying something new, randomness introduces creativity or innovation, which can lead to a self-organized system. Kennedy et al. (2001) also state that Gregory Bateson called evolution and mind the 'two great stochastic systems', they adapt by making (random) adjustments. The prevalence of these two stochastic systems suggests that randomness is an important property in nature, and as such in complex systems.

The occurrence of randomness in complex systems is arguably more a limitation of the observer than an intrinsic property of a system. If the observer would be able to perceive every part of the system and completely understand the relations between the parts, the randomness would disappear. This is similar to the concept of the 'edge of chaos' as was discussed in sections 2.2.2 and 2.2.4. However, this is difficult and this is precisely what makes these systems so hard to analyze, as it is sometimes nearly impossible to observe every part, let alone to fully understand the parts. This perceived randomness could be a result of the non-linearity as discussed in section 2.4.3, in this case, what is really meant with randomness is that a certain process is 'unpredictable' or that it has an 'unexpected' outcome (Kennedy et al., 2001). This randomness caused by non-linearity is also demonstrated in cellular automata as was discussed in section 2.2.4.

However, whether randomness truly exists is not of importance for this discussion, what matters is that it works in complex systems. It is a useful property, for example, randomness enables the discovery of new solutions as stated by Schut (2010). Even though randomness in nature might be a product of non-linear systems, it is sometimes useful to simulate this with a random variable generated by a *pseudorandom* generator.

2.6 Adaptation

In biology, adaptation is the process whereby an organism fits itself to its environment (Holland, 1995, page 9). In the sense that we use it here, the system takes the place of the organism and the environment is defined by the system's input and output. De Wolf and Holvoet (2005) state that to be adaptable, a system needs to make a selection between behaviors and at the same time consider a variety of behaviors. This is a balance, where an abundance of variety makes the system uncontrollable, and too much selection

yields in a system that is not flexible enough. They relate this to balancing the system on the edge of order and chaos.

According to (Kennedy et al., 2001, page 9) an important difference between animate and inanimate things is the ability to adapt, this suggests that adaptation is the key to producing intelligent behavior. Furthermore, the "two great stochastic systems" (evolution and mind) mentioned earlier are in essence continuously adapting.

In (Salehie and Tahvildari, 2009) it is stated that adaptation properties are sometimes called self-* properties, examples of these properties were given in section 2.2.5. Each of these self-* properties is similar to one or more of the properties that are discussed in this section. It is stated by Heylighen (1999c) that in a sense, self-organization implies adaptation. This section describes the causes of adaptation more precisely by referring to the properties of both self-organization and emergence that were discussed in the previous sections. Through these emergent and self-organizing properties, adaptation is linked to complex systems in general.

2.6.1 Anticipation

Anticipation or prediction can be thought of as developing rules that anticipate the consequences of certain responses (Holland, 1992). Internal models are introduced in (Holland, 1995, page 31) as a means to anticipate future events. Models are abstractions of the real world, they are a very useful construct that allows reasoning about the world. It is further stated that anticipation is not exclusively used by higher order organisms, for example, a bacterium moves in the direction of a chemical gradient, thereby (implicitly) predicting that in this direction something valuable can be found.

Anticipation and an internal model can be the result of the interaction structure between the individuals and a useful design of feedback loops. Key here is the specific *design* of this structure, this is what really makes anticipation possible.

2.6.2 Autocatalysis

The concept of autocatalysis originally comes from chemistry, a catalyst is a substance that enables a chemical reaction. In autocatalysis the catalyst is one of the products of the reaction (Parunak and Brueckner, 2004), thereby enabling itself. They furthermore state that an autocatalytic set is a set of reactions that are not individually autocatalytic, but who catalyze each other. In the sense that it is used here, the concept represents the exploitation of positive feedback (Kennedy et al., 2001, page 103).

This mechanism allows the amplification of apparently trivial effects into significant ones (Kennedy et al., 2001, page 103). It also allows the system to deviate from one stable state to another state, it can be triggered by (external) circumstances to which the system needs to adapt itself. This is in essence a result of the feedback loops as described in section 2.5.2.

Marco Dorigo makes some interesting notions about autocatalytic processes. He states that their interaction can lead to rapid convergence to a subspace of the solution space that contains many good solutions, enabling the finding of a very good solution quickly, without getting stuck in it (Dorigo et al., 1991). Because of this quality, autocatalysis is one of the 'building blocks' of Ant Colony Optimization (Dorigo and Socha, 2006). This translates in the ACO algorithm to be able to adapt to changes in the problem space in real time, which is obviously a very practical function.

2.6.3 Resilience

Resilience refers to the flexibility of a system, it is the capacity to restore itself (Heylighen, 1999c). For example, a forest (an ecosystem) that has suffered damage from a fire, will in general recover relatively quickly. Resilience is strongly related to negative feedback loops, as it is a counteracting force which stabilizes the system, as such it can be seen as the opposite of autocatalysis. This same concept is called homeostasis in living organisms.

2.6.4 Robustness

Robust systems are relatively insensitive to perturbations or errors (Heylighen, 1999c), they have a high fault-tolerance. A robust system is expected to deal with continuous change and to maintain its organization autonomously (De Wolf and Holvoet, 2005).

Decentralized systems have a natural robustness, there is no central controller that on failure would cripple the system. However, increasing damage will decrease performance, but degradation will be 'graceful', the quality of the system will decrease gradually instead of with sudden drops of quality (De Wolf and Holvoet, 2005).

The system's performance is robust against the loss of a few individuals (Parunak and Brueckner, 2004), this is due to its inherent *redundancy* as is discussed in section 2.6.5.

2.6.5 Redundancy

A system is redundant when it has the same knowledge represented in a number of different places (Schut, 2010). This is a kind of robustness as a system shows a certain insensitivity to damage due to replication of components (Di Marzo Serugendo et al., 2006). The failure of a single entity will not cause a complete failure, the system is flexible enough such that the emergent structure can remain (De Wolf and Holvoet, 2005). For example ants in an ant colony can be killed but the colony still remains, and its (long term) operation will not be affected.

2.7 Designing a complex system

The design of a complex system is a delicate task, that is, the design of a *useful* complex system is very complicated. The crux lies in the usefulness, that is, how to make a complex system behave in a way that it's actually beneficial. This difficulty can be explained by the properties which were discussed in the previous section. Changing or designing a local component can yield unforeseen global system behavior, this is an effect of a complex system's inherent non-linearity and randomness.

Holland (1998) states that when parts interact in less simple ways, reduction does not work, both the interaction as well as the parts have to be studied (note that this is similar to what Minsky (1988) said, see section 2.1.4). By analogy, this means that when *designing* a complex system, the local components, the local interactions, and the global perspective all need to be taken into account.

Two options for designing such systems naturally appear, designing such a system by hand, or designing it automatically by using algorithms. Automatic design obviously transfers the design task from the complex system to the algorithm, instead of thinking about the design of a complex system an algorithm that creates a complex system has to be invented.

2.7.1 Design by hand

Several attempts have been made to create a type of complex system, for example, Mitchell and Hofstadter (1994) have created an *emergent* architecture that is similar to a complex system. Their system is capable of making analogies, they claim that their system is best suited to mimic human analogy-making. Subsequent research has shown that their idea is transferable to a number of other applications such as music prediction, geometric puzzles (Foundalis, 2006), and auditory perception (Dor and Rothkrantz, 2008).

Another interesting approach is to see the issue as an algorithm design problem, McLurkin and Demaine (2009) describe a distributed boundary detection algorithm for multi-robot systems. Their algorithm is designed on the local level which yields desired behavior on the global level. A similar approach for designing a distributed coverage control algorithm is described by Schwager et al. (2008). These implementations show that it is possible to design such a system by hand, however, these systems are relatively simple, as their domain of application is limited.

Generally, designing this kind of software is a complicated task, this is emphasized in the following statement:

> "*The traditional software engineering techniques are insufficient, since they are based on interfaces fixed at design time, or well established ontology* "
>
> — Di Marzo Serugendo (2003)

In other work on cooperating robot swarms Uny Cao et al. (1997) states that finding the correct values for control parameters that lead to a desired cooperative behavior can be difficult and time-consuming for a human designer. This suggests that using an automated approach might be more feasible for the design of complex systems.

2.7.2 Design by computer

Designing a complex system using an algorithm isn't necessarily easier compared to designing it by hand. However, by designing a program that creates a complex system there is hope of developing a general approach for the creation of complex systems.

Artificial evolution seems to be the most suitable computational paradigm for designing a complex system. This follows from the observation that all natural complex systems are the product of natural evolution. Simulating evolution might therefore be an adequate approach for designing these systems.

The earlier mentioned work of Sims (1994) is a good example of using evolution to create a complex system. Evolution has also successfully been applied for the design of hardware (Torresen, 1998), the design of robots (Römmerman et al., 2009), and the development of an acoustic communication system between robots (Tuci and Ampatzis, 2007).

The work of Mitchell and Hofstadter (1994) (as discussed in the previous section) is based on the assumption that analogy making is closely related to intelligence. Using evolution such an underlying assumption is not needed, as evolution is by definition assumption less. However, if analogy is indeed the most important creative force, it is plausible that evolution will design such properties in a complex system. Of course, creating an artificial evolutionary process that yields a desired outcome isn't a straightforward task, evolutionary computing is explained in detail in chapter 3 .

2.7.3 Design choices

Whether designing a complex system by hand or using evolution, several design choices have to be made. By now it is clear that complex systems consist of a set of entities or individuals. The design of these entities determines the overall design of a complex system.

Uny Cao et al. (1997) lists several design choices that need to be made for designing a robot collective, the choices which are relevant for complex systems are discussed in this section.

Centralization or decentralization The choice between a centralized or decentralized structure is actually a simple one. As was discussed in section 2.3, self-organization (and as such decentralization) is an important feature of complex systems. A decentralized design also introduces a local and a global level (section 2.4.1), and as such it facilitates *emergence*. Furthermore, Uny Cao et al. (1997) states that decentralized architectures have several inherent advantages over centralized architectures, including fault tolerance, natural exploitation or parallelism, reliability, and scalability. They further state that many (artificial) systems do not conform to a strict centralized/decentralized dichotomy, for example, many largely decentralized architectures use 'leader' agents (Uny Cao et al., 1997).

Differentiation This choice is also acknowledged by Floreano et al. (2008). It concerns the homogeneity of the individuals in the complex system, all individuals can be the same or different. It seems that natural complex systems of both types exist, ants in an ant colony are (largely) the same, while there are several types of antibodies in the immune system. Groß et al. (2008) show that homogenous robots (in term of morphology and brain) are capable of a division of labor, meaning that robots perform different kind of tasks. As such, it seems that both options are equally viable for complex systems.

An advantage of using *homogenous* individuals is that the design task is much simpler as it has to be done only once. This is contrary to the *differentiated* situation, where each individual has to be designed separately. This makes the search space for homogenous individuals smaller than for heterogenous individuals (Potter et al., 1996). Additionally, a hybrid option is also possible, the use of parameterized homogenous individuals. This means that individuals are almost the same except for a few parameterized differences. This reduces the design task enormously as only a few parameters have to be set for each individual. We call this option *semi-homogenous*, this option is preferred since it is a good tradeoff between the size of the search space and possible differentiations.

Communication structure As was stated before interaction and feedback are important properties of complex systems, as such it is essential that this is included in a model of complex systems. There are three types of communication (Uny Cao et al., 1997), one of them is *interaction via environment*, an example is stigmergy using pheromones as is employed by ants. The difficulty of this communication

structure lies in the definition of the environment. For ants searching for the shortest path, the environment is a natural part of the problem, but for more abstract problems the role of the environment is hard to define. The second type of communication is *interaction via sensing*, an example is that of the school of fish discussed in section 2.1.3. Fish use visual input to adapt their position to that of the school. For this type of communication a similar problem arises as with stigmergy, if the problem is difficult to translate to the physical domain, how can 'sensing' be defined? The third type is *interaction via communications*, this can be direct or broadcast messages. This type of communication can be characterized as a network topology, this is convenient as it requires no additional modeling. This form of communication is used in the most powerful complex system known: the human brain.

Because it doesn't require additional modeling and is generally applicable, *interaction via communications* is the preferred communication structure.

Modeling of other agents Uny Cao et al. (1997) state that modeling the intentions, beliefs, actions, capabilities, and states of other agents can lead to more effective cooperation between robots. However, this is not necessary for a complex system, as it's obvious that bacteria, antibodies or individual neurons do not posses this capability. As such it is not a preferred ability for an artificial complex system.

2.8 Conclusion

In section 1.3 several hypotheses were listed, and in section 1.2 several research questions were listed. The first hypothesis and research question specifically target complex systems:

H_1 Complex systems have properties that are desirable in computing.

Q_1 What are the strengths and weaknesses of natural complex systems?

This chapter has been investigating all properties of complex systems, many of which are desirable in computing, this means that this hypothesis can be accepted. The strengths of complex systems are the properties discussed in section 2.6: *anticipation*, *autocatalysis*, *resilience*, *robustness* and *redundancy*. These are all highly desirable properties for software, as such, complex systems provide a promising alternative model for software. These properties are particularly suited for classifiers since difficult classification problems (such as speech recognition) require adaptivity. And as was stated in chapter 1, traditional statistics based classifiers lack these adaptive properties. However, the design of complex systems is hard, their emergent properties are difficult to steer in a preferred direction.

This chapter also investigated a number of design choices, the following properties were identified as being desirable when creating complex systems.

- decentralized structure
- semi-homogenous individuals
- interaction via communications

These properties are used as guidelines for the design of a model for complex systems which is discussed in section 4.1.

3. Evolutionary Computing

When Charles Darwin published his *On the Origin of Species* in 1859, he identified and described a previously unknown class of natural problem solvers (Darwin, 1859). Around the same time Mendel started developing the theory of genetics which was later connected to Darwin's selection theory.

Evolutionary Computing (EC) is the subfield of Computer Science which employs techniques inspired by Darwin's theory of evolution. Evolution offers an explanation for the biological diversity we observe in the world. Through the mechanism of natural selection the number of individuals in a given environment is kept under control. Where the individuals that are most fit (adapted to the environment) are more likely to be able to reproduce themselves, this is called survival of the fittest. In natural evolution there are two different ways in which variation in individuals is introduced, through reproduction (genetic recombination) and through gene mutation. The idea of EAs is to mimic the process of evolution by implementing fitness pressure and variational operators which operate over a set of individuals (problem solutions), which will lead to a (near) optimal solution over a number of generations.

As was made clear in the previous chapter, there is an evident need for a program that 'designs' complex systems, the following statement corresponds with the view on EC of this chapter.

> "*Evolutionary computation paradigms provide tools to build intelligent systems that model intelligent behavior.* „
>
> — Kennedy et al., 2001

This chapter first establishes common concepts in the EC literature and describes the overall layout of an evolutionary algorithm. In the later sections of this chapter, the design task at hand is more specifically studied, and several possibly fruitful techniques are explored to design a complex system using evolution.

3.1 Definition

Evolutionary algorithms simulate the evolution of a population. The population consists of possible solutions for the problem at hand, which evolve until the final solution is found. The overall process starts in a chaotic (random) state, and starts to organize from that point. Eiben and Smith (2007) use a summarized description to characterize an EA, in this section the most relevant concepts are explained.

There are several branches in EC, all follow the same general outlines (Eiben and Smith, 2007) but differ in technical details. In this section a unified approach is chosen, highlighting the differences between the branches only for additional background information when it is relevant to the discussion.

In Figure 3.1 the general outline of an evolutionary algorithm is shown. This section will describe every component of the algorithm, highlighting the common implementations, the possible complications and usefulness of the component.

3.1.1 Representation

The representation of the problem's solution is the bridge between original problem context and the problem-solving space. Objects forming possible solutions within the original problem context are referred to as phenotypes (or individual, candidate solution), their encoding, the individuals in the EA are called genotypes (or individual, chromosome). So the genotypes represent the phenotypes. A solution (a good phenotype) is obtained by decoding the best genotype after termination. A chromosome usually has elements, a position is called gene or locus and an object in such a place is called an allele.

Bentley (2000) states that evolution, like all search algorithms, is limited and constrained by the representation it can modify. In other words, choosing a suitable representation is essential for solving

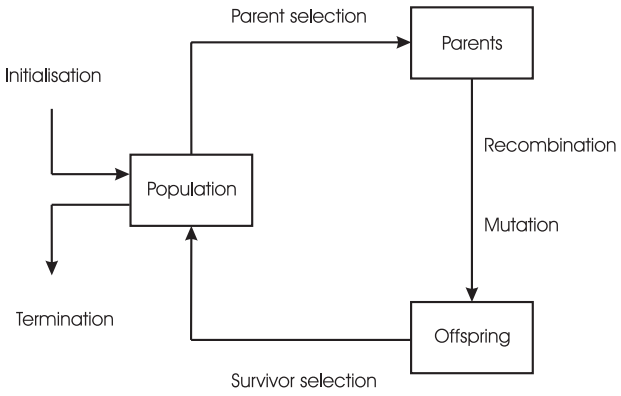


Figure 3.1: The general scheme of an evolutionary algorithm as a flowchart (adapted from Eiben and Smith (2007)).

the problem at hand. There are several common representation encodings which are associated with a particularly branch of EC: strings over a finite alphabet are commonly used by Genetic Algorithms (GA), vectors of real numbers by Evolution Strategies (ES) and application parse trees by Genetic Programming (GP). Of the four evolutionary variants, only GA uses a mapping from phenotype to genotype, the representations used in the other variants are usually direct problem solutions.

For designing good representations there are five major requirements listed in (Kicinger et al., 2005a). These are:

- *Non-redundancy*, the mapping between genotype and phenotype should be 1-to-1.
- *Legality*, standard variation operators should always create a legal solution.
- *Completeness*, for every phenotype there exists a corresponding genotype.
- *Lamarckian property*, meaning that offspring can inherit goodness from the parent (Cheng et al., 1996). The meaning of alleles for a gene is not context dependent (Kicinger et al., 2005a).
- *Causality* or *continuity*, small variations in the genotype space should result in small variations in the phenotype space.

Also in (Kicinger et al., 2005a) a taxonomy of representations is given, the different attributes are listed in Table 3.1.

Attribute	**Possible attribute values**
EA level	*Genotypic* (using a mapping) or *phenotypic* (direct representation)
Structure	*Linear* (e.g. a string) or *nonlinear* (e.g. a tree)
Length	*Fixed* or *variable*
Change during evolution	*Static* or *dynamic*, this includes changes on the structure.
Encoding scheme	*Direct* representation encodes actual design concepts, or *indirect* encodes rules that construct these concepts.
Accuracy of solution specification	*Parameterization* the design is specified in sufficient detail, or *open-ended*, the design is changeable.
Ability to reuse encoding	*Non-generative* (no reuse) or *generative* (reuse)
Genotype-phenotype correspondence	*Explicit* generative representations are similar to procedural programs for constructing designs, *implicit* generative representations consist of a set of simple rules that implicitly specify a design.

Table 3.1: A taxonomy of evolutionary representations, inspired by (Kicinger et al., 2005a, Table 2).

3.1.2 Fitness function

The fitness function or evaluation function is a function which indicates the quality of a solution. It forms the basis for selection, and so it facilitates improvements. The fitness function can be seen as the driving force or the pressure for optimization of the solution. For numerical optimization problems, the fitness function is usually proportional to the output of the function being optimized.

However, for problems of a more complex nature a function that needs to be optimized doesn't always exist. Take for example the evolution of a robot controller, examining this program by itself is difficult, it has to be tested in its context, its domain of application. This is exactly what Sims (1994) did in his pioneering work on evolving virtual creatures. He constructed a virtual three-dimensional world which simulated forces like gravity, collision detection, collision response, etc. The fitness is determined based on the solution's (robot) performance in this simulation.

The fitness function can be seen as the driving force, the pressure for optimization and novelty. This survival of the fittest, or the most adapted individual to the environment, leads to solutions of an ever increasing quality. However, this doesn't always lead to the global optimum. There is a risk of getting trapped in a local optimum, this algorithm behavior is called *premature convergence* (Eiben and Smith, 2007). This can best be explained by visualizing fitness functions using the metaphor of a landscape, one dimension usually depicts the fitness of the solution and one or more dimensions represent the dimensions of the configuration, or biological traits. An example is shown in Figure 3.2, here the vertical dimension depicts the fitness values, and the other two dimensions the biological features.

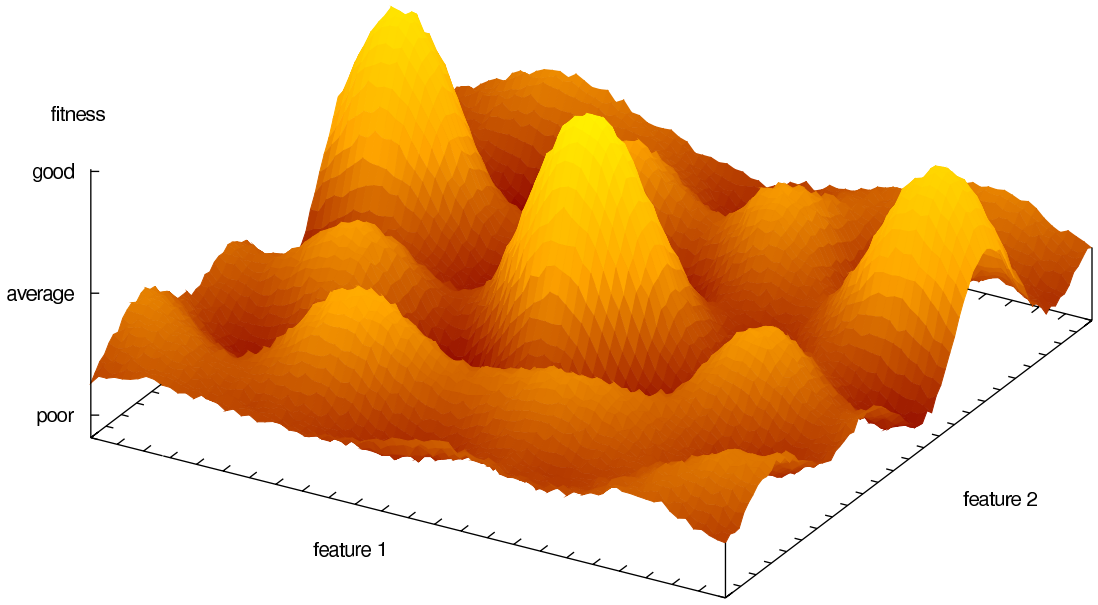
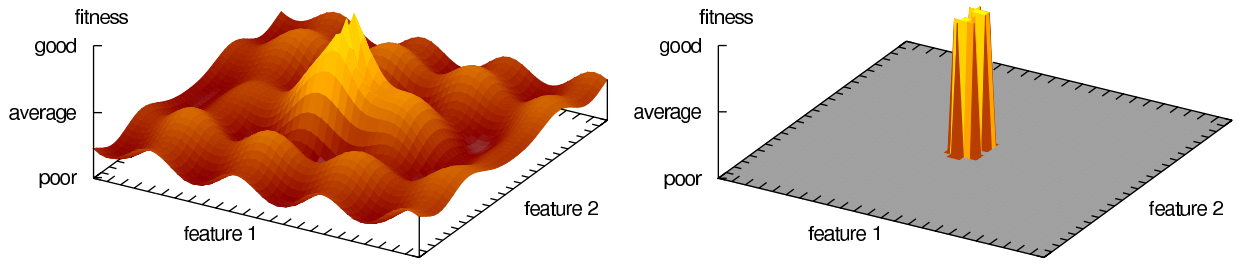


Figure 3.2: Illustration of fitness landscape with two features, the higher (yellow) the better, peaks are (local) optima.

The design of the fitness function influences the fitness landscape enormously. An important property of the landscape is its smoothness, if a landscape is smooth, small steps usually result in small increases or decreases of the fitness value. In contrary, if a landscape is less smooth small steps might yield bigger fitness changes. A rather extreme example is given in Figure 3.3, these landscapes represent the same problem domain. Now, consider the area around the global optimum which leads directly (when ascending) to the peak, in Figure 3.3(a) this area is significantly greater compared to that of Figure 3.3(b). This area directly influences the probability of finding the global optimum, as such a fitness function producing a smooth fitness landscape is generally more desirable.

Fitness functions are in many cases the most computationally intensive component of the EA (Kennedy et al., 2001). Which is amplified by the fact that it has to be computed for every individual in every generation. To avoid getting stuck in local optima, and to distinguish EC from ordinary hill climbing methods, variational operators are used such as recombination and mutation.



(a) smooth (b) bumpy

Figure 3.3: Different fitness landscapes for the same (fictional) problem domain: a smooth landscape (a) and a bumpy landscape (b).

3.1.3 Variational operators

A variational operator literally brings variation in the population, two kinds are described, the binary recombination operator and the unary mutation operator.

Recombination Recombination, or crossover, is a variational operator which merges information from two parent genotypes into one or two offspring genotypes. Various implementations are available most of them are stochastic. The implementation of the recombination operator largely depends on the type of representation used. In Figure 3.4 an example of a crossover operator is displayed.

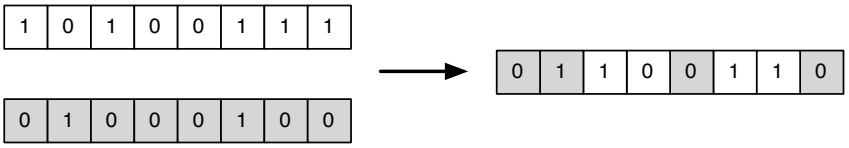


Figure 3.4: Example of a uniform crossover operator, two parents (binary strings) create one child. Each position is chosen uniformly from one of the parents.

A recombination operator ideally preserves information which is contained in both parents, this property is called *respect*. Whether this is easy to implement depends entirely on the representation used, this is more extensively discussed in (Eiben and Smith, 2007).

Mutation Mutation is also a stochastic operator, it modifies the genotype to create a mutated child. It is used to cause a random, unbiased change.

3.1.4 Selection

Parent selection Parent selection is the mechanism on which the individuals are chosen which are used to create children. Generally, more fit individuals have a higher probability of being selected to become a parent.

Survivor selection Survivor selection can be used to keep the population size constant. There are generally two types of survivor selection: *generational* in which all individuals are replaced and *steady-state* which is deterministic and biased by fitness.

3.1.5 Parameters

Population Usually the population has a fixed number of individuals. Determining the population size is not a clear cut case, a large population will perform an extensive search through the search space but it comes with an additional computational cost compared to a smaller population (Kennedy et al., 2001).

Initialization Usually the first population is randomly generated. Although time can be invested to construct a relatively good solution before the evolution starts, this is generally not practiced because in the first few generations of an evolutionary run progress is very steep. As such the efforts of generating a good initial population are quickly surpassed by the EA.

Termination condition The termination condition determines when the evolution is brought to an end. An obvious example is to terminate when the best possible solution is found. Other measures can be to stop after a predetermined number of generations or when the fitness hasn't increased for a number of generations.

3.2 Evolutionary design

> "*It is often presumed that, in order to automate any part of the design process, one must start with a cognitive theory of how humans design. This is based on the assumption that humans offer the only example of a successful design system. However, alternative successful design systems do exist. One such system is biological evolution in nature, which has been evolving biological designs that far exceed any human designs in terms of complexity, performance and efficiency. Evolutionary programs use biological evolution in nature as a source of inspiration, rather than a phenomenon to be accurately modeled.* „
>
> — Janssen et al., 2002

This expression exactly resembles our view on evolution, it is a solution, a tool, and an inspiration. Now that the most common evolutionary concepts are clear, we can focus on approaches that are more design specific and that can be used for the design of complex systems. Traditionally, evolutionary algorithms are seen as an optimization tool. However, according to Bentley (2000), they can also be seen as an exploration tool. Using evolution as an explorer or designer is a non-trivial task. This challenge is apparent in the following quote:

> "*Current evolutionary algorithms are widely successful optimization methods but are bad engineers.* „
>
> — Mouret and Doncieux, 2009a

From this statement it is clear that in order to successfully use evolution as a designer, the evolutionary algorithm itself needs to be adapted. For the design of complex systems, two questions of importance are:

- How to reduce the search space to speed up the evolutionary process?
- How to guide evolution towards a desired level of complexity?

Section 3.3 deals with possible answers to the first question. In the literature several attempts have been made to increase the efficiency of the evolutionary process for design tasks. In section 3.4 several approaches to generate complexity using evolution are discussed, it describes methods to 'push' evolution in the right direction.

Simon (1962) states that evolution using hierarchies is much faster compared to evolution without hierarchies. This is explained using several parables. The key of the theory comes down to the fact that hierarchies consist of stable intermediate forms, and as such facilitate a much faster evolution of complex systems. He states the hypothesis that 'complexity will be hierarchic'. Hierarchies are key to the success of evolution and will play a prominent role in the answering of these two questions in sections 3.3 and 3.4. However, this also means that the answers to these questions partially overlap. It turns out that guiding evolution towards more complex solutions also implies an acceleration of evolution.

3.3 Accelerating evolution

Accelerating evolution is obviously something that is desired. Sometimes, artificial evolution is so slow that it seems that the process does not converge to an useful solution. This problem, which is sometimes

called the *bootstrap problem* (Mouret and Doncieux, 2009b), can be caused by a poor performance of the first random generated population. For example, when the search space is large and the population size is relatively small, it can be such that the entire population is stuck on the lowest fitness value. A simple solution would be to significantly increase the population size. However, this would yield in an equally significant computation time increase, and could make the problem computationally intractable. Possible solutions for the bootstrap problem are a reduction or transformation of the solution space or a more efficient search process.

For optimization problems, the representation is usually of a fixed length, which limits the solution space. In optimization the task is to find the most optimal values for this particular representation. However, in design problems the length of the representation is usually not known, making the bootstrap problem even more apparent. Determining the length of the chromosome poses a problem since choosing an insufficient length makes it impossible for evolution to come with a (near) optimal solution, choosing a representation length that is substantially larger than needed might make the search space too large, and the problem intractable. This problem is also acknowledged in the following statement:

> "*A major obstacle to using fixed-length encodings is that heuristically determining the appropriate number of genes becomes impossible for very complex problems.* „
>
> — Stanley and Miikkulainen (2004)

So how to solve this problem? When looking at natural evolution and the history of our planet, one trend can be easily identified. The first organisms were extremely simple, but throughout the millennia their complexity gradually increased, to eventually result in the complex animals that inhabit our planet today (Dawkins, 2006). One idea is to mimic this process by allowing evolution to dynamically increase the representation length during evolution, this assumes that if a bigger representation is useful it will be found. Several variants on this solution are described in sections 3.3.1 and 3.3.2.

3.3.1 Incremental Structure Evolution

In (Pasemann et al., 2001) incremental structure evolution is introduced as a way to create a modular robot control system. As the robot controller they use two neural networks, A and B. A solves a particular task which B also needs to be able to solve. In fact, A is a subnetwork of B. Since B contains A, it can use the already evolved networks of A. They discuss three different variants, *restricted*, *semi-restricted* and *free* module expansion. The restricted variant limits the architecture and parameters of module A, the semi-restricted variant only fixes the architecture of A and the free variant doesn't restrict A. It is expected that these strategies lead to the evolution of more complex behaviors by recombining behaviors in a hierarchical way.

The work of Hülse et al. (2004) uses *neuro-modules* as the basic non-linear dynamic building blocks for larger systems. They state that their coupling might result in emergent properties. They address the problem of large search spaces by introducing two neuro-module combination methods, fusion and expansion. These methods keep the search space as small as possible by starting with at least one already existing neuro-module, which can then be incrementally expanded. Furthermore, the modules can be fused to solve more complex tasks, and according to their claim, this method generates complex behavior in a fast and effective way.

A very interesting statement is:

> "*[..] the (recurrent) coupling of non-linear subsystems can lead to many undesired or unexpected behaviors of the composed system. But it is exactly the emergence of such unpredictable attributes on which our hope to find interesting brain structures rests.* „
>
> — Pasemann et al., 2001

In an attempt to find the minimal requirements necessary for the emergence of communication within a population of robots during the evolution of their control architectures, Wischmann and Pasemann (2006) used the semi-restrictive incremental method. They showed that even after many repetitions of the evolutionary process, no cooperation emerged. They attribute this behavior to the bootstrap problem, discussed earlier. However, they showed that for very simple environments emergence of cooperation through communication did occur, i.e. in a simple environment the bootstrap problem is probably less significant.

MacLeod et al. (2009) assume that the brain grew in evolutionary time by adding small numbers of neurons on top of existing structures, which increases the overall complexity of the brain. In their paper they successfully translate this assumption into an evolutionary algorithm. The aim of the algorithm is to create a neural network robot controller. They start with a very simple robot body plan, which is gradually made more complex during the evolutionary process. New neural modules are added to the robot brain until it reaches a certain fitness threshold, once this threshold is reached, the robot body plan is made more complex and the module adding continues. This incremental approach allows an unbounded growth of complexity.

Complexification In natural evolution a process called *complexification* occurs; the occasional addition of new genes which leads to increased phenotypic complexity. Stanley and Miikkulainen (2004) have implemented this concept; they state that complexification elaborates strategies by adding new dimensions to the search space. Note that this is very similar to the earlier discussed concept of *incremental structure evolution.*

3.3.2 Increased Complexity Evolution

In his paper about Evolving Hardware (EHW) designs, Torresen (1998) notes that evolving a relatively small hardware design requires about 2-3 weeks (in 1998) of computation time. In the same paper he proposes a solution for this remarkable long computation time, called *increased complexity evolution.* Instead of evolving the entire system at once, various stages are defined in a divide-and-conquer style, each stage builds upon the previous stage. It can be said that every stage creates building blocks which are used to reach higher levels of complexity in the course of evolution.

The major challenge in this approach is in designing the fitness function such that it facilitates different levels. This was shown first for a character recognition system, the problem domain is divided by giving a subsystem only a subset of the training characters. Several subsystems were evolved together such that as a whole the system had access to all the training characters. Torresen (1998) shows that the number of generations can be drastically reduced by evolving sub-systems instead of evolving a complete system at once.

In (Torresen, 2002) a controller for Prosthetic Hand Control is evolved using increased complexity evolution. Here a different approach is chosen, six subsystems are first evolved separately while each subsystem has access to all training vectors. The number of six subsystems correspond to the six prosthetic motions that need to be controlled. In the second phase the subsystems are evolved together to become the final system. Furthermore, in the second phase, only half of the 32 outputs are used to compute the fitness, Torresen (2002) argues that this improves the generalizability of the system.

Hierarchical fitness function A similar approach used for aerodynamic shape optimization, is called hierarchical evolution by Whitney et al. (2002). They define three different layers, the *bottom layer* which carries out the exploration in the search space using a very approximate model for fitness evaluation. The *intermediate layer* which uses a slightly better model, and the *top layer* which is used to refine the solution and which employs the most accurate model. They have shown that using this hierarchical model the evolutionary computation time can be optimized.

3.4 Towards complexity

> "*Evolution is interested in immediate advantage, rather than in complexity or generality, and there seems to be no general relationship between the two: To reliably obtain the latter, one must forcibly connect it to the former by means of algorithmic contrivance.* "
>
> — Miconi, 2008

In short, this means that evolution is unbiased towards higher or lower complexity. So to reach a complex solution, evolutionary pressure towards higher complexity needs to be present. Heylighen (1999b) underscores this by saying that a structurally more complex environment requires a more complex set of functions to cope with it. However, complexity is not a goal on its own, in fact the *least* complex but *working* solution is desired. Furthermore, Miconi (2008) states that complexity doesn't need to be favored in the fitness function, it can even discriminate against it.

Creating an evolutionary algorithm that does this is non trivial. Bentley (2004) compares evolution to a river flowing down a landscape; it always finds the easiest path, even if that means finding no solution at all. That's why efficient evolutionary guidance is needed. In the next sections, several approaches for evolving towards more complex systems are presented.

3.4.1 Generative Representation

An interesting approach for generating solutions of a higher complexity is that of a generative representation. A generative representation is inspired by the process of morphogenesis occurring in nature. In natural evolution, rules for growing complex organisms, called 'genetic plans' (e.g. DNA), are manipulated (Kicinger et al., 2005b). Organisms are built from these plans via a developmental process which is called morphogenesis.

Angeline and Pollack (1994) claim that the reason that most newer work has not managed to surpass the complexity of the work of Sims is the difference in representation. Kicinger et al. (2005b) also emphasize the importance of evolutionary representation, furthermore, they consider parameterized representations as inadequate when novel designs are sought. This suggests that different representations such as generative representations might have a better performance. In a study by Hornby and Pollack (2002) it was found that a generative representation is able to produce robots with a significantly greater fitness. They claim that generative representation captures a useful bias from the design space and allows viable large scale mutations in the phenotype, which leads to a better performance. Furthermore, generative representations enable encapsulation, coordination and reuse of assemblies of parts.

A generative representation or 'growing' designs, is what Bentley (2000) and Bentley and Kumar (1999) call *embryogenies*, it is a mapping procedure from genotypes to phenotypes. An embryogeny is a special kind of mapping process. The genotype is a set of 'growing instructions' which grows the phenotype.

Existing generative representations include cellular automata (as discussed in section 2.2.4) and Lindenmayer systems (Roggen et al., 2006). Lindenmayer systems (Lindenmayer, 1968a,b) are a grammatical rewriting system introduced to model the biological development of multicellular organisms. They are hierarchical, allowing reuse, and specialization. Many studies have been conducted using generative representations, e.g. for the evolution of hardware Tufte and Haddow (2004) introduced a developmental approach with the aim to improve the complexity of the electronic circuits that were evolved.

Based on their research Kicinger et al. (2005b) made two conclusions about generative representations. They concluded that their representations based on cellular automata proved to scale well with the size of the considered design problems. Also, their generative representations outperformed parameterized representations in their problem domain by generating better designs and finding them faster. These two advantages are further explained in the following paragraph.

Advantages An advantage of using a generative over a non-generative representation is their ability to reuse elements (as building blocks) of an encoded design (Kicinger et al., 2005a). These designs have fractal-like self-similarities, giving them an organic look and better scaling properties (Hornby and Pollack, 2001a). Fractals show how simple formula can create very complex results. This property is desired in a representation as it limits the search space, and is capable of generating complexities. This reuse and subsequent reduction of the search space leads to an increased performance over direct representations.

Many of the aforementioned authors use some kind of (neural) networks in their representation. However, using a direct encoding for evolving neural networks has a severe scalability problem (Gruau, 1994). For a network with n neurons, the chromosome has to be of size n^2. Generative representations do not have this problem as they can reuse solutions. This limitation of direct representations is also noted by Roggen et al. (2006); they state that when using direct genetic encodings the size of the genetic string grows with the size of the phenotype which leads to large search spaces. In the study by Hornby and Pollack (2001b) using L-systems for the design of physical structures, they show that their generative system evolves faster and produces designs of higher fitness compared to a non-generative EA.

3.4.2 Multi-objective evolutionary algorithms

Exaptation also called *preadaptation* is the shifting of the function of a biological trait. It denotes that a trait originally 'intended' for a particular function can later be adapted to be used for another function. An example of exaptation given by Darwin (1859) is that of the swimbladder in fishes, this organ is used for flotation, with it fish can stay at the current water depth without having to waste energy in swimming. Darwin (1859) states that the swimbladder is 'ideally similar' in position and structure with the lungs of the higher vertebrate animals, hence there is no reason to doubt that the swimbladder has actually been converted into lungs. Another example is that of bird feathers, initially these were used for temperature regulation, but later they adapted for flight.

In (Mouret and Doncieux, 2009a) exaptation is implemented using a multi-objective EA (MOEA). They defined several fitness gradients and a modular genotype-phenotype map, which says how genotypic modules relate to phenotypic ones. Each module is linked to a fitness function, one of the fitness functions is the main objective, which rewards the ability to solve the problem we are interested in. A consequence of the use of a MOEA is that solving each sub-task is not necessary, the evolutionary process is allowed to find alternative paths to the best solution. In (Mouret and Doncieux, 2009b) they also used a MOEA, and they state that evolving a light-seeking robot in a complex arena without first evolving an obstacle-avoidance reflex is hard, this suggests why they use exaptation. Compared to the methods discussed in section 3.3, MOEA is a less strict method as it doesn't enforce a particular order in which capabilities have to be developed.

3.4.3 Coevolution

Biological coevolution is the name for the evolutionary interplay between two or more biological 'objects'. These can be species, or organisms within species (e.g. males vs females). Their individual fitness is a subjective function of their interactions with individuals from coevolving populations (Kicinger et al., 2005a). An example is that of *sexual selection* given by Darwin (1859). In a species, he says, usually the strongest males have the most offspring. Not for every species this is the case, e.g. humans select partners based on a number of factors such as money, social status, personality and appearance. With sexual selection usually females decide with whom they are going to mate, this gives rise to the evolutionary interplay between males and females, called coevolution.

Generally there are two kinds of coevolution, cooperative and competitive coevolution. These variants are discussed in the following two paragraphs.

Cooperative Coevolution

Axelrod and Hamilton (1981) give an interesting example of natural cooperative coevolution: a mutualistic symbiosis, a lichen is composed of two different types of organisms, namely fungus and algae. Both organisms benefit from each other, sometimes their interdependence is so strong that they cannot exist on their own.

From an evolutionary perspective, it is interesting how such a strong cooperative relationship could have come into existence. Both Axelrod and Hamilton (1981) and Dawkins (2006) share an individualistic point of view of Darwin's theory, advocating the 'selfishness' of genes. From this selfish point of view, the emergence of such cooperative behavior is not obvious. However by studying a problem from game theory called the iterated prisoner's dilemma, Axelrod and Hamilton (1981) showed that the best strategy wasn't always to short sighted try to optimize one's owns gain. They found that there are circumstances where cooperation paid off, these findings were further translated to the evolutionary domain, which suggest how natural cooperative coevolution might have begun.

Inspired by this phenomenon, several authors have created algorithms that mimic cooperative co-evolution. According to Maniadakis and Trahanias (2008), an ordinary evolutionary scheme usually underestimates or overlooks the structural nature of the problem. They state that in order to facilitate the exploration of cooperative dynamics, components of the solution should be considered explicitly by the evolutionary process. Coevolutionary algorithms address this issue by evolving each component separately, while also enforcing cooperation upon the components.

The paper by Maniadakis and Trahanias (2005) describes a method for the incremental modeling of robotic cognitive mechanisms. They define two types of neural agents, a 'cortical' agent representing a brain area and a link agent to connect the cortical modules. Two levels are defined, the lower level

containing several populations (or 'species') representing distinct elements which can be reused on the higher level in composite structures. The agents (components) on the lower level consist of the cortical and link agents. The fitness of the system is assessed by evaluating the higher level.

In (Maniadakis and Trahanias, 2008) this method is extended to a hierarchical multilevel architecture. Instead of defining just two levels, multiple levels are possible, each higher level is capable of using components of the lower levels as the building blocks for more complex behaviors.

Potter and De Jong (2000) argue that in order to use evolutionary algorithms for designing more complex systems a division in subcomponents is required. They present an architecture that does this, and which co-evolves the subcomponents as separated cooperating species.

Competitive Coevolution

Natural examples of competitive coevolution are parasites and their hosts (Bull, 1998) and predators and preys both which are involved in a coevolutionary process. Dawkins (2006) calls this process an *arms race*, because it usually results in species continually trying to outdo each other.

This is the essence of what Stanley and Miikkulainen (2004) try to mimic in competitive coevolution. However, interesting strategies will only evolve if the arms race continues for a significant number of generations. Which, they claim, is very difficult in practice as evolution tends to find the simplest solutions that can win. Sims (1994) also used this principle, he states that competition has shown to facilitate complexity, specialization or even social interactions. Another approach is taken by Hillis (1990), they use a model based on parasites in which the solutions coevolve with the test set; both species have an inverse fitness function.

Part II

Model Design

4. Evolutionary Design of a Complex System Classifier

To design a classifier using a complex system and an evolutionary algorithm a model of a complex system based on the theory discussed in chapter 2 is presented. Furthermore, an evolutionary algorithm is designed based on the concepts presented in chapter 3. This chapter aims to give a high level description of the model and algorithm that are used in the remainder of this thesis and chapter 5 gives a more detailed description of the solution. Eiben and Smith (2007) state that when looking for the most powerful natural problem solver, the two most obvious candidates are:

- The human brain
- The evolutionary process (that created the human brain)

Since the human brain is a complex system, this observation is entirely in accordance with our view. In this thesis it is argued that complex systems can be evolutionary designed to act as a solution for classification problems. The two main questions that remain are:

- How can the theory of complex systems be transformed into a practical computer model?
- How can the theory of evolutionary computing be used to design a complex system classifier?

In section 4.1 the first question is treated by giving a model for the implementation of complex systems. The second question is discussed in section 4.2 by presenting an evolutionary algorithm that can design a complex system.

4.1 Complex systems

This section presents a model which can represent a complex system. In section 2.3 complex systems were defined as being both *emergent* and *self-organized*. In section 2.7.3 several design choices were listed, which should result in such a complex system. It was concluded that the following properties are desired when creating a complex system:

- decentralized structure
- semi-homogenous individuals
- interaction via communications

Furthermore, the system needs to have a possibility to interact with the outside world, it needs inputs and outputs. As was stated before, interaction via communications can be established by using a communication network. A network of semi-homogenous individuals is decentralized as long as each individual is autonomous. Such a decentralized network can be represented by a graph. Several graph based representations already exist such as hidden Markov models (HMMs), bayesian networks (BNs), neural networks (NNs) and random boolean networks (RBNs). HMMs and BNs are both probabilistic graphical models, since complex systems are not limited to mere statistics, these models are not adequate to model complex systems.

Neural networks and especially recurrent neural networks (RNNs) are interesting since they are a networked computational model, with semi-homogenous individuals. The choice for RNNs over NNs is

because recurrent neural networks have the ability to memorize values through recurrent loops, which is of great importance for time sequence processing (Pujol and Poli, 1998). An interesting observation on recurrent neural networks is that knowledge in the system can be seen as an emergent property, it emerges from the structure of the network.

A random boolean network is a similar emergent computational model, it is basically a series of interconnected boolean gates such as AND, OR and XOR gates. An important difference with NNs is that RBNs only deal with boolean values, and that each gate (node) can implement a different boolean function.

A novel model that combines properties of both RNNs and RBNs is proposed. This model is called a *computational network* (CN). Computational networks can be seen as a generalization of NNs and RBNs, their properties are compared in Table 4.1. In the table it can be seen that CNs do not have to

	CN	**RNN**	**RBN**
structure	unstructured	structured	unstructured
mathematical functions	diverse	similar	diverse
values	real	real	boolean

Table 4.1: Comparison of three different computational models: computational networks, recurrent neural networks and boolean networks.

be structured in any kind of form, in contrast to the structures of RNNs which are usually designed in advance. Further, both RNNs and RBNs are graphs of interconnected mathematical functions. In RNNs these functions are usually all similar while in RBNs these can be different. Since *semi-homogenous* individuals are required, CNs allow the use of different mathematical functions using real values. RNNs also use real values while RBNs are limited to booleans. An example of a computational network is shown in Figure 4.1. A node can be defined as a mathematical function $f(x_1, x_2, .., x_n)$ with a number

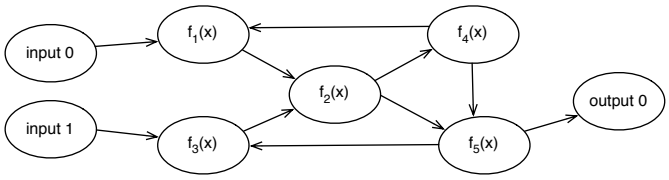


Figure 4.1: Example of a computational network, with two inputs and one output, all weights and parameters are omitted.

of n inputs and one output, each node can implement a different function.

Why are CNs chosen over BNs and RNNs? As said above, CNs are a generalization of RNNs and BNs, while RNNs usually allow just one type of function in the network, a computational network does not pose restrictions on the functions. For example, consider a classification problem which looks for the minimum value of 10 real input values. In a computational network this can be represented using one single node containing the *min* function. When using neural networks at least an array of neurons with a sigmoid function is required to represent the same function, this is obviously much more complex compared to the CN solution. This means that computational networks are more expressive, more advanced computations can be done with a small number of nodes, thereby reducing the solution space. Furthermore, this particular problem cannot be solved by using boolean networks since it involves *continuous* values, boolean networks are in this sense 'unrealistic' as it is hard to describe the world in terms of true or false. The mathematical functions that are used in computational networks are listed in section 5.2.1.

An important aspect of computational networks is the order of computation, circular dependencies are allowed, as it allows recurrent loops, which can be seen in Figure 4.1. Which node is computed first? As a node can be executed when all input values are set, circular dependencies pose a problem. When no executable node can be found, the first node that is found which is directly connected to an executed node (or input) is executed. For example in Figure 4.1 the first node to be executed would be node $f_1(x)$, even though not all its input values are set. When an input value is not set, the value of the previous time step is used instead. A more detailed description of this procedure is given in section 5.2.2.

Earlier it was said that complex systems are comprised of *autonomous* entities, as such, the chosen encoding should also have autonomous individuals. Nodes and their mathematical functions are in their simplicity not first to come to mind when thinking about autonomy. However, it is clear that their behavior is not dictated by some outside force. As such they can be considered autonomous.

Furthermore it has been proven that recurrent neural networks are equivalent to a universal Turing machine (Hyötyniemi, 1996). The same holds for a type of cellular automata (CA), which are a subclass of boolean networks (Kennedy et al., 2001). A universal Turing machine is capable of simulating any other Turing machine from the description of that machine (Sipser, 2006). Because the concept of a Turing machine forms the basis of the modern computer, this basically means that recurrent neural networks and boolean networks can compute anything a computer can. As such, it is safe to assume that computational networks are also universal Turing machine equivalent. This means that a computational network is at least *capable* of representing a good classifier. Since RNNs have proven to be powerful classifiers, and CNs are a generalization of RNNs, it is expected that CNs are good classifiers themselves. A classifier can be described as a mathematical function which divides a problem space into several subspaces (classes), since a computational network is in essence a set of interconnected functions, it is appropriate to use it as a classifier.

4.1.1 Model summary

Table 4.2 shows the summary of the model for a complex system. The next section describes how this model can be designed using an evolutionary algorithm.

Component	**Type**
Representation	unstructured network
Values	real valued
Nodes	diverse mathematical functions
Weights	real valued

Table 4.2: Summarized description of the model of a complex system, called *computational networks*.

4.2 Evolutionary design

Now that we have established how a complex system will be modeled we can focus our attention on the design task. For designing neural networks there are several methods available. However, most methods do not work with the recursive variant as it is more difficult to design RNN's (Pujol and Poli, 1998). It has been argued before that evolution is capable of designing complex systems, since this is how natural complex systems have originally emerged. This is why we turn to artificial evolution for the design of complex system classifiers.

Evolutionary computing mostly focuses on optimization problems, suggesting room for improvement for evolutionary design tasks. In this section the general outline of an evolutionary design algorithm is presented and several directions for further research are chosen in which evolutionary design might possibly be improved.

4.2.1 Outline

In order to be able to design a complex system, a purpose needs to be specified. In this thesis the purpose is to design a complex system which acts as a classifier. The purpose of a classifier is to predict (classify) the class of a *sample* based on its input features. A classifier is usually trained on a dataset, for each sample in the dataset a number of features (input values) and the expected class (output value) is available. A fitness score can be computed using the number of correct classified samples. In the evolutionary algorithm an individual is a classifier (a complex system), each generation the individuals are evaluated by testing them on the samples. In Figure 4.2 the outline of the proposed algorithm is shown, note that this image corresponds with Figure 3.1. In chapter 5 a more detailed description of the algorithm is given.

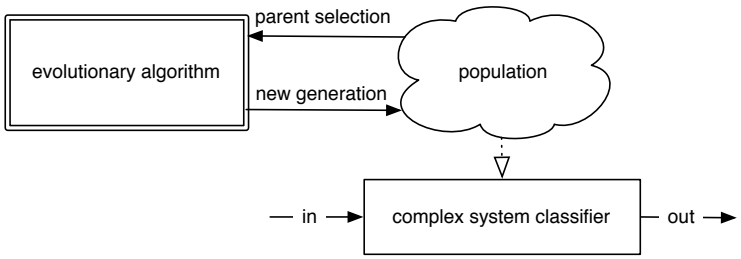


Figure 4.2: Outline of the EA that designs complex system classifiers, it is shown that each generation the population of classifiers is modified by the evolutionary algorithm.

4.2.2 Selected components

Section 3.3 and 3.4 describe several possible improvements for evolutionary design. Each of these improvements make a number of assumptions regarding evolutionary design, an important goal of this research is to determine whether these assumptions are justified. These assumed solutions bound the problem space with the goal of making evolution more efficient. However, it could be that these solutions transform the problem space in an undesirable way. It is difficult to find a balance between bounding the problem space and removing assumptions. Too many assumptions can lead to a problem space which doesn't contain a desirable solution, too few assumptions can lead to an immense problem space which does contain a desirable solution but with a chance of finding it approaching zero. In other words:

> "*Evolution like all search algorithms, is limited and constrained by the representation it can modify.* "
>
> — Bentley and Corne (2002)

In Table 4.2.2 a list of possible evolutionary design improvements is shown, next to each concept is the name of its related algorithm component. In Figure 4.3 a more detailed outline of the algorithm is

Evolutionary concept	**section**	**Algorithm component name**
Smooth Fitness Function	3.1.2	fitness distance metric
Incremental Structure Evolution	3.3.1	boosting
Increased Complexity Evolution	3.3.2	increasing complexity
Generative Representation	3.4.1	embryogeny
Multi-Objective Evolutionary Algorithms	3.4.2	increasing complexity
Coevolution	3.4.3	-

Table 4.3: A listing of evolutionary concepts which were discussed in chapter 3 with their corresponding component name which is used in the remainder of this thesis.

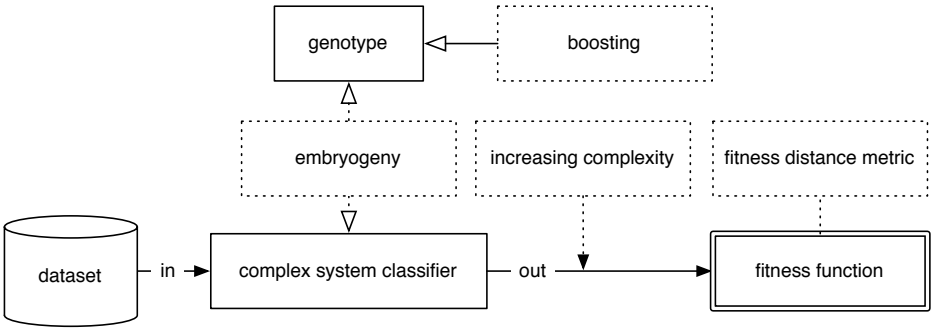


Figure 4.3: Illustration of the evolutionary algorithm that designs a complex system classifier, the dotted blocks indicate algorithm adaptations that are researched in this thesis.

shown, also displaying the algorithmic components which usefulness is researched. The illustration shows that the increasing complexity component modifies the input of the classifier and the fitness distance metric is a variant of the fitness function. Boosting operates on the genotype and the embryogeny component is a mapping between genotype and phenotype (the classifier).

4.2.3 Genotype

Two different representations are proposed, a 'default' representation which is a direct representation of the computational network and a generative representation called *embryogeny* which is a mapping. Figure 4.4 shows an example of the direct representation, Figure 4.5 shows the same phenotype represented in a genotypic representation called embryogeny. The *rules* in Figure 4.5(b) are used to substitute the nodes

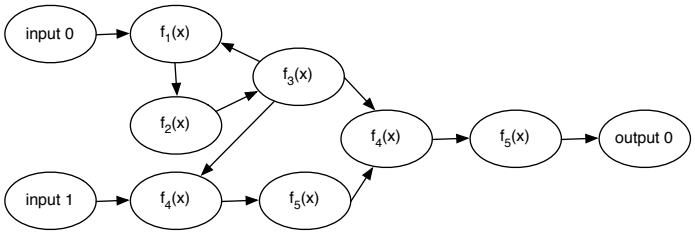
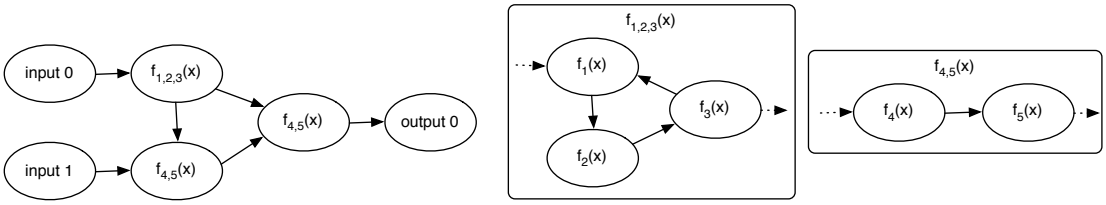


Figure 4.4: Direct (phenotypic) representation of a computational network.



(a) seed (b) rules

Figure 4.5: Genotypic representation (embryogeny) of a computational network.

.

with the same name in the *seed* in Figure 4.5(a). The seed and the rules produce the same computational network presented in Figure 4.4. A more detailed description of the embryogeny algorithm is given in section 5.3. In Table 4.4 a characterization of both representations is shown using the attributes which were listed in section 3.1.1. As these two representations both encode computational networks, they can

Attribute	**default representation**	**embryogeny**
EA level	phenotypic	genotypic
Structure	non-linear	
Length	variable	
Change during evolution	dynamic	
Encoding scheme	direct	indirect
Accuracy of solution specification	parameterized	
Ability to reuse encoding	non-generative	generative
Genotype-phenotype correspondence	-	explicit

Table 4.4: A comparison of the two proposed representations.

be compared to each other by conducting experiments. The advantage for using an embryogeny is the earlier mentioned use of 'building blocks'. These building blocks are implemented using the replacement rules visualized in Figure 4.5(b), which allow the reuse of substructures throughout the network.

4.2.4 Boosting

The boosting method is inspired by Kearns (1988) who researched whether a group of weak classifiers can be used to create a single strong classifier. It is similar to *incremental structure evolution*. However, it differs in the way Pasemann et al. (2001) implement it. They use predefined structures, while boosting aims to do this on the fly. In Figure 4.6 a computational network is shown just after it was boosted. The blue components already existed before, the black nodes are newly added. After the boost, the blue nodes are no longer changed, it is a frozen part of the computational network. However, the black part of the network can (re)use parts of the blue network. In section 5.4 boosting is discussed in more detail. This method has obvious potential, as it can recombine partial solutions into higher level (partial)

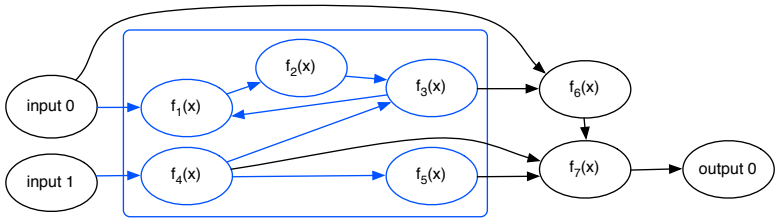


Figure 4.6: Boosting example

solutions, which can theoretically result in increasingly more complex solutions. Boosting can be applied on a conditional basis, for example after a number of generations or when a specific fitness level has been reached.

4.2.5 Fitness distance metric

Using a fitness distance metric is likely to be beneficial as was explained in section 3.1.2. However, this function is dependent on the problem domain. Creating it can be a non trivial task. It is possible that it transforms the search space in a less beneficial way, which is why its performance should be tested against an absolute fitness function.

4.2.6 Increasing complexity

This component combines features of *increased complexity evolution* and *multi-objective evolutionary algorithms*. It tries to divide the search space in several natural stages. The stages should be in order of increasing complexity, where a solution for each previous stage can be used for the next. This is similar to the concept of preadaptation which can also be used in MOEA's, as was discussed in section 3.4.2. As

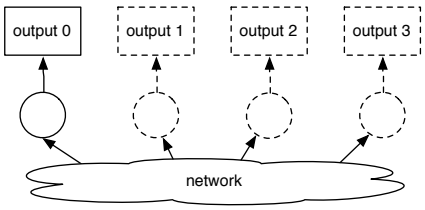


Figure 4.7: Increasing complexity example, showing a problem domain which requires four outputs, in the current view only one output is used, the other outputs can be used in subsequent phases.

can be seen in Figure 4.7, this component changes the number of outputs which are used for the fitness function, in a sense it is a hierarchic fitness function which gradually increases the fitness pressure.

4.2.7 Coevolution

In Table 4.2.2 coevolution is not related to an algorithm component. The benefits of cooperative coevolution would be that several separate components are evolved that cooperatively solve the problem at

hand. However, two cooperating computational networks can also be represented as one single network, the only thing that coevolution adds is the hard coded separation of the networks. But this also comes with a computational overhead, since this requires two evolutionary populations operating in tandem. For this reason it was chosen not to investigate coevolution in this thesis.

5. Detailed design

This chapter describes the algorithm and the model introduced in the previous chapter in more detail. First the architecture of the system is briefly discussed in section 5.1, then the computational network is discussed extensively in section 5.2. Subsequent sections elaborate on the components of the EA that is used. In section 5.9 a few details about the development process are presented and in section 5.10 the chapter is finished with a discussion of the validity of this design.

5.1 Architecture

In Figure 5.1 a high level class overview is shown with only the most important classes displayed. The function of each class is discussed in this chapter. An `CNIndividual` consists of exactly one `Genome` with a corresponding `CNFitness`. The `Genome` consists of one seed (a `ComputationalNetwork`) and multiple `Rule` instances, where each `Rule` contains two `ComputationalNetworks`. A `ComputationalNetwork` is a network of `Nodes`, each of a certain `NodeType`.

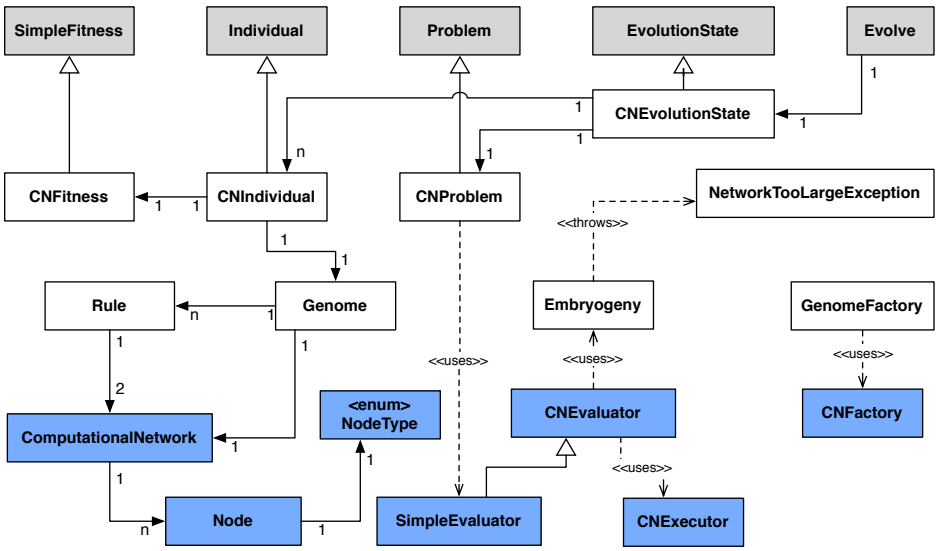
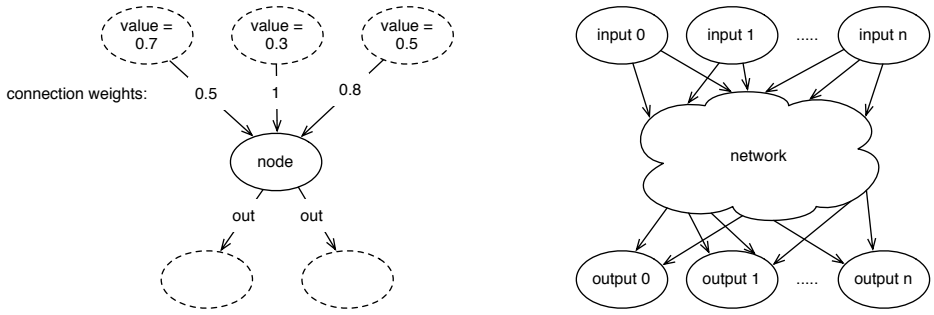


Figure 5.1: Class diagram showing only the most important classes. The gray colored classes are classes from the ECJ library, the white classes are evolution related and the blue classes are computational network related. This coloring also indicates the Java package organization of these classes.

The `CNEvaluator` evaluates `CNIndividuals`, which are supplied to it. If needed the representation is first grown using the `Embryogeny` class, subsequently it is executed using `CNExecutor`. `GenomeFactory` and `CNFactory` are two of the most important utility classes. They deal with genome and neural network operations respectively.

5.2 Phenotype: computational network

As we will see the `ComputationalNetwork` class plays a double role in the implementation, it is used both for the phenotype as well as in the genotype (discussed in section 5.3). This section focusses on it's role as a phenotype. The class consists of a number of `Node` instances each with a `NodeType`. In Figure



(a) node close-up (b) example network

Figure 5.2: Illustration of a node and its connections (a) and of an example network (b).

5.2(a) a node with three incoming connections and two outgoing connections is shown. The values in the striped nodes indicate the incoming values, the values at the center of the line indicate the weight of the connection, the values can be both real and integer values. In Figure 5.2(a) an example of a computational network is shown. The input of a node is called its *activation*. In the following example the *activation* for the center node in Figure 5.2(a) is computed:

$$\begin{aligned} activation &= \{\forall\ con \in connections : (con\ value \cdot con\ weight)\} \\ &= \{0.7 \cdot 0.5,\ 0.3 \cdot 1,\ 0.5 \cdot 0.8\} \\ &= \{0.35,\ 0.3,\ 0.4\} \end{aligned} \tag{5.1}$$

5.2.1 Node types

There are 10 different types of nodes (`NodeType`). Each node has a *weight* and can use the *activation* values as described above, which are shown in the following equations:

Type	Function	
SIGMOID	$\frac{1}{1+e^{-t}}$, where $t = \frac{\sum activation}{weight}$	(5.2)
LINEAR	$\left(\sum activation\right) \cdot weight$	(5.3)
STEP	$\begin{cases} 0 & \text{if } \sum activation < weight \\ 1 & \text{if } \sum activation \geq weight \end{cases}$	(5.4)
SUM	$\sum activation$	(5.5)
POWER	$\left(\sum activation\right)^{weight}$	(5.6)
SIN	$\sin\left(\left(\sum activation\right) \cdot weight\right)$	(5.7)
E	$\left(\sum activation\right) \cdot e^2 \cdot \pi \cdot weight$	(5.8)
MIN	$\min\left(activation\right)$	(5.9)
MAX	$\max\left(activation\right)$	(5.10)
DELAY	$\sum activation^{t-1}$, where $t =$ the current time	(5.11)

These 10 different types were chosen to ensure a wide variety of available functions. The rationale behind this is that evolution will choose the appropriate node types, while avoiding undesired types.

5.2.2 Execution order

An important property of the network is the order in which the nodes are computed, the so called execution order. Figure 5.3 shows an example of a network, the number of each node indicates its position in the execution order. Figure 5.4 shows how the execution of a network can be parallelized, while this

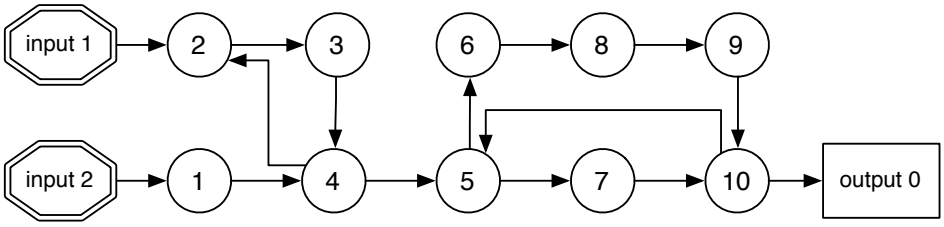


Figure 5.3: Example network showing only connections between nodes, the index of the node indicates the order in which the values of the nodes are set.

is a relatively small network, it is expected that bigger networks can be even further parallelized. Figure

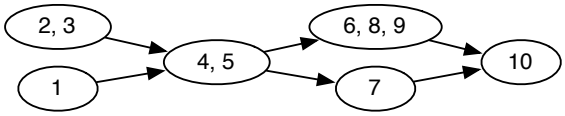


Figure 5.4: Example how the execution of a network could be parallelized.

5.5 shows the pseudo code of the algorithm that computes the order in which the nodes are computed, this is implemented in `CNExecutor`.

```
1. while there are unexecuted nodes do
2.     find executable nodes
3.     if no executable nodes then
4.         find nodes directly connected to the executed nodes
5.         force execution of the first found node
6.     else
7.         execute executable nodes
8.     end if
9. end while

a node is executable when all of its activation values are set
```

Figure 5.5: Pseudo code for the algorithm that determines the node execution order.

5.2.3 Execution output

In Figure 5.6 a very simple example of a network is shown. This network can be described by the following equations:

$$node\ 1 = ((0.7 \cdot input\ 1) + (0.2 \cdot input\ 2))^2 \quad (5.12)$$

$$node\ 2 = min((0.6 \cdot input\ 1), (0.3 \cdot input\ 2)) \quad (5.13)$$

$$node\ 3 = ((0.1 \cdot node\ 1) + (0.9 \cdot node\ 2)) \cdot 3.2 \quad (5.14)$$

Note that the value of *node* 3 is the output value. Executing these equations in the correct order results in a correct output, this is in essence how `CNExecutor` executes a `ComputationalNetwork`.

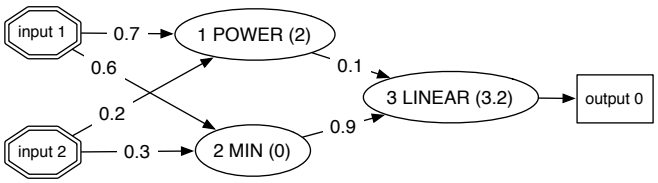


Figure 5.6: A simple example of a computational network.

5.3 Genotype

In section 3.1.1 it was stated that two different genotypes were implemented, a *phenotypic* and a *genotypic* representation. The phenotypic representation is as the name implies, a direct representation of the phenotype presented in the previous section.

This is contrary to the genotypic representation which is a mapping to a phenotype, it grows the genotype into a phenotype. This particular generative representation is called 'embryogeny' in this thesis.

5.3.1 Embryogeny

The mapping process is implemented in `Embryogeny`. The genome is contained in the `Genome` class. The genotype consists of a `ComputationalNetwork` called the *seed* and a set of substitution rules (`Rule` instances). In the genotype a special node type is available called 'abstract'. Abstract nodes are used as references for the substitution rules. Each rule consists of an abstract condition node and a substitution network. If a node of the condition node type is found in the seed it is replaced with the substitution network in the rule.

A very simple example of this principle is illustrated in Figure 5.7. In this figure, the genotype is shown consisting of one rule and a seed containing only one node. This seed node is abstract and has type 0, this corresponds to the condition part of the rule, meaning that this abstract seed node is to be replaced by the substitution network of rule 0. In Figure 5.7(c) the resulting computational network (phenotype) is shown, the red box indicates the substitution of rule 0.

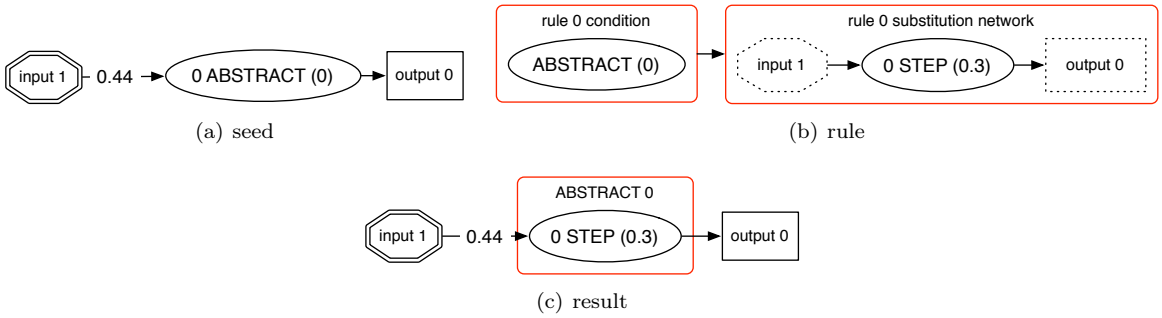
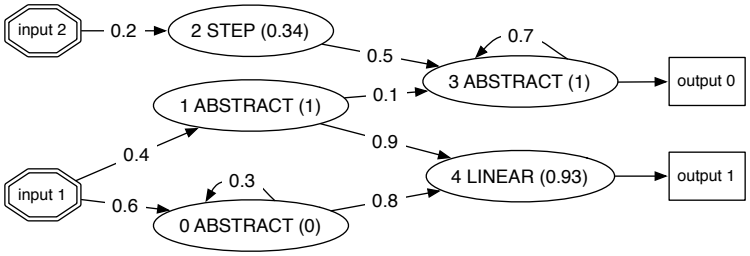


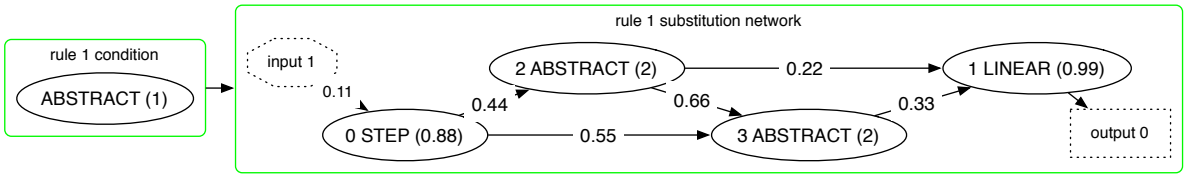
Figure 5.7: An example of a very simple genotype consisting of the seed (a) and just one rule (b). The corresponding phenotype is shown in (c).

In Figure 5.8 a more complex example of a gene is shown. The network it generates (its phenotype) is shown in Figure 5.9. As can be seen, the seed consists of five nodes, three of them are abstract. There are also three rules, interestingly, rule 0 and rule 1 both contain an abstract node in their substitution network. This shows how components can be reused throughout a genotype, this is especially clear in Figure 5.9, here the colored boxes around the nodes indicate which rule produced that particular subnetwork. It can be seen that the embryogeny facilitates the reuse of multiple levels of subnetworks, this is a kind of hierarchical representation. Through this reusage of nodes, relatively small genotypes can encode enormous phenotypes. In the course of some tests phenotypes consisting of over 6 million nodes have been observed which were generated from less than 20 rules containing a total of just 200 nodes.

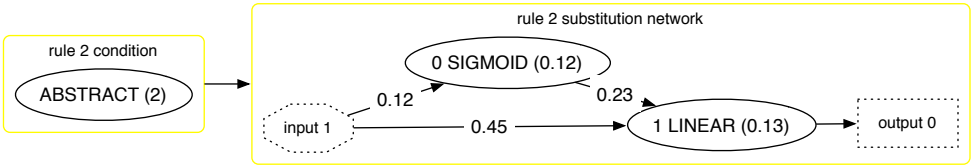


(a) seed

(b) rule 0



(c) rule 1



(d) rule 2

Figure 5.8: In (a) the seed of a more complex example is shown, and in (b), (c) and (d) the rules are shown.

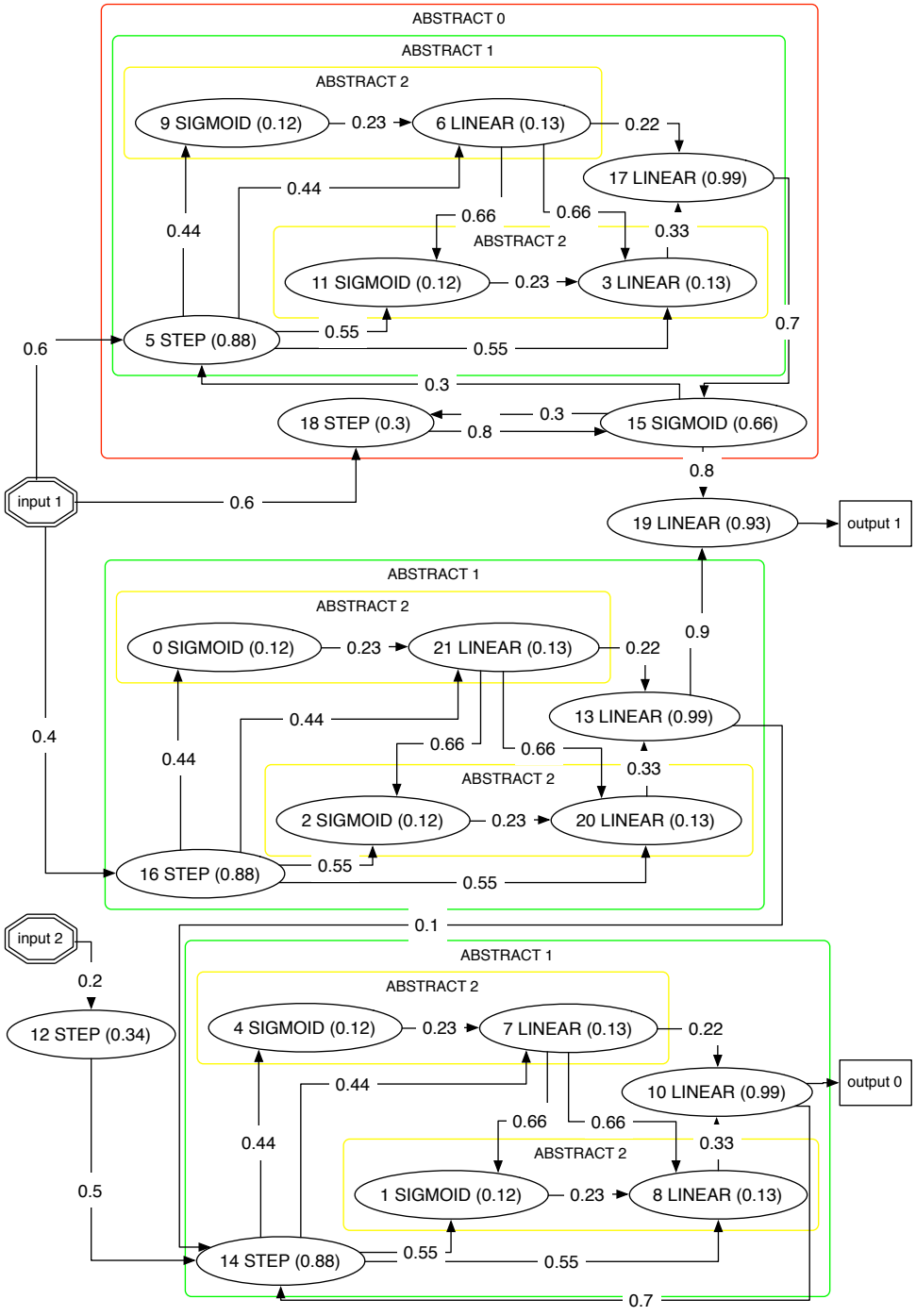
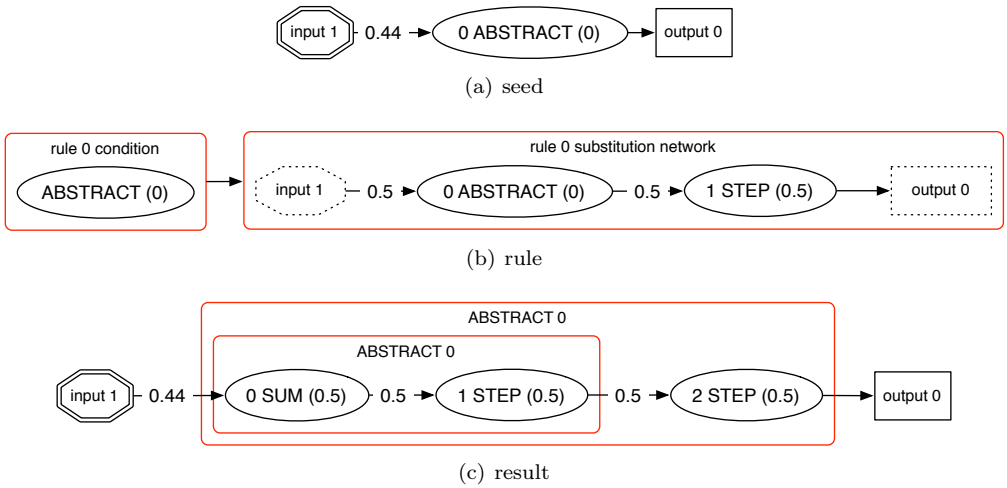


Figure 5.9: This image shows the resulting phenotype generated from the genotype illustrated in Figure 5.8, note that the colored rectangles correspond to the rules in the genotype.

5.3.2 Recursion

An important consequence of this embryogenic representation is that self recursion is also possible. The representation permits that a rule subnetwork contains a reference to itself. To avoid infinite recursion a restriction is applied. The recursion depth is not allowed to go beyond 2, this allows exactly one copy of itself in itself. When this limit is reached, any subsequent abstract nodes of this type are replaced by a node of the type SUM.

This is illustrated with an example of self recursion in Figure 5.10. In Figure 5.10(b) it can be seen that the rule replaces abstract node of type 0 with two nodes, one of which itself is an abstract node of type 0. In Figure 5.10(c) the resulting phenotype is shown, it can be seen that the substitution stopped at depth two, the node labeled "0 SUM (1.0)" replaces the abstract node at this recursion depth. A SUM node with weight 1.0 was chosen as this is the most neutral substitution, because it simply passes its inputs to its outputs. This can also be seen in the aforementioned example, here the SUM node doesn't change the signal for the subsequent node.



(a) seed

(b) rule

(c) result

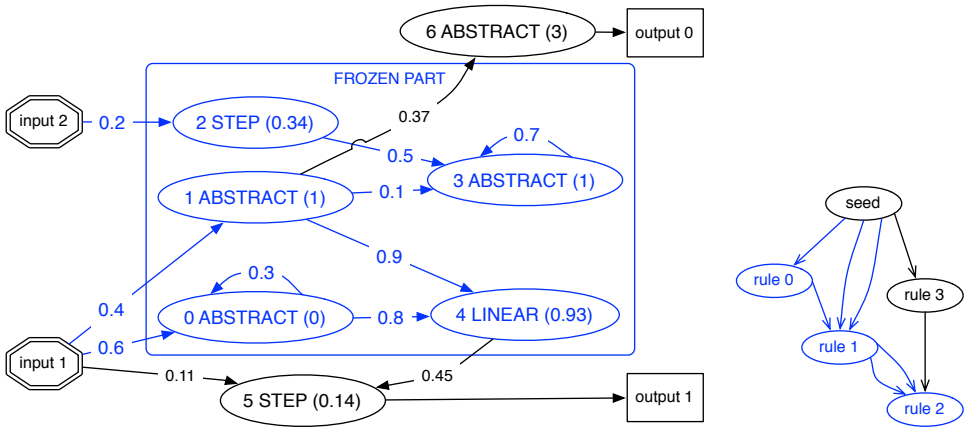
Figure 5.10: An example of a genotype that uses self recursion.

5.4 Boosting

As was explained in section 4.2.4, boosting is an incremental structure increase. After a number of generations (or generally any other imaginable condition) the boosting process can be initiated. Of all individuals (genomes) in the population at that particular time, the one with the best fitness is selected. This best individual is used as the starting point for the *entire* next generation. Every new genome contains this best genome and some randomly created rules are added. Also a number of randomly created nodes are added to the seed. The rules and part of the seed that belongs to the best genome are frozen, that means that they are not changed by the variation parameters, they remain constant for the rest of the evolutionary run.

An example of this process is given in Figure 5.11. Note that the frozen part of this genome (drawn in blue) is derived from the genome in Figure 5.8. In Figure 5.11(a) it can be seen that the original network remains untouched, and that there are two newly added nodes which hook into the frozen part. In Figure 5.11(b) an outline of the rules is shown, the frozen rules are drawn in blue, one new rule (rule 3) is added that reuses some of the frozen rules.

As can be seen from the example given above, this particular use of boosting is different from the original concept of boosting in a number of ways. An important difference is that the original concept uses several distinct classifiers which are combined. In contrast, boosting as discussed in this section fuses two classifiers during the construction phase.



(a) seed (b) rules

Figure 5.11: Boosting example, the blue parts are 'frozen', they cannot be changed. In (a) the seed is shown containing frozen (blue) and active (black) nodes and connections, in (b) an outline of the rules in the genome is shown.

5.5 Fitness function

As was discussed in section 3.1.2, smoothness is an important property of a fitness function. In general, two different fitness functions were implemented, one 'default' and one called 'fitness distance metric', where the first is non-smooth and the latter is smooth. Because the fitness function is problem dependent, the exact implementation for each problem will be discussed in chapter 6.

5.6 Increasing complexity

This component divides the evolutionary process in a number of stages, in each stage the number of outputs in the solution is increased. Generally, the first stage starts with a relatively low number of outputs, this is indicated by the *output-amount-init* parameter. The next stage can be reached after a certain fitness level has been reached or after a number of generations. In Figure 4.7 the concept of increasing complexity is illustrated. In this example the problem requires four outputs, but in the first stage only one output is used (output 0). In subsequent stages additional outputs are connected to the network (as is illustrated by the striped nodes and connections). This adding of new outputs increases the evolutionary pressure as well as the complexity (hence the name). In the first phase, all effort can be put in solving a subproblem, in subsequent phases additional subproblems are added until the entire problem is targeted. It is expected that once a subproblem is reasonably solved it might be partially reused for the solution of the following subproblems, similar to preadaptation in natural evolution.

5.7 Variation operators

To introduce variation in the population, two variational operators were designed that modify a genome. The two operators, mutation and crossover, are discussed in this section.

5.7.1 Mutation

The mutation operator consists of eight different mutation variants. Some of these variants affect the entire genome while others only affect a small part of the genome. These different mutation variants to not exclude each other, multiple of these mutations can occur during one mutation operation, except for the reset mutation. The mutation variants that modify nodes and connections operate on both the seed as on the rule substitution networks.

Reset The reset mutation destructively creates an entire new genome. This resets both the seed as all rules in the genome. This mutation variant makes any further mutations during the same mutation operation unnecessary, as such the reset mutation prohibits any other mutation variant.

Add rule If the current genome has less than five *unused* rules, this mutation variant adds a new rule to the genome. To increase the effectiveness of this new rule, a node of the same type as the condition node of the rule just added is added to the genome seed. The restriction of the unused rules is deliberately added to avoid the occurrence of bloat. Bloat is the phenomenon where the average representation size grows over time, sometimes called the "survival of the fattest" (Eiben and Smith, 2007). Over time, bloat can significantly hinder the evolutionary process.

Remove rule This removes a rule from the genome. Any existing node of the same type as the condition node of the rule that is removed is replaced by a SUM node with weight 1.0.

Node weight This mutation variant either increases or decreases a node weight by 0.01.

Connection weight Similarly, this mutation variant either increases or decreases a connection weight by 0.01.

Connection struct Either adds or removes a connection between two nodes in the network.

Add node This mutation variant adds a node of a randomly chosen type. The possible types are the types discussed in 5.2.1 and the abstract type discussed in section 5.3. For every newly added node one incoming and one outgoing connection is added as well. These connections are added to increase the likelihood that the newly constructed node is connected to the network in a useful way. This mutation variant is illustrated in Figure 5.12. In this image the dotted components are added by an add node mutation.

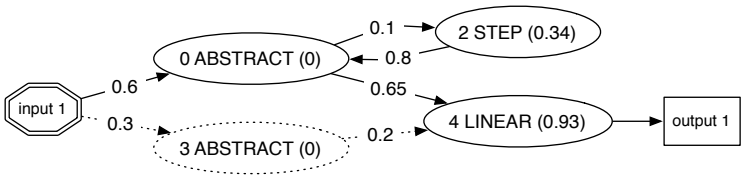


Figure 5.12: Example of an add node mutation. The dotted node and connections are newly added.

Remove node Removes a node and all its incoming and outgoing connections.

5.7.2 Crossover

The crossover operation is designed such that it only exchanges rules between two genomes, as such it is only applicable when *ebmryogeny* is used. The pseudocode of this operation is shown in Figure 5.13. In section 3.1.3 the notion of *respect* was introduced, it says that any information carried in both parents should also be present in the offspring. A moment thought suggests that this notion is very difficult to keep with the embryogenic genotype. It is hard to determine what 'information' is present in both parents, this is because the function of single nodes, or rules, is very dependent on the connections it has. This is closely related to the emergent property of neural networks, when transferring parts of a network to another network it is unlikely that they function in the same way.

1. **crossOver**(*thisGenome*, *otherGenome*)
2. A = randomly select $\frac{1}{2}$ the rules of *thisGenome*
3. B = randomly select $\frac{1}{2}$ the rules of *otherGenome*
4. *newGenome* = construct a new genome using $A \cup B$ and the *seed* of *thisGenome*
5. *lonelyNodes* = select every abstract node in *newGenome* that is not linked to a rule
6. **for** every node *abs* in lonelyNodes **do**
7. randomly set the weight of *abs* such that it links to a rule in *newGenome*
8. **end for**

Figure 5.13: Pseudo code for the crossover operation.

5.8 Selection

For parent selection *tournament selection* is used. Tournament selection repeatedly applies the following process: it selects n individuals of the population and adds the best one to the set of parents. This process is continued until the required number of parents is reached. Generational evolution with elitism is used. This means that the best n individuals are preserved in the population while the others are replaced. Elitism ensures that the maximum fitness never declines.

5.9 Development

This section gives a very brief description of the development process.

5.9.1 ECJ

As was mentioned earlier this chapter, the ECJ toolkit (Luke et al., 2010) was used. ECJ was chosen for its excellent multi-threaded design and very flexible architecture. The usage of ECJ made it possible to immediately focus on the actual problem instead of having to build an evolution environment first. The choice for ECJ also implied the programming language to be Java.

5.9.2 Google Collections Library

The Google Collections Library (Bourrillion and Levy, 2009) is a Java library that extends the Java Collections Framework. It provides numerous utilities and data structures which proved to be very convenient during development, especially for the `Embryogeny` class.

5.9.3 Testing

In total about 10.000 lines of code were written. During development, test classes were written for the most critical parts of the application. This made sure that the code works exactly as was specified, this was a crucial part in the development of the `Embryogeny`, `GenomeFactory` and `NNFactory` classes. Furthermore Java *assert* statements were used in many parts of the code. These statements check values, if an input doesn't match what is expected it throws an error. In other words, fail early, fail often. If these classes would not have been tested properly, and a bug would exist in the code, it would probably not have been found. Instead, the bug would probably have been blamed to be the source of the evolutionary process not working correctly. Test driven development was used from the start, which made the development of the model part effortless.

5.9.4 Command line utilities

To keep track of the progress of the evolutionary process several tools were used. The *dot* language was used for automatically creating graphs of neural networks, the graphs were drawn with the Graphviz

library Bilgin et al. (2010). At the end of an evolutionary run a report consisting of several performance graphs was automatically created using *gnuplot* (Williams and Kelley, 2010).

5.10 Validity

In section 3.1.1 a list of requirements for successful representations is listed, this section investigates whether the implementation presented in this chapter meets these requirements.

5.10.1 Non-redundancy

The direct representation is clearly non-redundant, there is an exact 1-to-1 mapping. For the embryogenic representation this is not the case. Different genotypes can theoretically encode the same phenotype. The most simple example of this is the following: consider a single node phenotype. A direct representation is possible, but also a representation using one rule containing that single node is possible. However, an advantage of the embryogenic representation is that it can encode very large phenotypes with a small number of rules. This advantage seem to outweigh the violation of this requirement, and if it does not, it is likely that a comparison between the two representations will show that embryogeny is inadequate.

5.10.2 Legality

Since the representation is basically an unordered set, an *illegal* solution is very hard to construct. There is, nonetheless, a (hypothetical) way to create an illegal solution for the embryogeny representation. This can be done be putting an abstract node in the seed without a corresponding rule. However, this scenario is explicitly avoided in the remove rule operation (section 5.7.1). Also, there exist several network configurations which might look illegal, but which really aren't. For example, a neural network containing dead ends, or a network without any connections is still legal, however, such networks will probably not be useful.

5.10.3 Completeness

For both the direct and embryogeny representation it is clear that the *completeness* requirement is met. For every phenotype there exists a corresponding genotype because there is always a direct representation, and since the embryogeny representation is in fact a *superset* of the direct representation it is also complete.

5.10.4 Lamarckian property

This requirement particularly concerns the crossover operator and as such the embryogeny representation. The Lamarckian property says that the meaning of alleles of a gene should not be context dependent, this is partially held for the crossover operator. The crossover operator exchanges rules, within rules most of the information is kept intact (except possibly for any abstract nodes in these rules). However, because the rules interact in the creation of the phenotype, they are in fact context dependent. This seems to be a consequence of the choice for a generative representation. In short, the Lamarckian property is not held completely but it is still possible for offspring to inherit 'goodness' from its parent (in the form of good rules).

5.10.5 Causality

The requirement of the *causality* or *continuity* of the search space is kept in the direct representation, every mutation in genotype space results in the exact same variation in phenotype space. However, for the embryogeny representation this doesn't necessarily hold, consider a mutation in a rule which abstract node occurs in several places in the genotype, now this single mutation has effect in every occurrence of this rule in the phenotype. This means that small variations in the search space for the embryogeny representation can result in large variations in the phenotype search space. However, this is also an incredible advantage of the embryogeny representation, as was discussed in section 5.3.

Part III

Experiments

6. Experimental Design and setup

This chapter describes the design and setup of the experiments that are conducted with the model and algorithm presented in the previous chapters. The experiments are setup in a way that the results will likely lead to answers on the research questions presented in section 1.2. These questions can be answered by examining the hypotheses of section 1.3. In this chapter, the design and setup of the experiments are described. Chapter 7 presents the results of these experiments, and in chapter 8 the results are analyzed and the hypotheses are investigated.

Before the main experiments can be conducted a tuning experiment is done (section 6.1). The outcome of this experiment is then used for the component experiment (section 6.2). The component experiment finds the best component setting which is then used for a performance experiment which design is described in section 6.3.

6.1 Tuning experiment

The tuning experiment is conducted to find the optimal values for several evolutionary parameters. For each parameter various values are tried, the best parameter setting is used for subsequent experiments. Each parameter setting is tested for a number of runs with different seed values for the random number generators.

6.1.1 Independent variables

The variables that are varied (called the independent variables) are the following:

crossover probability	This is the probability that the crossover operation (as explained in section 5.7.2) takes place on an individual.
mutation probability	This is the probability that the mutation operation (as explained in section 5.7.1) takes place on an individual.
reset probability	This is the conditional probability $P(reset\|mutate)$, it indicates the probability that an individual is reset given that this individual is mutated.
node weight mutation probability	$P(node\ weight\ mutation\|mutate)$ indicates the probability that a node weight is mutated given that this individual is mutated.
connection weight mutation probability	$P(connection\ weight\ mutation\|mutate)$ is a conditional probability that indicates the probability that a connection weight is mutated given that this individual is mutated.

6.1.2 Dependent variables

The datasets are split in a train and a test part, the train set is used during the evolutionary run, the test set is used after the evolutionary training phase is finished. For each experiment there are two measurements, these are:

train score	This is the score of the best individual found in the evolutionary run on the train set.
test score	This is the score of the best train individual on the test set.

6.2 Component experiment

The component experiment is designed to test the selected components from section 4.2.2. The aim is to find which of these components improve the solution and which of them does not. The results will also be investigated using statistical tests to see whether the components have a significant effect on the results, this is described in chapter 8.

6.2.1 Independent variables

As was mentioned earlier, the algorithm consists of a number of independent components which can be applied independently of each other. These components are the independent variables in the experiments, where each component can either be turned on or off. The components are:

embryogeny	This component is either turned on or off, it's conceptual working is described in section 5.3
boosting	This component is either turned on or off, see section 5.4.
increasing complexity	This component is either turned on or off, see section 5.6.
fitness distance metric	This component is either turned on or off, see section 5.5.

6.2.2 Dependent variables

Again the following scores can are used as dependent variables:

train score	This is the score of the best individual found in the evolutionary run on the train set.
test score	This is the score of the best train individual on the test set.

6.2.3 Other variables

For the boosting and the increasing complexity components there are several influential variables:

boost-interval	Every time this interval (in generations) has elapsed, boosting takes place.
max-boost	This is the maximum number of boosting operations that takes place.
output-amount-init	The number of outputs that are initially used.
output-amount-step	The number of outputs that are added each time the threshold is reached.
output-amount-step-threshold	The fitness threshold. When the current best fitness value reaches the threshold, additional outputs are added.
output-amount-time-threshold	The time threshold. When the number of generations exceeds this value, additional outputs are added.

6.3 Performance experiment

The performance experiment uses the best settings for a dataset (as established in the previously described experiments). The result of this experiment is compared to the results of the classifiers of the Weka toolkit (Hall et al., 2009) on the same dataset. The goal of this experiment is to see the performance of the proposed method relative to that of state of the art classifiers. This means that for this experiment the *independent variables* are the classifiers themselves, the *dependent variable* is the train score.

6.4 Datasets

This section describes the datasets which are used for the experiments, and it describes implementation details which are specific for each dataset.

6.4.1 Banana set

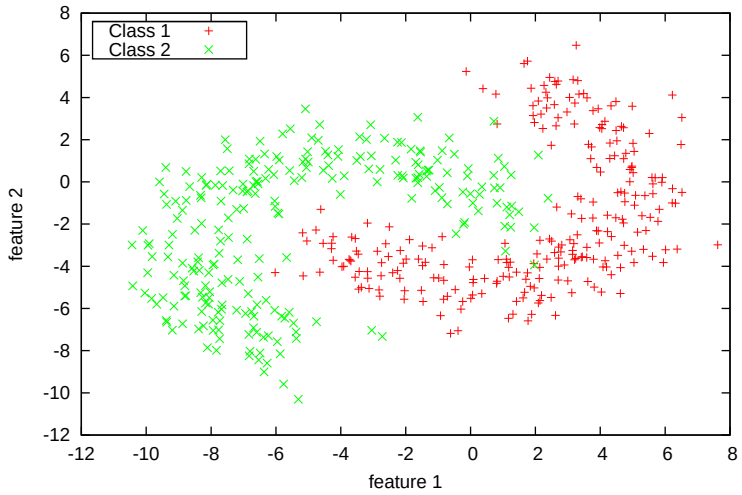


Figure 6.1: Banana set visualization, showing two classes defined in two features in a banana like shape.

With only two features and two classes the banana set is the easiest problem that is considered in this thesis. However, Figure 6.1 shows that the two classes cannot be separated by a simple linear classifier. Given two real values (feature 1 and 2), the task for a classifier is to determine the class of that particular data point. This dataset was generated using PRTools[1], a pattern recognition library for MatLab from Delft University of Technology.

Fitness function

The fitness function that is used for the banana set is shown in equation 6.5.

$$\mathbf{S} = \{s_0, s_1, ..., s_n\} \quad \text{(set of samples)} \tag{6.1}$$

$$exp(s_i) = \{e_0, e_1, ..., e_n\} \quad \text{(the expected outputs of sample } s_i\text{)} \tag{6.2}$$

$$obt(s_i) = \{o_0, o_1, ..., o_n\} \quad \text{(the obtained outputs of sample } s_i\text{)} \tag{6.3}$$

$$exact(\mathbf{O}, \mathbf{E}) = \frac{|\{\forall o_i \in \mathbf{O}, e_i \in \mathbf{E} : o_i = e_i\}|}{|\mathbf{O}|} \quad \text{(computes the ratio of correct outputs)} \tag{6.4}$$

$$fitness = \frac{\sum_{i=0}^{|\mathbf{S}|} exact(obt(s_i), exp(s_i))}{|\mathbf{S}|} \tag{6.5}$$

It essentially computes the mean 'correctness' for each output.

6.4.2 Iris dataset

The Iris dataset is an often-used dataset taken from the UC Irvine Machine Learning Repository[2]. The data contains measurements extracted from a set of irises, based on these values a classifier has to determine to which particular subclass of the iris species it belongs. The set consists of 150 data points of four features each: sepal length, sepal width, petal length and petal width. The data is evenly divided among three classes, Iris Setosa, Iris Versicolour and Iris Virginica.

[1] `http://prtools.org/prtools.html`
[2] `http://archive.ics.uci.edu/ml/index.html`

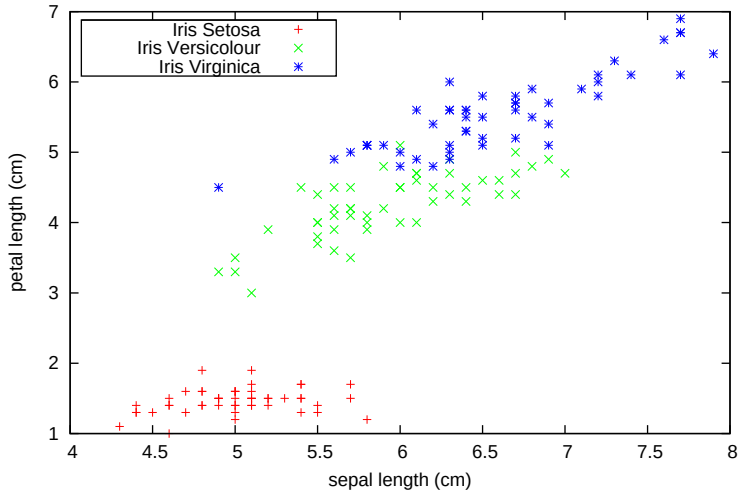


Figure 6.2: Scatterplot of the Iris dataset, only two of the four available features are used for this plot.

Fitness function

The default fitness function for the Iris dataset is the one displayed in equation 6.5. The *fitness distance metric* is displayed in equation 6.7, note that this function relies on equations 6.1, 6.2 and 6.3.

$$irisDist(s) = \begin{cases} 1 & \textbf{if } max(exp(s)) = max(obt(s)) \\ 0 & \textbf{if } max(exp(s)) = min(obt(s)) \\ 0.5 & \textbf{else} \end{cases} \tag{6.6}$$

$$fitness = \frac{\sum_{i=0}^{|\mathbf{S}|} irisDist(s_i)}{|\mathbf{S}|} \tag{6.7}$$

This second fitness function expects an ordering of outputs. For example, consider outputs $\{0.3, 0.9, 0.1\}$. When sorted in descending order this gives the following index order: $\{1, 0, 2\}$. In Table 6.1 three hypothetical expected outputs are shown. For each of these outputs the distance using the $irisDist(s)$ function (eq. 6.6) is computed.

$exp(s)$	expected highest index	$irisDist(s)$
$\{1, 0, 0\}$	0	0.5
$\{0, 1, 0\}$	1	1
$\{0, 0, 1\}$	2	0

Table 6.1: Example scores for the $irisDist(s)$ function for the following obtained output sequence: $obt(s) = \{0.3, 0.9, 0.1\}$.

6.4.3 TIMIT

TIMIT is a corpus of English read speech created by a collaboration of Massachusetts Institute of Technology (MIT), Stanford Research Institute (SRI) and Texas Instruments (TI). The dataset contains read speech divided over 8 dialect regions. For each dialect region a number of speakers is available. The aim is to create a classifier which can classify the spoken phonemes, this is a relatively simple but essential step for creating a speech recognizer. For each speaker 10 audio files are available, for each frame in these files the spoken phoneme is transcribed. This makes it a useful dataset for phoneme classification.

The audio files are in the Wave format, which are converted to Mel Frequency Cepstral Coefficients (MFCC) (Jurafsky and Martin, 2008) using the Hidden Markov Model Toolkit[3] (HTK). This conversion has several benefits, the first is that MFCC files are smaller compared to the raw wav files. On this particular dataset a overall file size decrease of over 84% has been observed for the MFCC conversion. For the conversion the HTK default values are used, these are shown in section A.1. It can be seen that the data is sampled over an interval of 10ms, the data is transformed to 12 MFCC outputs and one energy output, thus requiring 13 inputs in a neural network.

For the experiment only a subset of the TIMIT set was used, 32 sentences were available, they were split up in 16 test and 16 train sentences. In both sets there are two sentences for each dialect region.

Fitness function

The TIMIT dataset contains 53 different phonemes, as such 53 outputs are used. The fitness function implemented is shown in equation 6.8.

$$fitness(\mathbf{S}, d) = \frac{\sum_{i=0}^{|\mathbf{S}|} sampleFitness(s_i, d)}{|\mathbf{S}|} \tag{6.8}$$

This function has two parameters: the set of samples $\mathbf{S}$ and a distance d. When d is set to 1 the value is computed differently compared to values of d greater than 0. This can be seen in equation 6.9.

$$sampleFitness(s_i, d) = \begin{cases} absSampleFitness(s_i) & (d = 1) \\ distSampleFitness(s_i, d) & (d > 1) \end{cases} \tag{6.9}$$

When d is equal to 1, the following equation is applied:

$$absSampleFitness(s_i) = \frac{|\{\forall f_j \in frm(s_i) : max_{index}(exp(f_j)) = max_{index}(obt(f_j))\}|}{|s_i|} \tag{6.10}$$

This equation counts all perfect results and divides them by the total number of samples, thus yielding the average number of correct samples. Equation 6.10 relies on equations 6.11, 6.12 and 6.13.

$$frm(s_i) = \{f_0, f_1, ..., f_n\} \quad \text{(set of frames of sample } s_i\text{)} \tag{6.11}$$
$$exp(f_i) = \{e_0, e_1, ..., e_n\} \quad \text{(the expected outputs of frame } f_i\text{)} \tag{6.12}$$
$$obt(f_i) = \{o_0, o_1, ..., o_n\} \quad \text{(the obtained outputs of frame } f_i\text{)} \tag{6.13}$$

When d in equation 6.9 is greater than 1 the following equation is used:

$$distSampleFitness(s_i, d) = \sum_{j=0}^{|frm(s_i)|} \begin{cases} 1 - \left(\frac{rank(f_j)}{|obt(f_j)|}\right)^2 & rank(f_j) \leq d \\ 0 & rank(f_j) > d \end{cases} \tag{6.14}$$

This formula rewards answers which are *nearly* correct, this is based on the rank (distance of correctness) defined by:

$$rank(f_i) = index(descSort(obt(f_i)), max_{index}(exp(f_i))) \tag{6.15}$$

6.5 Setup

All experiments are run on either of two identical servers, which run the Ubuntu Linux (version 10.04) operating system. They have an x86 64 bit architecture and feature two quad core processors with a clock speed of 2.66 Gigahertz each. This means that each system has a total of eight cores. Furthermore it has access to 16 Gigabyte of RAM. In this section the setup of all experiments are described. All experiments share a set of parameters, these are shown in section A.2.

[3]HTK is a toolkit for creating hidden Markov models, it is primarily used for speech recognition and it provides training, testing and analysis tools. The website is: `http://htk.eng.cam.ac.uk/`.

6.5.1 Banana parameter tuning experiment

To find the optimal evolutionary parameter configuration, an initial test was conducted on the banana set. The best settings found in this experiment are used in the other experiments. In Table 6.2 the parameters that were varied are shown. Each of the listed parameter value was tested against every combination of the other parameters, meaning that a total of $6 \cdot 4 \cdot 2 \cdot 2 \cdot 2 = 192$ combinations were tested. Each combination was tested for 10 different seeds, resulting in a total of 1920 experiments. Because of this massive number of experiments, a population size of 100 was used and each experiment was executed for only 251 generations.

parameter	values	number of values
crossover-prob	0.4, 0.5, 0.6, 0.7, 0.8, 0.9	6
mutation-prob	0.4, 0.5, 0.7, 0.9	4
reset-prob	0.005, 0.01	2
node-weight-mutation-prob	0.5, 0.25	2
connection-weight-mutation-prob	0.5, 0.25	2

Table 6.2: Parameters that are varied in the parameter tuning experiment

In Appendix A.3 the complete parameter file used for this experiment is shown.

6.5.2 Banana component experiment

Since the increasing complexity and the fitness distance metric components are not implemented for the banana set, only the embryogeny and boosting components are tested on the banana set. This results in $2 \cdot 2 = 4$ experiments, each experiment is repeated 10 times, yielding a total of 40 experiments. In section A.4 the entire parameter file for this experiment is shown.

6.5.3 Iris component experiment

For the Iris dataset, all components are implemented, and can be tested. This gives a total of $2^4 = 16$ different experiments (as shown in Table 6.3), each parameter setting is repeated in a series of length 10 resulting in a total of 160 experiments for the Iris dataset. In section A.5 the entire parameter file for the Iris component experiment is displayed.

experiment #	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
embryogeny	off	on	off	on	off	on	off	on	off	on	off	on	off	on	off	on
boosting	off	off	on	on	off	off	on	on	off	off	on	on	off	off	on	on
incr. complexity	off	off	off	off	on	on	on	on	off	off	off	off	on	on	on	on
fit. dist. metric	off	off	off	off	off	off	off	off	on	on	on	on	on	on	on	on

Table 6.3: The experiment setup, showing the 16 different experiments that are conducted.

6.5.4 TIMIT component experiment

Each parameter setting is repeated 10 times. The same component setup as for the Iris dataset is used, see Table 6.3. For performance reasons, only 200 individuals were tested for 500 generations. In section A.6 all used parameters are listed.

6.5.5 Iris performance experiment

For this experiment the Iris dataset is divided in 10 equal parts using stratified sampling. Stratified sampling of a data set means that the relative frequency of the classes is preserved within its subsets, this reduces the sampling error. Ten fold cross validation is used resulting in ten different experiment setups. For each setup the train set contains 9 folds and the tenth fold is used as test set. Each

Weka classifier is thus executed 10 times, taking its average performance as its result. The evolutionary algorithm is executed 10 times for *each* fold combination (using a different random seed), this results in a total of 100 experiments. Again the average over all experiments is taken as its result. In section A.7 all used evolutionary parameters are displayed.

6.5.6 TIMIT performance experiment

This experiment is conducted in two flavors, using 500 and 5000 individuals in the population. Both experiments are run for 500 generations and repeated 10 times with different random seeds. In section A.8 all evolutionary parameters are listed.

7. Results

This chapter lists the results of the experiments discussed in the previous chapter. A detailed analysis is presented in chapter 8.

7.1 Parameter tuning experiment

In Table 7.1 the 8 best results of the parameter tuning experiment using the Banana set are presented, the settings are sorted on the mean (μ) test score. It can be seen that the best crossover rate is 0.9 and the best mutation rate is 0.7. Interestingly, the reset rate of either 0.005 or 0.01 doesn't influence the scores. For node weight, it is clear that the best rate is 0.25, and for connection weight it is 0.5. In Figure 7.1 all results of the parameter tuning experiment are plotted. In this graph, the values in the corner on the right top are the best results.

crossover	mutation	reset	node weight	connection weight	μ test score	μ train score
0.9	0.7	0.0050	0.25	0.5	0.865	0.908
0.9	0.7	0.01	0.25	0.5	0.865	0.908
0.8	0.9	0.0050	0.25	0.5	0.855	0.902
0.8	0.9	0.01	0.25	0.5	0.855	0.902
0.4	0.9	0.0050	0.25	0.25	0.851	0.905
0.4	0.9	0.01	0.25	0.25	0.851	0.905
0.5	0.9	0.0050	0.25	0.5	0.847	0.895
0.5	0.9	0.01	0.25	0.5	0.847	0.895

Table 7.1: The ten best results of the parameter tuning experiment sorted on test score. This experiment was conducted using the Banana set, note that μ stands for the mean score.

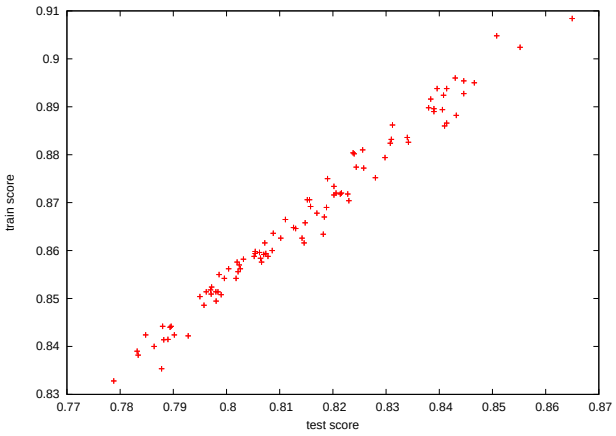


Figure 7.1: Scatter plot of all 192 results of the parameter tuning experiment. Each cross represents the mean of an experiment, its position on the x-axis indicates its mean test score, its position on the y-axis indicates its mean train score.

7.2 Banana component experiment

In Table 7.2 the mean results of the banana component experiment are shown, the raw scores are plotted in Figure 7.2 and box plots are shown in Figure 7.3.

#	boosting	embryogeny	μ test score	μ train score
0	false	false	0.892	0.951
1	true	false	0.892	0.948
2	false	true	0.915	0.964
3	true	true	0.916	0.967

Table 7.2: Results of the component experiment for the banana dataset.

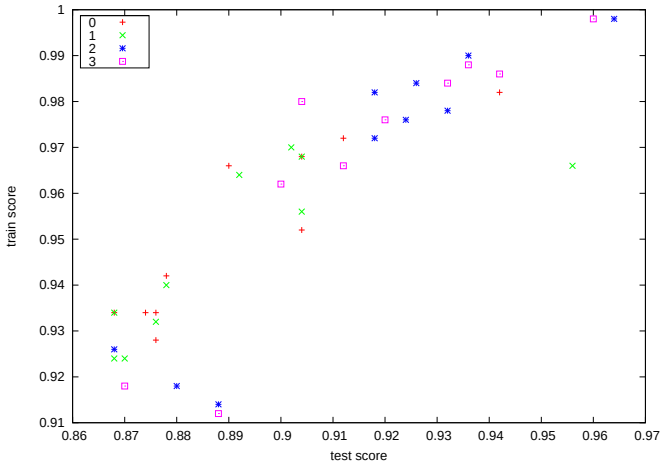


Figure 7.2: Scatter plot of the component experiment for the banana dataset. The position of each data point is defined by its test score (x-axis) and train score (y-axis). The different colors indicate the experiment setting. The experiment numbers from the legend correspond to the first column of Table 7.2.

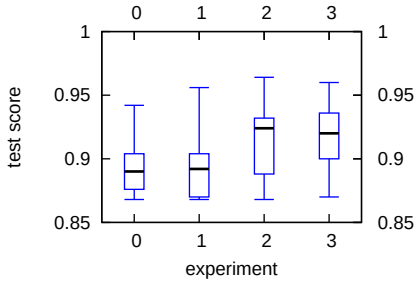


Figure 7.3: Banana component experiment box plots, the experiment numbers correspond to the first column of Table 7.2.

7.3 Iris component experiment

The results of the Iris component experiment are shown in Table 7.3 and Figure 7.4. As the number of experiments is high the scatterplot is omitted as this doesn't give insight in the results.

#	fit-dist-metric	boosting	embryogeny	incr-complexity	μ test score	μ train score
0	false	false	false	false	0.805	0.816
1	true	false	false	false	0.974	0.993
2	false	true	false	false	0.782	0.795
3	true	true	false	false	0.972	0.991
4	false	false	true	false	0.836	0.864
5	true	false	true	false	0.967	0.991
6	false	true	true	false	0.829	0.852
7	true	true	true	false	0.951	0.963
8	false	false	false	true	0.833	0.859
9	true	false	false	true	0.977	0.993
10	false	true	false	true	0.797	0.818
11	true	true	false	true	0.962	0.975
12	false	false	true	true	0.833	0.864
13	true	false	true	true	0.971	0.991
14	false	true	true	true	0.828	0.851
15	true	true	true	true	0.971	0.989

Table 7.3: Results of the component experiment for the Iris dataset.

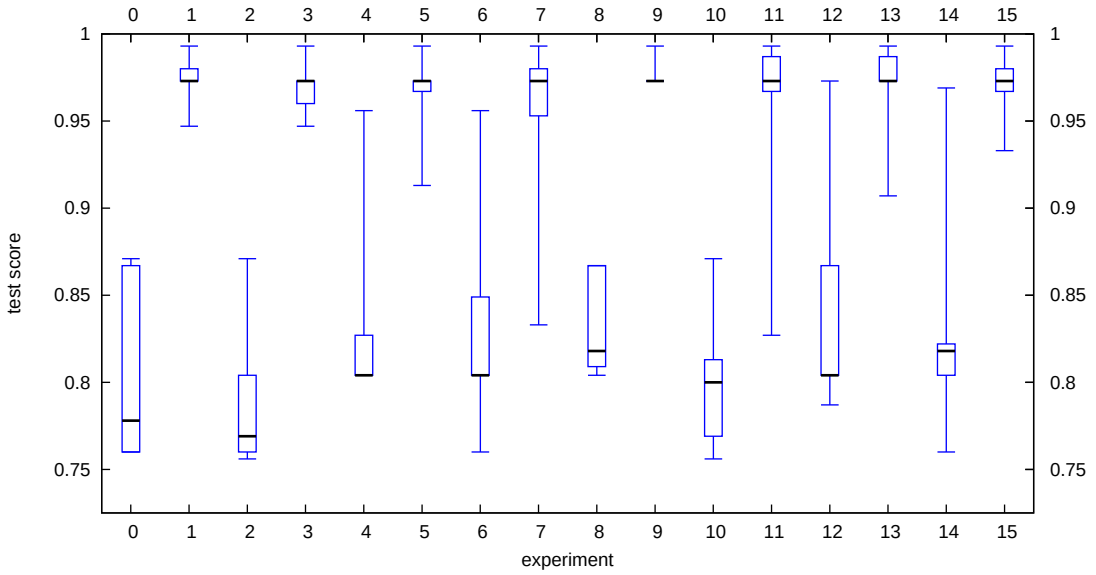


Figure 7.4: Box plots for the Iris component experiment.

7.4 TIMIT component experiment

In Table 7.4 the results for the TIMIT component experiment are shown, showing both the default fitness values as the values for the fitness distance metric.In Figure 7.5 and 7.6 the same results are shown as box plots.

#	fit-dist-metric	boosting	embryogeny	incr-complexity	μ test score	μ rel test score
0	false	false	false	false	0.17	0.154
1	true	false	false	false	0.013	0.436
2	false	true	false	false	0.177	0.182
3	true	true	false	false	0.014	0.396
4	false	false	true	false	0.193	0.256
5	true	false	true	false	0.031	0.424
6	false	true	true	false	0.185	0.236
7	true	true	true	false	0.011	0.405
8	false	false	false	true	0.191	0.189
9	true	false	false	true	0.033	0.435
10	false	true	false	true	0.169	0.183
11	true	true	false	true	0.013	0.43
12	false	false	true	true	0.184	0.234
13	true	false	true	true	0.01	0.42
14	false	true	true	true	0.175	0.271
15	true	true	true	true	0.01	0.423

Table 7.4: Results of the component experiment for the TIMIT dataset, the last column shows that fitness distance metric values.

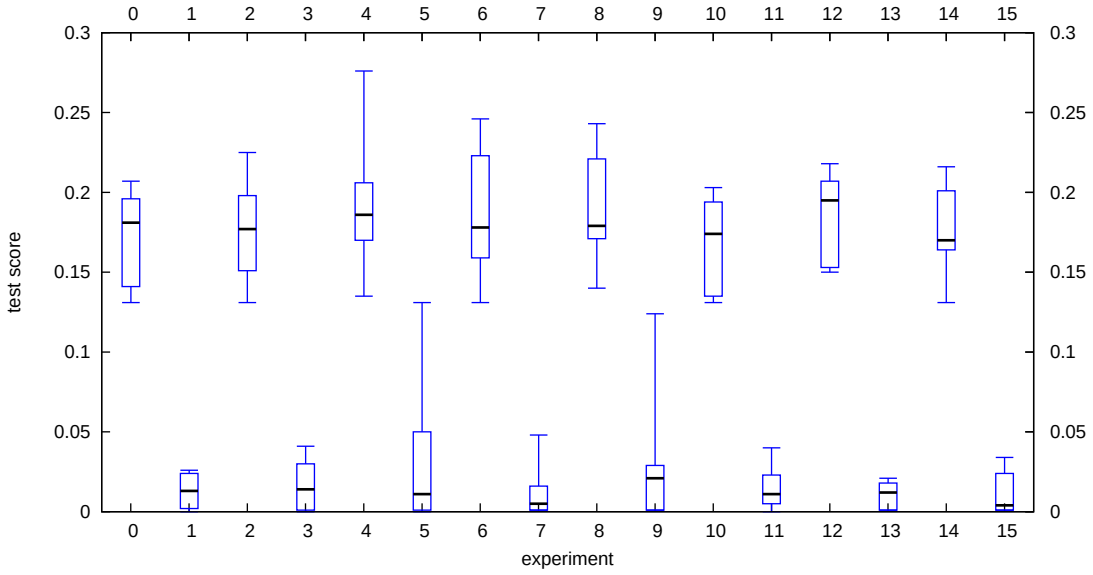


Figure 7.5: Box plots for the TIMIT component experiment, the experiment numbers correspond to the first column of Table 7.4.

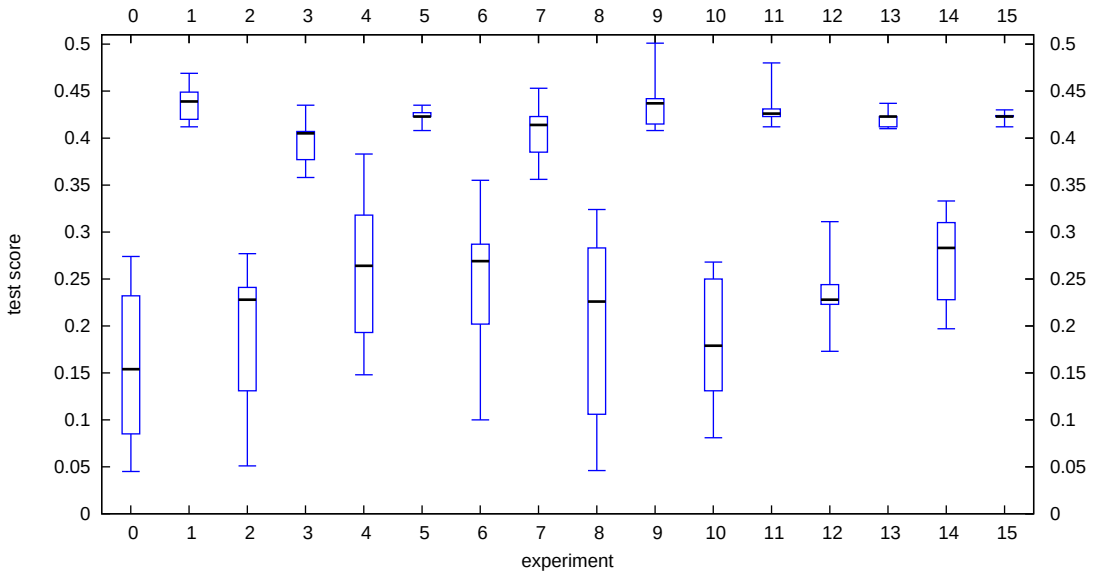


Figure 7.6: Box plots for the TIMIT component experiment using the fitness distance metric values, the experiment numbers correspond to the first column of Table 7.4.

7.5 Iris performance experiment

In Table 7.5, the 8 best performing classifiers from the Weka toolkit are shown compared to the evolutionary constructed complex system classifier. For each data fold, a computational network classifier was constructed using ten different random seeds. In the table there are two performance values for the complex system classifier, the *best* score takes the best score per fold and averages that over the folds. The *avg* score takes the means per fold and averages that over the folds. From this table it is clear that the complex system classifier can solve the Iris problem completely, in contrary to the Weka classifiers, furthermore, it is clear that even the average performance of the algorithm is very competitive. Appendix B.1 lists all Weka results. Table 7.6 gives a short description of the eight best performing Weka classifiers.

Classifier name	μ score	σ score
Complex System Classifier (best)	1.0	0.0
Logistic	0.98	0.045
LibSVM	0.98	0.032
Complex System Classifier (avg)	0.975	0.031
MultilayerPerceptron	0.973	0.047
SimpleLogistic	0.967	0.047
FT	0.967	0.047
LMT	0.967	0.047
SMO	0.967	0.035
MultiClassClassifier	0.967	0.035

Table 7.5: Summary of the results, the classifiers are sorted on the mean (μ) score, also the standard deviation (σ) of the score for each classifier is shown.

In Figure 7.7 one of the networks that was created for the Iris performance experiment is shown, the displayed network scores 100% on one of the folds.

Classifier name	description
Logistic	A multinomial logistic regression model with a ridge estimator (le Cessie and van Houwelingen, 1992).
LibSVM	Uses the LIBSVM library (Chang and Lin, 2001), which is a library for support vector machines.
MultilayerPerceptron	A neural network classifier that uses back propagation to classify instances, the nodes in the network are all sigmoid.
SimpleLogistic	Classifier for building linear logistic regression models (Landwehr et al., 2005).
FT	Classifier for building 'Functional trees', which are classification trees that could have logistic regression functions at the inner nodes and/or leaves. The algorithm can deal with binary and multi-class target variables, numeric and nominal attributes and missing values. (Gama, 2004)
FMT	A "logistic model tree" classifier (Landwehr et al., 2005).
SMO	Implementation of John C. Platt's sequential minimal optimization algorithm for training a support vector classifier using polynomial or RBF kernels (Platt, 1998).
MultiClassClassifier	Class for handling multi-class datasets with 2-class distribution classifiers. It uses the Logistic classifier.

Table 7.6: Descriptions of the eight best performing Weka classifiers.

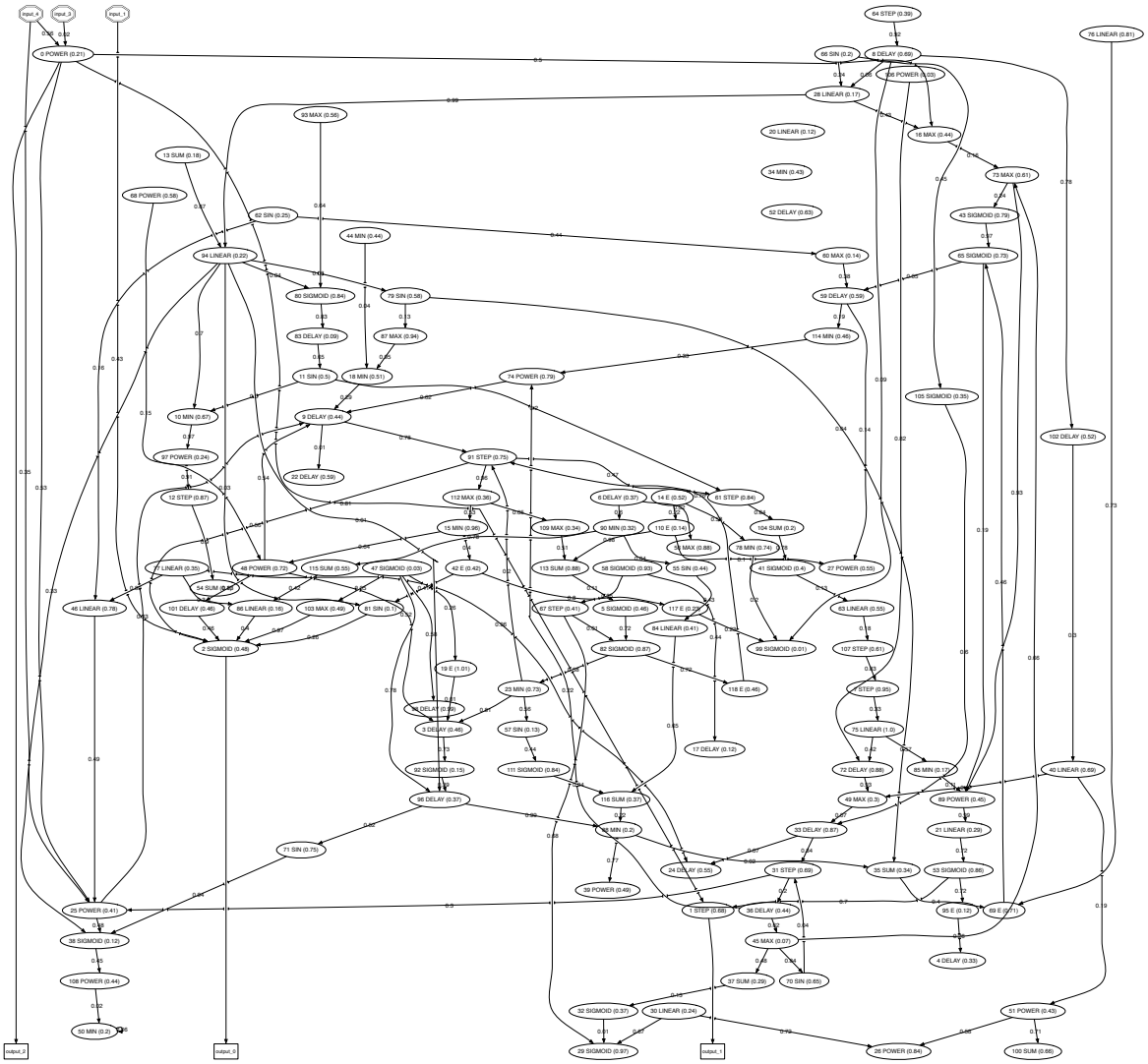


Figure 7.7: Resulting network from the Iris performance experiment, this network scores 100% on one of the folds.

7.6 TIMIT performance experiment

Table 7.7 shows the results of the TIMIT performance experiment.

population size	500	5000
maximum score	0.282	0.357
average score	0.211	0.306

Table 7.7: Results for the TIMIT performance experiment on two different population sizes.

8. Analysis

In this chapter, the results from the previous chapter are analyzed in detail. The component experiments are investigated using an *independent two sample T-test* (Robson, 2002). The tests were executed using R (Gentleman and Ihaka, 2010). A t-test checks whether two means are significantly different.

$$t = \frac{\bar{X_1} - \bar{X_2}}{\sqrt{\frac{\sigma_1^2 + \sigma_2^2}{2}} \cdot \sqrt{\frac{2}{n}}} \tag{8.1}$$

where n is the number of participants in each group. Using the value for t the p-value can be looked up in a *Student t distribution* table. Special attention is given to the hypotheses which were introduced in section 1.3, it is investigated whether they can be accepted or rejected in section 8.6.

8.1 Banana component experiment

component	on / off	μ score	σ	n	p-value	improves	significant
embryogeny	on	0.892	0.024	20	0.008	no	yes
	off	0.916	0.027	20			
boosting	on	0.904	0.028	20	0.944	no	no
	off	0.904	0.029	20			

Table 8.1: t-tests for the banana component experiment, investigating the effect of the embryogeny and boosting components.

In Table 8.1 the results of the t-tests are shown. It can be seen that the embryogeny component significantly worsens the evolutionary design. The boosting component seems to have no effect at all on the performance, as such it isn't necessary. In Table 8.2 the conclusions are shown.

component	conclusion
embryogeny	$--$
boosting	$-$

Table 8.2: Conclusions for the banana component experiment.

8.2 Iris component experiment

component	on / off	μ score	σ	n	p-value	improves	significant
embryogeny	on	0.898	0.08	80	0.073	yes	no
	off	0.888	0.091	80			
boosting	on	0.886	0.089	80	0.005	no	yes
	off	0.899	0.082	80			
fit-dist-metric	on	0.968	0.028	80	0	yes	yes
	off	0.818	0.05	80			
incr-complexity	on	0.896	0.083	80	0.269	yes	no
	off	0.889	0.088	80			

Table 8.3: t-tests for the Iris component experiment, investigating the effect of every component.

Table 8.3 gives an overview of the t-tests which were conducted for every component. From this table it is clear that three of four components improve the algorithm, and one of them, fitness distance metric, does so significantly. The boosting component significantly worsens the design, this indicates that boosting should not be used.

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
fit.dist.metric	1	0.90	0.90	569.16	**0.0000**
boosting	1	0.01	0.01	4.22	**0.0417**
embryogeny	1	0.00	0.00	2.75	0.0996
incr.complexity	1	0.00	0.00	1.25	0.2656
fit.dist.metric:boosting	1	0.00	0.00	0.58	0.4483
fit.dist.metric:embryogeny	1	0.01	0.01	6.90	**0.0096**
boosting:embryogeny	1	0.00	0.00	0.87	0.3534
fit.dist.metric:incr.complexity	1	0.00	0.00	0.20	0.6529
boosting:incr.complexity	1	0.00	0.00	0.02	0.8818
embryogeny:incr.complexity	1	0.00	0.00	0.08	0.7739
fit.dist.metric:boosting:embryogeny	1	0.00	0.00	0.84	0.3616
fit.dist.metric:boosting:incr.complexity	1	0.00	0.00	0.12	0.7347
fit.dist.metric:embryogeny:incr.complexity	1	0.00	0.00	2.36	0.1271
boosting:embryogeny:incr.complexity	1	0.00	0.00	0.77	0.3828
fit.dist.metric:boosting:embryogeny:incr.complexity	1	0.00	0.00	0.09	0.7587
Residuals	144	0.23	0.00		

Table 8.4: Analysis of variance (Anova) for the Iris component experiment, specifically investigating possible interaction effects.

The Anova table for the Iris component experiment is shown in in Table 8.4. The table shows that fitness distance metric and boosting are significant, this resembles the results of the t-tests in Table 8.3. An additional finding is that there is a significant interaction effect between fitness distance metric and embryogeny. This result demands an additional investigation for the embryogeny component.

component	on / off	μ score	σ	n	p-value	improves	significant
embryogeny	on	0.971	0.024	10	0.5	no	no
	off	0.976	0.007	10			

Table 8.5: Additional t-test to investigate the interaction effect between fitness distance metric and embryogeny, it can be seen that this effect is negative but not significant. The settings for the other components were: fit.dist.metric ON, boosting OFF, incr.complexity ON

In Table 8.5 an additional t-test is shown, investigating the interaction effect of the embryogeny and fitness distance metric component. It can be seen that this interaction effect is negative, this argues for

the exclusion of the embryogeny component in the model. Apparently, embryogeny does not benefit the outcome when used with the fitness distance metric for this problem.

Conclusion The conclusions of the Iris component experiment are shown in Table 8.6. The best setting is to use fitness distance metric and increasing complexity.

component	conclusion
embryogeny	$-$
boosting	$--$
fit.dist.metric	$++$
incr.complexity	$+$

Table 8.6: Conclusions for the Iris component experiment.

8.3 TIMIT component experiment

The two different fitness functions (indicated by the fit-dist-metric component) used for this experiment did not function as expected. Therefore, instead of only the absolute scores, the relative fitness scores are discussed in this section as well.

8.3.1 Absolute fitness function score

variable	on / off	μ score	σ	n	p-value	improves	significant
embryogeny	on	0.1	0.09	80	0.608	yes	no
	off	0.098	0.085	80			
boosting	on	0.094	0.086	80	0.019	no	yes
	off	0.103	0.088	80			
fit-dist-metric	on	0.017	0.025	80	0	no	yes
	off	0.181	0.032	80			
incr-complexity	on	0.098	0.087	80	0.84	no	no
	off	0.099	0.088	80			

Table 8.7: t-tests for the TIMIT component experiment, investigating the effect of every component.

In Table 8.7 the results of the t-tests for the TIMIT component experiment is shown. As can be expected the fit-dist-metric worsens the result significantly, because this component is used with a different fitness metric, which apparently is not helping the absolute score. Also the boosting component is (again) worsening the classifier design. The incr-complexity components seems to have almost no effect and the embryogeny improves the design only slightly.

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
fit.dist.metric	1	1.07	1.07	1281.92	**0.0000**
boosting	1	0.00	0.00	3.71	0.0561
embryogeny	1	0.00	0.00	0.26	0.6079
incr.complexity	1	0.00	0.00	0.05	0.8271
fit.dist.metric:boosting	1	0.00	0.00	0.02	0.8828
fit.dist.metric:embryogeny	1	0.00	0.00	1.14	0.2879
boosting:embryogeny	1	0.00	0.00	0.01	0.9217
fit.dist.metric:incr.complexity	1	0.00	0.00	0.00	0.9695
boosting:incr.complexity	1	0.00	0.00	0.71	0.4009
embryogeny:incr.complexity	1	0.00	0.00	4.14	**0.0437**
fit.dist.metric:boosting:embryogeny	1	0.00	0.00	0.00	0.9956
fit.dist.metric:boosting:incr.complexity	1	0.00	0.00	0.59	0.4418
fit.dist.metric:embryogeny:incr.complexity	1	0.00	0.00	0.07	0.7975
boosting:embryogeny:incr.complexity	1	0.00	0.00	3.62	0.0589
fit.dist.metric:boosting:embryogeny:incr.complexity	1	0.00	0.00	0.13	0.7227
Residuals	144	0.12	0.00		

Table 8.8: Analysis of variance (Anova) of the TIMIT component experiment.

To investigate possible interaction effects, an analysis of variance was conducted. The Anova table is shown in Table 8.8. It shows that fit-dist-metric has a significant effect. More interestingly, there seems to be an interaction effect between embryogeny and incr-complexity.

In Table 8.9 only the scores of the experiments with the different combinations of distance metric and embryogeny are shown. This table shows that the interaction effect of these components is negative, the best scores are reached with either embryogeny or distance metric on, but not both.

Conclusion Best setting for TIMIT dataset: fitness distance metric OFF, boosting OFF, either incr-complexity or embryogeny on, not both. This is shown in Table 8.10.

embryogeny	incr. complexity	μ score	σ	n
on	on	0.184	0.026	10
on	off	0.193	0.044	10
off	on	0.191	0.034	10
off	off	0.17	0.029	10

Table 8.9: Investigation of interaction effect between fitness distance metric and embryogeny. It was run with fitness distance metric off and boosting off.

component	conclusion
embryogeny	+/−
boosting	−−
fit.dist.metric	−−
incr.complexity	+/−

Table 8.10: Conclusions for the TIMIT component experiment using absolute fitness.

8.3.2 Fitness distance metric score

In Table 8.11 the results of the t-tests for the TIMIT component experiment using the fitness distance metric score are shown. The fitness distance metric significantly improves the solution as can be expected when using its score. Embryogeny also improves the solution significantly.

variable	on / off	μ score	σ	n	p-value	improves	significant
embryogeny	on	0.334	0.097	80	0	yes	yes
	off	0.3	0.138	80			
boosting	on	0.316	0.113	80	0.769	no	no
	off	0.318	0.127	80			
fit-dist-metric	on	0.421	0.023	80	0	yes	yes
	off	0.213	0.081	80			
incr-complexity	on	0.323	0.118	80	0.233	yes	no
	off	0.311	0.122	80			

Table 8.11: t-tests for the TIMIT component experiment, investigating the effect of every component.

In Table 8.12 an analysis of variance is shown. Fitness distance metric and embryogeny show significant effects, similar to that of the t-test. There is also an interaction effect between these two components, this demands for an investigation of this effect.

In Table 8.13 it can be seen that the effect between fitness distance metric and embryogeny is negative, this means that when fitness distance metric is used embryogeny worsens the solution.

Conclusion This takes us to the final conclusion for the TIMIT component experiment using the fitness distance metric score. Embryogeny and boosting do not add to the solution, but increasing complexity does, as is shown in Table 8.14. The fitness distance metric is clearly not beneficial in its current form.

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
fit.dist.metric	1	1.73	1.73	585.67	**0.0000**
boosting	1	0.00	0.00	0.09	0.7669
embryogeny	1	0.04	0.04	15.04	**0.0002**
incr.complexity	1	0.01	0.01	1.87	0.1733
fit.dist.metric:boosting	1	0.01	0.01	2.18	0.1421
fit.dist.metric:embryogeny	1	0.06	0.06	20.82	**0.0000**
boosting:embryogeny	1	0.00	0.00	0.12	0.7339
fit.dist.metric:incr.complexity	1	0.00	0.00	0.00	0.9930
boosting:incr.complexity	1	0.00	0.00	1.38	0.2426
embryogeny:incr.complexity	1	0.00	0.00	0.38	0.5361
fit.dist.metric:boosting:embryogeny	1	0.00	0.00	0.26	0.6132
fit.dist.metric:boosting:incr.complexity	1	0.00	0.00	0.24	0.6214
fit.dist.metric:embryogeny:incr.complexity	1	0.00	0.00	0.00	0.9536
boosting:embryogeny:incr.complexity	1	0.00	0.00	1.30	0.2556
fit.dist.metric:boosting:embryogeny:incr.complexity	1	0.01	0.01	2.35	0.1271
Residuals	144	0.42	0.00		

Table 8.12: Analysis of variance (Anova) of the TIMIT component experiment using the fitness distance metric score.

embryogeny	fit. dist. metric	μ score	σ	n
on	on	0.42	0.008	10
on	off	0.234	0.036	10
off	on	0.435	0.027	10
off	off	0.189	0.103	10

Table 8.13: Investigation of interaction effect between fitness distance metric and embryogeny. It was run with incr.complexity on and boosting off.

component	conclusion
embryogeny	−
boosting	−
fit.dist.metric	++
incr.complexity	+

Table 8.14: Conclusions for the TIMIT component experiment using fitness distance metric.

8.4 Iris performance experiment

This test was executed using boosting, and without embryogeny. In the comparison with the Weka classifiers the complex system obtained the first place with a score of 100%. This indicates that evolutionary design of complex systems can create very competitive classifiers. Furthermore, it shows that the mean score was 97.5% which means that the evolutionary algorithm produces solutions which have on average very good scores. Furthermore, one of the best Weka classifiers is the multilayer perceptron. This is interesting since this representation lies close to that of the complex system which was evolutionary designed.

An interesting observation that can be made from examining Figure 7.7 is that there are some unconnected nodes as well as nodes which have no incoming or no outgoing connections. These 'unnecessary' nodes can be the result of bloat, or they can be unfinished partial solutions which are beneficial for the evolutionary process.

8.5 TIMIT performance experiment

In section 7.6 the results for the TIMIT performance experiment were shown. With a maximum score of 35.7% the results are far below an acceptable level for everyday usage.

However, in speech recognition it is common to also use the first and second order derivatives of the features which can significantly increase the recognition rate. Additionally only a small amount of training data has been used, in this light, 35.7% is not too bad.

A possible cause of the disappointing performance is the complexity of the problem. Since there are 53 outputs in the system, the initial number of nodes is at least 53. It is likely that the required number of nodes is at least multiple times the number of outputs. Now consider the following equation:

$$configurations = 2^{n^2} \cdot n^{10}, \quad \text{where } n \text{ is the number of nodes} \tag{8.2}$$

This gives a very conservative lower bound of the size of the problem space as the possible values of the weights of the connections and the weights of the nodes are not counted. Only the connection configurations are counted and the 10 different types of nodes. Now consider a network of only 10 nodes, the lower bound of the search space would already be $1.268 \cdot 10^{40}$, for 50 nodes it is $3.670 \cdot 10^{769}$, for 100 nodes, $1.995 \cdot x10^{3030}$ and 1000 nodes: $9.9 \cdot 10^{301059}$. These calculations show that this size grows quickly out of hand, suggesting that it might be infeasible to search for a solution in this space, this is also stated by Holland (1995) who says that 10^{30} is already infeasible. In fact this search space is far greater than the estimated number of atoms in the universe[1].

However, this huge search space was not a problem for the more simple datasets. This suggests that the difference in quality of 'good' and 'bad' structures is easily detected. Furthermore, it is likely that a good classifier contains many repeating substructures. It is expected that those 53 phonemes are similar (this is an assumption of HMMs). As such, it can be expected that this search space can be searched effectively using the embryogeny component (which allows reuse of substructures). The actual cause of the poor performance can be the small amount of data compared to the number of inputs and outputs.

Another cause of the poor performance might be the bootstrap problem discussed in section 3.3. It is possible that with an even bigger population, there is enough variety to get a significant performance increase, similar to the increase between 500 and 5000 individuals.

8.6 Hypotheses

This section discusses the hypotheses presented in section 1.3, based on the analysis of the experiments it is argued whether each hypothesis can be accepted or rejected.

8.6.1 Hypothesis 1

H_1 Complex systems have properties that are desirable in computing.

This hypothesis is accepted, as described in section 2.8.

[1] The number of atoms in the universe is calculated to be around 10^{80}: `http://en.wikipedia.org/wiki/Observable_universe`

8.6.2 Hypothesis 2

H_2 A complex system can be modeled as a classifier.

The presentation of a computational network as a model for complex systems in chapter 4 shows that complex systems can indeed be modeled as a classifier. As was stated in chapter 4 a classifier can be described as a mathematical function which divides a problem space into several subspaces (classes). Since a computational network is in essence a set of interconnected functions, it is appropriate to use it as a classifier. Furthermore, the results shown in chapter 7 demonstrate that a complex system can be designed to behave as a classifier. H_2 is accepted.

8.6.3 Hypothesis 3

H_3 Artificial evolution is a promising method for the design of complex systems.

The results of the Iris performance experiment in section 7.5 show that an evolutionary design computational network can reach a 100% recognition rate. Furthermore, comparisons with the Weka classifiers have shown that the computational network is better than any of the classifiers, as such it can be concluded that artificial evolution is a promising method for the design of complex systems. H_3 is accepted.

8.6.4 Hypothesis 4

H_4 The evolutionary design of complex systems can be improved.

To be able to accept or reject H_4, the sub hypotheses have to be considered first. In Table 8.15 the conclusions for the components for each experiment are displayed.

component	Banana	Iris	TIMIT	TIMIT (fit. dist. metric)
embryogeny	−−	−	+/−	−
boosting	−	−−	−−	−
fit.dist.metric		++	−−	++
incr.complexity		+	+/−	+

Table 8.15: Conclusions for the all component experiments.

$H_{4.1}$ Embryogenies enhance the evolutionary design of complex systems.

This hypothesis can be accepted as it was shown that for the TIMIT experiment it *can* contribute to a better solution. Interesting is that for the most simple problem (banana set) the boosting component degrades the solution significantly, for the more complex problem of the Iris dataset embryogeny worsens the score only slightly, and for the most complex problem it can even improve the score. This suggests, that embryogeny is only used when a certain level of complexity is required. When complexity is not required, it might be that because it violates the non-redundancy requirement (as described in section 5.10.1), it worsens the solution. Additionally it also violates the causality requirement described in section 5.10.5, possibly hindering evolution. But when complexity is required, the benefits of embryogeny might outweigh these disadvantages.

$H_{4.2}$ Boosting enhances the evolutionary design of complex systems.

This hypothesis can be rejected, for every experiment boosting degrades the solution. This result is remarkable as Pasemann et al. (2001) have reported good results using boosting with evolving neural networks. A possible explanation is that when the boosting process has frozen a part of the genome and the genome is increased, the search space is also increased. However, whether this is the true cause for the bad performance is uncertain.

$H_{4.3}$ Distance metrics enhance the evolutionary design of complex systems.

This hypothesis can be accepted. For the Iris dataset we have seen that fitness distance metrics can be very useful. By smoothing the fitness landscape, the evolutionary algorithm can more easily find a global optimum. However, the results of the TIMIT dataset have shown that the particular design of this metric is non-trivial.

$H_{4.4}$ Increasing complexity enhances the evolutionary design of complex systems.

This hypothesis can be accepted. It was seen that this improves the result for the Iris dataset, however not significantly. For the TIMIT dataset increasing complexity was shown to improve the performance of the solution. Again, it is likely that this has to do with the complexity of the problem. The TIMIT dataset requires a much more complex solution. The increasing complexity component can facilitate a higher level of complexity. This is in accordance with the hypothesis of Simon (1962) which says that 'complexity will be hierarchic' (as was discussed in section 4.2).

H_4 The evolutionary design of complex systems can be improved.

This brings us to the main hypothesis which can be accepted because several sub hypotheses have shown that there is room for improvement. H_4 is accepted.

Part IV

Final Results

9. Conclusion

In this chapter the conclusions of this thesis are stated. The research questions are answered and the research objectives are discussed.

The first research objective and research question are:

- Investigate natural complex systems.

Q_1 What are the strengths and weaknesses of natural complex systems?

In chapter 2 natural complex systems were investigated to find the properties which define such a system. It was shown that the self-organizing and emergent capabilities of complex systems lead to the following adaptive abilities: *anticipation*, *autocatalysis*, *resilience*, *robustness* and *redundancy*. These abilities are responsible for the success of complex systems, they can survive in dynamically changing environments. These strengths of complex systems also come with a cost, the decentralized emergent nature of these systems make complex systems very hard to understand or design. Nevertheless, three preferred properties for the design of complex systems were identified: a decentralized structure, semi-homogenous individuals and networked communications.

The second research question is:

Q_2 Can complex systems model a classifier?

A new model for a complex system that behaves as a classifier was presented in chapter 4, it is called a *computational network*. Computational networks are a generalization of neural networks and boolean networks. A computational network is essentially a graph of interconnected mathematical functions. In principle any mathematical function can be used. There are no restrictions on the structure of the network, and it uses real values. Computational networks naturally incorporate the desired properties discussed in chapter 2. Furthermore they are equivalent to a universal Turing machine, meaning that in principle, they are capable of computing any *computable* function (Russell and Norvig, 2003). This is clearly an advantage of complex systems, as this means that their applicability has no limits within computer science. Complex systems are known to be appropriate for classification problems, because classifying is what many natural complex systems do (e.g. a brain). However, a classifier can also be described as a mathematical function which divides a problem space into several subspaces, as such a computational network, being a set of interconnected mathematical functions, is appropriate to use as a classifier. Furthermore CNs are expected to be good classifiers since it is a generalization of RNNs (which are proven classifiers). Additionally, because CNs have more function types, a smaller network is needed compared to an RNN.

The second objective is:

- Research how complex systems can be designed using evolution.

In chapter 3 it was argued that evolution can be used as a designer, two topics of concern were identified: how evolution could reach a desired level of complexity and how the evolutionary process could be speed up. The research of these two topics led to the investigation of several adaptations of the standard evolutionary algorithm, four of them were chosen for implementation. A novel generative representation (embryogeny) was presented which allows reuse and recursion of parts of the solution. The reuse of a partial solution allows the generative representation to describe a bigger solution space using a relatively small genotype space. Boosting is a component which tries to boost the classifier performance by dividing the evolutionary process in discrete phases. In each phase a part of the structure is frozen. Another component, fitness distance metric, tries to smooth the fitness landscape. The increasing complexity

component divides the problem space in several parts. The number of parts used for evaluation is slowly increased, which increases the complexity of the problem.

The third objective is:

- Create an evolutionary algorithm that can automatically construct classifiers.

An unsupervised evolutionary algorithm which designs a complex system classifier using computational networks is constructed using a direct representation. Several mutation operators manipulate the graph structure of the computational network, thereby introducing variation in the population. Tournament selection with elitism was used as selection method. The EA components discussed above were also implemented, each component setting was tested.

The third research question and fourth objective are closely related:

Q_3 Can artificial evolution be used to design a complex system classifier?

- Test the algorithm on several problems including the problem of automatic speech recognition.

The performance of evolutionary designed complex system classifiers has been investigated in two experiments. One experiment was conducted on the Iris dataset, the results show that the best evolutionary designed computational network had a recognition rate of 100%. That is 2% better than the best scoring classifier in the Weka classifier collection tested on the same data. Furthermore, the average performance of all the designed computational networks is 97.5%, this shows that on average the evolutionary design process produces designs of a very high quality. This is even more apparent if you consider that the Weka classifier collection contains many state-of-the-art classifiers.

The other performance experiment was conducted on the TIMIT speech dataset, the score of the best evolutionary designed computational network was 35.7% correct classified phonemes. Although this is not a correctness ratio which makes the classifier usable in practice, it is still a very remarkable result considering the small size of the train set and the relatively few input features.

The fourth research question:

Q_4 Can artificial evolution be improved for the design of complex systems?

A thorough investigation has been conducted for this research question. Each evolutionary component mentioned above is investigated separately. The experiments have shown that the problem domain defines which particular setting of the algorithm performs best. The complexity of a problem seems to be an important factor for performance of the proposed components.

Experiments with the generative representation called embryogeny have shown that its use can lead to an improved evolutionary design for the TIMIT dataset. This is contrary to the results for the Banana and Iris datasets where embryogeny worsens the score. It was argued that embryogeny only improves the evolutionary design of classifiers for complex problems because it facilitates more complex solution through its reuse of partial solutions. When this level of complexity is not required it was argued that it is disadvantageous because it violates the non-redundancy requirement of a representation, thereby hindering the evolutionary process. Furthermore the embryogeny component also violates the causality requirement, small mutations in genotype space can lead to big mutations in the phenotype space.

The results of the boosting component were very disappointing, for each experiment the boosting component degrades the designed computational network. It was argued that the particular implementation of boosting that is used suffers from the increasing search space which it causes. Thereby slowing the evolutionary process down.

The distance metric has shown to perform very well for the Iris dataset. The computational networks that were constructed using this metric were shown to be significantly better compared to the networks that were created without this metric. So, for the Iris dataset the smoothing of the fitness landscape resulted in better performance. However, for the TIMIT dataset the distance metric didn't work well, resulting in a significantly lower performance. This result shows that the success of the distance metric is very data and implementation dependent.

The increasing complexity component has shown to improve evolutionary designed computational networks. It starts by evaluating only a small number of outputs, slowly increasing this number over time. As it improved the solution for both datasets this slowly increasing of complexity is an improvement of the evolutionary design algorithm

Since three of the four algorithm components improve the evolutionary design of complex systems, the answer to the fourth question is positive.

The experimental results, together with the theoretical findings in this thesis, indicate that complex systems are a promising answer to the need for adaptive software. It was shown that complex systems have all abilities that are needed for adaptive systems, this asks for a new perspective, from a hard to comprehend problem domain towards a very powerful method of encoding a solution. Combined with the creative force of artificial evolution, complex systems can be designed to solve virtually any problem. An advantage of the complex systems perspective is that they are so abundant, there are complex systems everywhere in nature. Many of these systems are more easily understood than the human brain, this gives complex systems an edge over the traditional neural networks perspective, as there are many more places to get inspiration from.

10. Future work

During the writing of this thesis, many alternative theories, explanations and solutions arose. This section describes the most interesting of these 'thought experiments', which could function as the starting point for future work.

10.1 Adaptivity

This thesis showed that complex systems can be very adaptive in theory, it would be interesting to investigate this further. In order to gain *adaptive* complex systems the application domain should implicitly or explicitly reward adaptivity. This section describes several interesting strategies for achieving this.

10.1.1 Competitive coevolution

Competitive coevolution can be used to coevolve a 'hostile' or 'parasitic' program together with a complex system. The sole purpose of this hostile program would be to disrupt the complex system in a way such that its performance suffers. For example, the program could deliberately send noise towards the input nodes of the complex system during evaluation, after a number of generations the complex system will adapt to this nuisance, forcing the hostile program to change its strategy, as it has an inverse fitness function. The goal of this coevolution is that the complex system and the hostile program embark into an arms race which leads to an increasingly adaptive system.

10.1.2 Lesions

The concept of a stroke in the human brain could be simulated in a complex system. For example, in a random location in the system a group of nodes could be removed, or changed such that they behave randomly. When a stroke occurs in a complex system, it is likely to harm its performance. However if lesions are part of the natural condition in which the complex system is evolved, it is likely that the complex system will evolve a strategy to avoid any performance degradation. For example, the complex system could be designed redundant, if a node fails, there are one or more nodes as backup. It could also lead to self-organizing (or reorganizing) abilities, just like in a human brain, the complex system could move harmed functions to other regions. The ultimate result would be that complex systems can be as resilient to lesions similar to the resilience of the human brain.

10.2 Model extensions

The proposed computational network model is a very good starting point for investigating extensions of a model.

10.2.1 Mathematical functions

An interesting extension to the computational network would be the use of more sophisticated functions. A limitation of the functions in a computation network is that all inputs are treated equally. This rules out functions such as *division* that divide one value by another. In order to use such functions, the computational network needs to be able to distinguish between several input values. This requires an adaptation in the model as well as in the algorithm, it would facilitate many other functions such as if-then-else, $\frac{A}{B}$, A^B, $A \cdot B$, where A and B are both input values.

10.3 Algorithm extensions

The four different evolutionary components which were investigated in this thesis, are just a starting point. The possibilities are endless, and much work needs to be done. This section lists several approaches for improving the evolutionary design algorithm.

10.3.1 Hybrid: boosting complexity

An interesting variation on both increasing complexity and boosting is the combination of the two. Each time an additional output is added to the complex system, the previous nodes can be frozen similar to the boosting process. The expectation is that this reduces the search space since an existing working solution is frozen, as such, no mutations can occur in that part anymore. This means that all variation is focussed in the new part of the complex system, the part which targets the new outputs.

10.3.2 TIMIT fitness distance metric

A possible alternative for the TIMIT fitness distance metric can be to change the distance. For example, at the beginning of the evolution the distance could be set such to include all outputs, this results in a very smooth fitness landscape. When after a couple of generations, the performance increases, the distance could be lowered to create a slightly less smooth landscape. The expectation is that this increases the difficulty of the problem over time, and pushes the solution towards the global optimum.

10.3.3 TIMIT increasing complexity

To improve the results for the TIMIT dataset, the increasing complexity component can be improved. This could be done by starting with a very simple sound classification problem. For example, the first phase could consist of distinguishing between noise (non-speech and silence) and speech. Subsequent phases could try to distinguish between disparate phonemes, slowly increasing the total number of phonemes, until all phonemes are recognized.

When most phonemes are recognized, distorted speech samples can be added to the dataset. For example, noise can be added to the signal, gaps can be introduced or the pitch of the sound can be changed. These distorted speech samples should lead to a more adaptive (robust) speech recognizer.

10.3.4 Different rules

One of the limitations of the embryogeny mapping is that a rule can have only one input and one output. It is possible to work around this by substituting a rule with more than one node. This could facilitate more advanced rules requiring the occurrence of several abstract nodes which are also connected to each other. Another possibility is to eliminate the need for a dedicated abstract node, and instead allow rules to substitute any type of node.

10.3.5 Bloat investigation

The occurrence of bloat in computational networks can be further investigated. In this thesis, unused nodes were observed in the designed computational network for the Iris dataset (Figure 7.7). It would be interesting to investigate whether these unused nodes are beneficial for evolution, or whether pruning them would result in a performance increase.

10.4 Future

Several interesting directions for future work are described in this section.

10.4.1 Natural inspiration

One of the advantages of the complex systems perspective is that examples can be found everywhere in nature. These natural complex systems can inspire improvements in the complex system design,

and a closer study on natural evolution could lead to improving artificial evolution. This suggests a cross disciplinary approach of biologists, computer scientists, electrical engineers, mathematicians, psychologists, physicists, physicians, ecologists and so on.

10.4.2 Transcending the Von Neumann Architecture

The current standard in computer architecture is the von Neumann architecture, it employs a central processing unit (CPU) and separate memory. In essence it is an implementation of a universal Turing machine. A problem with this architecture is the bottleneck between CPU and memory, called the von Neumann bottleneck. The problem is that there is limited bandwidth between CPU and memory, forcing the CPU to wait for needed data to be transferred to or from memory[1].

The distributed design of complex systems can in principle transcend this bottleneck. In a complex system, computation and memory are hard to distinguish, if a computer is modeled as a complex system it might be more powerful compared to traditional designs, and have all the advantages of complex systems: adaptivity, redundancy, etc. For design of such a computer, evolution is a very obvious candidate, the fitness function could try to minimize the power consumption and optimize the computation speed.

10.4.3 Computing power

Evolutionary design requires a lot of computing power, fortunately, once the design is complete computation is no longer needed. Given the ever continuing trend of miniaturization and the resulting performance increase, and price drop, of computers, the future of evolutionary design looks bright. Furthermore, evolution is a naturally distributed process, as such it can easily harness the power of modern multi-core systems.

10.5 Possible application areas

Several application areas of the methods presented in this thesis naturally arise, these are discussed here. However, I think that the possible areas of applications are not limited to the ones presented here, in fact, I think the possible areas of application are endless.

10.5.1 Classifiers

Several classifications problems have been discussed in this thesis, however, they present only the tip of an iceberg of classification problems. Other examples are, optical character recognition, facial recognition, automatic lip reading, medical diagnosis, and so on. Many of these classification problems still pose enormous challenges before they can be solved. However, evolutionary designing complex systems might be a viable new angle to approach these problems.

An interesting experiment would be to test evolutionary designed computational networks on a large set of classification datasets. For each problem the computational network classifier can be compared to a state of the art classifier. The results of this experiment will indicate strong and weak points of evolutionary designed computational networks. The weak points could be further investigated to improve evolutionary algorithm.

10.5.2 Robot controllers

Another topic which deserves some attention is that of robot controllers. Robot controllers are a natural application domain for designing complex systems, mostly because the evolutionary 'environment' is easy to visualize and to manipulate. When designing a robot controller which has to solve some difficult task, it is easier to divide the problem in subproblems compared to for example the problem of speech recognition. Examples of research in this direction are (Tuci and Ampatzis, 2007; Fehérvári and Elmenreich, 2010) and (Hülse et al., 2004).

[1] `http://en.wikipedia.org/wiki/Von_Neumann_architecture#von_Neumann_bottleneck`

Bibliography

Angeline, P. J. and Pollack, J. B. (1994). Coevolving High-Level Representations. *Artificial Life*, III:55–71.

Axelrod, R. and Hamilton, W. D. (1981). The Evolution of Cooperation. *Science*, 211(27 March).

Baele, G., Bredeche, N., Haasdijk, E., Maere, S., Michiels, N., Van de Peer, Y., Schmickl, T., Schwarzer, C., and Thenius, R. (2009). Open-ended on-board Evolutionary Robotics for robot swarms. *2009 IEEE Congress on Evolutionary Computation*, pages 1123–1130.

Bentley, P. J. (2000). Exploring Component-based Representations - The Secret of Creativity by Evolution? In Parmee, I. C., editor, *Fourth International Conference on Adaptive Computing in Design and Manufacture (ACDM 2000)*, pages 161–172. University of Plymouth: UK.

Bentley, P. J. (2002). Why Biologists and Computer Scientists Should Work Together. In Collet, P., editor, *Lecture notes in computer science*, pages 3–15. Springer-Verlag Berlin Heidelberg.

Bentley, P. J. (2004). Fractal Proteins. *Genetic Programming and Evolvable Machines*, 5(1):71–101.

Bentley, P. J. and Corne, D. W. (2002). *An Introduction to Creative Evolutionary Systems*, chapter 1, pages 1–75. Morgan Kaufmann Publishers, San Francisco.

Bentley, P. J. and Kumar, S. (1999). Three ways to grow designs: a comparison of embryogenies for an evolutionary design problem. In Banzhaf, W., Daida, J., Eiben, A. E., Garzon, M. H., Honavar, V., Jakiela, V., and Smith, R. E., editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'99)*, volume 99, pages 35–43, Orlando, Florida, USA. Morgan Kaufmann Publishers.

Best, J. B. (1999). *Cognitive Psychology*. Wadsworth Publishing Company, Belmont, CA, USA, 5th edition.

Bilgin, A., Ellson, J., Gansner, E., Hu, Y., and North, S. (2010). Graphviz. *http://www.graphviz.org/*.

Bourrillion, K. and Levy, J. (2009). Google Collections Library. *http://code.google.com/p/google-collections/*.

Branke, J., Mnif, M., Müller-Schloer, C., Prothmann, H., Richter, U., Rochner, F., and Schmeck, H. (2006). Organic Computing Addressing Complexity by Controlled Self-Organization. *Second International Symposium on Leveraging Applications of Formal Methods, Verification and Validation (isola 2006)*, pages 185–191.

Bull, L. (1998). Coevolutionary Computation: An Introduction [unpublished].

Camazine, S., Deneubourg, J.-L., Franks, N. R., Sneyd, J., Theraulaz, G., and Bonabeau, E. (2003). *Self-Organization in Biological Systems*. Princeton University Press, Princeton, second edition.

Chang, C.-C. and Lin, C.-J. (2001). LIBSVM: a library for support vector machines. *Software available at http://www.csie.ntu.edu.tw/∼cjlin/libsvm*.

Cheng, R., Gen, M., and Tsjujimura, Y. (1996). A Tutorial Survey of Job-Shop Scheduling using Genetic Algorithms - I. Representation. *Elsevier Science*, 30(4):983–997.

Corning, P. A. (2002). The re-emergence of "emergence": a venerable concept in search of a theory. *Complexity*, 7(6):18–30.

Damper, R. (2000). Emergence and levels of abstraction. *International Journal of Systems Science*, 31(7):811–818.

Darwin, C. R. (1859). *On the Origin of Species*. John Murray.

Dawkins, R. (2006). *The Selfish Gene*. Oxford University Press, New York, 30th anniv edition.

De Wolf, T. and Holvoet, T. (2005). Emergence Versus Self-organisation: Different Concepts but Promising When Combined. *Engineering Self Organising Systems: Methodologies and Applications*, 3464:1–15.

Di Marzo Serugendo, G. (2003). Engineering Emergent Behaviour: A Vision. In Hales, D., editor, *Multi-Agent-Based Simulation III*, pages 1–7, Berlin Heidelberg. Springer-Verlag.

Di Marzo Serugendo, G., Gleizes, M.-P., and Karageorgos, A. (2006). Self-organization in multi-agent systems. *The Knowledge Engineering Review*, 20(02):165.

Dor, R. and Rothkrantz, L. J. M. (2008). The Ear's Mind An emergent self-organising model of auditory perception. *JETAI*, (2008):1–23.

Dorigo, M., Maniezzo, V., and Colorni, A. (1991). Positive feedback as a search strategy.

Dorigo, M. and Socha, K. (2006). An Introduction to Ant Colony Optimization.

Eiben, A. E., Schut, M. C., and Nitschke, G. S. (2005). Evolving an Agent Collective for Cooperative Mine Sweeping. *The 2005 IEEE Congress on Evolutionary Computation*, 1.

Eiben, A. E. and Smith, J. E. (2007). *Introduction to Evolutionary Computing (Natural Computing Series)*. Springer-Verlag Berlin Heidelberg, corrected edition.

Fehérvári, I. and Elmenreich, W. (2010). Evolving Neural Network Controllers for a Team of Self-organizing Robots Self-organizing Systems. *Journal of Robotics*, vol. 2010:10 pages.

Floreano, D., Mitri, S., Perez-Uribe, A., and Keller, L. (2008). Evolution of Altruistic Robots. In Zurada, J., editor, *Proceedings of the WCCI 2008*, pages 232–248. Springer-Verlag Berlin Heidelberg.

Foundalis, H. E. (2006). *Phaeaco: A Cognitive Architecture Inspired by Bongard's Problems*. PhD thesis.

Gama, J. (2004). Functional Trees. *Machine Learning*, 55(3):219–250.

Gentleman, R. and Ihaka, R. (2010). R: A Language and Environment for Statistical Computing.

Groß R., Nouyan, S., Bonani, M., Mondada, F., and Dorigo, M. (2008). Division of Labour in Self-organised Groups. In *Proc. of the* 10^{th} *Int. Conf. on Simulation of Adaptive Behavior*, pages 426–436. Springer Verlag, Berlin, Germany.

Gruau, F. (1994). *Neural Network Synthesis Using Cellular Encoding and the Genetic Algorithm*. Phd thesis, L'universite Claude Bernard-lyon I.

Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., and Witten, I. H. (2009). The WEKA Data Mining Software : An Update. *SIGKDD Explorations*, 11(1):10–18.

Heylighen, F. (1999a). Collective Intelligence and its Implementation on the Web: Algorithms to Develop a Collective Mental Map. *Computational Mathematical Organization Theory*, 3:253–280.

Heylighen, F. (1999b). The growth of structural and functional complexity during evolution. In Heylighen, F., Bollen, J., and Riegler, A., editors, *The Evolution of Complexity*, volume evolutiono, pages 17–44, Dordrecht. Kluwer Academic.

Heylighen, F. (1999c). The Science Of Self-Organization And Adaptivity. In *in: Knowledge Management, Organizational Intelligence and Learning, and Complexity, in: The Encyclopedia of Life Support Systems, EOLSS*, pages 253–280. Publishers Co. Ltd.

Hillis, W. D. (1990). Co-evolving Parasites Improve Simulated Evolution As An Optimization Procedure. *Physica D: Nonlinear Phenomena*, 42:228–234.

Holland, J. H. (1992). Complex Adaptive Systems. *Daedalus*, 121(1, A New Era in Computation):17–30.

Holland, J. H. (1995). *Hidden Order: How Adaptation Builds Complexity*. Perseus Books, New York.

Holland, J. H. (1998). *Emergence: From Chaos To Order*. Perseus Books, New York.

Homer-Dixon, T. (2009). The Newest Science. *Alternative Journal*, 35(4):9–11,38–39.

Hornby, G. S. and Pollack, J. B. (2001a). Evolving L-systems to generate virtual creatures. *Computers & Graphics*, 25(6):1041–1048.

Hornby, G. S. and Pollack, J. B. (2001b). The advantages of generative grammatical encodings for physical design. In *Congress on Evolutionary Computation*, pages 600–607. Citeseer.

Hornby, G. S. and Pollack, J. B. (2002). Creating High-Level Components with a Generative Representation for Body-Brain Evolution. *Artificial Life*, 8(3):223–246.

Hülse, M., Wischmann, S., and Pasemann, F. (2004). Structure and function of evolved neuro-controllers for autonomous robots. *Connection Science*, 16(4):249–266.

Hyötyniemi, H. (1996). Turing Machines are Recurrent Neural Networks. In Alander, J., Honkela, T., and Jakobsson, M., editors, *STeP'96 - Genes, Nets and Symbols*, pages 13–24, Vaasa, Finland. Finish Artificial Intelligence Society.

Janssen, P., Frazer, J., and Ming-xi, T. (2002). Evolutionary design systems and generative processes. *Applied intelligence*, 16(2):119–128.

Jurafsky, D. and Martin, J. H. (2008). *Speech and Language Processing (2nd Edition)*. Prentice Hall.

Kearns, M. (1988). Thoughts on hypothesis boosting. *Unpublished manuscript*.

Kennedy, J., Eberhart, R. C., and Shi, Y. (2001). *Swarm Intelligence (The Morgan Kaufmann Series in Artificial Intelligence)*. Morgan Kaufmann.

Kicinger, R., Arciszewski, T., and De Jong, K. A. (2005a). Evolutionary computation and structural design: A survey of the state-of-the-art. *Computers and Structures*, 83(23-24):1943–1978.

Kicinger, R., Arciszewski, T., and De Jong, K. A. (2005b). Parameterized versus Generative Representations in Structural Design : An Empirical Comparison. In Beyer, H.-G. and O'Reilly, U.-M., editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2005)*, pages 2007–2014, Washington, DC. The Association for Computing Machinery.

Landwehr, N., Hall, M., and Frank, E. (2005). Logistic Model Trees. *Machine Learning*, 59(1-2):161–205.

le Cessie, S. and van Houwelingen, J. (1992). Ridge Estimators in Logistic Regression. *Applied Statistics*, 41(1):191–201.

Lindenmayer, A. (1968a). Mathematical models for cellular interactions in development, Part I: Filaments with One-sided Inputs. *Journal of Theoretical Biology*, 18:280–299.

Lindenmayer, A. (1968b). Mathematical models for cellular interactions in development, Part II: Simple and Branching Filaments with Two-sided Inputs. *Journal of Theoretical Biology*, 18:300–315.

Luke, S., Panait, L., Balan, G., Paus, S., Skolicki, Z., Popovici, E., Sullivan, K., Harrison, J., Bassett, J., Hubley, R., Chircop, A., Compton, J., Haddon, W., Donnelly, S., Jamil, B., and O'Beirne, J. (2010). ECJ 19 - A Java-based Evolutionary Computation Research System. *http://www.cs.gmu.edu/∼eclab/projects/ecj/*.

MacLeod, C., Maxwell, G., and Muthuraman, S. (2009). Incremental growth in modular neural networks. *Engineering Applications of Artificial Intelligence*, 22(4-5):660–666.

Maniadakis, M. and Trahanias, P. (2005). CoEvolutionary Incremental Modelling of Robotic Cognitive Mechanisms. In Capcarrere, M., Freitas, A. A., Bentley, P. J., Johnson, C. G., and Timmis, J., editors, *Proceedings of the VIIIth European Conference on Artificial Life (ECAL-2005)*, pages 200–209, Canterbury, UK. Springer-Verlag Berlin Heidelberg.

Maniadakis, M. and Trahanias, P. (2008). Hierarchical Co-evolution of Cooperating Agents Acting in the Brain-Arena. *Adaptive Behavior*, 16(4):221–245.

McLurkin, J. and Demaine, E. D. (2009). A Distributed Boundary Detection Algorithm for Multi-Robot Systems. *IEEE Intelligent RObots and Systems (IROS)*, (to appear.

Miconi, T. (2008). Evolution and complexity: the double-edged sword. *Artificial life*, 14(3):325–44.

Minsky, M. (1988). *The Society of Mind*. Simon & Schuster.

Mitchell, M. and Hofstadter, D. R. (1994). The Copycat Project: A Model of Mental Fluidity and Analogy-making. In Holyoak, K. and Barnden, J., editors, *Advances in Connectionist and Neural Computation Theory*. Ablex.

Mouret, J.-B. and Doncieux, S. (2009a). Evolving modular neural-networks through exaptation. In *Proceedings of the Eleventh conference on Congress on Evolutionary Computation*, pages 1570–1577, Trondheim, Norway. IEEE Press.

Mouret, J.-B. and Doncieux, S. (2009b). Overcoming the bootstrap problem in evolutionary robotics using behavioral diversity. In *IEEE Congress on Evolutionary Computation (CEC)*, pages 1161 – 1168, Trondheim, Norway.

Parunak, H. V. D. and Brueckner, S. A. (2004). Engineering swarming systems. *Methodologies and Software Engineering for Agent Systems*, pages 1–29.

Pasemann, F., Steinmetz, U., Hu, M., and Lara, B. (2001). Robot Control and the Evolution of Modular Neurodynamics. *Theory in Biosciences*, 120:311–326.

Platt, J. (1998). Fast Training of Support Vector Machines using Sequential Minimal Optimization. In Schoelkopf, B., Burges, C., and Smola, A., editors, *Advances in Kernel Methods - Support Vector Learning*. MIT Press.

Potter, M. A. and De Jong, K. A. (2000). Cooperative coevolution: An architecture for evolving coadapted subcomponents. *Evolutionary Computation*, 8(1):1–29.

Potter, M. A., Meeden, L. A., and Schultz, A. C. (1996). Heterogeneity in the Coevolved Behaviors of Mobile Robots: The Emergence of Specialists. *Evolution*.

Pujol, J. C. F. and Poli, R. (1998). Efficient Evolution of Asymmetric Recurrent Neural Networks Using a PDGP-inspired Two-Dimensional Representation. *Lecture Notes in Computer Science*, 1391/1998:130–141.

Robson, C. (2002). *Real World Research*. Blackwell Publishing, Malden, MA, USA, 2nd edition.

Roggen, D., Federici, D., and Floreano, D. (2006). Evolutionary morphogenesis for multi-cellular systems. *Genetic Programming and Evolvable Machines*, 8(1):61–96.

Römmerman, M., Kühn, D., and Kirchner, F. (2009). Robot design for space missions using evolutionary computation. *2009 IEEE Congress on Evolutionary Computation*, pages 2098–2105.

Russell, S. J. and Norvig, P. (2003). *Artificial Intelligence: A Modern Approach*. Prentice Hall, New Jersey, 2nd edition.

Salehie, M. and Tahvildari, L. (2009). Self-adaptive software: Landcape and Research Challenges. *ACM Transactions on Autonomous and Adaptive Systems*, 4(2):1–42.

Scheidler, A. and Middendorf, M. (2009). Evolved cooperation and emergent communication structures in learning classifier based organic computing systems. *Genetic And Evolutionary Computation Conference*, page 7.

Schut, M. C. (2007). *Scientific Handbook for Simulation of Collective Intelligence.*

Schut, M. C. (2010). On model design for simulation of collective intelligence. *Information Sciences*, 180(1):132–155.

Schwager, M., Mclurkin, J., Slotine, J.-j. E., and Rus, D. (2008). From Theory to Practice: Distributed Coverage Control Experiments with Groups of Robots. In *Proceedings of the International Symposium on Experimental Robotics (ISER 08)*, number 1, Athens, Greece.

Selouani, S.-A. and O'Shaughnessy, D. (2003). On the Use of Evolutionary Algorithms to Improve the Robustness of Continuous Speech Recognition Systems in Adverse Conditions. *EURASIP Journal on Advances in Signal Processing*, 2003(8):814–823.

Simon, H. A. (1962). The Architecture of Complexity. *Proceedings of the American Philosophical Society*, 106(6):467–482.

Sims, K. (1994). Evolving virtual creatures. *Proceedings of the 21st annual conference on Computer graphics and interactive techniques - SIGGRAPH '94*, (July):15–22.

Sipser, M. (2006). *Introduction to the Theory of Computation.* Thomson Course Technology, Boston, Massachusetts, USA, 2nd, int. edition.

Stanley, K. O. and Miikkulainen, R. (2004). Competitive Coevolution through Evolutionary Complexification. *Journal of Artificial Intelligence Research*, 21:63–100.

Torresen, J. (1998). A Divide-and-Conquer Approach to Evolvable Hardware. In *Evolvable Systems: From Biology to Hardware. Second Int. Conf., ICES 98*, pages 57–65. Springer-Verlag.

Torresen, J. (2002). A Dynamic Fitness Function Applied to Improve the Generalisation when Evolving a Signal Processing Hardware Architecture. In *In Applications of Evolutionary Computing: EvoWorkshops 2002*, pages 267–279. Springer-Verlag.

Tuci, E. and Ampatzis, C. (2007). *Evolution of Acoustic Communication Between Two Cooperating Robots*, volume 4648 of *Lecture Notes in Computer Science*, pages 395–404. Springer-Verlag Berlin Heidelberg, Berlin, Heidelberg.

Tufte, G. and Haddow, P. C. (2004). Biologically-inspired: A rule-based self-reconfiguration of a virtex chip. In M. Bubak, G., Albada, D., and Van Sloot, P. M. A., editors, *Proceedings of the 4th International Conference on Computational Science (ICCS 2004)*, pages 1249–1256, Heidelberg. Springer-Verlag.

Uny Cao, Y., Fukunaga, A. S., and Kahng, A. B. (1997). Cooperative Mobile Robotics: Antecedents and Directions. *Autonomous Robots*, 4:7–27.

Whitney, E. J., Sefrioui, M., Srinivas, K., and Périaux, J. (2002). Advances in Hierarchical, Parallel Evolutionary Algorithms for Aerodynamic Shape Optimisation. *JSME International Journal Series B*, 45(1):23–28.

Wiggers, P. (2008). *Modelling context in automatic speech recognition.* PhD thesis.

Williams, T. and Kelley, C. (2010). gnuplot - An interactive plotting program. *http://gnuplot.info/.*

Wischmann, S. and Pasemann, F. (2006). The emergence of communication by evolving dynamical systems. In Nolfi, S., Baldassarre, G., Calabretta, R., Hallam, J., Marocco, D., Meyer, J.-A., and Parisi, D., editors, *From animals to animats 9: Proceedings of the Ninth International Conference on Simulation of Adaptive Behaviour. LNCS (LNAI)*, pages 777–788. Springer-Verlag Berlin Heidelberg.

Part V

Appendices

A. Parameter files

A.1 MFCC configuration

```
# Coding parameters
SOURCEFORMAT    = NOHEAD
SOURCERATE      = 625
TARGETKIND = MFCC_0
TARGETRATE = 100000.0
SAVECOMPRESSED = F
SAVEWITHCRC = T
WINDOWSIZE = 250000.0
USEHAMMING = T
PREEMCOEF = 0.97
NUMCHANS = 26
CEPLIFTER = 22
NUMCEPS = 12
ENORMALISE = F
```

A.2 Other variables

```
# constants
verbosity           = 0
breedthreads        = 8
evalthreads         = 8

select.tournament.size      = 5

pop.subpop.0.species            = nl.rinde.graduation.ec.NNSpecies
pop.subpop.0.species.ind        = nl.rinde.graduation.ec.NNIndividual
pop.subpop.0.species.fitness    = nl.rinde.graduation.ec.NNFitness

checkpoint          = true
checkpoint-modulo   = 50
prefix              = ec
quit-on-run-complete =          true

state           = nl.rinde.graduation.ec.NNEvolutionState
finish          = nl.rinde.graduation.ec.NNFinisher
stat            = nl.rinde.graduation.ec.NNStatistics
init            = nl.rinde.graduation.ec.NNInitializer
breed           = ec.simple.SimpleBreeder
eval            = ec.simple.SimpleEvaluator
eval.problem    = nl.rinde.graduation.ec.NNProblem
exch            = ec.simple.SimpleExchanger
pop             = ec.Population
pop.subpops     = 1
pop.subpop.0    = ec.Subpopulation
pop.subpop.0.duplicate-retries =    0

pop.subpop.0.species.pipe                   = nl.rinde.graduation.ec.breed.
    NNMutationPipeline
pop.subpop.0.species.pipe.source.0          = ec.select.TournamentSelection
pop.subpop.0.species.pipe.source.0          = nl.rinde.graduation.ec.breed.
    NNCrossoverPipeline
```

```
pop.subpop.0.species.pipe.source.0.source.0 = ec.select.TournamentSelection
pop.subpop.0.species.pipe.source.0.source.1 = ec.select.TournamentSelection
```

A.3 Banana tuning experiment parameter file

```
# banana probabilities experiment
parent.0 = ../grad.params
target-folder = /banana-variation-probabilities-experiment

breed.elite.0        = 5
pop.subpop.0.size    = 100
generations          = 251

seed.0 = 8022,4119,7466,6880,4481,4189,7713,9251,1748,2841
seed.1 = 7050,3433,1378,699,7939,188,28,4934,4397,8016
seed.2 = 6563,9059,3195,3127,6412,8920,3634,5484,8774,2740
seed.3 = 8740,8506,8681,2680,7497,5042,8911,999,6855,8144
seed.4 = 6889,5521,9948,2744,9385,6878,3641,2697,2391,6835
seed.5 = 9920,7984,8836,220,1055,4640,751,4853,9738,9163
seed.6 = 8849,5663,4181,9737,5387,8350,2609,8816,784,7506
seed.7 = 2138,4322,5555,1540,5674,8760,3559,5458,7940,245

pop.subpop.0.species.crossover-prob.*    = 0.4,0.5,0.6,0.7,0.8,0.9
pop.subpop.0.species.mutation-prob.*     = 0.4,0.5,0.7,0.9

pop.subpop.0.species.reset-prob.*                          = 0.005,0.01
pop.subpop.0.species.node-weight-mutation-prob.*     = 0.5,0.25
pop.subpop.0.species.connection-weight-mutation-prob.*= 0.5,0.25
pop.subpop.0.species.connection-struct-prob          = 0.5
pop.subpop.0.species.add-node-prob               = 0.5
pop.subpop.0.species.remove-node-prob            = 0.5
pop.subpop.0.species.add-rule-prob                   = 0.05
pop.subpop.0.species.remove-rule-prob                = 0.05
pop.subpop.0.species.mutate-rule-prob                = 0.99
pop.subpop.0.species.mutate-seed-prob                = 0.99

eval.problem.type            = BANANA
eval.problem.num-inputs      = 2
eval.problem.num-outputs     = 1
eval.problem.train-set-dir   = /files/datasets/banana/banana-trainset.txt
eval.problem.test-set-dir    = /files/datasets/banana/banana-testset.txt

eval.problem.embryogeny      = true
eval.problem.boosting        = false
eval.problem.incr-complexity= false
eval.problem.fit-dist-metric= false
```

A.4 Banana component experiment parameter file

```
# banana probabilities experiment
parent.0 = ../grad.params
target-folder = /banana-component-experiment

breed.elite.0        = 5
pop.subpop.0.size    = 500
generations          = 501

seed.0 = 8022,4119,7466,6880,4481,4189,7713,9251,1748,2841
seed.1 = 7050,3433,1378,699,7939,188,28,4934,4397,8016
seed.2 = 6563,9059,3195,3127,6412,8920,3634,5484,8774,2740
seed.3 = 8740,8506,8681,2680,7497,5042,8911,999,6855,8144
seed.4 = 6889,5521,9948,2744,9385,6878,3641,2697,2391,6835
seed.5 = 9920,7984,8836,220,1055,4640,751,4853,9738,9163
seed.6 = 8849,5663,4181,9737,5387,8350,2609,8816,784,7506
```

```
seed.7 = 2138,4322,5555,1540,5674,8760,3559,5458,7940,245

pop.subpop.0.species.crossover-prob                  = 0.9
pop.subpop.0.species.mutation-prob                   = 0.7

pop.subpop.0.species.reset-prob                      = 0.01
pop.subpop.0.species.node-weight-mutation-prob       = 0.25
pop.subpop.0.species.connection-weight-mutation-prob= 0.5
pop.subpop.0.species.connection-struct-prob          = 0.5
pop.subpop.0.species.add-node-prob               = 0.5
pop.subpop.0.species.remove-node-prob            = 0.5
pop.subpop.0.species.add-rule-prob                   = 0.05
pop.subpop.0.species.remove-rule-prob                = 0.05
pop.subpop.0.species.mutate-rule-prob                = 0.99
pop.subpop.0.species.mutate-seed-prob                = 0.99

eval.problem.boost-interval = 250
eval.problem.max-boost = 1

eval.problem.type            = BANANA
eval.problem.num-inputs      = 2
eval.problem.num-outputs     = 1
eval.problem.train-set-dir   = /files/datasets/banana/banana-trainset.txt
eval.problem.test-set-dir    = /files/datasets/banana/banana-testset.txt

eval.problem.embryogeny.*        = false,true
eval.problem.boosting.*          = false,true
eval.problem.incr-complexity     = false
eval.problem.fit-dist-metric     = false
```

A.5 Iris component experiment parameter file

```
# banana probabilities experiment
parent.0 = ../grad.params
target-folder = /iris-component-experiment

breed.elite.0        = 5
pop.subpop.0.size    = 500
generations          = 501

seed.0 = 8022,4119,7466,6880,4481,4189,7713,9251,1748,2841
seed.1 = 7050,3433,1378,699,7939,188,28,4934,4397,8016
seed.2 = 6563,9059,3195,3127,6412,8920,3634,5484,8774,2740
seed.3 = 8740,8506,8681,2680,7497,5042,8911,999,6855,8144
seed.4 = 6889,5521,9948,2744,9385,6878,3641,2697,2391,6835
seed.5 = 9920,7984,8836,220,1055,4640,751,4853,9738,9163
seed.6 = 8849,5663,4181,9737,5387,8350,2609,8816,784,7506
seed.7 = 2138,4322,5555,1540,5674,8760,3559,5458,7940,245

pop.subpop.0.species.crossover-prob                  = 0.9
pop.subpop.0.species.mutation-prob                   = 0.7

pop.subpop.0.species.reset-prob                      = 0.01
pop.subpop.0.species.node-weight-mutation-prob   = 0.25
pop.subpop.0.species.connection-weight-mutation-prob= 0.5
pop.subpop.0.species.connection-struct-prob          = 0.5
pop.subpop.0.species.add-node-prob               = 0.5
pop.subpop.0.species.remove-node-prob            = 0.5
pop.subpop.0.species.add-rule-prob                   = 0.05
pop.subpop.0.species.remove-rule-prob                = 0.05
pop.subpop.0.species.mutate-rule-prob                = 0.99
pop.subpop.0.species.mutate-seed-prob                = 0.99

eval.problem.boost-interval = 250
eval.problem.max-boost = 1
eval.problem.output-amount-init = 2
eval.problem.output-amount-step = 1
eval.problem.output-amount-step-threshold = 0.8
```

```
eval.problem.output-amount-time-threshold = 250

eval.problem.type                = IRIS
eval.problem.num-inputs          = 4
eval.problem.num-outputs         = 3
eval.problem.train-set-dir       = /files/datasets/iris/iris-train.txt
eval.problem.test-set-dir        = /files/datasets/iris/iris-test.txt

eval.problem.embryogeny.*        = false,true
eval.problem.boosting.*          = false,true
eval.problem.incr-complexity.*   = false,true
eval.problem.fit-dist-metric.*   = false,true
```

A.6 TIMIT component experiment parameter file

```
# timit component experiment
parent.0 = ../grad.params
target-folder = /timit-component-experiment

breed.elite.0        = 5
pop.subpop.0.size    = 200
generations          = 501

seed.0 = 8022,4119,7466,6880,4481,4189,7713,9251,1748,2841
seed.1 = 7050,3433,1378,699,7939,188,28,4934,4397,8016
seed.2 = 6563,9059,3195,3127,6412,8920,3634,5484,8774,2740
seed.3 = 8740,8506,8681,2680,7497,5042,8911,999,6855,8144
seed.4 = 6889,5521,9948,2744,9385,6878,3641,2697,2391,6835
seed.5 = 9920,7984,8836,220,1055,4640,751,4853,9738,9163
seed.6 = 8849,5663,4181,9737,5387,8350,2609,8816,784,7506
seed.7 = 2138,4322,5555,1540,5674,8760,3559,5458,7940,245

pop.subpop.0.species.crossover-prob                  = 0.9
pop.subpop.0.species.mutation-prob                   = 0.7

pop.subpop.0.species.reset-prob                      = 0.01
pop.subpop.0.species.node-weight-mutation-prob  = 0.25
pop.subpop.0.species.connection-weight-mutation-prob= 0.5
pop.subpop.0.species.connection-struct-prob          = 0.5
pop.subpop.0.species.add-node-prob               = 0.5
pop.subpop.0.species.remove-node-prob            = 0.5
pop.subpop.0.species.add-rule-prob                   = 0.05
pop.subpop.0.species.remove-rule-prob                = 0.05
pop.subpop.0.species.mutate-rule-prob                = 0.99
pop.subpop.0.species.mutate-seed-prob                = 0.99

eval.problem.boost-interval = 250
eval.problem.max-boost = 1

eval.problem.embryogeny.*        = false,true
eval.problem.boosting.*          = false,true
eval.problem.incr-complexity.*   = false,true
eval.problem.fit-dist-metric.*   = false,true

eval.problem.type                = TIMIT
eval.problem.num-inputs          = 13
eval.problem.num-outputs         = 53

eval.problem.output-amount-init = 5
eval.problem.output-amount-step = 5
eval.problem.output-amount-step-threshold = 0.8
eval.problem.output-amount-time-threshold = 40
eval.problem.phoneme-fit-dist    = 5

eval.problem.train-set-dir = /files/datasets/timit-set-divided/train/
eval.problem.test-set-dir = /files/datasets/timit-set-divided/test/
```

A.7 Iris performance experiment parameter file

```
# banana probabilities experiment
parent.0 = ../grad.params
target-folder = /iris-performance-experiment-big

breed.elite.0        = 5
pop.subpop.0.size    = 5000
generations          = 501

seed.0 = 8022,4119,7466,6880,4481,4189,7713,9251,1748,2841
seed.1 = 7050,3433,1378,699,7939,188,28,4934,4397,8016
seed.2 = 6563,9059,3195,3127,6412,8920,3634,5484,8774,2740
seed.3 = 8740,8506,8681,2680,7497,5042,8911,999,6855,8144
seed.4 = 6889,5521,9948,2744,9385,6878,3641,2697,2391,6835
seed.5 = 9920,7984,8836,220,1055,4640,751,4853,9738,9163
seed.6 = 8849,5663,4181,9737,5387,8350,2609,8816,784,7506
seed.7 = 2138,4322,5555,1540,5674,8760,3559,5458,7940,245

pop.subpop.0.species.crossover-prob                  = 0.9
pop.subpop.0.species.mutation-prob                   = 0.7

pop.subpop.0.species.reset-prob                      = 0.01
pop.subpop.0.species.node-weight-mutation-prob  = 0.25
pop.subpop.0.species.connection-weight-mutation-prob= 0.5
pop.subpop.0.species.connection-struct-prob          = 0.5
pop.subpop.0.species.add-node-prob               = 0.5
pop.subpop.0.species.remove-node-prob            = 0.5
pop.subpop.0.species.add-rule-prob                   = 0.05
pop.subpop.0.species.remove-rule-prob                = 0.05
pop.subpop.0.species.mutate-rule-prob                = 0.99
pop.subpop.0.species.mutate-seed-prob                = 0.99

eval.problem.boost-interval = 250
eval.problem.max-boost = 1
eval.problem.output-amount-init = 2
eval.problem.output-amount-step = 1
eval.problem.output-amount-step-threshold = 0.8
eval.problem.output-amount-time-threshold = 250

eval.problem.type            = IRIS
eval.problem.num-inputs      = 4
eval.problem.num-outputs     = 3
num-datasets = 10
eval.problem.train-set-dir   = /files/datasets/iris/folds/iris-*-train.txt
eval.problem.test-set-dir    = /files/datasets/iris/folds/iris-*-test.txt

eval.problem.embryogeny.*        = false
eval.problem.boosting            = false
eval.problem.incr-complexity     = true
eval.problem.fit-dist-metric     = true
```

A.8 TIMIT performance experiment parameter file

```
# timit component experiment
parent.0 = ../grad.params
target-folder = /timit-performance-experiment-big

breed.elite.0        = 5
pop.subpop.0.size    = 5000
generations          = 501

seed.0 = 8022,4119,7466,6880,4481,4189,7713,9251,1748,2841
seed.1 = 7050,3433,1378,699,7939,188,28,4934,4397,8016
seed.2 = 6563,9059,3195,3127,6412,8920,3634,5484,8774,2740
seed.3 = 8740,8506,8681,2680,7497,5042,8911,999,6855,8144
seed.4 = 6889,5521,9948,2744,9385,6878,3641,2697,2391,6835
```

```
14 seed.5 = 9920,7984,8836,220,1055,4640,751,4853,9738,9163
15 seed.6 = 8849,5663,4181,9737,5387,8350,2609,8816,784,7506
16 seed.7 = 2138,4322,5555,1540,5674,8760,3559,5458,7940,245
17
18 pop.subpop.0.species.crossover-prob                    = 0.9
19 pop.subpop.0.species.mutation-prob                    = 0.7
20
21 pop.subpop.0.species.reset-prob                       = 0.01
22 pop.subpop.0.species.node-weight-mutation-prob  = 0.25
23 pop.subpop.0.species.connection-weight-mutation-prob= 0.5
24 pop.subpop.0.species.connection-struct-prob          = 0.5
25 pop.subpop.0.species.add-node-prob               = 0.5
26 pop.subpop.0.species.remove-node-prob            = 0.5
27 pop.subpop.0.species.add-rule-prob                    = 0.05
28 pop.subpop.0.species.remove-rule-prob                 = 0.05
29 pop.subpop.0.species.mutate-rule-prob                 = 0.99
30 pop.subpop.0.species.mutate-seed-prob                 = 0.99
31
32 eval.problem.boost-interval = 250
33 eval.problem.max-boost = 1
34
35 eval.problem.embryogeny.*        = true
36 eval.problem.boosting.*          = false
37 eval.problem.incr-complexity.*   = false
38 eval.problem.fit-dist-metric.*   = false
39
40 eval.problem.type                = TIMIT
41 eval.problem.num-inputs          = 13
42 eval.problem.num-outputs         = 53
43
44 eval.problem.output-amount-init = 5
45 eval.problem.output-amount-step = 5
46 eval.problem.output-amount-step-threshold = 0.8
47 eval.problem.output-amount-time-threshold = 40
48 eval.problem.phoneme-fit-dist    = 5
49
50 eval.problem.train-set-dir = /files/datasets/timit-set-divided/train/
51 eval.problem.test-set-dir = /files/datasets/timit-set-divided/test/
```

B. Results

B.1 Iris performance experiment

Classifier name	μ score	σ score
Complex System (best)	1.0	0.0
Logistic	0.98	0.045
LibSVM	0.98	0.032
Complex System (avg)	0.975	0.031
MultilayerPerceptron	0.973	0.047
SimpleLogistic	0.967	0.047
FT	0.967	0.047
LMT	0.967	0.047
SMO	0.967	0.035
MultiClassClassifier	0.967	0.035
NNge	0.967	0.035
VFI	0.96	0.056
RBFNetwork	0.96	0.047
LogitBoost	0.96	0.047
ClassificationViaRegression	0.96	0.034
RotationForest	0.96	0.034
NaiveBayesMultinomial	0.953	0.045
IB1	0.953	0.045
IBk	0.953	0.045
AdaBoostM1	0.953	0.045
Ridor	0.953	0.045
BFTree	0.953	0.045
LADTree	0.953	0.045
RandomTree	0.953	0.045
NaiveBayes	0.953	0.032
NaiveBayesSimple	0.953	0.032
NaiveBayesUpdateable	0.953	0.032
REPTree	0.947	0.053
KStar	0.947	0.042
RandomCommittee	0.947	0.042
RandomSubSpace	0.947	0.042
RandomForest	0.947	0.028
Decorate	0.94	0.058
OneR	0.94	0.058
BayesNet	0.94	0.049

Classifier name	μ score	σ score
Bagging	0.94	0.049
EnsembleSelection	0.94	0.049
J48	0.94	0.066
NBTree	0.94	0.049
DTNB	0.94	0.038
LWL	0.933	0.054
AttributeSelectedClassifier	0.933	0.07
FilteredClassifier	0.933	0.054
PART	0.933	0.07
SimpleCart	0.933	0.054
END	0.933	0.063
ClassBalancedND	0.933	0.063
ND	0.933	0.063
J48graft	0.933	0.063
FLR	0.927	0.058
MultiBoostAB	0.927	0.049
DataNearBalancedND	0.927	0.066
OrdinalClassClassifier	0.927	0.066
DecisionTable	0.927	0.049
JRip	0.927	0.038
HyperPipes	0.92	0.053
Dagging	0.887	0.077
ComplementNaiveBayes	0.667	0.0
ClassificationViaClustering	0.667	0.0
ConjunctiveRule	0.667	0.0
DecisionStump	0.667	0.0
CVParameterSelection	0.333	0.0
Grading	0.333	0.0
MultiScheme	0.333	0.0
RacedIncrementalLogitBoost	0.333	0.0
Stacking	0.333	0.0
StackingC	0.333	0.0
Vote	0.333	0.0
ZeroR	0.333	0.0

Table B.1: Results of 67 classifiers from the Weka toolkit compared to the classifier (called 'Complex System' in this table) generated by the program of this thesis.