Tiago Rodrigues Baptista

COMPLEXITY AND EMERGENCE IN SOCIETIES OF AGENTS

Thesis submitted to the University of Coimbra in partial fulfilment of the requirements for the degree of Doctor of Philosophy in Informatics Engineering, supervised by Doctor Ernesto Jorge Fernandes Costa, Full Professor.

July 2012

· U C ·

University of Coimbra

Complexity and Emergence in Societies of Agents

COMPLEXITY AND EMERGENCE IN SOCIETIES OF AGENTS

Tiago Rodrigues Baptista

UNIVERSITY OF COIMBRA

Thesis submitted to the University of Coimbra in partial fulfilment of the requirements for the degree of Doctor of Philosophy in Informatics Engineering.

This work was performed under the supervision of Doctor Ernesto Jorge Fernandes Costa, Full Professor of the Department of Informatics Engineering of the Faculty of Sciences and Technology of the University of Coimbra.

University of Coimbra
Paço das Escolas
3004-531 Coimbra
Portugal

Financial support by Fundação para a Ciência e a Tecnologia (FCT) through PhD grant SFRH/BD/18401/2004

To Rita

Acknowledgments

The successful completion of this thesis, and of the work leading to it, would not have been possible without the support of a number of people to whom I owe proper acknowledgment.

First and foremost, thank you to my advisor Ernesto Costa for all the support, mentorship, contributions, and for giving me the freedom to pursue my ideas. A global thank you to all the members of the ECOS research group for all the inspiring conversations, with a special thank you to Telmo Menezes for the companionship throughout a large part of this journey, and for all the thought-provoking discussions.

During this time, I have had the privilege of collaborating with other researchers. Thank you to Philippe Caillou from the LRI research group at the Université Paris Sud XI, for his contributions and hospitality. Thank you to all the people I met at the CASCI research group of the Indiana University, Bloomington, especially to Luis Rocha for the important input and contribution to a part of this thesis, and for his and his family's hospitality in Bloomington.

A very special thank you goes obviously to my family and close friends, without whose support this work would not have been possible, especially my parents and my brothers. A special thank you to my mother not only for all the unconditional support and encouragement during this phase of my life, but also for the overall support and upbringing. A special thank you must go also to my brother Ricardo who sometimes felt first hand the frustrations and difficulties I went through.

Last, but certainly not the least, thank you to my soon to be wife, Rita, who accompanied me in all this journey, sharing with me all the emotional turmoil it caused.

Tiago Baptista
Coimbra, August 2011

Abstract

Throughout the last decades, Darwin's theory of natural selection has fuelled a vast amount of research in the field of computer science, and more specifically in artificial intelligence. The majority of this work has focussed on artificial selection, rather than on natural selection. In parallel, a growing interest in complexity science brought new modelling paradigms into the scene, with a focus on bottom-up approaches.

By combining ideas from complex systems research and artificial life, we present a multi-agent simulation model for open-ended evolution, and a software framework (BitBang) that implements it. We also present a rule list based algorithm implemented for the brain component of the agents. Genetic variation operators were created to drive the evolution of the rule list brains.

Several simulation environments were created using the BitBang framework. Experimental results are presented and analysed, validating our model. The results presented show that the model is capable of evolving agents' controllers in an open-ended evolution simulation. We see that populations evolve sustainable reproduction behaviours, without hard-coding the reproduction conditions into the simulations. By providing evolutionary pressure through the modelling of the environment, we see that on increasingly complex environments, agents evolve increasingly complex behaviours. The rule list brain is shown to provide an important analysis advantage by having readability into the agents' evolved behaviours. This feature proved to be especially important when unexpected behaviours emerged.

Keywords: Artificial Life, Open-Ended Evolution, Complex Systems, Multi-Agent Systems, Computational Intelligence.

Contents

List of Figures

List of Tables

Chapter One

Introduction

In this chapter we present the motivation behind the work described in this document. We then detail the problem we aimed to solve and our contributions to that regard. Finally, we set out the roadmap for the rest of this document.

1.1. Motivation

Ever since the dawn of its time, Man has always wondered about the diversity and complexity found in Nature. This interest has driven a lot of research work in a multitude of sciences.

The work of Charles Darwin and his theory of the evolution of species (1859) is certainly one great example of this interest. In his seminal work, Darwin has showed that all species found in Nature descended from common ancestors, as a result of a process he called natural selection. This process determines that organisms that possess traits advantageous to their survival and reproduction will gradually overtake a population. Darwin's theory was rather successful in convincing most biologists and the general public that evolution had indeed occurred. However, he did not include in his theory a detailed explanation of how new species arise.

It was only later in the beginning of the twentieth century that the recognition of Gregor Mendel's work on genetics (1985), brought a better explanation for the process of speciation. Combining Mendel's and Darwin's ideas, the modern evolutionary synthesis theory provided a widely accepted and unified explanation of evolution. Later, the discovery of the double helix structure of DNA by Watson and Crick (1953), and of the molecular structure of nucleic acids by Wilkins (1953), paved the way to a better understanding of the processes involved in the evolutionary exploration of possibilities, like genetic mutation and crossover.

On a different path, but also with a deep interest in the complexity found in Nature, is the recent research field of Complex Systems. Although one could consider that this type of system has been the subject of study since ancient times, it was only in the twentieth century that the specific study of the properties of these systems became a field on its own, with application in a multitude of sciences. A formal definition of complex system is

yet to receive consensus. However a generally accepted definition is that a complex system is a system composed of a large number of interacting parts that as a whole exhibits properties not directly inferred from the properties of the individual parts. Some examples of such systems include ant colonies, human social structures, economic markets, climate, the human brain, or the Internet.

In computer science, the pursuit to reproduce the complex behaviours found in Nature is also a recurrent research topic. Most of the work done on this topic can be attributed to the field of Artificial Intelligence (AI) (Russel and Norvig 2002), whose broad aim is to mimic Human intelligence and implement it in machines. Some of this effort has used inspiration on Biology to create intelligent computational systems. One example of such an endeavour was the creation of Artificial Neural Networks (ANN) that try to mimic the functionality of the Human brain. This direction of research involves the study and reproduction in silico of structures found in Nature.

A more recent trend in AI research deals not as much with the structures found in Nature, but with the processes by which they came to be. This kind of nature inspired system has an example on the field of Evolutionary Computation (Eiben and Smith 2003). Using some concepts of Darwin's evolutionary theory and Mendelian genetics, Evolutionary Computation techniques are mostly used to solve combinatorial optimization problems. A number of different methods were developed, like Lawrence J. Fogel's Evolutionary Programming, John H. Holland's Genetic Algorithm, Ingo Rechenberg and Hans-Paul Schwefel's Evolution Strategies, or the more recent Genetic Programming method. In a nutshell, these techniques evolve populations of solutions to a given problem, by promoting diversity using mutation and recombination operators, and selecting the best ones by evaluating their fitness through a fitness function (or cost function).

Using most of the same principles of the afore mentioned models, Artificial Life (ALife) has a generally different goal. Instead of being used to solve combinatorial optimization problems, ALife deals mostly with the imitation of traditional biology, trying to recreate not only the natural

phenomena, but also synthetic life. In fact, Christopher Langton (1989) defines Artificial Life as modelling not only *life-as-we-know-it*, but also *life-as-it-could-be*.

Equally, on the field of complex systems, computer science has had a major role in the creation of tools for research. When systems are too complex to model using standard mathematical tools, one common solution has been to model the system using a computer simulation. One such simulation method, gaining momentum, are multi-agent systems. These systems are composed of multiple interacting intelligent agents who share several important attributes, of which, the most important are autonomy, local views, and decentralization.

A special case of multi-agent systems are John H. Holland's (1995) Complex Adaptive Systems (CAS), where both the agents and the whole system are adaptive. Some of the interesting properties of CAS include emergence and self-organization. These properties are common to most complex systems, and constitute a major source of inspiration when designing artificial simulations.

Today the process of evolution of species is well understood thanks to the natural selection theory proposed by Darwin, and complemented by the laws of inheritance discovered by Mendel. Evolution is a slow, time consuming process, that started more than three billion years ago. In the last years Darwin's theory and the molecular biology central dogma — "DNA made proteins, and proteins made us" — has been under criticism and revision, with some researchers pointing out the importance of other dimensions (e.g. epigenetic, behavioral, symbolic) to the process of natural evolution. Notwithstanding, nobody denies that there is no goal, no plan and no end in evolution: it is an open-ended process. An established definition of open-ended evolution has not yet surfaced, however, some authors consider that one of the major requirements is the absence of an explicit fitness function (Channon 2000; Channon and Damper 2000). In other words, to have open-ended evolution, a system should be based on Natural Selection rather than Artificial Selection.

In the last few decades, both the fields of complex systems and evolutionary computation have been the focus of a large quantity of research work, both theoretical and applied. However, not as much on the interplay of the two fields. We believe that, by combining aspects of both complex systems and evolutionary computation, it is possible to evolve complex behaviours in an open-ended evolution environment.

1.2. Goals

The main goal of our work is to develop a mechanism to evolve complex behaviours in societies of agents. Namely, we aim to develop a multi-agent simulation environment where agents are able to evolve complex behaviours from initial simple configurations. Rather than relying on fitness functions to provide evolutionary pressure, we favour open-ended evolution in the sense that the evolutionary pressure should be exerted by the specificities of the environment and of the agent's makeup. We strive to remove the burden of over-specifying from the researcher.

Another goal of our work is to produce a software framework that implements the concepts proposed. This tool should be flexible enough to enable its use in different fields of research on complex systems, e.g. Biology, Social Sciences, Physics, or Mathematics. We aim to make this tool available to the community, so that our work can have application in all those fields.

The entertainment industry has been using tools developed by artificial intelligence research. For example, in most modern computer games some form of artificial intelligence algorithm is used to control the non-player characters and logic. In more recent developments, these concepts have also had application in movies' special effects. We believe the work proposed here can also be applied to these areas. For example, the artificial intelligence of non-player characters in computer games could be evolved rather than hard coded. Regarding the use in movies' special effects, one possible application could be the evolution of more accurate behaviours for characters in crowd simulations.

1.3. Contributions

The main contribution of our work is the model developed to accomplish the goals proposed above. The design of this model encompassed several steps, each giving rise to an important contribution on its own. Thus, we can list the several contributions of our work as:

- A novel multi-agent model for the open-ended evolution of complex behaviours, based on principals from Complex Systems, Evolutionary Computation, and Artificial Life;
- A specific agent brain model to use in our simulation environments. The agent brain model created is a rule list;
- An implementation of genotype editing for the brain model used, based on previous work by Luis Rocha. Genotype editing aims to capture the essential aspects of the RNA editing process in biology.
- A software framework – BitBang – that implements the multi-agent model developed, and combines it with both a graphical three-dimensional engine and a physics engine to enable the simulation and visualization of three-dimensional worlds. Due to its object-oriented and abstract architecture, the BitBang Framework can be used in different complex systems simulation scenarios;
- Several different simulation environments were created for validation of the model. The several considerations and conclusions that came out of these simulations also constitutes an important contribution of this work.

Part of the work presented here has also been published in several peer-reviewed conferences and journal (Baptista and Costa 2006, 2007, 2008, 2011; Menezes et al. 2006a, 2006b).

To allow widespread use of the software created, both for complex systems and artificial life research, the BitBang Framework is made available to the community under the open source license GPL version 2[1].

1 *The text of this license can be reviewed at http://www.gnu.org/licenses/old-licenses/gpl-2.0.html*

1.4. Roadmap

The rest of this document will be organized as follows. In chapter two we will describe a series of background concepts required to better understand our work, as well as present the state of the art in open-ended evolution multi-agent models for artificial life and complex systems research. In chapter three we detail the architecture of our model. We describe the multi-agent model, the rule list brain model, and also the BitBang software framework. In the next two chapters we will present the simulation scenarios developed to test our model, and analyse the results of the experiments conducted. Finally, in chapter six, we present our conclusions and discussion of future work.

We also provide a companion Compact Disc with the source code and binaries of the BitBang Framework, the source code and binaries of the simulation scenarios created, the logs of the experiment results, and the papers published.

Chapter Two

Background

In this chapter we present some background concepts, material to the research described in the following chapters. We will also present a study on the state of the art in multi-agent systems for artificial life and complex systems research.

2.1. Core Concepts

In this section we give a brief introduction to the core concepts underlying the work presented in this thesis. We will start with an introduction to the concepts of complex systems, emergence, self-organization, and multi-agent systems, and proceed into concepts more specific to artificial intelligence, like evolutionary computation, artificial life, and open-ended evolution.

2.1.1. Complex Systems

The study of complex systems has been gaining momentum over the last decades (Kauffman 1993; Holland 1995; Per Bak 1997; Strogatz 2003). However, a definitive definition of complex system, or some of the related concepts, like emergence and self-organization, has not gathered consensus yet. Nevertheless, we can define a complex system as one composed of several individual elements that interact in such a way that the system as a whole will exhibit properties, not directly derived from the individual components.

An important characteristic of complex systems is their usual hierarchical structure. That is, a set of components of a system interact in such a way as to form higher level structures, which in turn, interact to also form other higher level structures. We can name these hierarchical layers as levels of complexity. In a complex system, whenever phenomena observed on a given level of complexity, that is not directly derived from the properties of the lower level, we say those phenomena are emergent. This kind of levelled structure and emergent phenomena can be easily found on a variety of natural systems, like physical, biological or human social systems (Per Bak 1997).

Another important concept related to complex systems is that of self-organization. Whenever a pattern or structure can be observed in a system,

without it being enforced or planned by a central authority or external element, the system is said to have self-organization. Again, we can find a number of examples of self-organization in Nature (Camazine et al. 2001; Strogatz 2003), like the swarming behaviours of insects, molecular self-assembly, or the evolution of life.

2.1.2. Multi-agent Systems

The dynamics of complex systems, being nonlinear, are usually difficult to model using standard mathematical tools, like differential equations. One tool that has recently gained widespread adoption is that of computer simulation. By simulating the several individual components and their interactions of one level of a complex system, it is possible to observe the dynamics and emergent phenomena at the higher levels. One specific type of simulation used for complex systems research is the multi-agent system.

The concept of multi-agent system (MAS) can be broadly defined as a system composed of several interacting intelligent agents. As defined by Wooldridge (2002), an intelligent agent has four main properties: autonomy, reactivity, pro-activity, and social-ability. An agent is autonomous if it is capable of independent action, i.e., there is no central control mechanism dictating their actions. The property of reactivity means that an agent is able to react to changing conditions in the environment. Pro-activity is defined as the ability to generate and attempt to achieve goals. And lastly, social-ability is the power to interact with other agents via cooperation, coordination, or negotiation.

Although other mechanisms can be used to implement MAS (e.g robotics), most use computer simulations. The ongoing increase in computation power available to researchers, has fuelled a growing popularity of this tool. A variety of research areas have used MAS to study complex systems, namely biology, economics and other social sciences, or the most relevant to our work, artificial intelligence.

A special kind of complex system, highly related to MAS, are complex adaptive systems (CAS). This field of research was born at the highly interdisciplinary environment of the Santa Fe Institute, with one the main pro-

ponents being John H. Holland. The main characteristic that defines CAS is that of coherence under change (Holland 1995). A generic definition of CAS given by Holland (2006) is: "Cas are systems that have a large number of components, often called agents, that interact and adapt or learn". This adaptation to changes in the system occurs in such a way that some type of coherence is preserved.

2.1.3. Evolutionary Computation

Evolutionary Computation (EC) is a research field within computer science (Eiben and Smith 2003), and more specifically artificial intelligence. EC applies the concept of natural evolution to computational problem solving, like combinatorial optimization problems. The core inspiration to EC comes from Darwin's theory of the evolution of species (1859), and the genetics theory pioneered by Mendel (1865), later combined to form the *modern evolutionary synthesis* (Huxley 1942). Despite the fact that currently the field of EC is considered to encompass a greater number of techniques, those know collectively as Evolutionary Algorithms (EAs) form the core of the field.

Historically, a number of sub-categories of EAs have been developed, for the most part independently of each other. In the USA, Fogel, Owens and Walsh introduced evolutionary programming (Fogel et al. 1966), while Holland called his method genetic algorithm (Holland 1975). Meanwhile, in Germany, Rechenberg and Schwefel proposed evolution strategies (Rechenberg 1973; Schwefel 1981; Schwefel 1995). These three methods were later joined by a fourth alternative called genetic programming, proposed by Koza (1992). At the core, all these techniques share a common generic algorithm that we present in pseudo-code in Listing 2.1.

```
1.  population = GenerateRandom()
2.  fitnesses = Evaluate(population)
3.  WHILE not stop_condition DO
4.      parents = Select(population, fitnesses)
5.      offspring = Recombine(parents)
```

```
6.      offspring = Mutate(offspring)
7.      fitnesses = Evaluate(offspring)
8.      population = Select(offspring, fitnesses)
```

Listing 2.1 *Pseudo-code of the generic procedure of evolutionary algorithms*

Given a specific problem to be solved, the EA process starts by generating a population of random solutions to the problem, and then will continuously select and vary that population of solutions, biased by a fitness function. This generic algorithm can be changed by employing different techniques or parameters to its most important components (Eiben and Smith 2003):

- representation of individuals
- fitness function
- population
- parent selection
- variation operators
- survivor selection
- initialization procedure
- termination condition

The different types of EAs will vary mainly on the techniques or parameters employed in one or more of these prime components. One example can be the representation of individuals. In genetic algorithms, individual solutions are typically represented as strings over a finite alphabet (e.g. bit strings), whereas in evolutionary strategies they are represented as real-valued vectors. In evolutionary programming, individuals are classically, finite state machines, and in genetic programming, trees are used.

In EAs the representation of individuals is thought of as the mapping from genotypes to phenotypes. Following the inspiration from biology, an individual has a genotype and a phenotype. Its phenotype is the candidate solution to which the fitness function can be applied, and the genotype is an encoding used to represent the phenotype. For example, considering that we want to maximize the value of a function $f(x)$, where x is an integer value. Considering the candidate solution $x=34$ (the phenotype), we could

use its binary code as its representation, making its genotype the bit-string 100010. The variation operators will be applied to the genotype of solutions, and the fitness function operates on their phenotypes.

The representation can either be direct or indirect. We say that we have a direct representation if there is in practice no mapping between phenotype and genotype. In that case the individual's genotype is equal to its phenotype. In indirect representations, as exemplified above, an encoding is used to map the phenotype to the genotype.

Because the variation operators are applied to the genotype of candidate solutions, they are highly related to the representation chosen. Both these choices, representation and variation operators, can have a high impact on the performance of the evolutionary algorithm and are tightly coupled to the specific problem to be solved (Tavares 2007).

The variation operators used can either be recombination operators or mutation operators. Recombination operators are applied to two parents to produce one or more offspring by combining the parents genotypes, and represent the biological concept of sexual reproduction. The mutation operators are used to vary one individual by randomly changing its genotype. On a given instance of application, we can use both recombination and mutation, or only the latter.

The selection mechanisms in EAs are responsible for providing the necessary evolutionary pressure towards higher quality solutions. Two separate selection steps may be defined, occurring at different times in the algorithm: *parent selection* and *survivor selection*. Typically, parent selection is stochastic, whereas survivor selection is usually deterministic. Having a probabilistic selection step allows low quality individuals to have a small, but positive chance of being selected, avoiding the search process to get stuck in a local optimum. Both selection steps are usually fitness biased, where fitter individuals have a higher probability of being selected. A number of different parent selection strategies exist, of which probably the most common are *roulette wheel* and *tournament*. In *roulette wheel*, the probability of an individual of being selected as a parent is biased by its relative fitness in the population. Considering that f_i is the fitness value of an individual, and that

N is the size of the population, the probability p_i of the individual i being selected is given by:

$$p_i = \frac{f_i}{\sum_{j=1}^{N} f_j} \tag{2.1}$$

In *tournament selection*, a number of individuals is randomly chosen from the population, and from those, the one with the highest fitness is selected as a parent. This procedure is repeated until all parents are selected. By changing the number of individuals that are randomly chosen each time (tournament size), the selection pressure can be adjusted.

The initialization procedure determines the generation of the initial population. The most common is to randomly generate the individuals, although, in some cases, heuristics can be employed to ensure an initial population with higher fitness, or that uniformly covers the search space. The termination condition will determine when to stop the search process. In some cases, there is a known optimum fitness that we look to achieve, providing a natural stopping condition. However, being a stochastic process, the EA may not reach that optimum in a reasonable time frame, and usually, other termination conditions are also used. For example, we can stop the search after a number of generations have been created, after a specified CPU time has elapsed, or after a specified number of fitness evaluations.

2.1.4. Artificial Life

Artificial Life (ALife) is considered by some as a sub-field of evolutionary computation. Though it does share its inspiration in Biology with EC, especially regarding the theory of evolution, at its core, ALife has a different purpose. Instead of trying to solve optimization problems, ALife's main goal is to replicate the mechanisms found in Nature to study not only *life-as-we-know-it,* but also *life-as-it-could-be* (Langton 1989). This means that the research in ALife is not restricted to the imitation of life in order to study its properties, as it also studies the synthesis of life-like systems from the bottom-up. This type of synthesis can lead us to create structures that have life-like properties, but do not have a direct equivalent in the real

world. We could envision this as exploring the different paths that evolution could have taken.

ALife experiments are not constrained to those performed in computers. In fact, the field can be divided into soft, hard, and wet ALife, refering to the medium used to perform the experiments. Soft ALife uses software, hard ALife, hardware, and wet ALife, biochemistry (Bedau 2003). Of all these mediums, the one that is mostly related to the our work is that of soft ALife.

The term "artificial life" was coined by Christopher Langton (1989), but the research field has its deepest roots in John von Neumann's self-reproducing, computation-universal cellular automata (von Neumann and Burks 1966). The usage of cellular automata, and other low-level simulations, like artificial chemistries, has had a strong presence in ALife research. Another popular path has been the study of higher-level phenomena, mainly through the already introduced concept of agent and the theory of evolution, similar to what is done in EC.

A number of different approaches to the evolution of agents in ALife research can be identified. For example, the coevolution of morphologies and controllers, as pioneered by Karl Sims (1994*a;* 1994*b*), and later developed upon by other researchers (Miconi and Channon 2006; Miconi 2007; Chaumont et al. 2007; Lassabe et al. 2007). Another important direction of research has been on a more generic evolution of digital organisms, using virtual CPUs as a metaphor, like in Tierra (Ray 1992) or Avida (Adami and Brown 1994; Ofria and Wilke 2004). Other fields of Alife like evolutionary robotics (Brooks 1990; Nolfi and Floreano 2000) or even artificial economics (Tesfatsion 2002), have also had developments in recent years. Some of the research referred can fall into the field of artificial life, evolutionary computation, or complex systems.

The work presented in this document falls on the category of artificial life systems, using a high-level multi-agent systems approach. A variety of similar systems have been developed under the umbrella of ALife, some of which were referenced above. The ones most relevant to our work will be further detailed later in this chapter.

2.1.5. Open-Ended Evolution

The concept of open-ended evolution is mainly related to artificial life research, and has been considered one of the open issues in alife research (Bedau et al. 2000). Although a formal definition of open-ended evolution has not yet gathered consensus, it can be broadly defined as the ability of an evolutionary system to continually produce novel forms (Standish 2003). To achieve open-ended evolution, a number of authors consider that a major requirement is the absence of an explicitly defined goal or notion of "better" individual (Channon 2000; Channon and Damper 2000).

Both evolutionary computation and artificial life research have a core inspiration on Darwin's concept of Natural Selection (1859). However, in the process of adapting this concept to computational problem solving, some compromises were made, namely the creation of an artificial evolutionary goal, usually via the fitness function. In the real world, there are clearly no explicit fitness functions governing the evolution of organisms. Thus, in these artificial systems we say that instead of Natural Selection, we have Artificial Selection. Packard (1989) makes this distinction when comparing extrinsic and intrinsic adaptation.

This distinction between natural and artificial selection is paramount to the construction of open-ended evolution systems (Channon and Damper 1998; Channon 2000). In fact, Darwin (1859) introduced the concept of artificial selection by referring to selective breeding of animals and plants carried out by mankind, which is an active selection mechanism. In contrast, natural selection is a passive selection mechanism. Channon (2001) further developed on this distinction between natural and artificial selection, by making the distinction between biotic and abiotic selection. Biotic selection is induced by living organisms (biota) whereas abiotic selection is not. In artificial systems, the most commonly used tool to implement selection is the usage of fitness functions to select the best individuals. Thus, one of the main requirements to achieve open-ended evolution is to free the artificial evolution systems from the fitness function.

Research into artificial evolution systems that use natural selection, and into open-ended evolution, has seen less adoption than into systems with

artificial selection. This trend might have some ground in the fact that open-ended evolution seems harder to model (Maley 1999) and heavier on the computational resources. Notwithstanding, there have been some important efforts into creating artificial systems that exhibit open-ended evolution. Von Neumann's self reproducing automata (1966) may be considered as the first computational system capable of open-ended evolution (Rocha 1998; McMullin 2000). Sannier's distributed GA (1987) might be one of the early examples of using what Ray (2001) would consider partial natural selection. Other examples include Tierra (Ray 1992), Polyworld (Yaeger 1994), ECHO (Holland 1995), Geb (Channon 2001), New Ties (Eiben et a. 2007), or Evosphere (Miconi 2007). Also, in the field of evolutionary robotics there have been some recent attempts to implement open-ended evolution (Bianco and Nolfi 2004; Baele et al. 2009). Some of these examples, the ones most relevant to the work described in this document, are detailed in the next section.

In our work, we will refer to open-ended evolution solely regarding the absence of an explicit fitness function, or any other type of artificial selection mechanism.

2.2. Models and Frameworks

In this section we present the state of the art in artificial life multi-agent systems. A larger number of systems could be described, however, we limit this analysis to the ones that are more closely related to our work or that served as inspiration to the model presented in this document.

2.2.1. Tierra and Avida

Tierra is an artificial life simulation created by the ecologist Thomas S. Ray (1992). In Tierra virtual creatures (computer programs) compete for central processing unit (CPU) time and access to memory in a virtual machine. Competing programs in virtual computers had been implemented before, most notably in the Core War game (Dewdney 1984, 1988).

The assembly language created for the virtual computer in Tierra was developed with a special care to allow evolution to be possible. This was

accomplished by creating a small instruction set, removing the use of operands, and using addressing by template.

The Tierran virtual computer has one block of memory, called the "soup" where all programs live. This block of memory can be viewed as the environment. Each creature is assigned one block of memory from the soup, over which it has exclusive write privileges, forbidding creatures to write to each others blocks. Read and execute privileges are not protected. All creatures can read and even execute each others code. Creatures can, through the use of a memory allocation instruction, a second block of memory can be reserved with write privileges. This second block can be used to grow or reproduce into. When the instruction divide is called, the creature looses its write access to this second block.

To enable the creatures to live simultaneously, the Tierran operating system must have multitasking. The system attributes time slices to each creature in the soup one by one, through the use of a circular queue. When a creature is born, it is given one virtual CPU and its placed in this queue. The number of instructions executed at each time slice is proportional to the size of the program, and this proportion can be configured. By doing this, it is possible to exert selective pressure for larger or smaller programs.

Another important process of the operating system is the *reaper*. This process is used to kill programs when the total memory of the soup is almost filled up. The reaper process uses a *first in first out* (FIFO) queue. Creatures are inserted at the beginning of the queue when they are born, and when the need to kill a program arises, the one at the end of the queue is selected. Programs are killed by deallocating their memory block, removing their virtual CPU, and removing them from both queues (slicer and reaper). Their instructions however, are not deleted from the soup.

In order for evolution to occur, a variation mechanism had to be introduced. Variation is implemented using bit-flip mutation. There are two situations when these mutations occur. First, in the background, the operating system randomly mutates one bit of the soup, with a low probability. Secondly, the copy instruction imperfectly copies the code. In this process, a bit flip can occur in the destination copy of the code. This mutation can

occur with a higher probability than the background mutation. Other than mutation, a second mechanism was introduced. The instructions have a low probability of being executed incorrectly, making the outcome of the programs nondeterministic. In Tierra, simulations are initially seeded with a hand-coded program that essentially has the ability to self-replicate. No other functionality was coded into this seed program.

Emergence of interesting organisms was show to occur in Tierran simulations. Most of these emerged behaviours relate to parasitism or the coevolution of hosts and parasites, resulting in a Lotka-Volterra cycle (Lotka 1956).

Another system closely related to Tierra is Avida (Adami and Brown 1994; Ofria and Wilke 2004). Avida extends the Tierra model by adding the two-dimensional grid where the creatures live, thus introducing the concept of spacial locality to the simulated world. Another extension included in Avida is the usage of fitness rewards. By executing predefined operations, creatures can be rewarded with energy. The slicer method is also different than the one used in Tierra. Although several slicing algorithms exist in Avida, the one that mostly relates to its goals is the "integrated" slicer. Agents are awarded time based on a merit measure, that is also related to the execution of predefined operations. Avida's time slicer and merit system superimpose a fitness landscape over the evolutionary process. In that regard, although arguable, Avida introduces artificial selection into the simulation.

Avida has gained great recognition in the artificial life community as a successful tool for research in theoretical biology, with several contributions to the field in recent years.

2.2.2. ECHO

Echo is generic ecosystem model, or class of models, developed to simulate complex adaptive systems (Jones and Forrest 1993; Holland 1994, 1995; Hraber et al. 1997). Echo is devised as a simple model that is then extended by incrementally adding mechanisms, as proposed by Holland.

The Echo world, shared by all extensions to the model, consists of a network of interconnected sites. The neighbourhood relation, or geographical disposition of these sites is left to the designer to choose. However, the most

common is to use a two-dimensional grid. In each site, a resource fountain continuously generates resources for that site. A site can hold many agents. The amount and type of resources generated, varies from site to site. The resources in the Echo models are represented by letters of the alphabet. For example, we might define for a given simulation an alphabet of {a,b,c,d}. Agents are constructed with strings on this alphabet.

In the basic Echo model, an agent is composed of a reservoir of resources, and a chromosome string. An agent can gather resources either from the site it occupies, or through interaction with other agents. The agent will need these resources to reproduce, as it will only be able to do so when it has gathered enough resources to make a copy of its chromosome string. This makes the implicit fitness be the agents' ability to gather resources. The chromosome in this basic model contains only the specification of two tags, an offense tag, and a defence tag. When two agents interact, the offense tag of one agent is compared to the defence tag of the other agent (and vice-versa), to determine the outcome of the encounter in terms of exchange of resources. This basic model is then augmented with the following five mechanisms: an interaction condition, a transformation segment, an adhesion tag, a mating condition, and a replication condition.

The interaction condition mechanism permits selective interaction between agents. An agent will only interact with other agents whose offense tag matches his own interaction condition. The transformation segment mechanism allows agents to transform resources. For example, an agent could transform a resource that is abundant in the environment into a resource it needs for reproduction. The adhesion tag mechanism allows agents to adhere to one another, thus forming multi-agent aggregates. The amount of adhesion will be determined by the degree of match between two agents adhesion tags. The mating condition mechanism is similar to the interaction condition one. It allows agents to selectively mate with one another. To accomplish this a mating condition is checked against the offense tag to determine if the agents mate. By adding this mechanism, the emergence of species is made possible. Finally, the replication condition mechanism is related to the concept of multi-agent aggregates. An agent will check its

replication condition against the offense tag of other active agents within the same aggregate to determine if it will try to reproduce. The addition of these extension mechanisms has the main objective of mimicking the process of producing a complex aggregate from a single seed organism.

To complete the simulation model, two steps are added to all the agent-agent and agent-site interactions: agent migrations and agent deaths. No specific recommendations are made on the nature of implementation of these simulation steps. A simple example given for both cases is to randomly choose agents to migrate, or to die.

The Echo model has been implemented, primarily at the Santa Fe Institute. Their implementation however does not cover all the extensions of the model. As of version 1.3 beta 2, the adhesion tag and replication condition extensions are not present in the tool. To assess if Echo does possess all the properties of CAS, Smith and Bedau (2000) analysed the results of a series of experiments using this simulation tool. They conclude that, although Echo does show some of the properties of CAS, it does not possess all of Holland's proposed properties, namely that of aggregation. These results can be attributed to the implementation not having all the extension mechanisms.

Recently, a new implementation of Echo was developed (McIndoe 2005) that added the adhesion tag mechanism. The author collected experimental data that indicates the emergence of aggregates, but also states that only with a final implementation of Holland's final extension (the replication condition) will it be possible to fully test the model.

2.2.3. Polyworld

Polyworld (Yaeger 1994) simulates the evolution of organisms living in a flat world. A three-dimensional visualization is provided but the simulation is actually in two dimensions. In this world agents can move freely, reproduce sexually, fight and kill each other, eat dead agents, and eat food that grows throughout the world. The environment can optionally contain barriers that constrain the agents movement in the world and can favour speciation via geographical isolation.

The controller of the agents in Polyworld is an artificial neural network. The input to this network, and thus the agent's world perception, is a rendered, one-dimensional, image of the world from the agent's point of view. Added to this world perception, other inputs to the neural network are the agent's own health-energy level and a random value. The output of this network will determine the agent behaviour from a set of seven possible actions: move, turn, attack, eat, mate, focus, and light. A Hebbian learning algorithm (Hebb 1961; Paulsen and Sejnowski 2000) is used to update the network's weights, allowing the agents to learn during the course of their lifetime.

Both the neural network topology and the agents' physiology are encoded in their genotype, composed of 8-bit genes. On the initial creation of the population the genotype of the agents is created randomly, and is then evolved through both mutation and crossover operators. Agents reproduce when two of them physically overlap and both execute the mating action. As the world is initially populated with random agents, it is highly probable that the agents will not successfully reproduce. To solve that, a fitness function is used to create new agents as the offspring of two highly fit individuals. This fitness measure is only used up to the point that agents exhibit what Yaeger calls a Successful Behaviour Strategy (SBS), which allows the population to have self-sustainable reproduction.

Agents store two types of energy: health-energy and food-value-energy. Both are replenished by eating, and both are spent by living. But only the health-energy is affected by fighting. In Polyworld all agent activity has an energy expenditure, that is affected by the organisms size and strength. Neural activity also carries an energy expenditure, proportional to the size of the network.

From the experimental simulation runs, Yaeger (1994) found a number of recurrent evolved behaviour strategies, that can he categorized as different species. The *frenetic joggers* are simply always running forward, eating and mating. The *indolent cannibals* are slow moving agents that live close together, mate with each other, kill and eat each other. The *edge runners* roam the world always following the edges. And the *dervishes,* that appear

when the environment is configured so that the crossing of an edge is deadly, explore the world in rapid turning movements to avoid the edges.

Yaeger also found a number of interesting individual behaviours evolved in some simulation runs: responding to visual stimuli by speeding up, responding to an attack by running away, responding to an attack by fighting back, slowing upon encountering each food patch, seeking out and circling food, and following other organisms.

2.2.4. Geb

Geb (Channon and Damper 1998; Channon 2001) was developed as a system where virtual organisms evolve by means of natural selection alone. It is in many aspects similar to Polyworld, despite having been created without prior knowledge of the former (Channon 2001). In fact, Geb could be seen as a simplified version of Polyworld.

Organisms live in a two-dimensional toroidal grid and are controlled by neural networks. Both time and space are discreet. Each grid cell can only be occupied by one organism, and organisms' actions are evaluated at discreet time steps. By only allowing one organism to live in each grid cell, the system creates an intrinsic limit to the maximum number of organisms in the population. There is no notion of energy in the system, so organisms only die due to fighting or if replaced by a new-born organism.

The organism's neural network outputs determine the action of the current time step. The actions available to the organisms are: reproduce, fight, turn anti-clockwise, turn clockwise, and move forward. An organism reproduces if, when executing the reproduce action, there is another individual in the grid cell directly in front (the organisms have an orientation). The offspring is placed in the cell beyond that individual, if empty, or replaces the organism being mated with, otherwise. Fighting works in a similar way. If the organism executes the fight action, the organism in front, if one exists, will be killed. The movement actions use an excitatory output of the corresponding neuron to determine the length or angle of the movement. Each network node has a bit-string character attached to it. These characters are used to match the network input nodes of an organism with the

network output nodes of other individuals. An organism's input nodes have their excitatory inputs set to the weighted sum of the excitatory output of matching output nodes from the other individuals in its neighbourhood.

Genotypes encode the neural network of an organism using a developmental system to provide the genotype to phenotype mapping. The developmental system uses a class of L-system (Prusinkiewicz and Lindenmayer 1990) with context-free production rules. The bit-string genotype encodes the production rules, that are then used to develop the neural network. Reproduction uses both crossover and mutation on this bit-string genotype. On initialization of a simulation, all cells on the grid have an individual created with a single bit 0 genotype.

A number of emergent behaviours have been identified from simulation runs. These range from very simple *do everything* and *always go forward and kill,* to somewhat more complex behaviours like turning to face a member of a dominant species and holding its direction while trying to reproduce and kill. However, identifying behaviours in Geb proved to be a difficult task as the evolved neural networks are hard to understand, and so relied solely on visual inspection of the running simulations.

2.2.5. New Ties

The New Ties system (Gilbert et al. 2006; Eiben et al. 2007) was developed to study socio-biological simulations. It implements a multi-agent system where the world is a discreet two-dimensional grid, and time passes in discreet time steps. The agents can perform actions on this environment like moving, eating, mating, talking, or picking up objects. Other actions are possible to implement, depending on the simulations intended. Agents also possess characteristics like weight, colour, shape, or sex.

The controller component of the agents are Decision Q-Trees (DQT). The DQT has three types of nodes: test, bias, and action nodes. The test and bias nodes are intermediary nodes and are used to traverse the tree up to an action node (leaf nodes). The test nodes are used to provide a situation description to the agents. Example test nodes could be my-energy-low, food-ahead, or female-nearby. When traversing the tree, if we reach a test

node, we continue to the branch corresponding to the answer to the test node. Bias nodes are traversed in a probabilistic manner, adjusted by the weights of its sub-trees.

In New Ties, the adaptation of the agents' controllers has three separate mechanisms which the authors name: evolutionary learning, individual learning, and social learning. All these types of adaptation occur on the DQT controller.

In evolutionary learning agents' controllers are subject to variation through recombination and mutation with genetic operators similar to those used in genetic programming. The tree itself is the genotype of the agent. Because the controller will be the subject of further adaptation via lifetime learning, the agents' initial controller is stored to adopt a non-Lamarckian notion of evolution. No fitness function is used in New Ties. Agents die due to age or lack of energy, and reproduce by choosing the mate action. In New Ties the initial population of a simulation is started with preprogrammed basic trees to allow the population to survive and reproduce.

The other adaptation mechanisms model lifetime learning. In individual learning the controller tree's weights are updated using a reinforcement learning algorithm, with rewards based on energy. In social learning the trees are modified by exchanging sub-trees with other agents. One of the main goals of the New Ties project is to study the interactions between these three types of learning in socio-biological simulations.

Although promising an interesting set of features, published data about the New Ties project is scarce and it mostly details the theoretical model. To our knowledge, no actual experimental data has been published, and the project seems to have stopped development.

2.2.6. Summary

We described a series of models and frameworks that can be used for artificial life simulations. The features of these systems are quite varied and mostly derive form the specific goals each one was developed for. In Table 2.1 we provide a comparison, showing the characteristics we consider are most relevant to our work.

One feature not shown in this table is that of flexibility. For example, the flexibility to simulate systems at different complexity levels, or the flexibility to change the structure of the agents or the environment. Tiera, Avida, and ECHO are all especially geared towards low level simulations. This type of simulation is more targeted towards thought experiments, rather than practical real-world scenarios. In Polyworld, a number of different scenarios can eventually be created, but all based on the same general type of agent and environment. In Geb, the structure of both the environment and the agents is fixed, and simulation is particularly geared towards evaluation of evolution through natural selection. New Ties seems to have a flexible structure for the agents, but not for the environment. However, we could not find a description of how the structure of the agents can be changed. It is also not clear if the three learning mechanisms can selectively be turned on and off.

Table 2.1 *Comparison of artificial life frameworks*

	Environment	***Brain***	***Reproduction****	***Artificial selection***	***Initial population***
Tierra	virtual machine	machine code	agent	no	hand-coded
Avida	2D grid	machine code	agent	yes (merit system)	hand-coded
ECHO	abstract / 2D grid	not applicable	system	no	hand-coded
Polyworld	2D continuous	neural network	system / agent	initially	random
Geb	2D grid	neural network	agent	no	random
New Ties	2D grid	decision trees	agent	no	hand-coded

* Reproduction is either triggered by the agent or by the system.

Other than the flexibility feature mentioned above, which we aim to provide with our model, there are two main characteristics where we focused our development: open-ended evolution, and the readability of the brain algorithm. Contributing to the former, we identify three of the char-

acteristics presented in the table. The reproduction mechanism should be triggered by the agents to avoid having to define the conditions when reproduction occurs. There should be no fitness function, or other artificial selection schemes conditioning evolution. And the initial population should be created randomly, to avoid having to define a-priori good behaviour. Regarding the readability of the brain algorithm, we wish to provide the capability of easily interpreting the evolved behaviours. To that end we chose to use a brain algorithm that is simple, yet powerful enough to enable the emergence of complex behaviours. Using a rule list, we get both those features, and also get the added benefit of being able to interpret the evolved behaviours. Considering all these characteristics, we built a new open-ended evolution, multi-agent framework, which we will detail later, since none of the described frameworks had all of our requirements.

Chapter Three

Conceptual Model and Framework

In this chapter we lay out both the conceptual model proposed to tackle the goals presented in chapter 1, and the framework implemented to test the model. We will begin by presenting a general overview of the agent-based conceptual model, and then detail all its components. Next, we will explain the evolutionary process applied to this model. On the succeeding section we detail the software framework developed — the BitBang Framework.

3.1. Conceptual Model

The model we propose has roots in both Artificial Life and Complex Systems. We use an agent-based approach to define our simulation world — a technique that is becoming more and more popular in both areas of research. Our aim was to develop a base model capable of sustaining a wide range of different simulation worlds. To that end, the components of this base model are fairly abstract.

In our model, the world is composed of entities. These can either be inanimate objects which we designate as "things", or entities that have reasoning capabilities and power to perceive and affect the world — the "agents". Both have traits that characterize them, such as colour, size, or weight. We designate these as "features". The agents perceive and change the world by means of their "perceptions" and "actions", making decisions using the "brain".

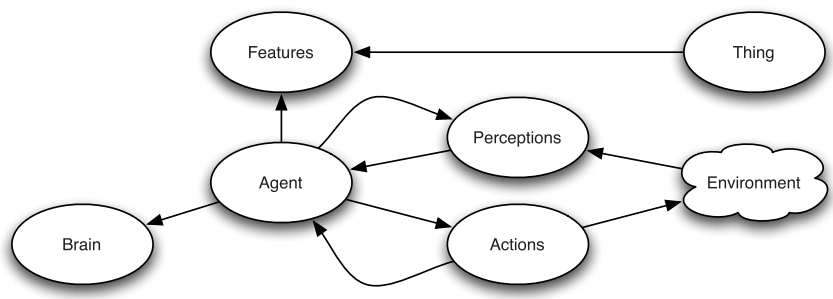

Fig. 3.1 *Diagram depicting the components of the conceptual model, and the relations between them.*

In Fig. 3.1 we show a diagram of these basic components and their relations. Note that the environment is illustrated in a different fashion than other components. This is due to the fact that the environment is a slightly different component than the rest. In fact, we don't consider the environment as a component per se, as it is made up of the set of all the agents and things in the simulation. Nonetheless, this "virtual" component is of no less importance than the rest. Actually, as we will discuss later, the environment has a major role in the facilitation of the evolutionary process.

With all these components specified for a given world, it is possible to simply start the simulation and observe the behaviours that emerge from their interaction. In this model, there is no definition of a simulation step, as there is no centralized control. In this regard, the simulation is asynchronous. The agents will, independently, perceive, decide, and act.

3.1.1. Components

In this section we will detail each component of our conceptual model, as presented in Fig. 3.1.

3.1.1.1. Agent

An agent is an object of the world that has the capability to affect the world. An agent has a set of perceptions, a set of actions, a set of features, and a brain component.

3.1.1.2. Perceptions

Perceptions are the input from the world to the agent. Being an abstract concept, we can have higher or lower order perceptions. A higher order perception would be, for example, the *friend is near* perception, and a lower order perception would be, for example, the *temperature* perception. One other thing to note is that the world referenced here also includes the agent himself, thus opening up the possibility to have a perception on oneself. That would be the case of, for example, a perception on the agent's own energy level.

One other interesting possibility is the use of direct rather than symbolic perceptions. An example of a direct perception is a render of the three-dimensional world as viewed through the agent's eyes, or the wave of the sound reaching the agent. These kind of perceptions can, at first, seem more difficult to deal with, but we believe that for a large enough world, they can prove easier to handle and also faster to compute. Also, using direct perceptions, one could aim to emerge new kinds of data processing schemes.

3.1.1.3. Actions

The actions are the output from the agent to the world. Again, we can have higher and lower order actions. We could have the action *go home* or the action *go forward*. This choice of granularity will have an impact on where on the zone of emergent phenomena we can place our world. This is explained later in this section. As with the perceptions, actions can also act on the agent himself. An example of such an action could be the action *store in memory*. Note that we categorize the perceptions and actions as higher or lower order just as an example of different possibilities for the abstract concepts. Other categorizations are also possible.

Both the perceptions and the actions can be seen as the capabilities of the agent. In a world with several species, each would have its intrinsic capabilities, and as such, what it can perceive from the world and how it can act on the world.

3.1.1.4. Features

The features are, again as an abstract concept, the characteristics of the agent or thing. It can be, for example, colour, energy, the 3D form, or anything that we want to characterize our agents and things with. A feature can then act as a source for a perception, be it a self-referenced perception like seeing your own colour, or a perception on other objects.

3.1.1.5. Brain

The brain is a decision making component. It receives the perceptions through the agent, and decides what actions the agent should take. The

brain is not bound to any kind of predefined artificial intelligence model. It is possible to implement the brain using for example, a rule system, a neural network, or anything that can take perceptions on the input and output a decision on the action to take.

3.1.1.6. Thing

A thing is an inanimate object. As opposed to the agents that are considered to be living, the things can be used to represent all nonliving entities in the simulated world. A thing has a set of feature that can be used to represent different types of objects, and different characteristics of these objects. Still, the concept of thing is of major importance to the model as it permits the artificial world to more closely mimic the real world. We can view the thing as having no active power to change the world, but make a difference in the phenomena that emerges from the interactions of the agents with the things and the perceptions the agents have of the things.

3.1.2. Evolution

In the past section we described our base conceptual model. Although the main goal of this project is to evolve complex behaviours, notice that there is no definition of an evolutionary mechanism in the base model. In an effort to maintain both its generic properties, and its biological rationality, we implement the evolutionary process by means of the agents' actions. We accomplish this by giving the agents the capability (action) of reproduction. With this method, we also add an extra flexibility to the model. The reproduction action can be implemented as a simple cloning mechanism, include variation promoters like mutation and cross-over, or any other process one would like to test.

Again, there is no central control bound to the evolutionary process. Each agent will independently choose when and, if it is the case, with whom to reproduce by deciding to execute the corresponding action. Moreover, there is no explicit fitness function or generations, like in standard genetic algorithms. The agents die due to lack of resources, predators, age, or any

other mechanism implemented in their world. All these techniques allow us to create an open-ended evolution system.

This method of implementing evolution on the model, being generic, has also the capability of being used at different levels of complexity (see section 2.1.1). It is generally accepted that complex systems are structured hierarchically, where each system is composed of subsystems, these subsystems are themselves composed of subsystems, and going on until we reach the most basic components, like sub-atomic particles. It is also generally accepted that most emergent properties appear across levels. That is, an emergent property observed at a given level, is the product of interactions of components on the level below. In our model, by implementing both the agents perceptions and actions accordingly, it is possible to simulate any level. At the limit, having enough computational power, it would be theoretically possible to simulate a system at the quantum level.

Another important theory that gathers some consensus is that evolution is not continuous, but happens mostly in large transitions (Smith and Szathmáry 1995). This theory states that, although small changes can occur and propagate in a population, the big changes are the ones that drive the evolutionary process. Some of the major transitions identified in the evolutionary history of our planet are: the origin of eukaryotes from prokaryotes, of meiotic sex, and of the genetic code itself.

3.1.2.1. Step Evolution

In our model, by combining aspects of the two aforementioned theories — levels of complexity and major transitions in evolution — we introduced in the model a technique we named *Step Evolution.* This technique consists of identifying the perceived complexity levels, and dividing the problem accordingly. We first simulate a scaled down version of the problem, and when the agents evolve to solve this simpler problem, we change the simulation by adding the features removed earlier, and continue the evolution. In other words, if a problem with a very large search space, has an extremely small number of possible solutions, we can simplify the problem, maintaining the search space, but increasing the number of solutions.

When the simpler version of the problem is solved, we then try to solve the initial problem, starting from the solutions already found. Obviously, not all problems are prone to this kind of decomposition. For example, when using a standard genetic algorithm to optimise a function, it may not be easy, or possible, to decomposed that function into simpler components. Nonetheless, we believe that evolutionary processes, especially when open-ended, are ideal candidates to apply this technique.

In our model, this technique consists of allowing the agents to evolve in steps, by periodically changing the environment and/or the agents' structure (perceptions, actions, features). For example, if an agent is in an environment where the possible survival behaviours are very few, the evolutionary search process will effectively be almost like a random search. We thus, change the environment in such a way that widens the space of good behaviours, for example by removing constraints. We then let the agents evolve and later reintroduce the removed constraints.

3.1.2.2. Genotype Editing

An additional technique we implemented in the evolutionary process is based on Luís M. Rocha's Agent-Based Model of Genotype Editing (Huang et al. 2007; Rocha and Kaur 2007), that tries to mimic the processes of RNA editing in an artificial evolution scenario, namely in Artificial Life.

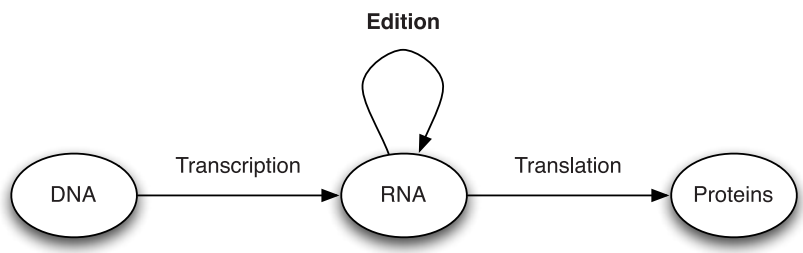

Fig. 3.2 *The process of RNA Editing in the context of gene transcription and translation.*

RNA editing (Benne 1993; Bass 2001) is a process of modification of the genetic information, after the transcription from DNA to RNA, and

before the translation to proteins. There have been identified several forms of RNA Editing in higher eukaryotes, such as the insertion and / or deletion of bases in the messenger RNA (mRNA), or the substitution of bases in the mRNA. One specific, vastly studied, example of one of these editing processes is the insertion and deletion of Uridine (u) in the mitochondrial mRNA in trypanosomes (Benne et al. 1986; Benne 1993; Stuart 1993). In this case, the editing process requires a special class of RNA molecules called guide RNA (gRNA), encoded mostly by what was previously thought of as non-functional or non-coding genetic material. This gRNA base-pairs with the unedited mRNA and inserts or, less commonly, deletes uridine into the mRNA.

Based on this these processes of RNA editing, Luis M. Rocha proposed an extension to the traditional GA, by stochastically editing the genotypes before the translation into solutions. This model was tested first with the *Genetic Algorithm with Editing* (GAE) (Huang and Rocha 2003; Rocha and Huang 2004; Huang and Rocha 2004), and later by an extended version called the *Agent-based Model of Genotype Editing* (ABMGE) (Huang et al. 2007; Rocha and Kaur 2007). In both instances, the models where tested against a series of standard fitness functions and some dynamic variations.

In our model, the goal is to apply the concepts in the ABMGE into an open-ended evolution environment. With our method of implementing evolution, the inclusion of genotype editing is fairly straightforward. All that is required is the implementation of the editing functions and an equivalent to the base-pairing mechanism. We will explain our implementation later in this chapter.

3.2. The BitBang Framework

To test our conceptual model, we developed a software framework — the BitBang Framework. In this section we'll detail the design and implementation of this software, and provide some examples on how to use it.

3.2.1. Global Architecture

The BitBang Framework is composed of two separate software components: the BitBang Core, and the BitBang Simulation Engine. The BitBang Core implements, in essence, the abstract conceptual model, and the BitBang Simulation Engine adds a three-dimensional graphics engine (Irrlicht Engine) and a physics engine (Bullet Physics). In Fig. 3.3 we show the essential connections between the various software components of the framework.

This two-part architecture was chosen mainly to facilitate the interchange of the environmental components, like the three-dimensional or physics engines. It is fairly straightforward to plug in a different engine, or even use only the BitBang Core and change the design possibilities of the environment. For example, it is possible to create a simulation on a grid world, or on a two-dimensional world. Moreover, this architecture also aids in our goal of applying our model to the entertainment industry. The BitBang Core, as an independent component, can be used as the artificial intelligence module of a game engine, or special effects framework.

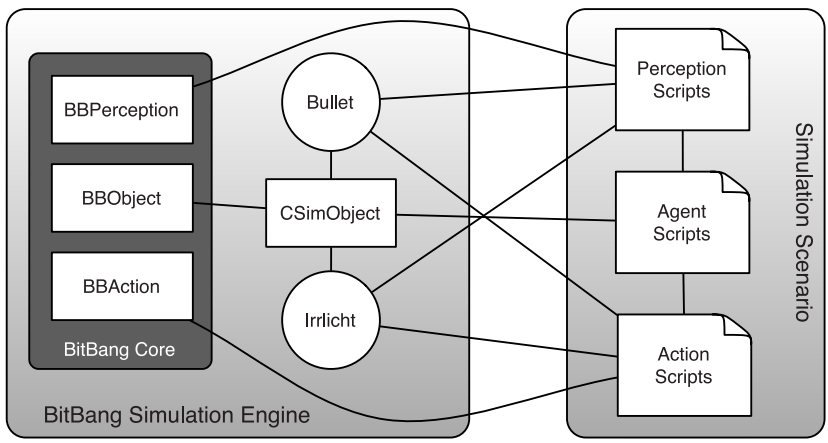

Fig. 3.3 *Global architecture diagram of the BitBang Framework.*

3.2.2. BitBang Core

As mentioned above, the BitBang Core essentially implements the components of the conceptual model. In fact, if one looks at Fig. 3.4, and compares it to Fig. 3.1, one can easily find the similarities. In this class diagram we only show the fundamental classes of the framework. The class *BBObject* represents both the Agent and Thing components. The class *BBWorld* is used to simulate the environment. The rest of the classes represented map directly to their conceptual model counterparts.

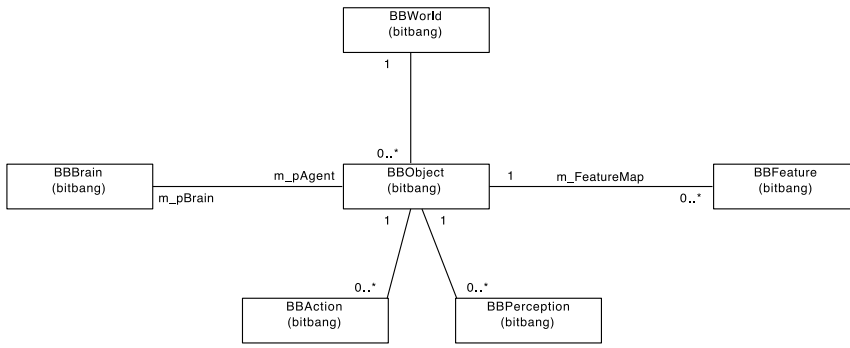

Fig. 3.4 *Class diagram of the fundamental classes of the BitBang Core.*

The BitBang Core library has, however, other secondary classes, mostly used to provide some supplementary services to the programmer. These classes are shown in the class diagram presented in Fig. 3.5. We provide through these classes a logging facility to record the simulation results and configuration, a timer to keep the simulation time and its correspondence to real time, and a scheduler to periodically run tasks. The *BBPerceptionSphere* class is a utility class that is used to enable the computation of special kinds of perceptions. One example, is the vision system implemented through the *BBVisionReach3DCone* class.

Most of the fundamental classes in BitBang Core are abstract, i.e. they are intended to be derived to implement specific cases of perceptions, actions, features, brains, objects, and worlds. For some of these, the frame-

work already provides some ready to use implementations. We provide a description of these in the next sections.

3.2.2.1. Perceptions

The *BBPerception* class only defines the base methods and variables that all perceptions should implement. To create a specific type of perception, one must specify the type of data that the perception holds, and the operators available to evaluate that data.

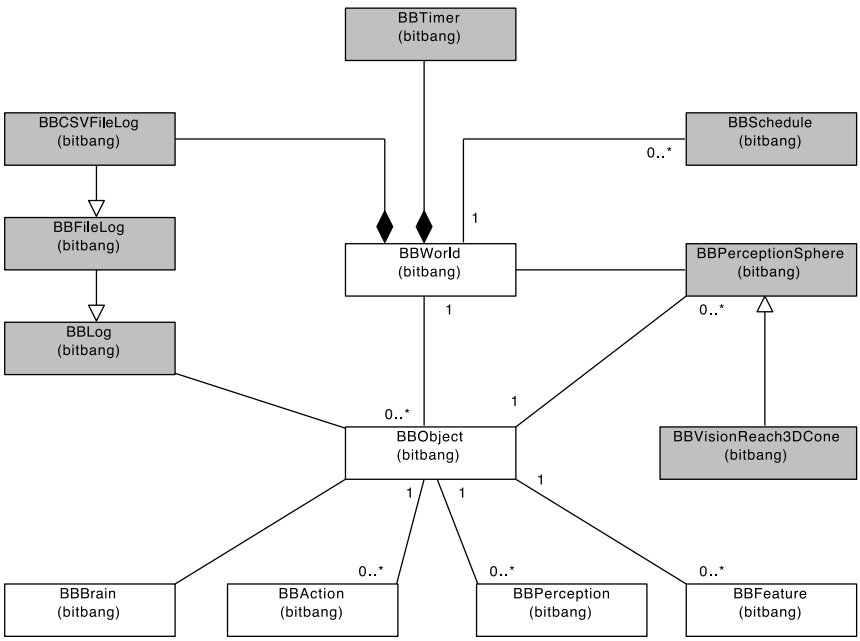

Fig. 3.5 *Expanded class diagram of the BitBang Core.*

In the framework we provide implementations of perceptions of the following types: boolean, number, and string. For the number and string perception types, we also provide a class to represent a fixed value perception (*BBPerceptionFixedNumber* and *BBPerceptionFixedString*). For each of the perception types we also implement a class that gets its perception value from an object's feature of the same type (e.g. *BBPerceptionFeatureBoolean*).

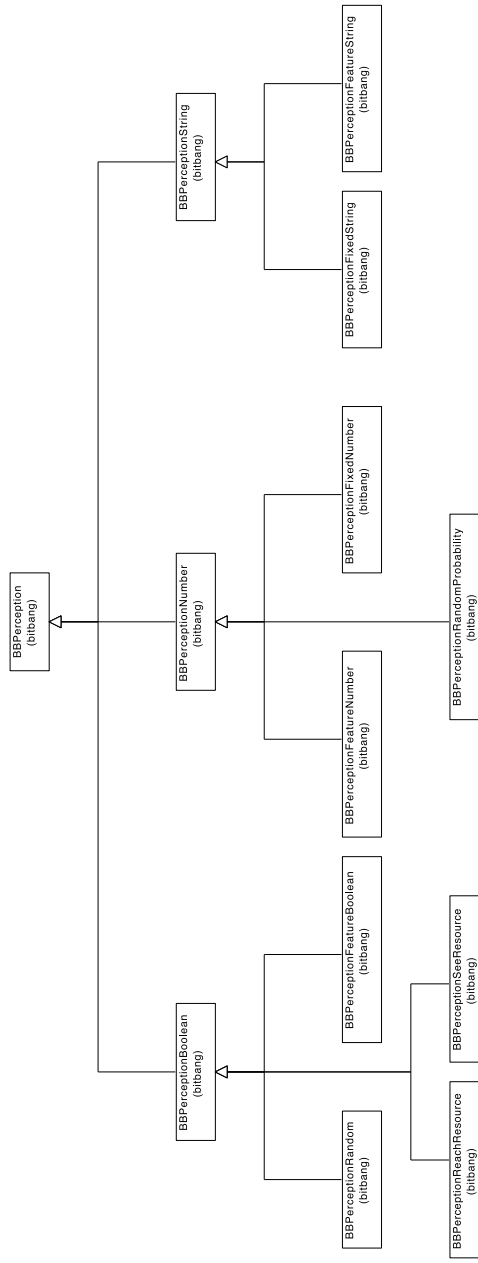

Fig. 3.6 *Class diagram of the perception classes.*

Each of these implemented perception types has a set of operators available. The boolean perceptions have two possible unary operators: *true* and *false*. Numeric perceptions have three possible operators: *equal*, *less than*, and *greater than*. String perceptions have two implemented operators: *equal* and *starts with*. Naturally, for any of these types of perceptions, new operators can be implemented.

In addition to these base type perception classes, the framework also provides some final perception classes, either as examples or as utility perceptions. An example of a utility perception is the *BBPerceptionRandom* class. This perception changes its data, on each update, to a random boolean value. The *BBPerceptionSeeResource* and the *BBPerceptionReachResource* perceptions are provided as examples on how to implement a perception class. The first evaluates to true when there is an object of type "resource" in the vision range of the agent, and the second evaluates to true if the object is within the agent's reach. All these perception classes are shown in the class diagram presented in Fig. 3.6.

3.2.2.2. Actions

Having also a base class from which all actions must derive *(BBAction)*, we do not however provide implementations of specific actions. This is due to the fact that actions are too problem and environment specific. Nonetheless, there are some important characteristics to take into account when deriving the base class.

There are two main types of actions: continuous and atomic. The atomic actions are executed in a single operation, when triggered by the brain. Continuous actions, on the other hand, are executed continuously, from the time they are triggered, and until a specific stop condition occurs. We provide three different options for these stop conditions. Actions can be timed, being automatically stopped when the defined timeout is reached. Another option is to make the action be rule-bound. This type of action keeps active as long as it is triggered by the brain, i.e. the action is stopped when, after a series of brain algorithm evaluations with this same triggered action, an evaluation triggers a different one. Finally, if the stop condition

does not fit into any of these options, it is possible to make the action "flaggable". This option allows the stop condition to be triggered from the outside, e.g. from the environment.

3.2.2.3. Features

Similarly to the perceptions, the features also derive all from the base *BBFeature* class. Again, to implement a specific type of feature, one must specify the type of data that the feature holds.

In the framework we provide some implementations of feature types. Once again, the types implemented are: boolean, number, and string. These feature type classes are displayed in the class diagram of Fig. 3.7.

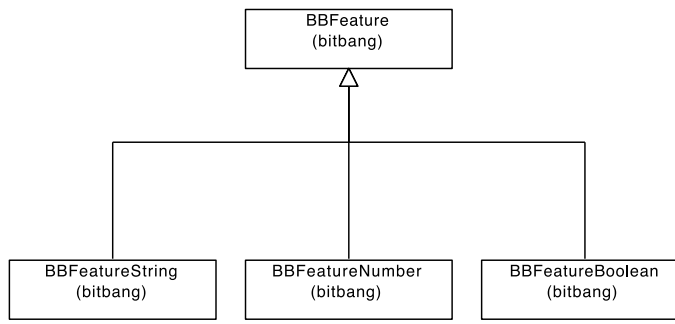

Fig. 3.7 *Class Diagram of the feature classes.*

3.2.2.4. Rule List Brain

The base class for the brain component *(BBRain)* is decision algorithm agnostic. However, to test our model, we needed to implement a specific algorithm to use in our simulations — the Rule List brain. We chose to use a rule list mainly to allow for an easy understanding of the evolved behaviours by directly inspecting the brains of the agents.

The Rule List brain is composed of an ordered list of rules. The reasoning process is straightforward. The rules are evaluated in order, and the first one whose conditions are all true, is selected. Each rule is composed of a

conjunction of conditions and an action. The structure of a rule is shown in Listing 3.1.

```
1.  <rule> ::= IF <condition-list> THEN <action>
2.  <condition-list> ::= <condition>
3.  <condition-list> ::= <condition> AND <condition-list>
4.  <condition> ::= <perception> <operator> <perception>
5.  <condition> ::= <perception> <operator>
```

Listing 3.1 *Syntax of the Rule List brain.*

Conditions have one or two perceptions and an operator. The possible operators are defined by the type of perceptions used. Several base perception types are already implemented, but new ones can be implemented if needed. The usage of one or two perceptions in a condition depends on the operator used. If the operator is unary, only one perception is needed. If the operator is binary, two perceptions must be provided. We only defined unary and binary perception operators, but the system can be easily augmented to accommodate higher arity operators.

To better illustrate, in Listing 3.2 we provide an example of a short rule list.

```
1.  IF reach_resource TRUE THEN eat
2.  IF resource_location = 2 THEN move_front
3.  IF resource_location = 1 THEN turn_right
4.  IF resource_location = 3 THEN turn_left
```

Listing 3.2 *Example of a Rule List brain.*

In this example we are using two different perceptions and four different actions. The reach_resource perception is of type boolean, and evaluates to true whenever a food source is within the agent's grasp. The resource_location perception is of type number, and provides an indication of where, within the vision range of the agent, a food source is. It evaluates to 1 if the resource is on the agent's left, 2 if the resource is in front of the agent,

3 if the resource is on the agent's right, and 0 if no resource is found on the vision range. The actions used in this example have self-explanatory names. This rule list codes a simple foraging behaviour. The agent will search for and move towards food items and eat whenever it finds one within its grasp. Remember that the rules are evaluated in sequence, and that the one that first evaluates to true will be chosen.

To be able to evolve this brain architecture we need to define its equivalent to the genotype, and the operators that modify it on reproduction. The brain's genotype is the rule list itself, no translation is applied, making it a direct representation. Maintaining our focus on simplicity, the variation operators defined are only mutation operators. These operators are listed in Table 3.1.

Table 3.1 *Mutation operators implemented for the Rule List Brain.*

Operator	***Description***
Add Rule	This operator adds a rule to the rule list.
Delete Rule	This operator deletes a rule from the rule list, randomly chosen.
Mutate List	This operator iterates through the rule list, and replaces a rule with a new random one.
Mutate Rules	This is the lowest level operator. It drills down to the perceptions on the conditions and mutates both the perceptions and their operators. It also mutates the action of the rules.
Mutate Order	This operator iterates through the rule list and moves a rule to the top of the list.
Mutate Order 2	This operator iterates through the rule list and moves a rule one position towards the top.

Having the structure of our rule list in mind, the rationale behind the creation of the mutation operators was providing a large coverage of the search space. As our rule list is ordered, we needed mutation operators that would affect the order. Two different operators were created. One with a more disruptive effect (Mutate Order), and another with less disruptive effect (Mutate Order 2). To provide variation of the inner structure of

rules, we created two operators. Again, one that completely replaces one rule (Mutate List), having a larger disruptive effect, and one with a higher granularity, affecting the components of the rules (Mutate Rules). Finally, to provide the capability of changing the size of the rule list, operators that add or delete rules were created. With these three groups of operators, we believe that the search space is adequately covered.

To test the genotype editing technique described earlier in the chapter, we also implemented a variation of the Rule List brain. In line with the definitions of the ABMGE model, we divided the genotype defined for the Rule List into a *codotype* part and an *editype* part. The former is the actual rule list, as was the case for the standard Rule List brain. The latter is composed of a series of editors of the form (E_j, F_j, v_j), again, in line with the ABMGE. The E_j editor string is defined here as a rule list condition. This rule condition is compared to the codotype's rules conditions to search for a match. Each editor E_j has an associated editing function F_j that specifies how that particular editor modifies the codotype. In our implementation we defined two editing functions. One substitutes the first perception of the condition with a new random one, and the other substitutes the operator of the condition with a new random one. More editing functions can be easily implemented for a specific simulation. Regarding the v_j parameter, we maintain the same definition as in ABMGE. It determines the probability of the E_j editor string being checked against the codotype.

The variation operators (mutation) applied to the codotype in this version of the Rule List brain are the same as those defined for the standard version (shown in Table 3.1). Furthermore, the editype itself is also subject to variation. As previously, we follow the definition of the ABMGE and only apply mutation to the editor string, and not to the editing function or the concentration parts of the editor. The mutation operator applied is the same as the one defined for the *Mutate Rules* operator, but affecting only the condition. Once more, this variation operation can be easily modified and extended to include the other editype parts, or with new variation operators.

In the class diagram shown in Fig. 3.8, we show the implementation classes of the Rule List brain, its genotype editing version, and its connections to the perception and action classes.

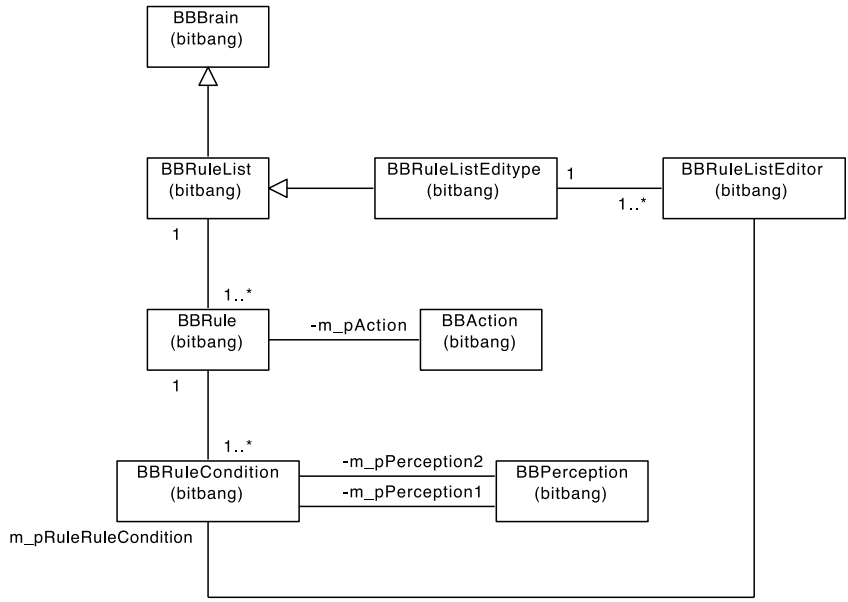

Fig. 3.8 *Class diagram of the Rule List brain classes.*

3.2.3. BitBang Simulation Engine

As mentioned previously, the main purpose of the BitBang SE (Simulation Engine) is the joining the BitBang Core with a three-dimensional graphics engine and a physics engine, and thus, provide a experimental simulation environment in which to test our model. In Fig. 3.9 we provide a class diagram of the main classes of the BitBang SE and its connections to the other engines. In this diagram we can see that there are two versions of the object class. This implementation allows the use of both 3D and physics (using the *CPhyscisObject* class), or to use only the 3D engine (using the *CSimObject* class).

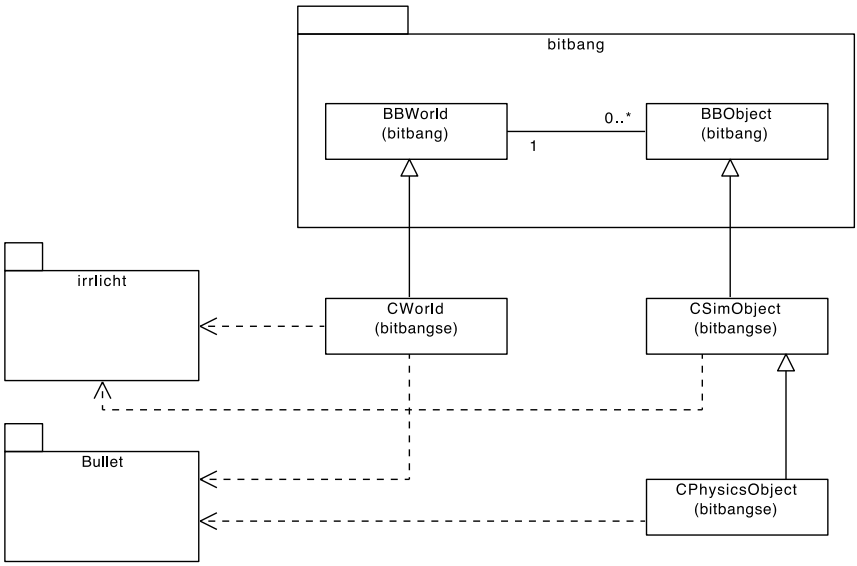

Fig. 3.9 *Class diagram of the BitBang Simulation Engine, and its connections with the BitBang Core classes, and the third-party 3D and physics engines.*

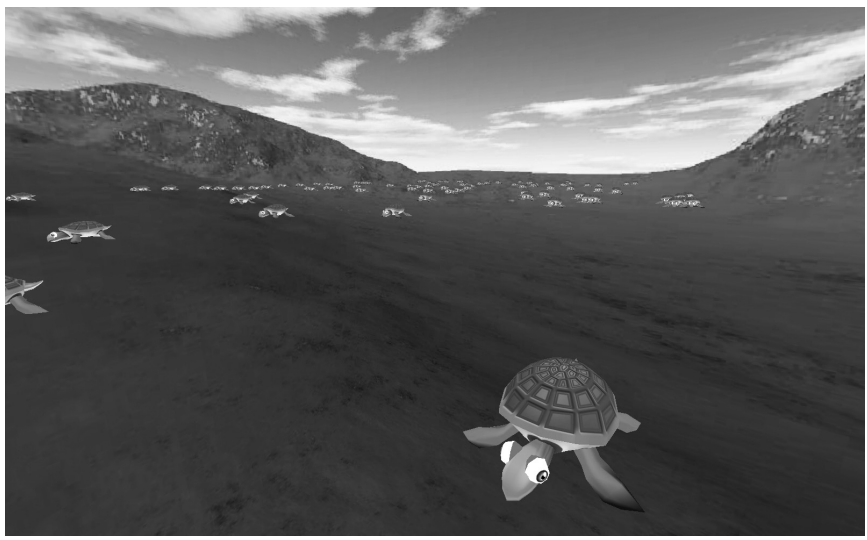

Fig. 3.10 *Screenshot of a running simulation on the BitBang Simulation Engine.*

In addition to the classes represented in the class diagram, there are others supplied in the engine. Those are mainly utility classes to provide some auxiliary services for visualisation, input event handling, and simulation configuration.

On a more technical note, both the BitBang Core and the BitBang SE are implemented in C++. This choice permitted a better optimization of the code to take advantage of all the power that modern processors can supply. When simulating open-ended evolution scenarios, one needs all the processing power available. The framework is cross-platform and is tested on Microsoft Windows, Mac OS X, and Linux. In Fig. 3.10 we show a screenshot of a running simulation using the three-dimensional graphics engine.

Chapter Four

Foraging Simulations

In this chapter we present the results from two different simulation scenarios, developed to test the model and framework implemented. Both scenarios are based on the foraging task. In Nature, the foraging task, is of vital importance to all organisms, big and small. We can find a great variety of different behaviours associated with this task. For example, social insects like bees and ants, have evolved complex collective foraging behaviours to cope with this task.

The first scenario's main purpose is the testing of the various simulation parameters, mainly those governing the evolution of the agents controllers — the mutation operators. Next, on the second scenario, we try to up the ante by making the environment more challenging to survive in, and thus forcing the evolution of more complex behaviours. That is also, ultimately, the motivation to develop the scenarios presented in the next chapter. We expect to find the emergence of increasingly complex behaviours, by creating increasingly complex environments.

For each of the scenarios presented, we will first describe the virtual world implemented, following the BitBang Framework architecture, and then we present the results of the simulations executed.

4.1. Basic Foraging

The experiments described in this section implement a world where agents and resources are randomly placed in the environment, and the agents will need to develop good navigation capabilities in order to find and eat the resources. This task may seem too simple, but consider that the agents are initialized with random brains, and that we do not have a fitness function to guide their evolution. Moreover, as these simulations serve the main purpose of benchmarking the effects of the configuration parameters on the evolutionary process, we didn't want to make them overly complicated.

4.1.1. World Definition

In this section we will set out all the implementation details and architecture of the virtual world created. As expected, these simulations were implemented using the BitBang framework, and therefore we will present

the architecture according to the framework's specifications. We begin by describing the simulation environment, then detail the agents' architecture (features, perceptions, actions, and brain), and then we present the things defined in this world.

4.1.1.1. Environment

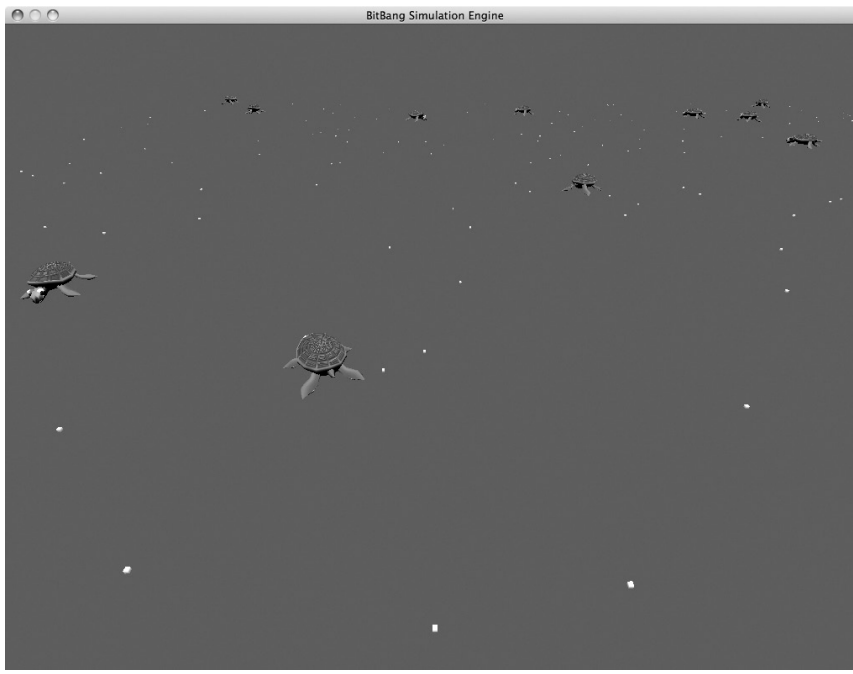

Fig. 4.1 *Screenshot of a running simulation. We can see the agents (turtles) and the food items (small boxes).*

Our world is a two dimensional world[1] where agents and resources are placed (see Fig. 4.1). The terrain is a square. This area restricts the placement of agents and resources, but does not restrict the movement of the

1 The world is two dimensional, but the visualisation is three dimensional, as the underlying BitBang Framework is three dimensional.

agents. The world is infinite, i.e., an agent can move past the boundaries of the populated terrain.

At startup, the field is populated with a configured amount of randomly placed food items. These are periodically replenished so that the total food count is maintained. As soon as the agents gain good foraging capabilities, they will deplete the world of resources, and ultimately the population would die out. As our model is open-ended, and evolution is continuos, we need to keep introducing energy into the environment to maintain the evolutionary process.

The number of resources available is configurable to be able to fine tune the system so as to allow agents to survive but also provide enough evolutionary pressure. If there are too few resources in the environment, the world could be impossible to live in. But if there are too many resources, the foraging task could become too easy and thus not provide any pressure to evolve better foraging behaviours. New resources are introduced using a schedule that periodically checks the number of resources in the environment and, if its below the configures threshold, creates new randomly placed ones.

On initialization, the world is populated with randomly placed, and randomly generated agents. At this time, having completely random brains, it is highly probable that the agents will not execute the reproduction action, either by not choosing it, or because they do not have enough energy to reproduce. Thus, if nothing is done at this stage, the population will most likely die out.

We identified two different paths to solve this problem. Either generate the initial agents with basic foraging and reproduction capabilities — similar to what is used in New Ties — or force the reproduction of agents during the initial stages of the simulation — similar to what is used in Polyworld. The first option would, to some extent, defeat our purpose of keeping the simulation open-ended, in the sense that we would need to initially define good behaviour and intrinsically define a goal. So we chose the second option.

To keep the population alive, whenever the number of agents in the world falls bellow a given threshold, new agents are created. If there are live agents in the environment, one will be picked for reproduction, otherwise a new random agent is created. Note that, as there is no explicit fitness function, the agent chosen for reproduction will be randomly selected from the population. Considering that an agent with a behaviour better adapted to the environment will live longer, it will have a greater probability of being chosen for forced reproduction.

This process is only active whenever the population falls below a configured threshold. This happens at the start of the simulation. As soon as the agents evolve behaviours capable of sustaining the population, no new agents will need to be created this way.

4.1.1.2. Agents

In this simulation only one type (species) of agent exists, and has the following architecture:

- *Features:* energy, metabolic rate, and birth date.
- *Perceptions:* energy, resource location, reach resource, light level.
- *Actions:* move front, turn left, turn right, sleep, eat, reproduce.
- *Brain:* rule list.

We now describe each one of these components.

Features

- *Energy:* This feature represents the current energy level of the agent. When this feature reaches zero, the agent dies. The feature is initialized with a predetermined value at agent birth.
- *Metabolic Rate:* The metabolic rate is the amount of energy the agent consumes per time unit. The agents energy decreases linearly, using the formula presented in equation (4.2), where $e(t)$ is the energy of the agent at time t, and m_r is the metabolic rate of the agent. The metabolic rate can vary and is composed of a base metabolic rate that increases by a configurable amount whenever the agent moves.

$$e(t + \Delta t) = e(t) - m_r \Delta t \tag{4.2}$$

- *Birth Date:* This feature is set to the current time at birth and remains constant. It is used to calculate the agent's age. When the agent reaches a given age, it dies. This procedure allows the evolution to continue past the moment when the agents have developed good navigation and eating capabilities, whilst maintaining an asynchronous and open-ended simulation. The maximum age of the agents is configurable.

Perceptions

- *Energy:* This is a self-referencing perception on the agent's current energy level. This perception is tied to the corresponding feature. This is a numerical perception, and the range of values is configured according to the corresponding feature.
- *Resource Location:* This is the agent's perception of vision, representing the position of the nearest resource, relative to the agent's position and orientation. The agent's vision field is defined by a given range and angle. An object is within the vision field of an agent if its distance to the agent is less than or equal to the vision range, and the relative angle to the agent is within the defined vision angle (see Fig. 4.2). This is a numerical perception with possible values 0, 1, 2, and 3. The value 0 means no resource is visible. The value 1 means there is a resource to the left. The value 2 means there is a resource directly in front of the agent. The value 3 means there is a resource to the right.

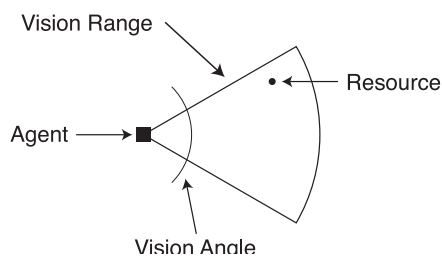

Fig. 4.2 *Diagram of the vision field of an agent.*

- *Reach Resource:* This is a boolean perception that evaluates to true whenever the agent has a resource within its reach. The distance the agent can reach is configurable.

Actions

- *Movement:* We define three actions for movement. One to walk forward, one to turn left, and one to turn right. These actions have a tie to the metabolic rate feature in such a way that whenever the agent is moving, the metabolic rate increases by a configured fix amount.
- *Eat:* This action enables the agent to eat a resource within its range. If no resource is in range when the action is executed, nothing happens. This action will add a configured amount of energy to the agent's energy feature.
- *Reproduce:* This action allows the agent to reproduce itself. The reproduction implemented is asexual. When the action is executed, a new agent is created and placed in the world. The new agent will be given a brain that is a mutated version of its parent's brain. Note that, as each mutation operator has been given a probability of being applied, the child's brain can be a perfect clone of its parent's brain. The action will also transfer energy from the parent to the offspring. The amount of energy consumed in the action is the sum of the initial energy for the new agent and a configurable fixed cost. It's important to have a cost of reproduction higher than the initial energy of an agent, so as to provide evolutionary pressure.

Brain

The agents' brain used in these experiments is a rule list, whose architecture was detailed in section 3.2.2. On initial creation of an agent, the brain is randomly initialized. This initialization conforms to some configurable parameters: the maximum number of rules, the minimum number of rules, and the maximum number of conditions per rule. Other configured values are the mutation probabilities used in the reproduction action.

4.1.1.3. Things

Only one type of thing is defined for this world: the resources that the agents eat to acquire energy. No features are associated with them. A configurable parameter defines the amount of energy each resource provides.

4.1.2. Results

In this section we present and analyse the results of the simulations performed with this foraging scenario. But first we present the various parameters and their configuration values used in these experiments. In Table 4.1 we present the base values used for this scenario. We later vary some of these parameters to study their influence in the evolutionary process. For each different set of configuration parameters tested, we ran thirty independent simulations, and collected the results.

Table 4.1 *Basic Foraging: Configuration values*

Parameter	***Values***
Terrain Size	1000 x 1000
Time Limit	100 000
Minimum Food	200
Food Energy Content	3
Minimum Agents	20
Vision Range	200
Vision Angle	60º
Agent Reach	20
Agent Initial Energy	10
Metabolic Rate	0.1
Move Metabolic Rate Increase	0.01
Maximum Age	500
Reproduction Cost	2
Minimum Rules	15
Maximum Rules	20
Maximum Conditions	2

Parameter	***Values***
Mutation List Probability	0.01
Mutation Rules Probability	0.01
Mutation Order Probability	0.01
Mutation Order 2 Probability	0.01
Mutation Add Probability	0
Mutation Delete Probability	0

A statistical analysis of these results is presented later. Now, we show some plots of the evolution of the population of agents throughout one simulation run. The plots shown are from an experiment with the configuration values presented in Table 4.1, and are typical of all simulation runs of that experiment.

In the plots shown in Fig. 4.3 we can clearly spot the time when agents start having a good foraging behaviour (at around 2000 time units), and then have a significant improvement around time 19 000. We can also spot the time where agents start reproducing by themselves (at around time 27 000), by looking at the evolution of the number of agents in the population. At this time we can see that the average gathered energy per agent drops. This happens due to the increase in population size, and the number of resources being kept constant.

Since we do not use fitness functions, we are not able to assign a fitness value to agents, and thus cannot compare runs based on fitness. We have, however, a number of different values that we could use to benchmark the experiments. From the analysis of the evolutionary plots presented above, we can see that there are two important events in these simulations: the evolution of foraging, and the evolution of self sustained reproduction. We will use the time when these events occur in the run as a performance measure. If in a given run, the agents never evolve either foraging or reproduction, the value used will be the time limit of the run. Naturally, using these measures of performance, the lower the value, the better the performance.

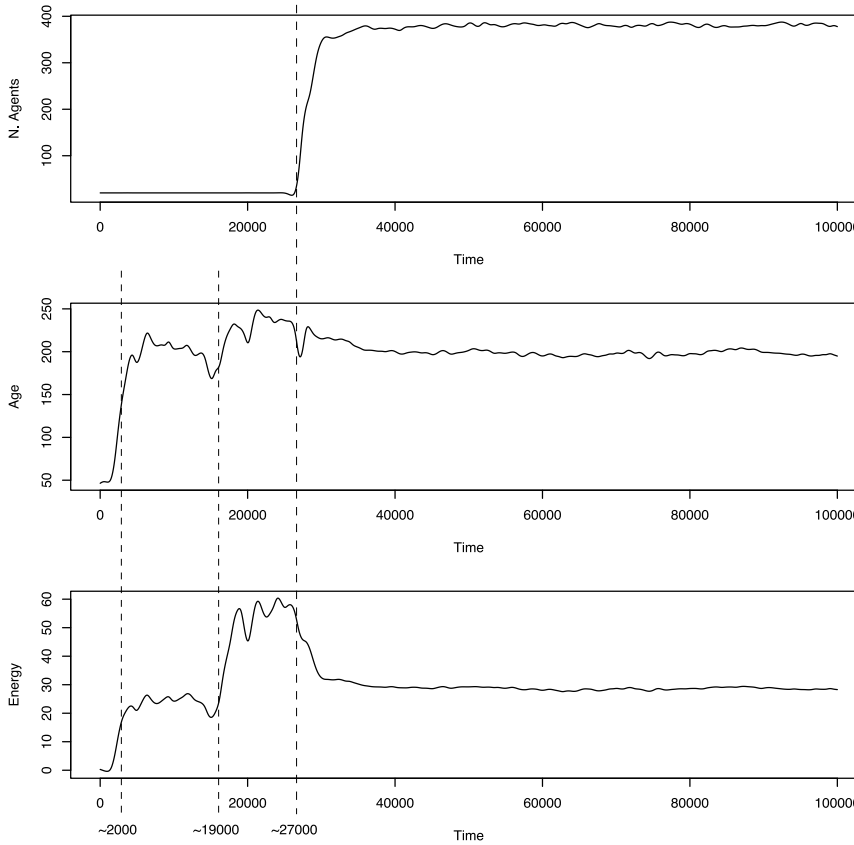

Fig. 4.3 *Plot of the evolution of the number of agents, their average age, and average gathered energy over the course of one simulation run.*

We divide the benchmarking experiments into three groups. First we test the effect of the time limit, then the environmental conditions, and finally we test the mutation operators. For each set of experiments, the parameter values not explicitly presented in the text, are set to the values show above in Table 4.1.

To analyse the results from the various parameter configuration variations, we determined the statistical significance of the null hypothesis of no difference with Mann-Whitney U test with α=0.05, whenever we have two experiment sets to test. In the cases where we have more than two sets of

experiments, we use Kruskal-Wallis ANOVAs with α=0.05. If a significant difference in a these sets of experiments is found, further pairwise Mann-Whitney U tests with Holm's p-value adjustment were conducted. These non-parametric tests were chosen because the data is not guaranteed to follow a normal distribution.

4.1.2.1. Time Limit

In these first experiments we tested two different values for the time limit of the simulation. On experiment one, we set the time limit to 100 000 time units, and on experiment two, we set it to 200 000. The results from these experiments are shown in Fig. 4.4. The plots show that both experiments present identical results. In fact, we didn't find any significant difference in the results with a p-value of 0.9646 for the time of evolution of foraging, and a p-value of 1 for the time of evolution of reproduction.

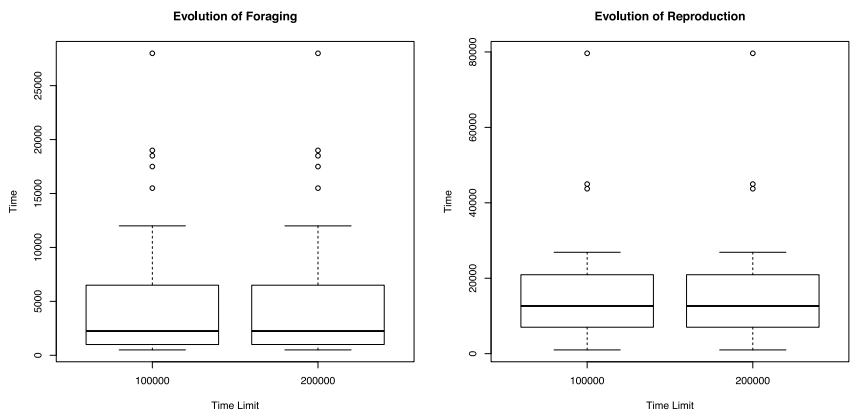

Fig. 4.4 *Results from experiments with different time limit. The results show the time when evolution of foraging or reproduction occur in the runs.*

4.1.2.2. Environment

In these experiments we tested the effect of changing the amount of food available in the environment. We run experiments with values of 200, 100,

and 50 for the minimum food parameter. The results from these experiments are shown in Fig. 4.5, Table 4.2 and Table 4.3.

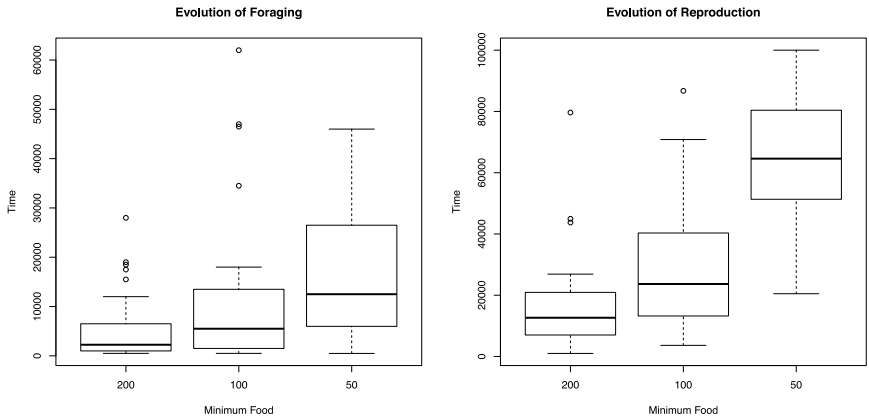

Fig. 4.5 *Results from experiments with different number of food items in the environment. The results show the time when evolution of foraging or reproduction occur in the runs.*

Table 4.2 *Pairwise comparisons: time of evolution of foraging*

	200	***100***
100	0.10372	—
50	0.00076	0.09027

Table 4.3 *Pairwise comparisons: time of evolution of reproduction*

	200	***100***
100	0.00838	—
50	2.93×10^{-9}	1.04×10^{-6}

We find that the best median values for both foraging and reproduction are obtained for 200 food items. The results were found to be significantly different with p-values of 0.00116 for foraging and 9.65×10^{-11} for reproduction. Pairwise tests reveal that performance significantly worse for 50 food items in both foraging and reproduction. We can also see that, in

terms of reproduction, the value of 200 food items is significantly better than both 100 and 50, and that 100 is significantly better than 50.

4.1.2.3. Mutation Operators

The experiments presented in this section test the effect of the mutation operators. The first set of experiments, whose results are shown in Fig. 4.6, Table 4.5 and Table 4.6, tests different values for the probability of mutation for all operators at the same time (except for the add and delete rule operators). In Table 4.4 we show the configuration for the various experiments of this set. In this first test of the mutation probability values, we change simultaneously all the mutation operators probabilities in order to test the systems sensibility to mutation. Later, we will test each mutation operator independently.

Table 4.4 *Mutation Operators: Experiments configuration*

Experiment	***Mutate Rules***	***Mutate Order***	***Mutate Order2***	***Mutate List***
1	0.01	0.01	0.01	0.01
3	0.001	0.001	0.001	0.001
4	0.005	0.005	0.005	0.005
5	0.02	0.02	0.02	0.02
6	0.05	0.05	0.05	0.05
7	0.1	0.1	0.1	0.1
8	0.2	0.2	0.2	0.2
9	0.5	0.5	0.5	0.5

The results show that the best medians were found for experiments one, four and five in the case of foraging, and for one and five in the case of reproduction. Tests show that the results are significantly different in terms of both foraging and reproduction with p-values of 0 in both cases. Further pairwise tests show that experiments one, four, and five perform significantly better than all other experiments in terms of foraging, and experiments one and five perform significantly better than all others in terms of reproduction.

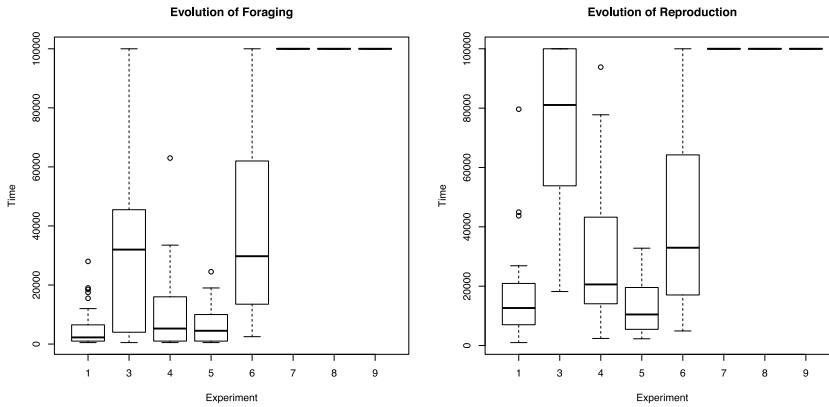

Fig. 4.6 *Results from experiments with different probabilities for the mutation operators, except for the Add Rule and Delete Rule operators. The results show the time when evolution of foraging or reproduction occur in the runs.*

Table 4.5 *Pairwise comparisons: time of evolution of foraging*

	1	*3*	*4*	*5*	*6*	*7*	*8*
3	0.00110	—	—	—	—	—	—
4	0.35482	0.02410	—	—	—	—	—
5	0.71000	0.00170	0.71000	—	—	—	—
6	1.2×10^{-6}	0.71000	0.00023	3.4×10^{-6}	—	—	—
7	3.0×10^{-11}	9.2×10^{-10}	3.0×10^{-11}	3.0×10^{-11}	8.1×10^{-9}	—	—
8	3.0×10^{-11}	9.2×10^{-10}	3.0×10^{-11}	3.0×10^{-11}	8.1×10^{-9}	—	—
9	3.0×10^{-11}	9.2×10^{-10}	3.0×10^{-11}	3.0×10^{-11}	8.1×10^{-9}	—	—

Table 4.6 *Pairwise comparisons: time of evolution of reproduction*

	1	*3*	*4*	*5*	*6*	*7*	*8*
3	9.1×10^{-9}	—	—	—	—	—	—
4	0.01903	9.3×10^{-7}	—	—	—	—	—
5	0.75235	1.9×10^{-9}	0.00310	—	—	—	—
6	0.00031	0.00181	0.13119	0.00008	—	—	—
7	3.0×10^{-11}	0.00002	3.0×10^{-11}	3.0×10^{-11}	9.1×10^{-9}	—	—
8	3.0×10^{-11}	0.00002	3.0×10^{-11}	3.0×10^{-11}	9.1×10^{-9}	—	—
9	3.0×10^{-11}	0.00002	3.0×10^{-11}	3.0×10^{-11}	9.1×10^{-9}	—	—

We can also see that experiments seven, eight, and nine perform significantly worse than all others. In fact, in these experiments agents never develop either foraging or reproductive behaviours in the assigned time limit. These results show that the best global values for the probability of mutation are 0.01 and 0.02.

Next, we tested the effect of only using each one of the mutation operators independently. In Table 4.7 we show the configuration values for these experiments. We present the results from these experiments in Fig. 4.7, Table 4.8, and Table 4.9.

Table 4.7 *Mutation Operators: Experiments configuration*

Experiment	***Mutate Rules***	***Mutate Order***	***Mutate Order2***	***Mutate List***
10	0.01	0	0	0
11	0	0.01	0	0
12	0	0	0.01	0
13	0	0	0	0.01

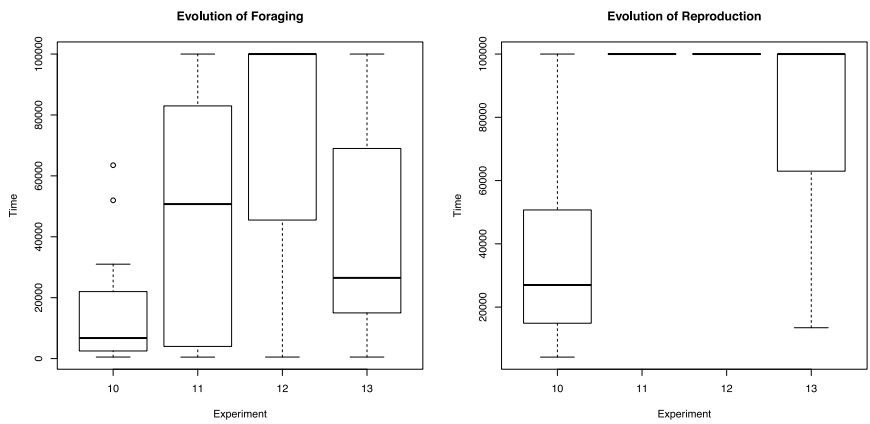

Fig. 4.7 *Results from experiments using each of the mutation operators independently. The results show the time when evolution of foraging or reproduction occur in the runs.*

The best median both in terms of foraging and reproduction was found for experiment ten, where we only use the Mutate Rules operator. Sets were

found to be significantly different both in term of foraging and reproduction, with p-values of 8.58 x 10^{-6} and 1.68 x 10^{-16} respectively. Pairwise tests show that the Mutate Rules operator performs significantly better than all other operators.

Table 4.8 *Pairwise comparisons: time of evolution of foraging*

	10	***11***	***12***
11	0.00994	—	—
12	0.00011	0.01697	—
13	0.00501	0.47279	0.01257

Table 4.9 *Pairwise comparisons: time of evolution of reproduction*

	10	***11***	***12***
11	2.9E-10	—	—
12	2.9E-10	—	—
13	0.00002	0.00029	0.00029

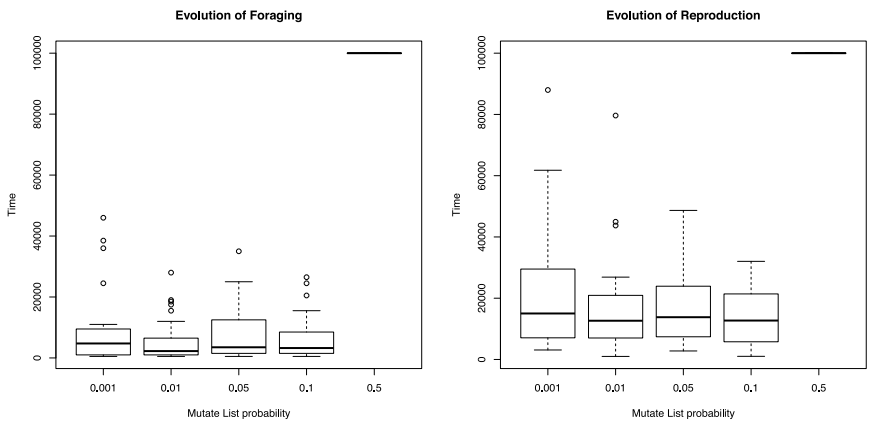

Fig. 4.8 *Results from experiments with different probabilities for the operator Mutate List. The results show the time when evolution of foraging or reproduction occur in the runs.*

In the experiments whose results are shown in Fig. 4.8, Table 4.10 and Table 4.11, we test different values for the probability of applying the Mutate List operator, whilst keeping all other mutation operators with a probability of 0.01. We tested the values 0.001, 0.01, 0.05, 0.1, and 0.5.

Tests show that there are significant differences in this set of experiments, with p-values of 4.52×10^{-15} for foraging and 4.17×10^{-15} for reproduction. Pairwise tests reveal that probability 0.5 has significantly worse performance than all other values. There was no significant difference found between the rest of the values.

Table 4.10 *Pairwise comparisons: time of evolution of foraging*

	0.001	***0.01***	***0.05***	***0.1***
0.01	1.00000	—	—	—
0.05	1.00000	1.00000	—	—
0.1	1.00000	1.00000	1.00000	—
0.5	1.2×10^{-11}	1.2×10^{-11}	1.2×10^{-11}	1.2×10^{-11}

Table 4.11 *Pairwise comparisons: time of evolution of reproduction*

	0.001	***0.01***	***0.05***	***0.1***
0.01	1.00000	—	—	—
0.05	1.00000	1.00000	—	—
0.1	1.00000	1.00000	1.00000	—
0.5	1.2×10^{-11}	1.2×10^{-11}	1.2×10^{-11}	1.2×10^{-11}

In the experiments whose results are shown in Fig. 4.9, Table 4.12 and Table 4.13, we test different values for the probability of applying the Mutate Order operator, whilst keeping all other mutation operators with a probability of 0.01. We tested the values 0.001, 0.01, 0.05, 0.1, and 0.5.

The best median in terms of both foraging and reproduction was found for the probability of 0.01. Tests show that there are significant differences in this set of experiments both in terms of foraging and reproduction, with p-values of 8.39×10^{-16} and 8.80×10^{-15} respectively. Pairwise tests reveal

that probability 0.5 has significantly worse performance than all other values, and that 0.01 is significantly better than 0.1 in terms of foraging.

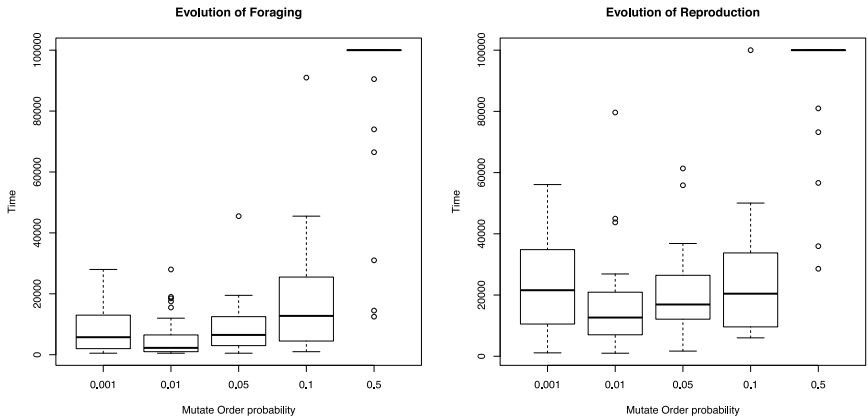

Fig. 4.9 *Results from experiments with different probabilities for the operator Mutate Order. The results show the time when evolution of foraging or reproduction occur in the runs.*

Table 4.12 *Pairwise comparisons: time of evolution of foraging*

	0.001	***0.01***	***0.05***	***0.1***
0.01	0.17199	—	—	—
0.05	1.00000	0.17199	—	—
0.1	0.10958	0.00079	0.10958	—
0.5	2.8×10^{-10}	1.9×10^{-10}	2.6×10^{-10}	1.8×10^{-9}

Table 4.13 *Pairwise comparisons: time of evolution of reproduction*

	0.001	***0.01***	***0.05***	***0.1***
0.01	0.17884	—	—	—
0.05	0.76347	0.35112	—	—
0.1	0.87755	0.08921	0.74343	—
0.5	2.3×10^{-10}	1.2×10^{-10}	1.2×10^{-10}	1.1×10^{-9}

In the experiments whose results are shown in Fig. 4.10 we test different values for the probability of applying the Mutate Order 2 operator, whilst keeping all other mutation operators with a probability of 0.01. We tested the values 0.001, 0.01, 0.05, 0.1, and 0.5. In this set of experiments we found that there are no significant differences either in terms of foraging and of reproduction, with p-values of 0.34371 and 0.37198 respectively.

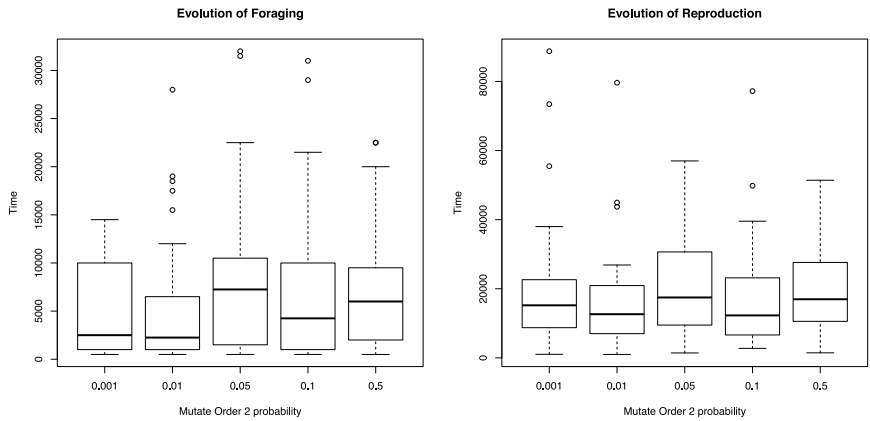

Fig. 4.10 *Results from experiments with different probabilities for the operator Mutate Order 2. The results show the time when evolution of foraging or reproduction occur in the runs.*

In the experiments whose results are shown in Fig. 4.11, Table 4.14, and Table 4.15 we test different values for the probability of applying the Mutate Rules operator, whilst keeping all other mutation operators with a probability of 0.01. We tested the values 0.001, 0.01, 0.05, 0.1, and 0.5.

The best median was found for the values 0.01 and 0.05, both in terms of foraging and reproduction. Tests show that there are significant differences in the set both in terms of foraging and reproduction, with p-values of 0 in both cases. Pairwise tests reveal that both values of 0.01 and 0.05 perform significantly better than all other values, and that 0.5 performs significantly worse than all others.

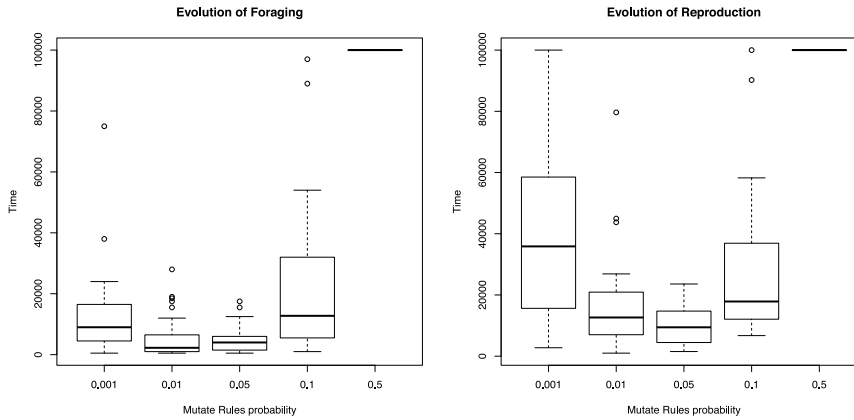

Fig. 4.11 *Results from experiments with different probabilities for the operator Mutate Rules. The results show the time when evolution of foraging or reproduction occur in the runs.*

Table 4.14 *Pairwise comparisons: time of evolution of foraging*

	0.001	***0.01***	***0.05***	***0.1***
0.01	0.01417	—	—	—
0.05	0.01272	0.46866	—	—
0.1	0.16998	0.00019	0.00014	—
0.5	1.2×10^{-11}	1.2×10^{-11}	1.2×10^{-11}	1.2×10^{-11}

Table 4.15 *Pairwise comparisons: time of evolution of reproduction*

	0.001	***0.01***	***0.05***	***0.1***
0.01	0.00178	—	—	—
0.05	0.00002	0.15695	—	—
0.1	0.15695	0.04460	0.00004	—
0.5	4.0×10^{-10}	1.2×10^{-11}	1.2×10^{-11}	3.7×10^{-11}

In the experiments whose results are shown in Fig. 4.12, we tested the effect of using either version of the Mutate Order operator. We ran experiments with either Mutate Order operator or Mutate Order 2 operator turned on, and with both operators turned on (see Table 4.16). No signifi-

cant differences were found in this set of experiments, both in term of foraging and reproduction, with p-values 0.66837 and 0.21955 respectively.

Table 4.16 *Mutate Order operators: Experiments configuration*

Experiment	***Mutate Rules***	***Mutate Order***	***Mutate Order2***	***Mutate List***
1	0.01	0.01	0.01	0.01
15	0.01	0.01	0	0.01
16	0.01	0	0.01	0.01

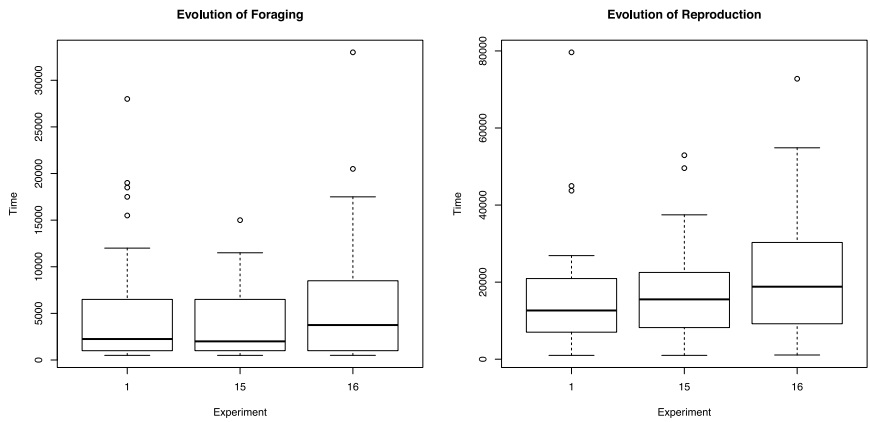

Fig. 4.12 *Results from experiments using either the Mutate Order, the Mutate Order2, or both operators. The results show the time when evolution of foraging or reproduction occur in the runs.*

In the experiments whose results are shown in Fig. 4.13, we tested the use of the Add Rule and Delete Rule mutation operators. To that end we run two experiments with these operators turned off (probability of zero) and turned on with probability of 0.01. No significant differences were found in the results with a p-value of 0.43307 in terms of foraging and a p-value of 0.09479 in terms of reproduction.

Owing to the fact that in these experiments we are using operators that affect the size of the agent's brains, by adding or removing rules, it is relevant to add performance measures related to brain size. Thus, for these experiments we also collected the performance values for the brain size and

for the number of used rules in the brain of the agents. Both these values are averages taken from the agents that lived in the last ten thousand time units of the simulation. We define used rules as the ones that have been selected at least once during the lifetime of the agent.

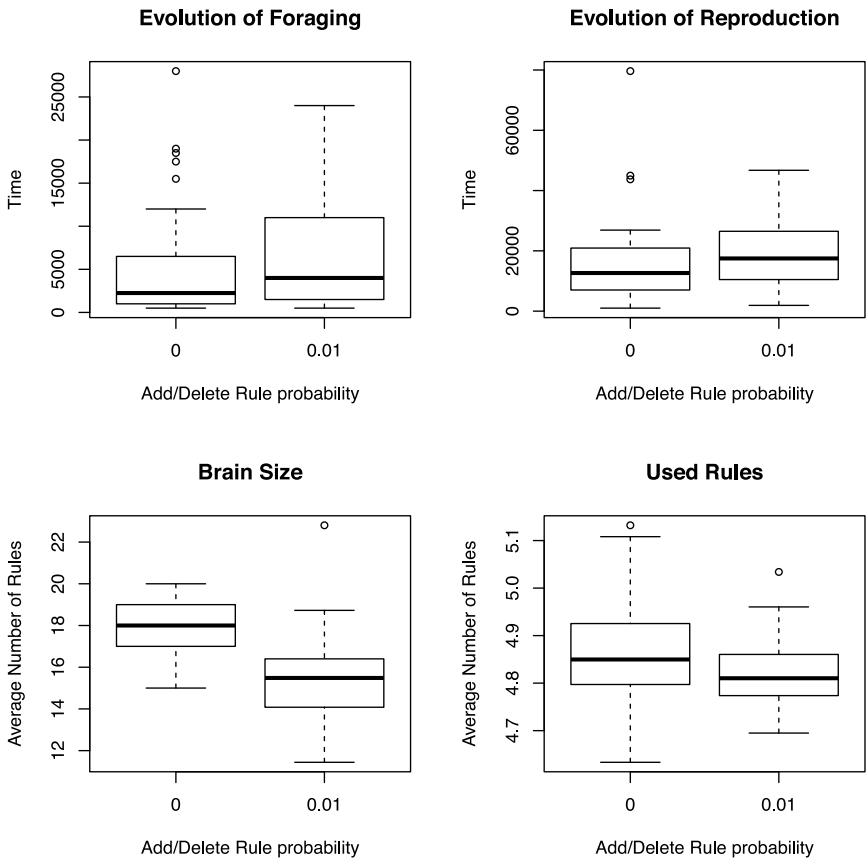

Fig. 4.13 *Results from experiments with the Add Rule and Delete Rule operators turned on or off. The results show the time when evolution of foraging or reproduction occur in the runs.*

One interesting phenomenon that we can check with these performance values, is if the brain size tends to increase in size, without a corresponding increase in the number of used rules. In genetic programming, and

other evolutionary computation techniques, an equivalent phenomenon is usually referred to as bloat. Such an increase in brain size could affect computational performance and eventually render the simulations too slow to complete in reasonable time.

The results show that, in terms of brain size, the lowest median is found for the case where the add and delete operators are used. This result is statistically significant, with a p-value of 0.00005. In terms of used rules, the sets are not significantly different, with a p-value of 0.0935.

4.1.3. Analysis of the Results

In this first experiment set, we tested various parameters of the simulation, to test how they affect the evolutionary process. We first tested the time limit parameter and found no significant difference in the results. This happens because all the thirty runs were successful in evolving good foraging behaviours and sustainable reproduction for the lower time limit. At least for the complexity of this world, 100 000 time units seem to be enough.

Next we tested the availability of resources in the environment. In this case, we found that the best performance attained for was for two hundred food items. This result was expected, as the least food we have in the environment, the harder the foraging task gets.

The rest of the experiments were used to test the various mutation operators implemented. Regarding the overall mutation probabilities test, we verified the best performance was obtained for the values 0.01 and 0.02. Next we tested each mutation operator individually. The results show that the only operator that could eventually be used alone is the *Mutate Rules* operator. This is probably due to the low level working of this operator. As it can mutate each component of the rules, it can eventually modify the rule list completely. If we had to choose only one operator to use, this would be the one to choose.

In the subsequent four sets of experiments, we tested the sensitivity to the mutation probability of each mutation operator, whilst keeping the rest of the operators with a probability of 0.01. In these experiments we were able to verify that for all operators, except for the *Mutate Order 2*, the prob-

ability of 0.5 has the worst performance. Another relevant result is that of the *Mutate Rules* operator, as this is the only operator that shows some sensitivity to the probability of being applied. In this case, the values 0.01 and 0.05 both have better performance than the rest. Again, this might be due to the high granularity of the operator. Overall, the results indicate that a value of 0.01 for all operators seems to be a good choice.

The next set of experiments tests the two operators that change the order of rules, in order to check if one might be better than the other. However, we could not find any significant differences between them, and also in applying both. This might indicate that we could drop one of these operators and maintain good performance.

On the final set of experiments we tested the usage of the add and delete rule operators. We were especially interested in checking if these operators would cause bloat. The results show that, in fact, these operators not only do not cause bloat, but reduce the size of the brain.

This foraging simulation was also used as a first test of our open-ended evolution model. From the analysis of the evolutionary plots at the beginning of this results section, we can that the agents do evolve good foraging behaviours and sustainable reproduction in this environment.

4.2. Ant Foraging

As mentioned before, the scenario described in this section is inspired by the known ant foraging behaviours fond in Nature. The study of these collective behaviours has crossed the discipline of biology and inspired a number of computational algorithms generally known as swarm intelligence (Bonabeau 1999). Specifically, in ants, both the use of a random walk and of pheromone as a tool to communicate and coordinate the foraging task, has fuelled a vast number of ant based algorithms like Dorigo's Ant Colony Optimization algorithms (2004, 2006).

A random walk is a process of defining a trajectory where at each step a random direction is chosen. The mathematical formalisation of the random walk has been the inspiration for a vast body of research in diverse areas. The term was first used by Karl Pearson (1905) on a letter to Nature jour-

nal. The random walk can be considered in n dimensions, however, of interest to our work is the two-dimensional random walk (McCrea and Whipple 1940). The most important property of the random walk, to our work, is that, given enough time, all points in space will eventually be visited. It has been observed that the foraging behaviour os some species of ants is analogous to a random walk, when in the absence of pheromone trails.

The other important aspect of the ant foraging behaviour is the use of pheromone trails to guide and recruit other ants to the task (Deneubourg 1983; Camazine et al. 2001). Pheromone is a chemical substance that ants deposit on the ground to mark the path to a food source. When an ant finds a food source, it will deposit pheromone on the way back to the nest. Other ants will then sense the pheromone and follow the trail to the food source. This description is a simplification of the possible variations of this behaviour, but encompasses the main aspects important to our research.

Although most research on these issues is based on the implementation of ant algorithms, using the known biological behaviours, some research onto the emergence of these behaviours also exists (Collins 1991; Kawamura 2000; Nakamichi and Arita 2005). In those papers, the authors use standard genetic algorithms to evolve populations of agents, controlled by artificial neural networks. In our research, however, we are concerned with the open-ended evolution of these behaviours.

In the scenario presented in this section, we look to make the foraging task harder for the agents, in order to see if more complex behaviours are evolved. To that end we changed the placement of food in the environment by gathering all the available resources in a patch far away from the agents' initial position. With this change, the agents will need to evolve a behaviour that allows them to find the patch of food in the environment, for example, a random walk.

4.2.1. World Definition

The world created for these experiments has some common components with the one described previously. Thus, in this section, we will only describe the new or changed components.

4.2.1.1. Environment

Again, we use a two dimensional world where agents and resources are placed. However, this time the terrain is a bounded square. By using a bounded square in this environment, we force the ants to stay within the populated terrain, thus allowing the better coverage of the whole environment when performing a random walk. Inside this arena we define a nest (the place where all the agents are born), and a feeding zone. This feeding zone is a circle of a given radius where all food items are placed. At startup, we fill the zone with food items, and these are periodically replenished so that the total food count is maintained. These different zones can be seen in the screenshot shown in Fig. 4.1. On initialization, the world is populated with randomly generated agents, placed at the nest. All the agents born during the simulation will also be placed at the nest.

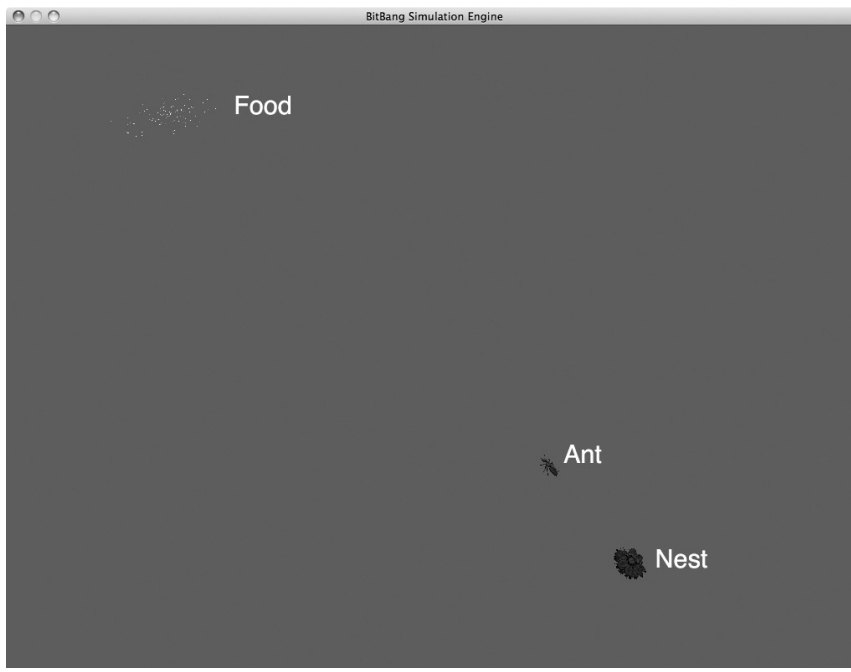

Fig. 4.14 *Screenshot of a running simulation. At the top left corner there is a feeding zone with food items and at the bottom right we see the nest and an ant agent walking away from the nest.*

To accommodate the placement and evaporation of pheromone, we created an influence map in the environment. The map divides the continuos environment into a grid. Each cell will have a value indicating the amount of pheromone it contains. Agents can deposit pheromone in the environment, and once placed, the pheromone will evaporate linearly at a constant rate. Each cell has a maximum value of pheromone it can contain. If an agent tries to place pheromone into a cell that already has the maximum amount, no more pheromone will be deposited.

4.2.1.2. Agents

In this simulation only one type (species) of agent exists, and has the following architecture:

- *Features:* energy, metabolic rate, birth date, and placing pheromone.
- *Perceptions:* energy, resource location, reach resource, pheromone location, placing pheromone, and random number.
- *Actions:* move front, turn left, turn right, sleep, eat, reproduce, and place pheromone.
- *Brain:* rule list.

We will now describe these components, but only those that are new in this world.

Features

- *Placing Pheromone:* This boolean feature indicates if the agent is currently placing pheromone on the environment. Whenever this feature is true, the agent will place pheromone at a given rate.

Perceptions

- *Pheromone Location:* This perception is similar in values to the *Resource Location* perception, but instead of identifying resources, it identifies pheromone values in the vicinity of the agent. To determine the value of this perception we will look to the nearby cells (1 cell neighbourhood) in the pheromone influence map, and find

the one with the maximum pheromone value. Cells are processed starting from the one at the front and left of the agent. In the case of a tie, the first processed cell will be selected. If none of the neighbouring cells have pheromone, the value of the perception will be 0. If the maximum is to the left of the agent (the agent's orientation is taken into account), the value of the perception will be 1. If it is to the right, the value is 3. And if it is in front of the agent, the value is 2.

- *Placing Pheromone:* This is a self-referencing perception that takes the same value as the corresponding feature. With this perception, the agent can perceive if it is currently placing pheromone.
- *Random Number:* This perception provides a source of randomness to the agent. Without some source of randomness, the agents would never be capable of having a random behaviour, and thus would not have the capacity to perform a random walk. The perception will take a new random number each time it is evaluated. The number is selected from the range 0 to 3. We use this range, as it is the same range of values that the location perceptions can take.

ACTIONS

- *Place Pheromone:* This action toggles the value of the *Placing Pheromone* feature, turning it on if it is off, and off if it is on.

4.2.1.3. Things

Only one type of thing is defined in this world, representing the food items that the agents can eat to acquire energy. These things have no associated features.

4.2.2. Results

In this section we present the main configuration values used in the experiments and then show the results from the simulation runs. Table 4.17 shows all the parameters used, and their configured values. If, for a given parameter, several values where tested, we present them separated by a semi-

colon. Most of the values used for these parameters are the result of previous experimentation.

Table 4.17 *Ant Foraging: Configuration values*

Parameter	***Values***
Terrain Size	1000 x 1000
Time Limit	100 000; 500 000; 5 000 000
Nest Location	100, 100
Food Location	800, 800
Minimum Food	100; 200
Food Energy Content	3
Minimum Agents	20
Vision Range	100
Vision Angle	100º
Agent Reach	20
Agent Initial Energy	10
Metabolic Rate	0.1
Maximum Age	500
Reproduction Cost	2
Minimum Rules	15; 20; 30
Maximum Rules	20; 30; 40
Maximum Conditions	2; 3
Mutation Probability	0.01
Pheromone Deposit Rate	50
Pheromone Evaporation Rate	10

The choice of the parameters to test was focussed on the evaluation of three main aspects. As introduced earlier, we know that the evolutionary process is very time consuming, especially when considering open-ended evolution. So, we tested three different values for the time limit of the simulations, one small, one medium, and one large. Secondly, we wanted to test if the size of the brain would influence the behaviours that emerge. So, we tested different values for the parameters that govern the creation of the

brains. And lastly, as discussed earlier, the configuration of the number of food items available is important to provide enough evolutionary pressure, but also creating a world where life is at all possible. In that regard, we also tested two different values for the number of food items in the environment. For each tested configuration, we ran thirty independent simulations. The combinations of tested configuration values are presented in Table 4.18.

Table 4.18 *Ant Foraging: Experiments configuration*

Exp. Num.	*Time Limit*	*Min. Rules*	*Max. Rules*	*Max. Cond.*	*Food*
1	100 000	15	20	2	100
2	500 000	15	20	2	100
3	500 000	20	30	3	100
4	500 000	30	40	3	100
5	5 000 000	15	20	2	100
6	500 000	15	20	2	200

As explained previously, in these experiments we are interested in finding out if the agents evolve good foraging behaviours in this environment. Namely we are looking for the random walk that characterizes the foraging behaviour of some species of ants. To that end we will first show some plots of an example typical simulation run. In Fig. 4.15 we show the data from a run of experiment two.

By inspecting the plots shown on Fig. 4.15, it's clear that the agents evolve foraging capabilities. We can see that at about 80 000 time units, both the average age and average energy of the agents rises, and in the case of the energy, it keeps improving up to time 150 000. At that time, we see that the agents start reproducing by themselves, and the population size quickly goes to about 250 agents. With the increase in population, the average energy per agent drops, because now there is greater competition for the same total amount of food.

Next, in Fig. 4.16, we show another group of plots from the same simulation run, showing the evolution of the average brain size and average

number of used rules. We define used rules as the ones that have been selected at least once during the lifetime of the agent.

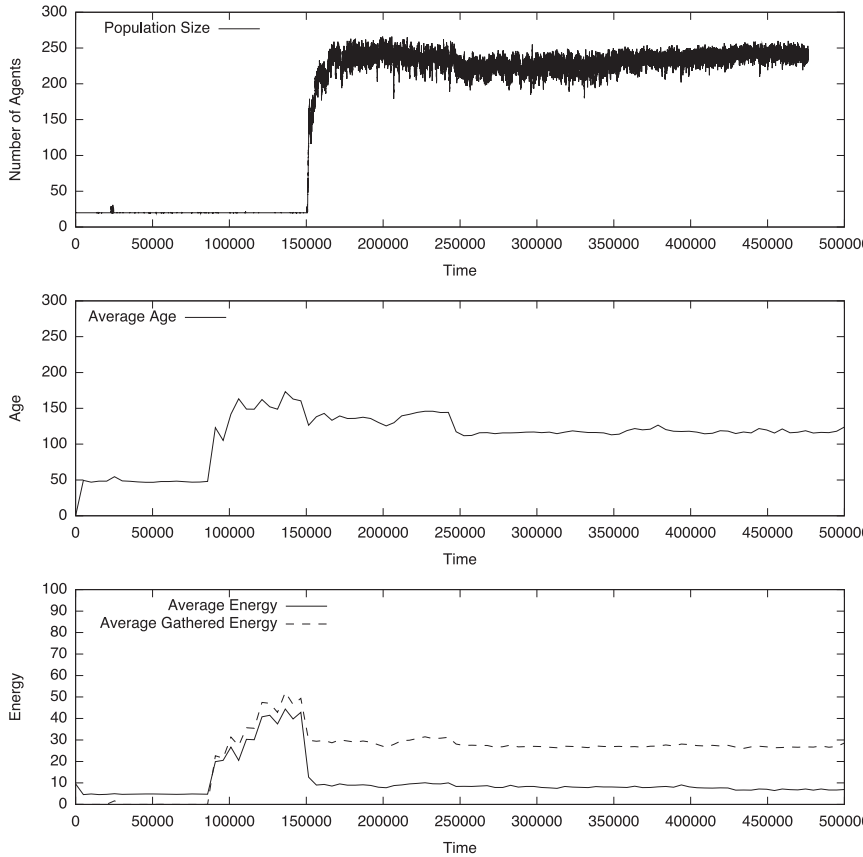

Fig. 4.15 *Plot of the evolution of the number of agents, their average age, average energy, and average gathered energy, over the course of one simulation run of experiment two.*

The results shown are from the same run of experiment two, and thus the number of rules of an agent is between fifteen and twenty. We can see from the plot that, at the start of the simulation, the average brain size varies between these two values. Then, after about 80 000 time units (the time when agents start gathering food, as seen on above), the average brain size stabilizes. From the analysis of all the simulations, we found that from the

point where an agent starts to have a good foraging behaviour, most of the subsequent agents will be from the lineage of this first ancestor. Moreover, considering that the mutation operators used do not change the number of rules, the brain size will consequently stabilize.

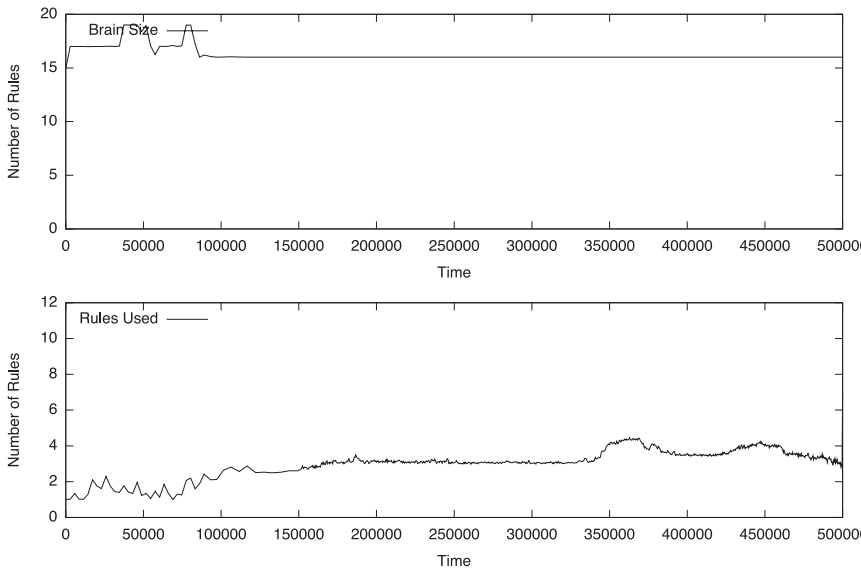

Fig. 4.16 *Plot of the evolution of the average brain size and average number of used rules, over the course of one simulation run of experiment two.*

However, if we examine the second plot, we see that the number of used rules does not completely stabilize. Thus, having a stable brain size, and an increasing number of used portions of that brain, indicates that there is an increase in the complexity of the agents and their behaviours. It is important to note that within the population of about 250 agents, there are a some that make it to the feeding zone, but also a large number that do not, pulling the average down. In the simulation run shown here, towards the end, the number of used rules for agents that do make it to the feeding zone is seven.

The analysis of the previous plots tells us that the agents do develop foraging capabilities, but does not show what kind of behaviours are emerging. These could be better observed by watching the running simulation, but that is not possible to show in this document. In any case, as the brain algorithm used in this experiments is fairly readable, we present an example of a brain taken from the population of agents from the same run as the plots shown. In Listing 4.1 we show an example of a brain taken from the end of the simulation.

```
1.  IF Feature energy > 18.4609 THEN reproduce
2.  IF istrue(Reaching Resource) THEN eat
3.  IF Resource Location > 1 THEN go front
4.  IF istrue(Feature placing pheromone) THEN turn left
5.  IF Resource Location = 1 THEN place pheromone
6.  IF istrue(Feature placing pheromone) THEN place
    pheromone
7.  IF Resource Location < Random Number THEN go front
8.  IF Feature energy > 25.0065 THEN go front
9.  IF Random Number = 2 THEN reproduce
10. IF istrue(Reaching Resource) THEN turn right
11. IF Random Number = Feature energy THEN go front
12. IF Resource Location = 0 THEN turn left
13. IF Feature energy > 34.5193 THEN turn right
14. IF istrue(Feature placing pheromone) THEN eat
15. IF Feature energy < Feature energy THEN turn right
16. IF istrue(Feature placing pheromone) THEN reproduce
```

Listing 4.1 *Example brain of an agent that evolved good foraging capabilities. This agent was born at time 499211. Used rules are set in bold.*

The example brain shown, allows us to decode some of its behaviour from the rule list presented. It is important to remember that the rule list is ordered, and the rule that is executed is the first one that evaluates to true. From rule one, we see that the agent has as first priority reproduction, and reproduces whenever it has more than 18.5 units of energy. And, from rule two, the agent eats whenever a food item is within its reach. The rest of the

behaviour can be divided into two parts: the movement behaviour when either there is some, or no food in the vision range. In other words, the behaviour is different when the agent finds the feeding zone. Examining rules three, four, and five, we can see that the agent will move forward if there is a food item in front or to the right, and turn left if there is a food item to the left. This is not the best possible behaviour because the agent does not turn right when the food is on its right, but it is a good enough behaviour. If there is no food item in the vision range, the value of the Resource Location perception is zero. Thus, if we consider the rules seven and twelve, we can see that the agent has a random walk behaviour. The agent will move forward if the *Random Number* perception is greater than zero (probability of 0.75), and will turn left otherwise. Again, this is not a perfect random walk, but it is good enough.

The data shown above is from one simulation run of one of the experiments conducted. Although the results are typical of all the runs where the agents evolve good foraging behaviours, we need to analyse global results from all the simulations. To that end we show in Table 4.19, for each experiment defined, the number of runs that were successful. We define a run as being successful if the agents evolve foraging and reproductive behaviours. This will give us an indication on how the conditions set by the parameters in each experiment influence the evolution of the agents.

Table 4.19 *Ant Foraging: number of successful runs.*

Experiment	***Total Runs***	***Successful***
1	30	0
2	30	13
3	30	11
4	30	10
5	30	30
6	30	27

To analyse the data from Table 4.19, we recall the three aspects considered for the definition of experiments: time limit, brain size, and environ-

mental resources. Considering the value of the time limit we will look at experiments one, two, and five. The results show that, when given enough time (5 000 000), all runs are successful. To analyse the effect of the brain size, we look at experiments two, three, and four. In these experiments there is no significant difference in the number of successful runs. That seems to indicate that the lower value used (fifteen to twenty rules, with one or two conditions) is sufficient to accommodate the evolved behaviours. Finally, considering the amount of available food in the environment, we'll examine experiments two and six. Again, we see that, as was the case for the time limit, the availability of resources has a significant influence in the evolutionary process. By increasing the food count from 100 to 200, the number of successful runs raised form thirteen to twenty seven.

Chapter Five

Day and Night Simulations

Most, if not all biological systems have some sort of adaptation to our planet's cycle of day and night. This adaptation is a current subject of scientific research (Rand et al. 2006; O'Neil and Reddy 2011). Despite having been extensively studied, these phenomena still have much to be investigated, but rather than wanting to learn more about the internals of this biological process, we use it as an inspiration to study the emergence of this kind of adaptation to a daily cycle. To that end we implemented a world with a day / night cycle, and analyse the ways the agents adapt to that cycle. In this chapter we describe and analyse the results of these experiments.

With these experiments, we aim to evolve different behaviours from those found in the foraging experiments. Namely, we test the agents' capability to adapt to the daily cycle. Again, we aim to evolve increasingly complex behaviours by modelling an increasingly complex world, while keeping the simulation free of any fitness function.

The chapter is divided into three sections. First we describe the base DayNight world, used for all the experiments in this chapter. Next, we describe a special dynamic version of this world created to test the genotype editing technique. And the third set of experiments adds more complexity to the environment by adding caves where the light is low as if it were night.

5.1. The DayNight World

In order to study the agents' adaptation to a daily cycle of day and night, we created a simulated world similar to the one described in section 4.1 (basic foraging), and added a daily cycle of lightness and darkness to the environment. This change in the light level of the environment will affect the agents' vision, making it harder to find food during the night. We also added to these simulations a varying level of metabolic rate for the agents. Rather than always having the same metabolic rate during their lifetime, the agents will vary their metabolic rate by moving or by sleeping. Naturally, when an agent moves, its metabolic rate will increase, and when it sleeps, its metabolic rate will decrease. This energy conservation achieved by sleeping will hopefully give an advantage to agents that evolve nightly sleeping behaviours.

By adding these extra characteristics to the simulated world, we aim to make the environment harder to evolve in and more complex. This added complexity will hopefully force the agents to evolve accordingly more complex behaviours adapted to their environment.

5.1.1. World Definition

The architecture of the experiments is analogous to that of the foraging simulations. We augmented the world defined in those experiments with the new requirements. We will now detail this architecture.

5.1.1.1. The Environment

Our world is a two-dimensional world where agents and resources are placed. The terrain is a square. This area restricts the placement of agents, caves, and resources, but does not restrict the movement of the agents. The world is infinite, i.e., an agent can move past the boundaries of the populated terrain. At startup, the field is populated with a configured amount of randomly placed food items.

There are three schedules defined. The first schedule keeps replenishing the world with resources. The number of resources available is configurable to be able to fine tune the system so as to allow agents to survive but also provide enough evolutionary pressure.

Another schedule generates new agents whenever the number of agents in the world falls below a given threshold. This schedule will pick agents that are still alive and create new agents based on them, as if they had reproduced. If there are no agents alive, new random agents are generated.

The last schedule generates the day / night cycle. It raises and lowers the light level at specified intervals. The light level oscillates between a configurable maximum and minimum. For each day the maximum light level is randomly calculated as the overall maximum minus a random value between zero and the delta. The same applies for the day's minimum. For example, if the maximum light level is 100, the minimum light level is 0, and the delta is 10, each day's maximum light level will be a random value between 90 and 100, and each day's minimum light level will be a random

value between 0 and 10. Additionally, the light level does not rise or fall abruptly, but rather changes linearly during a specified time interval, simulating dusk and dawn. To better illustrate, in Fig. 5.1 this variation of the light level can be observed. The duration of one day, can be configured, and remains constant for the full length of the simulation run.

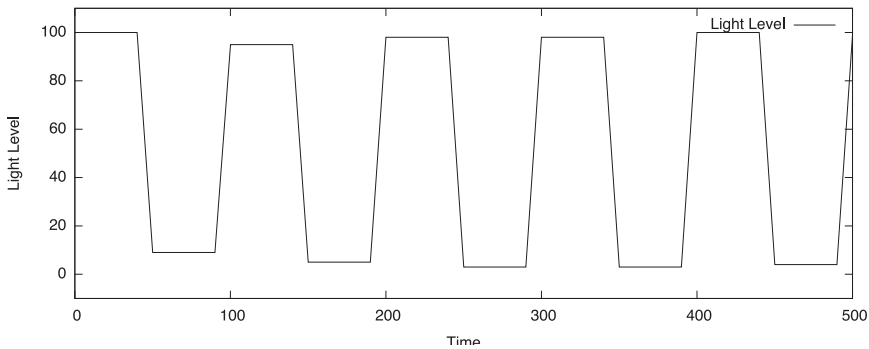

Fig. 5.1 *Example of the variation of the light level over the course of five days. In this example, the maximum light level is 100, the minimum is 0, and the delta is 10. The duration dawn and dusk is 10.*

5.1.1.2. Agents

In this simulation only one type (species) of agent exists, and has the following architecture:

- *Features:* energy, metabolic rate, and birth date.
- *Perceptions:* energy, resource location, reach resource, light level.
- *Actions:* move front, turn left, turn right, sleep, eat, reproduce.
- *Brain:* rule list.

We now describe each one of these components.

Features

- *Energy:* This feature represents the current energy level of the agent. When this feature reaches zero, the agent dies. The feature is initialized with a predetermined value at agent birth.

- *Metabolic Rate:* The metabolic rate is the amount of energy the agent consumes per time unit. This rate is initialized to its configured base value, and changes as the agent moves or sleeps. The increase or decrease amounts for move and sleep are configurable.
- *Birth Date:* This feature is set to the current time at birth and remains constant. It is used to calculate the agent's age. When the agent reaches a given age, it dies. This procedure allows the evolution to continue past the moment when the agents have developed good navigation and eating capabilities, whilst maintaining an asynchronous and open-ended simulation. The maximum age of the agents is configurable.

Perceptions

- *Energy:* This is a self-referencing perception on the agent's current energy level. This perception is tied to the corresponding feature. This is a numerical perception, and the range of values is configured according to the corresponding feature.
- *Resource Location:* This is the agent's perception of vision, representing the position of the nearest resource, relative to the agent's position and orientation. The agent's vision field is defined by a given range and angle. An object is within the vision field of an agent if its distance to the agent is less than or equal to the vision range, and the relative angle to the agent is within the defined vision angle (shown previously in Fig. 4.2). This is a numerical perception with possible values 0, 1, 2, and 3. The value 0 means no resource is visible. The value 1 means there is a resource to the left. The value 2 means there is a resource directly in front of the agent. The value 3 means there is a resource to the right. This perception is influenced by the light level of the environment. As the light level drops, so does the range of vision for the agent, using the formula shown in equation (5.3), where $V(t)$ is the vision range at time t, V_0 is the configured base vision range of the agents, $L(t)$ is the light level at time t, and L_0 is the configured maximum light level.

$$V(t) = V_0 \frac{L(t)}{L_0} \tag{5.3}$$

- *Reach Resource:* This is a boolean perception that evaluates to true whenever the agent has a resource within its reach. The distance the agent can reach is configurable.
- *Light Level:* This perception gives the agent the power of sensing the brightness of the environment. This can also be considered a perception of vision. The value of the perception is numeric and, at each time, is evaluated to the environment's current light level.

Actions

- *Movement:* We define three actions for movement. One to walk forward, one to turn left, and one to turn right. These actions have a tie to the metabolic rate feature in such a way that whenever the agent is moving, the metabolic rate increases.
- *Eat:* This action enables the agent to eat a resource within its range. If no resource is in range when the action is executed, nothing happens. This action will add a configured amount of energy to the agent's energy feature.
- *Sleep:* The agent can use this action to sleep. In this simulation, when an agent is sleeping, it will stand still and its metabolic rate will decrease, falling below the base metabolic rate and thus allowing the agent to conserve energy. As for the rest of the actions, it gets executed whenever the agent chooses to do so.
- *Reproduce:* This action allows the agent to reproduce itself. The reproduction implemented is asexual. When the action is executed, a new agent is created and placed in the world. The new agent will be given a brain that is a mutated version of its parent's brain. Note that, as each mutation operator has been given a probability of being applied, the child's brain can be a perfect clone of its parent's brain. The action will also transfer energy from the parent to the offspring. The amount of energy consumed in the action is the sum of the initial energy for the new agent and a configurable fixed cost.

It's important to have a cost of reproduction higher than the initial energy of an agent, so as to provide evolutionary pressure.

Brain

The agents' brain used in these experiments is a rule list, whose architecture was detailed in section 3.2.2. On initial creation of an agent, the brain is randomly initialized. This initialization conforms to some configurable parameters: the maximum number of rules, the minimum number of rules, and the maximum number of conditions per rule. Other configured values are the mutation probabilities used in the reproduction action.

5.1.1.3. Things

Only one type of thing is defined for this world: the resources that the agents eat to acquire energy. No features are associated with them. A configurable parameter defines the amount of energy each resource provides.

5.1.2. Results

In this section we present and analyse the results of the experiments. But first we give an overview of the main configuration values used for the simulations. In Table 5.1 we present all the values configured for the parameters of these experiments. These values were chosen empirically based on previous experimentation.

Table 5.1 *DayNight: Configuration values*

Parameter	Values
Terrain Size	1000 x 1000
Time Limit	100 000
Day Length	100
Day Transition Length	10
Maximum Light Level	100
Minimum Light Level	0; 10; 20
Light Level Delta	10

Parameter	Values
Minimum Food	200
Food Energy Content	3
Minimum Agents	20
Vision Range	200
Vision Angle	60º
Agent Reach	20
Agent Initial Energy	10
Base Metabolic Rate	0.1
Move Metabolic Increase	0.01
Sleep Metabolic Decrease	0.03
Maximum Age	500
Minimum Rules	15
Maximum Rules	20
Maximum Conditions	2
Mutation Probability	0.01

In these experiments, as we are testing the evolution of the agents' adaptation to the daily cycle, the main parameter to test is the minimum light level. For that parameter we show here the three tested values. These three experiments are numbered as shown on Table 5.2. Using these values, we ran thirty independent simulations for each configuration of minimum light level and collected the results.

Table 5.2 *DayNight: Experiments configuration*

Experiment	Min. Light Level
1	0
2	10
3	20

In Fig. 5.2 we show the results from a typical simulation run, and are able to examine the evolution of the agents ability to gather food and the point where they start to reproduce by themselves. By analysing the progress of

the average energy and average age of the agents, its clear that they learn to gather food. In this example, the agents start to eat at about 28 000 time units, then make a considerable improvement in their food gathering capabilities around time 38 000. At that time almost every agent will die of old age, as can be seen by the average age of the population reaching 250. Note that the maximum age of an agent is configured to be 500, and new agents are continually being born.

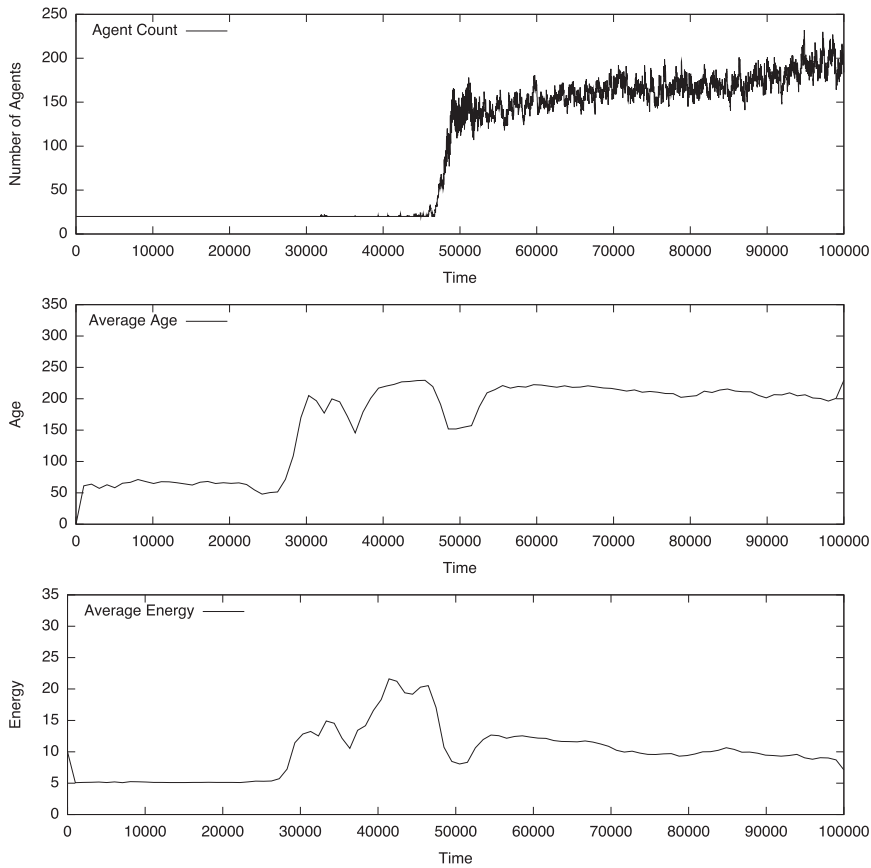

Fig. 5.2 *The evolution of the total number of agents in the population, their average age and average energy, over the course of one simulation run of experiment one.*

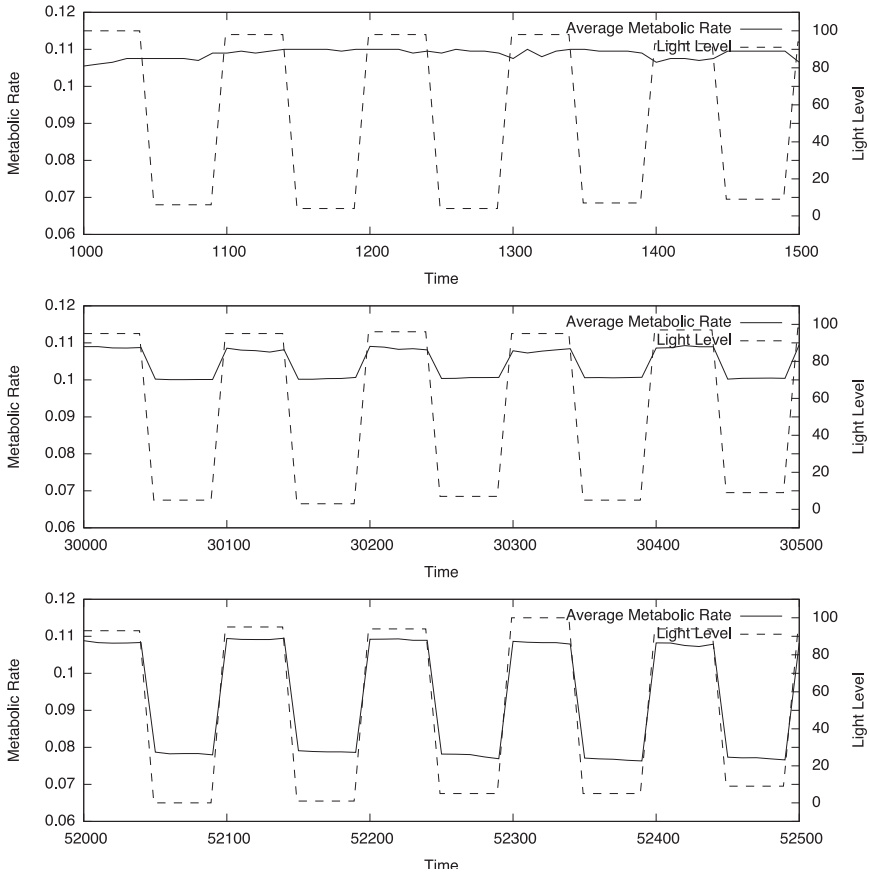

Fig. 5.3 *Extracts from a simulation run of experiment one, showing the average metabolic rate of the agents and the environments' light level over a period equivalent to five days. The three plots cover different times in the course of the simulation run.*

Then, at about time 48 000, the agents start to reproduce by themselves. When that happens, both the average age and the average energy of the agents fall. As the number of agents competing for the food increases, the quantity of food each one can harvest is naturally smaller. As for the average age of the agents, we notice that the fall is not as high as for the energy. This indicates that most agents are still capable of reaching old age. Some time after, at about time 52 000, both the average age and average energy, to some degree. This, again, suggests another improvement in the agents food

gathering capabilities. From here on the values stabilize, except for a slow increase in the population size.

In Fig. 5.3 we try to show the adaptation of the agents to the cycle. Note that the simulation run presented in these plots is not the same of Fig. 5.2. This phenomenon would be best observed by watching the real-time visualisation of a running simulation. In that case one can clearly see that the agents stop moving during the night and are active during the day. On the companion CD we provide a video of a running simulation where these behaviours can be observed. As that is not possible to show here, we can get a sense of that phenomenon by analysing the variation of the agents metabolic rate, which is affected by the variation of their movement and sleep patterns.

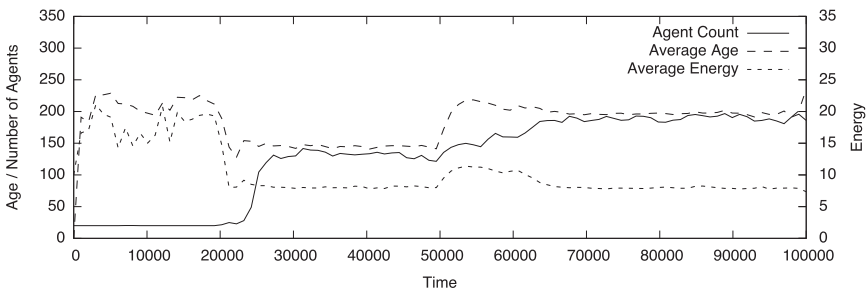

Fig. 5.4 *The evolution of the total number of agents in the population, their average age and average energy, over the course of one simulation run from experiment one.*

We can see that at the beginning of the simulation, the agents' behaviours have no connection to the light level. In the middle plot, we can see that, by the time 30 000, the agents have started to adapt to the daily cycle. It is interesting to note, however, that the average metabolic rate of the population does not fall to 0.07 during the night. Keep in mind that the base metabolic rate is 0.1, and that when sleeping it decreases 0.03, making the rate be equal to 0.07 when the agent sleeps. This can be due to any of two possibilities: first, at this time, there might still be some agents in the population that don't follow the cycle, and second, the agents have evolved to do nothing at night, rather than sleep. In the case of this simulation run,

it seems to be the latter, owing to the fact that an agent that does nothing will have a metabolic rate of 0.1. Then, by the time 52 000, the value of the average metabolic rate lowers, nearing the 0.07 value, as more and more agents follow a circadian rhythm, sleeping during the night.

To allow an overview of the effects this adaptation has on the population, we show a plot in Fig. 5.4, similar to that of Fig. 5.2, but with the data from this simulation run. It is interesting to verify that, once the agents start sleeping during the night, both the average age and average energy of the population go up, as does the number of agents, due to the increased energy efficiency.

In the previous figures we show the results of typical simulation runs. However, we ran thirty trials for each configuration scenario. Most simulation runs exhibited similar results, differing mainly on the time where the phenomena can be observed. In Table 5.3 we give an overview of the results from all simulation runs.

Table 5.3 *Overview of the success of runs*

Experiment	Foraging		Reproduction		Sync	
1	28	93%	21	75%	23	83%
2	23	77%	21	91%	15	65%
3	28	93%	24	86%	1	4%

By analysing the results from the simulation runs, it is clear that, the added evolutionary pressure introduced by having a lower minimum light level, forces the agents to adapt to the daily cycle. The percentages shown in the reproduction and sync columns consider only the runs where foraging was developed. In general we verify that the runs where agents develop foraging capabilities but don't synchronize are the ones where agents only gain foraging capabilities near the end of the simulation (about 90 000 time units), leaving no time to evolve the circadian rhythm.

As for the experiment with a minimum light level of twenty, the broader range of possible wining strategies reveals itself, as different behaviours are evolved, not synchronized to the daily cycle. Moreover, these behaviours

seem to be, to some extent, a better strategy to survive in this world with that value for minimum light level. One example of a different behaviour evolved is the case of agents that are always moving during day and night. In Fig. 5.5 we can see that the population size will grow to about 350, whereas in the previous examples shown, the population would only raise to about 200.

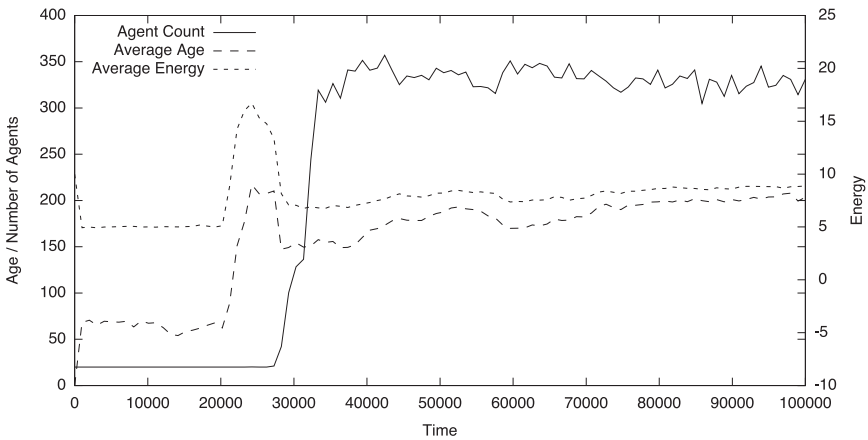

Fig. 5.5 *The evolution of the total number of agents in the population, their average age, average energy, and average gathered energy over the course of one simulation run from experiment three.*

5.2. Dynamic Version

In this section we present the simulations conducted to test the genotype editing evolutionary technique (see section 3.1.2.2). In the previous experiments that used this technique with standard genetic algorithms, the authors concluded that the method has best results in dynamic environments (Rocha et al. 2006; Huang et al. 2007).

Following these conclusions we developed a dynamic version of our day and night simulation. Although our original simulation environment already has some dynamic characteristics, namely the dynamics of the environment's light level, the evolutionary pressure does not change over the course of one simulation run. Thus, in order to create an environment that

has a dynamic evolutionary pressure, we introduced a periodic variation to the agents' vision capabilities. The agents will either have day vision, or night vision, and this will change periodically throughout the simulation run.

5.2.1. World Definition

As mentioned above, the simulated world in these experiments differs from the previous one mainly in the added environmental dynamics. Therefore, in this section we will only describe the changes introduced with these experiments.

5.2.1.1. The Environment

In order to impose a periodic change to the evolutionary pressure, we added to the environment a schedule that periodically changes the agents' vision capabilities. The agents will either have good day vision or good night vision, effectively changing its capacity to find food.

5.2.1.2. Agents

In these simulations, the agents have almost the same architecture as those defined in the standard DayNight world. We thus have one agent species with the following structure:

- *Features:* energy, metabolic rate, and birth date.
- *Perceptions:* energy, resource location, reach resource, light level.
- *Actions:* move front, turn left, turn right, sleep, eat, reproduce.
- *Brain:* editype rule list.

These agents differ from the former in the brain used. To test the genotype editing technique, we use a slightly modified version of the rule list brain that adds the required components to edit the genotype (see section 3.2.2.4). The other change is brought by the new environmental dynamics. The *Resource Location* perception will be affected by the periodic change. When an agent has good day vision, its vision range will be governed by the equation (5.3), presented above. Conversely, when an agent has good night vision, we will use the formula in the equation below.

$$V(t) = V_0 \left(1 - \frac{L(t)}{L_0}\right) \tag{5.4}$$

5.2.2. Results

In this section we present and analyse the results of the experiments, but first we give an overview of the main configuration values used in the simulations. As most of the configuration values are the same as seen in the previous experiments, we will focus on the what changed. In Table 5.4 we show all the new and changed parameter values used for these experiments.

Table 5.4 *Dynamic DayNight: Configuration values*

Parameter	Values
Time Limit	100 000; 200 000; 350 000; 600 000
Minimum Light Level	0
Use Dynamic Version	Yes; No
Dynamic Version Start	0; 100 000
Dynamic Version Period	10 000; 25 000; 50 000
Minimum Editors	1
Maximum Editors	2; 5
Use Genotype Editing	Yes; No

We introduced some new parameters in these experiments related to the generation and evolution of the agents' editypes. Upon initial creation of the agents, their editype has to be randomly generated. Similarly to what is defined for the generation of rules, we determine a minimum and a maximum of editors that can be generated for each agent. To evolve the editype we need to set the probability for the editype mutation operator.

The rest of the new parameters are related to the dynamic version. Namely, we can configure the time at which the periodic change in the environment starts, and the time interval for each period.

Despite having created a dynamic version of the DayNight world specifically to test the genotype editing technique, we first tested it with the standard world. To that end we executed two experiments using the stand-

ard DayNight world, but with the genotype editing version of the brain. The two experiments differ only the value for the parameter that defines the maximum number of editors. In Table 5.5 we show the main configuration values for these experiments. As usual, we performed thirty independent simulation runs for each of the experiments.

Table 5.5 *Genotype Editing: Experiments configuration*

Experiment	Dynamic	Min. Editors	Max. Editors
101	No	1	5
102	No	1	2

Analysing the results from the runs with the genotype editing technique turned on, we verify that the results are similar to those obtained with the technique turned off. In Fig. 5.6 and Fig. 5.7 we display plots of a typical run from experiment 101. Comparing with those shown on the previous section, we can find the same kind of evolutionary scenario. The agents first develop foraging capabilities, then start reproducing by themselves, and finally synchronize with the daily cycle.

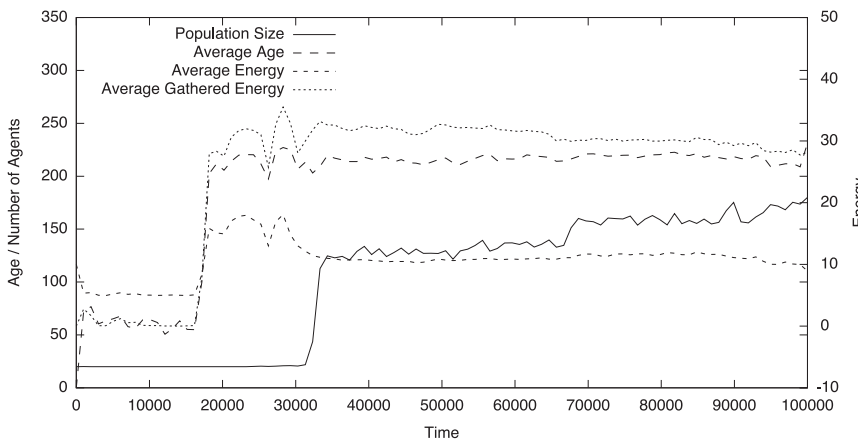

Fig. 5.6 *The evolution of the total number of agents in the population, their average age, average energy, and average gathered energy, over the course of one simulation run of experiment 101.*

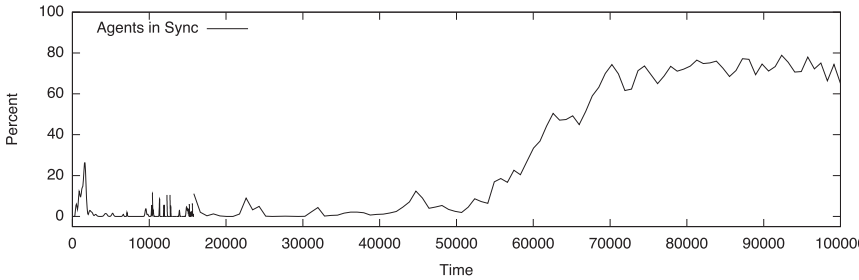

Fig. 5.7 *The evolution of the percentage of agents in sync over the course of one simulation run of experiment 101.*

From the analysis of the evolution plots, we cannot detect any significant difference in using genotype editing. So, we looked into the overall success of the runs from each experience to see if there is an advantage or disadvantage in using the technique. The summary of that data is presented in Table 5.6. We present the total number of runs that are successful in the evolution of foraging, reproduction, and synchronization. Regarding the percentages shown for the reproduction and synchronization success, we calculate it considering only the number of runs where foraging capabilities were evolved. In this table we also recall the results from experiment 1 of the standard DayNight world from the previous section, in order to have a comparison measure.

Table 5.6 *Comparison of the results from experiments with and without genotype editing, on the standard DayNight world.*

Experiment	Foraging		Reproduction		Sync	
1	28	93%	21	75%	23	82%
101	24	80%	17	71%	19	79%
102	25	83%	22	88%	22	88%

Again, from these results we cannot find a clear advantage or disadvantage in using genotype editing. The differences encountered may be attributed to the randomness of the initial population. We can however verify that the technique does not hinder the evolutionary capability of the system.

As mentioned earlier, to further test this technique we used the dynamic version of the world. In Table 5.7 we show the values tested for these experiments. We also run experiments of the dynamic version with the genotype editing technique turned off to permit the comparison of results.

Table 5.7 *Dynamic DayNight: Experiments configuration*

Exp.	Time Limit	Gen. Editing	Min. Editors	Max. Editors	Dynamic Start	Dynamic Period
103	200k	No	N/A	N/A	100k	10k
104	200k	Yes	1	5	100k	10k
105	200k	Yes	1	2	100k	10k
106	350k	Yes	1	5	100k	25k
107	600k	Yes	1	5	100k	50k
108	100k	No	N/A	N/A	0	10k
109	100k	Yes	1	5	0	10k
110	100k	Yes	1	2	0	10k

We can divide these experiments into two groups. The fist five shown in the table use a two step process, using the step evolution technique described in chapter 3. First we let the agents evolve in the standard version of the DayNight world for 100 000 time units, and then turn on the dynamic version and check how the agents perform in this new environment. In the second group (the last 3 experiments), we use the dynamic version from the start of the simulation, effectively making the initial evolution environment more challenging.

As usual, we show in Fig. 5.8 a series of plots from one simulation run. In this case, we show a run from experiment 103. Although not entirely typical of all runs form the first group of experiments, it provides an example of the agents re-adaptation to the new conditions after time 100 000. We say that the run is not entirely typical because the number of times the agents are able to recover from the abrupt change in the environment is not the same for all runs. Otherwise, the plots are similar for the periods where they do adapt. This is true for both the experiments without genotype editing and with genotype editing.

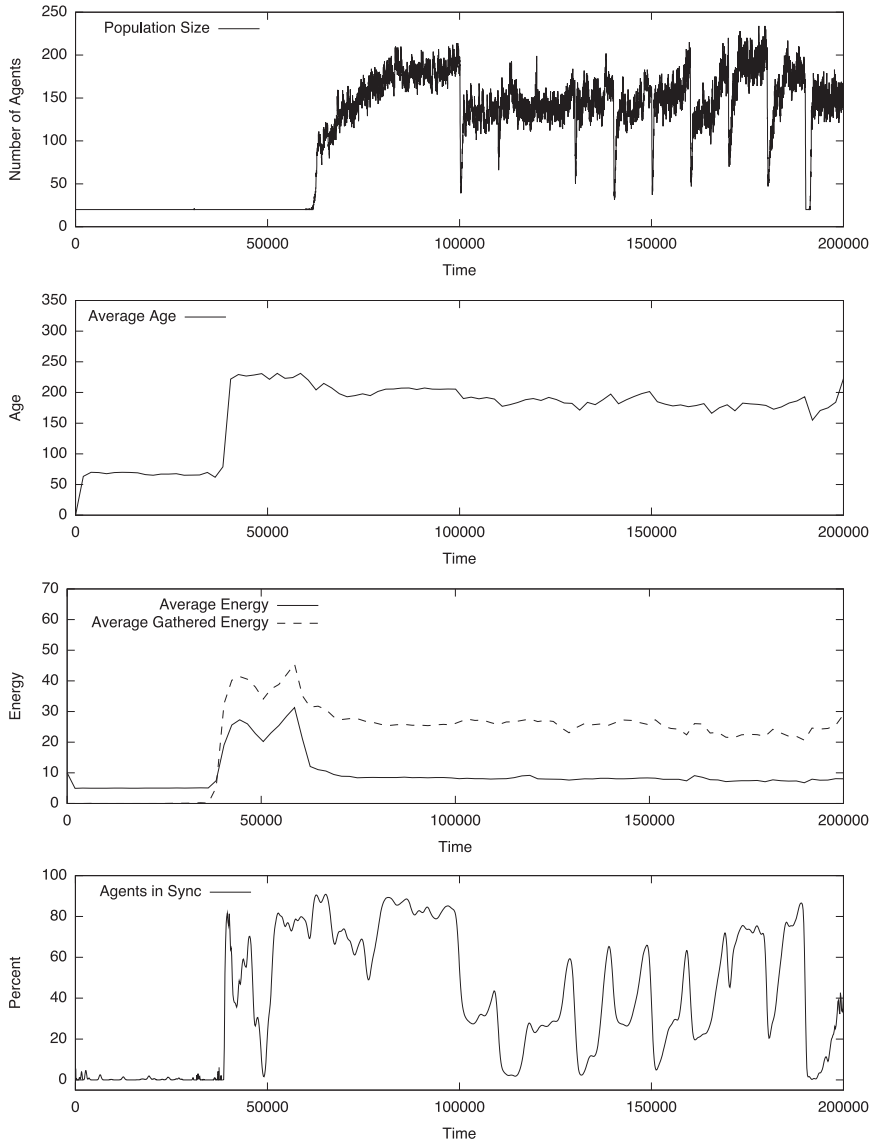

Fig. 5.8 *The evolution of the total number of agents in the population, their average age, average energy,, average gathered energy, and percentage of agents in sync over the course of one simulation run of experiment 103.*

Analysing the plots, we can see that the agents quickly recover after every change of the environment conditions. Beginning at time 100 000, which is when we turn on the dynamic version, its easy to pinpoint the time of each periodic change. We see that the population suddenly drops and also looses its synchronization, but then quickly recovers.

The second group of experiments, however, shows a different scenario. Although there are still cases where the agents develop foraging, reproduction, and synchronization, the number os periods where they can recover drops, and the number of successful runs also drops.

In Table 5.8 we provide an overview of the results from all runs of the experiments. In that table we show, for each experiment, the total number of successful runs and the average number of successful recovery periods per run. Note that the maximum number of recovery periods possible is ten. We consider a run to be successful if, in the first 100 000 time units the agents are able to adapt to the daily cycle. We also show in Fig. 5.9 boxplots of these results.

Table 5.8 *Comparison of the results from experiments with dynamic version*

Experiment	Successful Runs	Number of recovery periods	
		Average	Std. Dev.
103	24	6.42	2.21
104	21	6.24	3.24
105	21	5.95	2.64
106	21	7.24	2.36
107	21	7.81	2.20
108	20	2.10	1.33
109	18	2.89	1.23
110	16	2.69	2.15

Analysing the data from all the runs of the experiments, we can see that, again, there doesn't seem to be a clear difference in the results from the experiments with and without genotype editing. Although the number of

successful runs drops when we use genotype editing, the agents capability to recover is maintained on those runs that are successful.

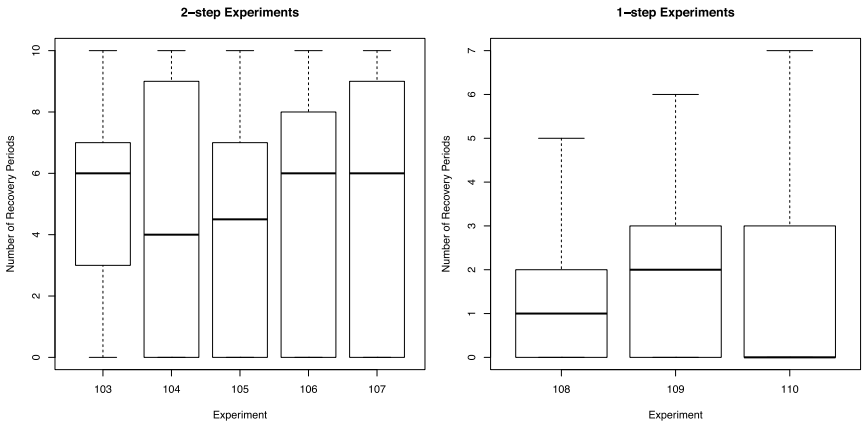

Fig. 5.9 *Boxplots of the number of recovery periods for experiments in one step and two steps.*

To further test if we can find differences in the these results, we determined the statistical significances null hypothesis of no differences with Kruskal-Wallis ANOVAs with α=0.05.

If we consider only the successful runs, comparing among the first group, we find there is no significant difference, with p-value of 0.130. Comparing among the second group, we also find no significant difference, with p-value of 0.242. If we consider unsuccessful runs as having 0 recovery periods and include them in the statistical tests, we get a p-value of 0.643 for the first group, and 0.526 for the second group.

Comparing across groups we do, however, find a significant difference with a p-value of 2.73×10^{-7}. Running further pairwise tests using the Mann-Whitney U tests with Holm's p-value adjustment, we get the results shown in Table 5.9. We can see that overall we can find a better performance of experiments of the first group, when compared with the second group, except for experiments 104 and 105 where significant differences are not found for all pairwise tests. Thus, generally, evolution in two steps had better performance than evolution in one step.

Table 5.9 *Pairwise comparisons across dynamic version groups*

	103	***104***	***105***	***106***	***107***	***108***	***109***
104	1	—	—	—	—	—	—
105	1	1	—	—	—	—	—
106	1	1	1	—	—	—	—
107	1	1	1	1	—	—	—
108	0.00049	0.06378	0.07956	0.02461	0.01344	—	—
109	0.00165	0.16326	0.15967	0.03595	0.02354	1	—
110	0.00071	0.04583	0.05019	0.01456	0.00728	1	1

5.3. DayNight with Caves

In an effort to add more complexity to the DayNight world, and consequently forcing the agents to evolve more complex behaviours, we introduced a new element to the environment: caves. Inside the caves, the light level is the same as if it were night. An agent entering a cave will, thus, not be able to distinguish between "night" and "cave". This will add an evolutionary pressure to the simulation, forcing the agents to develop behaviours that cope with this new reality.

The introduction of caves in the environment is to some extent based on the work of Mirolli and Parisi (2003). In that paper, a similar environment is used to investigate different types of circadian rhythm controllers. They employ standard genetic algorithm techniques to evolve the neural network of agents in this environment, with a fitness function that selects for both good foraging capabilities and low activity levels. In our work, as mentioned previously, we use open-ended evolution and so we do not directly select for any given behaviour.

5.3.1. World Definition

For these experiments the world created differs little from the one described in section 5.1.1. We simply added the caves and the effect they have on the light level perceived by the agents. As most of the architecture is maintained, in this section we only describe what was changed.

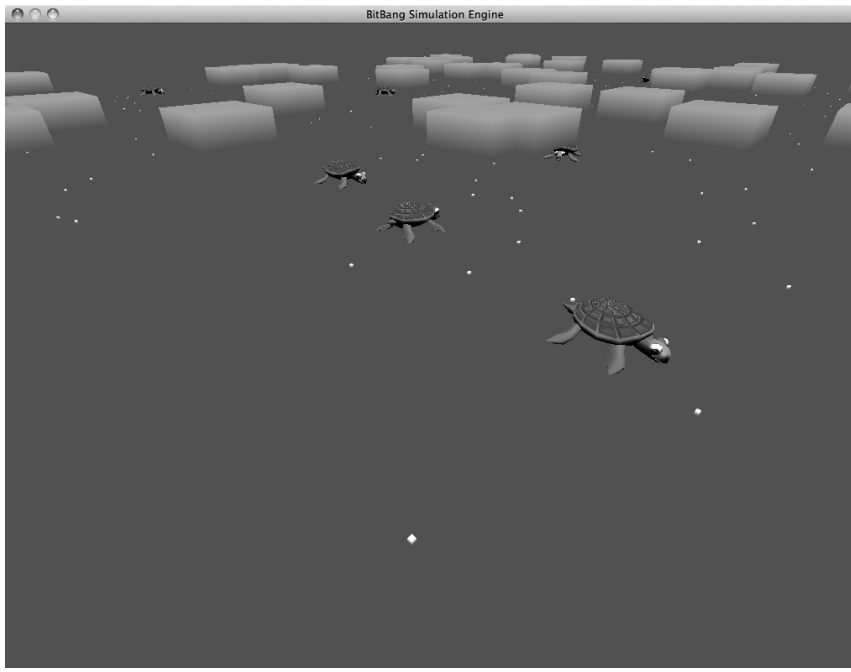

Fig. 5.10 *Screenshot of a running simulation. We can see the agents (turtles), the edible resources (very small boxes), and the caves (large boxes).*

5.3.1.1. The Environment

The total simulation time for these experiments was divided into two equal parts. For the first part of the simulation, the agents live in an environment that only has food items. Then, we randomly place in the environment a number of caves (implemented using a schedule), and continue the simulation in this new environment. In Fig. 5.10 we show a screenshot of a running simulation where the presence of caves in the environment can be seen. The number of caves that are placed in the environment is configurable.

When an agent enters a cave, the light level it perceives will be the same as if it were night, creating a new challenge for our agents. By the time the caves are created, the agents will have evolved a behaviour adapted to the light level, sleeping when the light is low, and being active when its high.

However, when an agent enters a cave, it will not be able to distinguish if the low light level means "night" or "cave". If it simply goes to sleep whenever the light level is low, it will never wake up again, as the light level in the cave will never rise. Our agents will now have to adapt their behaviour to this new environment.

5.3.1.2. Agents

In these simulations, the agents have almost the same architecture as those defined in the standard DayNight world. We thus have one agent species with the following structure:

- *Features:* energy, metabolic rate, and birth date.
- *Perceptions:* energy, resource location, reach resource, light level.
- *Actions:* move front, turn left, turn right, sleep, eat, reproduce.
- *Brain:* rule list.

The only difference is on the agents' perception of light level. As we now have caves in the environment, this perception will evaluate to the environment's light level whenever an agent is outside, but when inside a cave, the perception will evaluate to the experiment's configured minimum light level.

5.3.1.3. Things

In this version of the DayNight world, we added a new type of thing to represent the caves. Although we do use things to model the caves, they are not visible to the agents, or it would be easier for them to distinguish "night" and "cave".

5.3.2. Results

In this section we present and analyse the results of the experiments. But first we give an overview of the main configuration values used for the simulations. Again, most of the configuration values for this experiments are similar to those used in the first DayNight experiments, thus we will focus on the new parameters. Nevertheless, we also present here a table with all the values for the configured parameters in these experiments.

Table 5.10 *DayNight with Caves: Configuration values*

Parameter	Values
Time Limit	200 000
Cave Placement Time	100 000
Minimum Light Level	0
Number of Caves	10; 20; 50
Size of Caves	50 x 50

The main differences in configuration are in the simulation time limit, and in the new parameters for the caves. The time limit for the simulations is now 200 000, and we introduce the caves in the environment at 100 000 time units. This two step process follows what is done in Mirolli and Parisi's work (2003). Regarding the cave configuration parameters, we ran experiments with different values for the number of caves placed

Again, we ran thirty independent simulations for each configuration of the parameters. As is the case in the previous experiments, from those 30 simulations, there are some where the agents will not be successful in evolving good foraging behaviour, and thus will not be able to further evolve reproduction or synchronization to the day cycle. We will say that those simulations were unsuccessful.

Next, we will present and analyse the data of typical runs from the simulations. From all the simulations and runs analysed, we found mainly two different types of plot. In Fig. 5.11 and Fig. 5.12 we show the first type, and in Fig. 5.13 and Fig. 5.14 we show the second. These represent the majority of results from the runs, except for those that are considered unsuccessful. All these figures are taken from runs with the same parameter configuration (fifty caves).

The first type of plot, shown in the first two figures, is what we expected to find in this experiment. Here, the population collapses and loses synchronization when the caves are inserted into the environment. Examining this plot, we can clearly see the agents are successful in evolving food gathering, reproduction, and synchronization in the first environment (up to time

100 000). Then, when the environment changes, most of the population dies and never recovers food gathering capabilities, or synchronization.

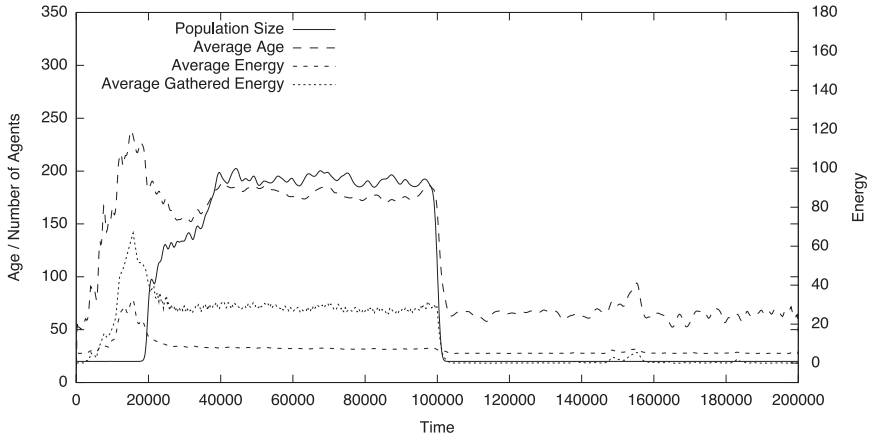

Fig. 5.11 *The evolution of the total number of agents in the population, their average age, average energy, and average gathered energy, over the course of one simulation run. The number of caves for this simulation run is fifty.*

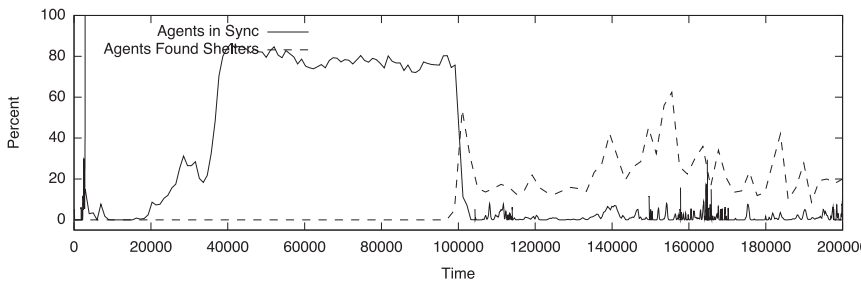

Fig. 5.12 *Plot showing the evolution of the percentage of agents in sync with the day cycle, and the percentage of agents that found caves during the course of one simulation run. The number of caves for this simulation run is fifty.*

In fact, this data is consistent with what is presented in Mirolli and Parisi's paper (2003). In that paper, the authors show that, when the agents only have an input of the light level, they are not able to differentiate the "caves" from the "night". They provide a possible solution to the problem,

by incorporating in the structure of the brain, a clock source. However, in our simulations, we found that on a significant number of runs (shown later in this chapter) the agents do recover.

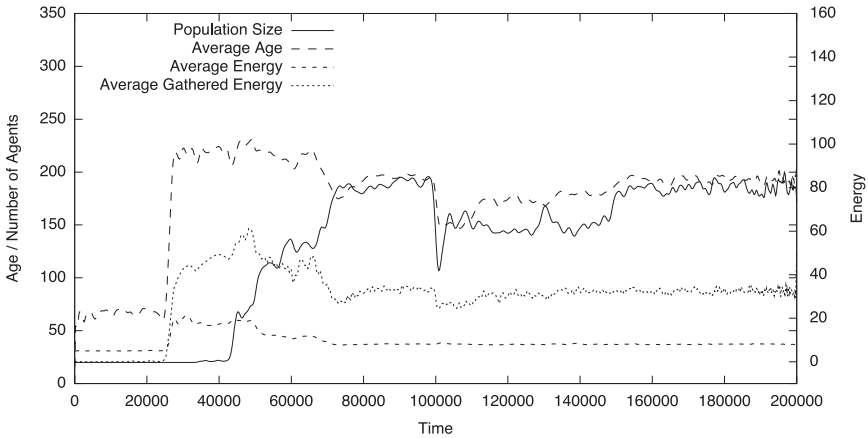

Fig. 5.13 *The evolution of the total number of agents in the population, their average age, average energy, and average gathered energy, over the course of one simulation run. The number of caves for this simulation run is fifty.*

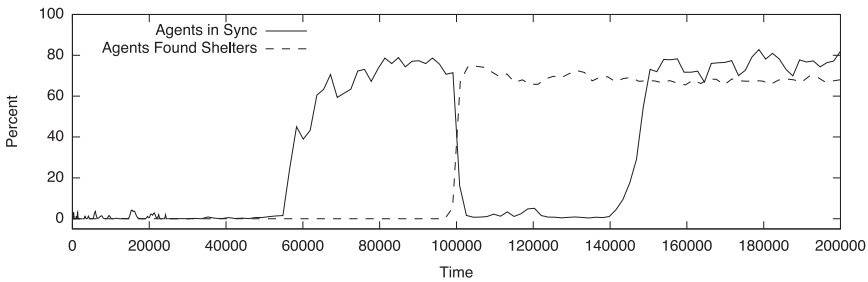

Fig. 5.14 *Plot showing the evolution of the percentage of agents in sync with the day cycle, and the percentage of agents that found caves during the course of one simulation run. The number of caves for this simulation run is fifty.*

In Fig. 5.13 and Fig. 5.14 we show an example of a typical run where the agents recover in the second environment. In the first environment, we find a plot similar to that of the previous example. The agents develop good

food gathering capabilities, reproduce, and synchronize with the daily cycle. When the environment changes at 100 000 time units, both the size of the population and the average gathered anergy drop, but then quickly recover. More importantly, by analysing the percentage of synchronized agents, we can see that, although it takes longer to recover, the agents also re-synchronize to the daily cycle. One might wonder if it is the case that the agents are simply not entering caves. But the plot also shows that about 80% of the agents find at least one cave during their lifetime. Note that, as these percentages are taken from the whole population at a given time interval, and there are constantly new agents being born, the percentage could never rise to 100%. New agents will normally need some time to move before they find a cave.

These results may seem counter-intuitive, as there doesn't seem to be any way for the agents to detect the caves. To clarify, we analysed some agents' brains. In Listing 5.1 and Listing 5.2 we show two examples of agents' brains from the simulation run shown in Fig. 5.14. The first one is taken from an agent living in the environment without caves (time 84 536), and the second is taken from the environment with caves and at a time where the agents have re-synchronized (time 183 513).

```
1.  IF Resource Location = 3 THEN turn right
2.  IF Light Level < Light Level THEN eat
3.  IF Resource Location > 3 THEN eat
4.  IF Light Level < 26.5859 THEN sleep
5.  IF Light Level < 36.3748 THEN sleep
6.  IF Light Level = 47.8427 THEN eat
7.  IF Feature energy > 16.7159 THEN reproduce
8.  IF Feature energy = 26.0336 THEN reproduce
9.  IF Resource Location = 3 THEN eat
10. IF Light Level < Feature energy THEN sleep
11. IF istrue(Reaching Resource) THEN eat
12. IF Resource Location < 2 THEN turn left
13. IF istrue(Reaching Resource) THEN turn left
14. IF not(Reaching Resource) THEN go front
15. IF Resource Location > 1 THEN sleep
```

```
16. IF istrue(Reaching Resource) THEN turn right
17. IF Light Level < 97.9668 THEN turn right
```

Listing 5.1 *Example of the structure of the brain of an agent born at time 84536, for the simulation run shown in Fig. 5.14.*

The analysis of the brain in Listing 5.1 allows us to see that the agent has good food gathering behaviour (rules 1, 11, 12, and 14), reproduces whenever it has more than 16.7159 energy (rule 7), and sleeps when the light level falls below 26.5859 (rule 4). If put in the environment with caves, this agent would clearly not survive, as it would fall asleep in a cave (from rule 4) and never wake up again.

```
1.  IF istrue(Reaching Resource) THEN eat
2.  IF Resource Location = Light Level THEN go front
3.  IF Light Level = 29.2361 THEN turn right
4.  IF Resource Location = 1 THEN turn left
5.  IF Resource Location < Resource Location THEN sleep
6.  IF Feature energy < Feature energy THEN sleep
7.  IF istrue(Reaching Resource) THEN eat
8.  IF Feature energy > 16.7159 THEN reproduce
9.  IF istrue(Reaching Resource) THEN reproduce
10. IF Light Level = Resource Location THEN eat
11. IF Light Level = 59.7254 THEN eat
12. IF istrue(Reaching Resource) THEN go front
13. IF Resource Location = 2 THEN go front
14. IF Light Level > 11.8449 THEN turn right
15. IF Resource Location < 0 THEN sleep
16. IF Resource Location < Light Level THEN sleep
17. IF Resource Location < 0 THEN reproduce
```

Listing 5.2 *Example of the structure of the brain of an agent born at time 183513, for the simulation run shown in Fig. 5.14.*

Examining the brain presented in Listing 5.2, we can finally see how the agents adapt to the environment with caves. The important rule in this case is the one at position 2. To explain the behaviour induced by this rule, we

need to take a closer look. Lets first consider that the agent is in a cave. If the agent doesn't have any resource within its vision range (very likely as in the cave the vision range is small), the value of *Resource Location* will be 0. As we know, in a cave the light level is equal to the minimum light level, which is 0. Therefore, this rule will make the agent move forward whenever it is inside a cave. In fact, we find this rule (or some small variation) in all the agents analysed in runs where there is recovery of synchronization.

The rest of the capabilities of the agent are also relatively easy to find in this rule list. From rules 1, 4, 13, and 14, we can see that the agent has a good foraging behaviour. Rule 8 provides reproduction capabilities. And rule 16, when combined with rule 14, makes the agent sleep if the light level of the environment is less than 11.8449 and greater than 3 (maximum value for the *Resource Location* perception).

Table 5.11 *Comparison of the success of runs*

Num. Caves	***Runs***	***Successful***	***Don't Rec.***		***Recover***	
50	30	23	9	39%	14	61%
20	30	22	3	14%	19	86%
10	30	24	1	4%	23	96%

In Table 5.11 we show an overview of the successful runs from all the tested simulation configurations. As stated earlier, we ran simulations with different values for the number of caves present in the environment. These results show that the configuration change doesn't seem to affect the number of successful runs out of the total of thirty runs. This was expected, as we were only changing the number of caves, which didn't affect the first environment. However, if we look at the number of runs where agents recover in the environment with caves, we find different results for the three configurations. Analysing the results, we see that with a smaller number of caves in the environment, the probability an agent has of finding a cave within its lifetime diminishes, making it easier to maintain the synchronization from the first environment.

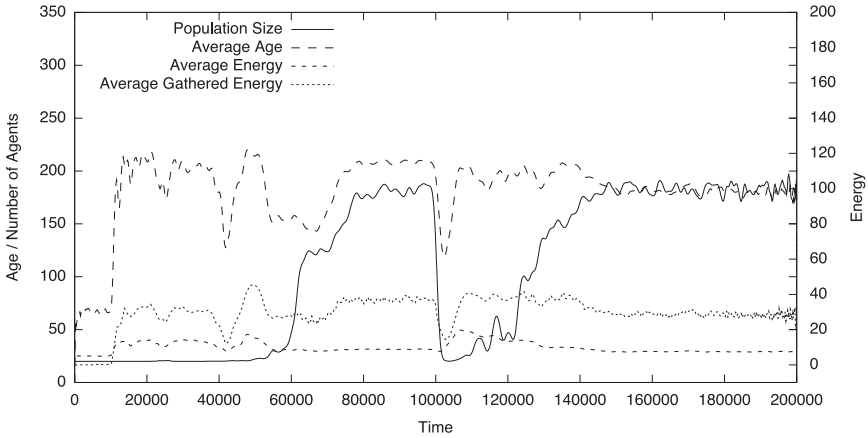

Fig. 5.15 *The evolution of the total number of agents in the population, their average age, average energy, and average gathered energy, over the course of one simulation run. The number of caves for this simulation run is twenty.*

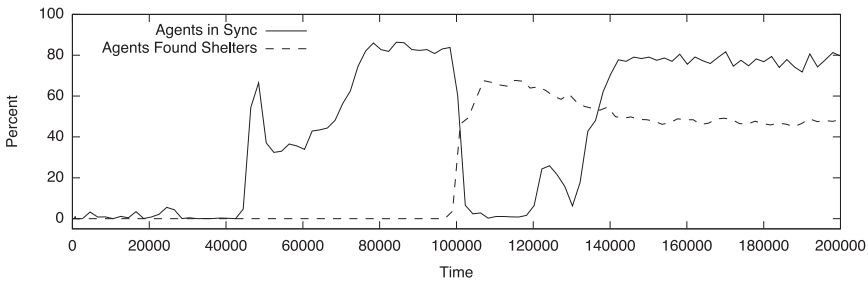

Fig. 5.16 *Plot showing the evolution of the percentage of agents in sync with the day cycle, and the percentage of agents that found caves during the course of one simulation run. The number of caves for this simulation run is twenty.*

In Fig. 5.15 and Fig. 5.16 we show a run of the simulation with twenty caves. The plots are similar to those shown previously, with the main difference being in the percentage of agents that find caves. In this case we can see that the percentage stabilizes at about 50%, whereas with the fifty cave configuration it stabilizes at about 80%. In the configuration with ten caves, the percentage falls to about 30%. As expected, the less caves in the environment, the lower the probability of an agent finding a cave in its lifetime.

Chapter Six

Conclusion

In this chapter we discuss the overall results of this work while also giving a summary of its contributions. We will also present some proposals of future work.

6.1. General Discussion

The goal of our work has been to research the evolution of complex behaviours in societies of agents, through open-ended evolution, by combining ideas from complex systems research and artificial life. We aimed to develop a highly abstract multi-agent model inspired by the evolutionary processes found in Nature, devoid of any centralized or supervised control, or selection mechanism.

We believe that the goal we proposed has been achieved through the modelling, implementation, and experimental validation of an artificial life framework capable of open-ended evolution.

Our main focus on devising our model was on allowing open-ended evolution. With that goal in mind, we consider that the principal feature needed is Natural Selection, achieved by freeing the system of any centralized control, like fitness functions. The model is centred on the agent, following Darwin's idea that the main unit of evolution is the organism. Towards this effort, we believe that the modelling of reproduction as an action that is triggered by the agents, and itself affected by evolution, is an important feature of the model. The other features of the model that we consider have contributed in the effort of achieving open-ended evolution are the following. First, the initial generation of the population is random, freeing the model from introducing, even if unintended, a-priori knowledge. Next, evolution is asynchronous, i.e. agents reproduce at any given time of the simulation, thus not having fixed generations. Next, two different features that both contribute to roughly the same goal of having situated agents. These are the modelling of the environment as part of the simulated world, and the incorporation of the notion of lifetime of the agent. Finally, the flexibility of the base model allows the simulation of different levels of complexity, and the easy incorporation of new techniques into the model.

In order to test the model we implemented a component for the brain of the agents, modelled as a rule list. The main focus on creating this component was on simplicity and readability. We believe that results show that being able to understand the inner workings of the agents' brains, gave us an important tool to analyse the results of the simulations. However, this brain architecture may have some limitations in the range of behaviours it can ultimately produce. This is a trade-off from the simplicity and readability requirements. Although we haven't found clear evidence of these limitations, this issue merits further research.

The incorporation of the genotype editing technique into our evolutionary process, also constitutes an important contribution. Although the results didn't show a clear advantage on using genotype editing in that specific simulated world, we consider that this development showed one of the advantages of having a base flexible model. The implementation of genotype editing was easy to fit into the system and required no specific tweaking of the simulations. It is also significant to note that, although not having shown an advantage, the technique also didn't hinder the evolutionary process.

The validation of our model followed a methodology common to complex systems research, which is the experimentation through computer simulations. To that end, the implementation of the BitBang Framework, which constituted a considerable part of this work, has proved to be an important tool to achieve our goals. The framework was made available to the community as open source software. As a result of the highly flexible nature of the framework, some research work has already been published in another complex systems research field (Caillou et al. 2008; Caillou et al. 2009; Curchod et al. 2009).

In the experimental scenarios created we set out to test the overall capability of the model to evolve agents' controllers in open-ended evolution. The first foraging scenario was developed to perform a benchmark of the parameters used in the simulations. Overall we found that the mutation operators chosen are capable of producing viable behaviours. We could also see that the operators are not very sensitive to variation of the probability

of mutation. The results show that there is a wide range of values that have good performance for most operators. However, having to choose, the value for probability of mutation that showed overall best performance was 0.01.

The second foraging scenario created, provided an environment where the agents had to develop more complex behaviours to cope with the foraging task. The environment is more challenging because the food is out of the initial vision range of the agents. The results show that the agents adapt to this environment by evolving a random walk behaviour, similar to what is found in Nature in some species of ants. We also included in this scenario the scenario the capability to communicate via pheromone. However, the results showed that the agents did not take advantage of this capability. The analysis of the results did not give us any indication as to why this happens. Nonetheless, we believe that in the environment created, the agents do not need that capability to have a good enough foraging behaviour. In Nature, some species of ants use this form of communication to aid in the foraging task. Though, whereas in our simulations the agents can eat the food as soon as they find it, in the real world we see that ants gather the food and return it to the nest. Without this added complexity in our simulated world, there will be no evolutionary pressure to take advantage of the pheromone.

With the day and night simulations, we were able to verify that, by creating increasingly complex environments, the agents evolve increasingly complex behaviours. We could also find that unexpected behaviours can be evolved, as was the case in the environment with caves. In that environment the agents adapted to distinguish the low light of the night from the caves, by taking advantage of a specificity of the environment. As the light level when inside a cave is always zero, and outside a cave, at night, it is between zero and ten, the agents adapted to that fact. It is important to note that the scenario was not designed with this in mind, and in that regard that behaviour was unexpected. Moreover, this result shows that in this kind of simulation, the agents are able to take advantage of environmental features not designed for the purposed they use them for. This result may have a parallel in the real world, where it is common to find species that take advantage of specific properties of the environment, creating niches.

The scenario developed to test the genotype editing technique, gave us an opportunity to also test the step evolution technique. Although in the first case, as we have already discussed, no clear advantage was found in using genotype editing, in the case of step evolution we did find that it can have an impact in the results. By using step evolution and gradually changing the environment by increasing its complexity, agents are able to more easily evolve adapted behaviours in the more complex environment.

Overall, the experimental results show that our model is capable of evolving agents' controllers, from initial random conditions, and without any supervised evolutionary process. Moreover, we have seen that, populations evolve sustainable reproduction behaviours, without needing to hard-code the reproduction conditions into the simulation.

As already discussed, the framework developed is highly flexible. We believe that this flexibility can contribute to solving a current issue of artificial life research. Alife has a strong basis in the biosciences, however in the last couple of decades, since alife inception, biology witnessed tremendous advances in our understanding of life, that have not been incorporated into alife research (Rocha 2007). We believe that the freedom of evolution based on natural selection, and the flexibility of our framework, can contribute to more easily incorporate new biological knowledge into alife simulations.

6.2. Future Work

As expected from the nature of the work presented in this document, many open research issues are still left to be explored. Some of these issues that we consider deserve further study are presented in this section.

The step evolution technique was tested only in one simulation, showing good promise. This technique should be further studied, and simulations with more evolutionary steps should be created.

In the ant foraging scenario we have successfully evolved random walk behaviours. However, further development of this scenario should be focused on finding the environmental conditions, or agent architecture needed to evolve communication using pheromone. The evolution of communication, or cooperation, has been extensively studied, but mostly in

evolutionary simulations that use artificial selection. To our knowledge, no successful attempts have been made in open-ended evolution simulations.

On all the simulations presented we used the rule list for the brain of the agents. As the framework was created with an abstract concept of brain, other architectures can be tested. We could implement several different algorithms, like neural networks, or other rule list architectures, and compare the results obtained. The simulation scenarios already developed could be easily used to test new brain architectures.

One extension to the rule list brain that is already being considered is to make the decision process stochastic. Instead of evaluating all rules in sequence, and choosing the first that evaluates to true, we could for example choose a random one among all rules that evaluate to true, eventually employing a bias. The choice of this bias presents a particular challenge, considering that we want to keep the evolution open-ended and thus can not directly reward good rules based on fitness. One possible solution to this problem is the implementation of a bucket brigade type algorithm, similar to what Holland (1985) proposed for classifier systems.

The add and remove rule mutation operators were tested only on the basic foraging scenario. Further tests should be made to assess the effects of these operators on new simulation scenarios. Special attention should be taken into checking if these operators are also bloat-free in the new scenarios.

One other interesting direction of research related to the rule list brain, and its mutation operators, is the study of the effects of neutral mutations, and of non-coding genetic information in our open-ended evolution process. From the visual inspection of the brains conducted, we found some parts of the brains that were never used, but could account for neutral mutations. We also found some cases where the non-used rules of the brain could serve as memory of past good behaviour. To better study these issues, specific experiments and analysis techniques should be created.

The flexibility of the framework can be used to apply this open-ended evolutionary model to a number of different simulation scenarios. For example, one that could show interesting results, is the coevolution of agents'

controllers and morphologies, following Karl Sims work. The added freedom that a natural selection system can give to such a simulation, may enable the evolution of an interesting and diverse set of behaviours and morphologies.

In conclusion, we believe that our work can be used in a variety of research areas, and hope that our initial effort can be developed upon by other researchers, mainly in the fields of artificial life and complex systems.

References

Adami, C. and Brown, C. T. (1994), 'Evolutionary Learning in the 2D Artificial Life System Avida', in R. Brooks and P. Maes (eds.), *Artificial Life IV: Proceedings of the Fourth International Workshop on the Synthesis and Simulation of Living Systems* (Cambridge, MA: MIT Press), 377–381.

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', *Evolutionary Computation, 2009. CEC '09. IEEE Congress on*: 1123–1130.

Bak, P. (1997), *How Nature Works: The Science of Self-Organized Criticality* (Oxford: Oxford University Press).

Baptista, T., Menezes, T., and Costa, E. (2006), 'Bitbang: A model and framework for complexity research', in *Proceedings of the European Conference on Complex Systems 2006* (Oxford, UK).

Baptista, T. and Costa, E. (2007), 'The emergence of a circadian rhythm in a multi-agent simulation', in *Proceedings of the European Conference on Complex Systems 2007* (Dresden, Germany).

Baptista, T. and Costa, E. (2008), 'Evolution of a multi-agent system in a cyclical environment', *Theory in Biosciences*, 127/2: 141–148.

Baptista, T. and Costa, E. (2011), 'The Evolution of Foraging in an Open-Ended Simulation Environment', in L. Antunes and H. S. Pinto (eds.), *Progress in Artificial Intelligence, 15th Portuguese Conference on Artificial Intelligence, EPIA 2011, Lisbon, Portugal, October 2011* (Springer-Verlag), 125–137.

Baptista, T. and Costa, E. (2011), 'The Evolution of Foraging in an Open-Ended Simulation Environment', in *Progress in Artificial Intelligence, 15th Portuguese Conference on Artificial Intelligence, EPIA 2011, Lisbon, Portugal, October 2011* (Springer-Verlag), 125–137.

Bass, B. L. (ed.) (2001), *RNA Editing* (Oxford: Oxford University Press).

Bedau, M. A. (2003), 'Artificial life: organization, adaptation and complexity from the bottom up', *Trends in Cognitive Sciences*, 7/11: 505–512.

Bedau, M. A., McCaskill, J. S., Packard, N. H., Rasmussen, S., Adami, C., Green, D. G., Ikegami, T., Kaneko, K., and Ray, T. S. (2000), 'Open problems in artificial life.', *Artificial Life*, 6/4: 363–376.

Benne, R. (ed.) (1993), *RNA Editing: The Alteration of Protein Coding Sequences of RNA* (New York: Ellis Horwood).

Benne, R., Van Den Burg, J., Brakenhoff, J. P. J., Sloof, P., Van Boom, J. H., and Tromp, M. C. (1986), 'Major transcript of the frameshifted coxll gene from trypanosome mitochondria contains four nucleotides that are not encoded in the DNA', *Cell*, 46/6: 819–826.

Bianco, R. and Nolfi, S. (2004), 'Toward open-ended evolutionary robotics: evolving elementary robotic units able to self-assemble and self-reproduce', *Connection Science*, 16/4: 227–248.

Bonabeau, E., Theraulaz, G., and Dorigo, M. (1999), *Swarm Intelligence: From Natural to Artificial Systems* (Oxford: Oxford University Press).

Brooks, R. A. (1990), 'Elephants Don't Play Chess', *Robotics and Autonomous Systems*, 6/1-2: 3–15.

Caillou, P., Baptista, T., and Curchod, C. (2008), 'Multi-Agent Based Simulation for Decision-Making: an Application to Rungis Food Market', in *Group Decision and Negotiation Meeting, GDN 2008* (Coimbra, Portugal).

Caillou, P., Curchod, C., and Baptista, T. (2009), 'Simulation of the Rungis Wholesale Market: Lessons on the Calibration, Validation and Usage of a Cognitive Agent-Based Simulation', in *IEEE/WIC/ACM International Joint Conferences on Web Intelligence and Intelligent Agent Technologies, 2009. WI-IAT '09,* 70–73.

Camazine, S., Deneubourg, J.-L., Franks, N. R., Sneyd, J., Theraulaz, G., and Bonabeau, E. (2001), *Self-Organization in Biological Systems* (Princeton: Princeton University Press).

Channon, A. D. (2000), 'Three evolvability requirements for open-ended evolution', in C. C. Maley and E. Boudreau (eds.), *Artificial Life VII Workshop Proceedings* (Portland, OR), 39–40.

Channon, A. D. (2001), 'Evolutionary Emergence: The Struggle for Existence in Artificial Biota', D.Phil. thesis (University of Southampton).

Channon, A. D. and Damper, R. I. (1998), 'Perpetuating evolutionary emergence', in *Proceedings of the fifth international conference on simulation of adaptive behavior on From animals to animats 5* (MIT Press).

Channon, A. D. and Damper, R. I. (2000), 'Towards the evolutionary emergence of increasingly complex advantageous behaviours', *International Journal of Systems Science*, 31/7: 843–860.

Chaumont, N., Egli, R., and Adami, C. (2007), 'Evolving virtual creatures and catapults', *Artificial Life*, 13/2: 139–157.

Collins, R. J. and Jefferson, D. R. (1991), 'AntFarm: Towards Simulated Evolution', in C. G. Langton et al. (eds.), *Artificial Life II: Proceedings of the Workshop on Artificial Life Held February, 1990 in Santa Fe, New Mexico* (Boulder, CO: Westview Press), 579–602.

Curchod, C., Caillou, P., and Baptista, T. (2009), 'Which Buyer-Supplier Strategies on Uncertain Markets? A Multi-Agents Simulation', in *Strategic Management Society 09* (Washington DC).

Darwin, C. (1859), *On the Origin of Species by Means of Natural Selection: Or the Preservation of Favoured Races in the Struggle for Life* (London: John Murray).

Deneubourg, J. L. and Pasteels, J. M. (1983), 'Probabilistic Behaviour in Ants: a Strategy of Errors?', *Journal of Theoretical Biology*, 105/2: 259–271.

Dewdney, A. K. (1984), 'In the game called Core War hostile programs engage in a battle of bits', *Scientific American*, 250/5: 14–22.

Dewdney, A. K. (1988), *The Armchair Universe: An Exploration of Computer Worlds* (New York: W. H. Freeman & Co).

Dorigo, M. and Stutzle, T. (2004), *Ant Colony Optimization* (MIT Press).

Dorigo, M., Birattari, M., and Stutzle, T. (2006), 'Ant colony optimization', *Computational Intelligence Magazine, IEEE*, 1/4: 28–39.

Eiben, A. E. and Smith, J. E. (2003), *Introduction to evolutionary computing* (Berlin: Springer-Verlag).

Eiben, A. E., Griffioen, A. R., and Haasdijk, E. (2007), 'Population-based Adaptive Systems: an Implementation in NEWTIES', in *ECCS 2007 - European Conference on Complex Systems* (Dresden, Germany).

Fogel, L. J., Owens, A. J., and Walsh, M. J. (1966), *Artificial Intelligence Through Simulated Evolution* (New York: John Wiley & Sons, Inc.).

Gilbert, N., Besten, den, M., Bontovics, A., Craenen, B. G. W., Divina, F., Eiben, A. E., Griffioen, R., Hevizi, G., Lorincz, A., Paechter, B., Schuster, S., Schut, M. C., Tzolov, C., Vogt, P., and Yang, L. (2006), 'Emerging Artificial Societies Through Learning', *The Journal of Artificial Societies and Social Simulation*, 9/2. <http://jasss.soc.surrey.ac.uk/9/2/9.html>, accessed 28 July 2011.

Hebb, D. O. (1961), 'Distinctive Features of Learning in the Higher Animal', in J. F. Delafresnaye (ed.), *Brain Mechanisms and Learning* (London: Oxford University Press), 37–46.

Holland, J. H. (1975), *Adaptation in Natural and Artificial Systems: An introductory analysis with applications to biology, control, and artificial intelligence* (Ann Arbor: University of Michigan Press).

Holland, J. H. (1985), 'Properties of the Bucket Brigade', in *Proceedings of the 1st International Conference on Genetic Algorithms* (Pittsburgh, PA), 1–7.

Holland, J. H. (1994), 'Echoing Emergence: Objectives, Rough Definitions, and Speculations for Echo-Class Models', in G. A. Cowan, D. Meltzer, and D. Pines (eds.), *Complexity: Metaphors, Models and Reality* (Reading, MA: Addison-Wesley).

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

Holland, J. H. (2006), 'Studying Complex Adaptive Systems', *Journal of Systems Science and Complexity*, 19/1: 1–8.

Hraber, P. T., Jones, T., and Forrest, S. (1997), 'The ecology of echo.', *Artificial Life*, 3/3: 165–190.

Huang, C.-F. and Rocha, L. M. (2003), 'Exploration of RNA editing and design of robust genetic algorithms', in *The 2003 Congress on Evolutionary Computation, 2003. CEC '03*, 2799–2806.

Huang, C.-F. and Rocha, L. M. (2004), 'A Systematic Study of Genetic Algorithms with Genotype Editing', in *Genetic and Evolutionary Computation – GECCO 2004* (Berlin: Springer Berlin / Heidelberg), 1233–1245.

Huang, C.-F., Kaur, J., Maguitman, A., and Rocha, L. M. (2007), 'Agent-based model of genotype editing.', *Evolutionary computation*, 15/3: 253–289.

Huxley, J. (1942), *Evolution: The Modern Synthesis* (London: Allen & Unwin).

Jones, T. and Forrest, S. (1993), *An Introduction to SFI Echo* (Santa Fe Institute).

Kauffman, S. A. (1993), *The Origins of Order: Self Organization and Selection in Evolution* (Oxford University Press, USA).

Kawamura, H. and Ohuchi, A. (2000), 'Evolutionary Emergence of Collective Intelligence with Artificial Pheromone Communication', in *IEEE International Conference on Industrial Electronics, Control and Instrumentation*, 2831–2836.

Koza, J. R. (1992), *Genetic Programming: On the Programming of Computers by Means of Natural Selection* (Cambridge, MA: MIT Press).

Langton, C. G. (ed.) (1989), *Artificial Life: Proceedings of an interdisciplinary workshop on the Synthesis and Simulation of Living Systems, Los Alamos, 1987* (Addison-Wesley).

Lassabe, N., Luga, H., and Duthen, Y. (2007), 'A New Step for Artificial Creatures', *IEEE Symposium on Artificial Life, 2007. ALIFE '07*, 243–250.

Lotka, A. J. (1956), *Elements of Mathematical Biology* (New York: Dover Publications).

Maley, C. C. (1999), 'Four Steps Toward Open-Ended Evolution', in W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. E. Smith (eds.), *GECCO-99: Proceedings of the Genetic and Evolutionary Computation Conference* (San Francisco, CA: Morgan Kaufmann Publishers Inc), 1336–1343.

McCrea, W. H. and Whipple, F. J. W. (1940), 'Random Paths in Two and Three Dimensions', *Proceedings of the Royal Society of Edinburgh*, 60: 281–298.

McIndoe, B. W. (2005), 'jECHO: A new implementation of Holland's Complex Adaptive Systems model with the aim of seeking answers to some fundamental open questions', Master's diss. (University of Liverpool).

McMullin, B. (2000), 'John von Neumann and the evolutionary growth of complexity: looking backward, looking forward...', *Artificial Life*, 6/4: 347–361.

Mendel, G. (1985), 'Experiments in Plant Hybridization', <http://www.mendelweb.org/Mendel.html>, accessed July 2011.

Menezes, T., Baptista, T., and Costa, E. (2006a), 'Towards Generation of Complex Game Worlds', in *Computational Intelligence and Games, 2006 IEEE Symposium on* (Reno, NV), 224–229.

Menezes, T., Baptista, T., and Costa, E. (2006b), 'BitBang - a Library for Modern Game AI', in *International Digital Games Conference (iDiG 2006)* (Portalegre, Portugal).

Miconi, T. (2007), 'The Road to Everywhere: Evolution, Complexity and Progress in Natural and Artificial Systems', D.Phil. thesis (University of Birmingham).

Miconi, T. and Channon, A. (2006), 'An Improved System for Artificial Creatures Evolution', in L. M. Rocha et al. (eds.), *Artificial Life X: Proceedings of the Tenth International Conference on the Simulation and Synthesis of Living Systems* (Cambridge, MA: MIT Press), 255–261.

Mirolli, M. and Parisi, D. (2003), 'Artificial Organisms That Sleep', in W. Banzaf et al. (eds.), *Advances in Artificial Life: Proceedings of the 7th European Conference on Artificial Life, ECAL 2003* (Springer Berlin / Heidelberg), 377–386.

Nakamichi, Y. and Arita, T. (2005), 'Effectiveness of Emerged Pheromone Communication in an Ant Foraging Model', in *The Tenth International Symposium on Artificial Life and Robotics 2005, AROB 10th '05* (Oita, Japan), 324–327.

Nolfi, S. and Floreano, D. (2000), *Evolutionary Robotics: The Biology, Intelligence, and Technology of Self-Organizing Machines* (MIT Press).

Ofria, C. and Wilke, C. O. (2004), 'Avida: A Software Platform for Research in Computational Evolutionary Biology', *Artificial Life*, 10/2: 191–229.

O'Neill, J. S. and Reddy, A. B. (2011), 'Circadian clocks in human red blood cells', *Nature*, 469/7331: 498–503.

Packard, N. H. (1989), 'Intrinsic adaptation in a simple model for evolution', in C. G. Langton (ed.), *Artificial Life: Proceedings of an interdisciplinary workshop on the Synthesis and Simulation of Living Systems, Los Alamos, 1987* (Addison-Wesley).

Paulsen, O. and Sejnowski, T. J. (2000), 'Natural patterns of activity and long-term synaptic plasticity', *Current Opinion in Neurobiology*, 10/2: 172–179.

Rand, D. A., Shulgin, B. V., Salazar, J. D., and Millar, A. J. (2006), 'Uncovering the design principles of circadian clocks: mathematical analysis of flexibility and evolutionary goals', *Journal of Theoretical Biology,* 238/3: 616–635.

Ray, T. S. (1992), 'Evolution, Ecology and Optimization of Digital Organisms', Working Paper 92-08-042 (Santa Fe Institute).

Ray, T. S. (2001), 'Artificial Life', in R. Dulbecco et al. (eds.), *Frontiers of Life, Volume One The Origins of Life* (Academic Press), 107–124.

Rechenberg, I. (1973), *Evolutionsstrategie, Optimierung technischer Systeme nach Prinzipien der biologischen Evolution* (Stutgard: Frommann-Holzboog).

Rocha, L. M. (1998), 'Selected Self-Organization and the Semiotics of Evolutionary Systems', in S. Salthe, G. Van de Vijver, and M. Delpos (eds.), *Evolutionary Systems: Biological and Epistemological Perspectives on Selection and Self-Organization.* (Kluwer Academic Publishers), 341–358.

Rocha, L. M. (2007), 'Reality is Stranger than Fiction', invited presentation for the "Biocomplexity" discussion section at the 9th European Conference on Artificial Life, September 12, 2007 in Lisbon, Portugal.

Rocha, L. M. and Huang, C.-F. (2004), 'The Role of RNA Editing in Dynamic Environments', in J. Pollack et al. (eds.), *Artificial Life IX: Proceedings of the Ninth International Conference on the Simulation and Synthesis of Living Systems* (Cambridge, MA: MIT Press), 489–494.

Rocha, L. M. and Kaur, J. (2007), 'Genotype Editing and the Evolution of Regulation and Memory', in F. A. E. Costa, L. M. Rocha, E. Costa, I. Harvey, and A. Coutinho (eds.), *Proceedings of the 9th European Conference on Artificial Life* (Berlin: Springer-Verlag), 63–73.

Rocha, L. M., Maguitman, A., Huang, C.-F., Kaur, J., and Narayanan, S. (2006), 'An Evolutionary Model of Genotype Editing', in L. M. Rocha, M. A. Bedau, D. Floreano, R. Goldstone, A. Vespignani, and L. Yaeger (eds.), *Artificial Life X: Proceedings of the Tenth International Conference on the Simulation and Synthesis of Living Systems* (Cambridge, MA: MIT Press), 105–111.

Russell, S. and Norvig, P. (2002), *Artificial Intelligence: A Modern Approach (2nd Edition)* 2nd ed. (Prentice Hall).

Sannier, A. V., II, and Goodman, E. D. (1987), 'Genetic Learning Procedures in Distributed Environments', in J. J. Grefenstette (ed.), *Genetic algorithms and their applications: proceedings of the second International Conference on Genetic Algorithms* (Hillsdale, NJ: L. Erlbaum Associates), 162–169.

Schwefel, H.-P. (1981), *Numerical Optimization of Computer Models* (John Wiley & Sons Ltd).

Schwefel, H.-P. (1995), *Evolution and Optimum Seeking* (New York: Wiley Interscience).

Sims, K. (1994a), 'Evolving 3D morphology and behavior by competition', *Artificial Life*, 1/4: 353–372.

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

Smith, J. M. and Szathmáry, E. (1985), *The Major Transitions in Evolution* (Oxford: Oxford University Press).

Smith, R. M. and Bedau, M. A. (2000), 'Is Echo a complex adaptive system?', *Evolutionary computation*, 8/4: 419–442.

Standish, R. K. (2003), 'Open-ended artificial evolution', *International Journal of Computational Intelligence and Applications*, 3/2: 167–175.

Strogatz, S. (2004), *Sync: The Emerging Science of Spontaneous Order* (London: Penguin Books).

Stuart, K. (1993), 'RNA editing in mitochondria of african trypanosomes', in R. Benne (ed.), *RNA Editing: The Alteration of Protein Coding Sequences of RNA* (New York: Ellis Horwood), 26–52.

Tavares, J. (2007), 'Evolvability in Optimization Problems: The Role of Representations and Heuristics', Ph.D. thesis (University of Coimbra).

von Neumann, J. (1966), *Theory of self-reproducing automata* (Urbana and London: University of Illinois Press).

Watson, J. D. and Crick, F. H. (1953), 'Molecular Structure of Nucleic Acids; A Structure for Deoxyribose Nucleic Acid.', *Nature*, 171/4356: 737–738.

Wilkins, M. H. F., Stokes, A. R., and Wilson, H. R. (1953), 'Molecular Structure of Deoxypentose Nucleic Acids', *Nature*, 171/4356: 738–740.

Wooldridge, M. (2002), *An Introduction to MultiAgent Systems* (John Wiley & Sons).

Yaeger, L. (1994), 'Computational genetics, physiology, metabolism, neural systems, learning, vision, and behavior or Poly World: Life in a new context', in C. G. Langton (ed.), *Artificial Life III: Proceedings of the Workshop on Artificial Life* (Addison-Wesley), 263–298.