COMPLEXITY IN ARTIFICIAL LIFE

Daniel Anders Gösta Högberg

(B.Sc. (Honours), Uppsala University)

A THESIS SUBMITTED FOR THE DEGREE OF

MASTER OF SCIENCE

DEPARTMENT OF COMPUTER SCIENCE

SCHOOL OF COMPUTING

NATIONAL UNIVERSITY OF SINGAPORE

2005

To my parents and my wife

Acknowledgments

First and foremost, I would like to express my deepest gratitude, and thank my supervisor Martin Henz, for his inspiration, support and always helpful comments. Working in a research area such as Artificial Life has indeed been a trip into the unknown, and Martin Henz has given invaluable support for such a journey.

I would also like to thank Janardan Mishra for many interesting discussions concerning life, the Earth and everything there between.

For moral support and everyday motivation, I would also like to thank my friend Magnus Ågren at Uppsala University.

Last but not least, I would like to thank E-Jean Tan for proof-reading this thesis and for being a tremendous support.

Contents

1 Introduction to Artificial Life **1**

1.1 Life is hard to study 2

1.1.1 Definitions of life 2

1.1.2 Weak or Strong Artificial Life 3

1.1.3 Complexity of life 3

1.2 History and background 4

1.2.1 Cellular automata 5

1.2.2 Assembler automata 6

1.3 Evolutionary complexity 8

1.4 Thesis outline 10

2 Evolution **12**

2.1 Biological evolution 13

2.2 An example of evolution 14

2.3 Artificial evolution 15

2.3.1 Replication 15

2.3.2 Variation 15

2.3.3 Selection 17

2.4 Chemostat 17

2.5 Merit function and fitness landscapes 19

2.5.1 Genetic drift 20

2.5.2 Müller's ratchet 21

3 A New Artificial Chemistry—SALTA **22**

3.1 Micro-level: Virtual CPU and program 23

3.1.1 CPU 23
3.1.2 Program 24
3.2 Macro-level: Dynamics 32
3.2.1 Self-replication and death (r) 33
3.2.2 Mutation ($\simeq$) 34
3.2.3 Merit (S) 35
3.2.4 Scheduling (α) 35

4 Mutational Evolution **37**
4.1 Information theory and quasi-species 38
4.1.1 Eigen's quasi-species 38
4.1.2 Strive for the smallest 40
4.2 Adaptive landscape—an example 40
4.2.1 Evolution, as a function of mutation rate 41
4.2.2 Evolution, as a function of population size 43
4.2.3 Evolution, as a function of topology 44

5 Genetic Recombination **47**
5.1 Composable evolution 48
5.2 Extending SALTA with recombination 49
5.2.1 Recombination for computation 49
5.2.2 Negative effects of recombination 50
5.3 Experiment: recombination versus mutation 50
5.3.1 Complex evolution with mutation 51
5.3.2 Complex evolution with recombination 52
5.3.3 Genotype length and diversity 55

6 Symbiosis and co-evolution **57**
6.1 Cost-benefit relations and symbiosis 58
6.2 Symbiosis and co-evolution in Artificial Life 59
6.3 Extending SALTA with symbiosis 59
6.3.1 I/O Buffer 61
6.3.2 Resources 61

6.4 Experiment: Sufficient conditions for symbiosis 61
6.4.1 (a) Single layer without resource relations 63
6.4.2 (b) Single layer with resource relations 64
6.4.3 (c) Multiple layers without resource relations 65
6.4.4 (d) Multiple layers with resource relations 66
6.5 Mutational pressure with symbiosis 69

7 Red-Queen Evolution **73**
7.1 Open-ended evolution 74
7.2 Red-Queen hypothesis and evolution 74
7.2.1 Model for evolution in an ever-changing environment 75
7.3 Modifying SALTA 75
7.3.1 Resources 76
7.3.2 I/O Buffer 76
7.3.3 Merit function 76
7.4 Experiment: open-ended evolution 77
7.4.1 Experimental observations 77

8 Conclusion **82**
8.1 Summary 82
8.2 Contributions 84
8.3 Discussion 84
8.4 Critique 84
8.5 Future work 85

A Settings and Parameters **93**
A.1 Command line arguments 93
A.1.1 Default SALTA 93
A.1.2 SALTA with symbiosis 95
A.1.3 SALTA with Red-Queen 95
A.2 Initialization file 96
A.3 Global settings 97
A.4 Output file 98

B Glossary **100**

Summary

This thesis introduces a new evolutionary platform called SALTA, for the study of evolutionary complexity in a finite population of *digital organisms*. These organisms execute assembler-like programs that run on individual CPUs, and have a clear distinction between "genotype" (program) and "phenotype" (execution). By imposing differential reproductive success, based on the organism's executional behavior (via a merit function), self-replication, natural selection and mutation introduce evolution in the population. Throughout the conducted experiments, a new notion of *evolutionary complexity* is defined as the product of (genotype or phenotype) diversity and average genotype size. The evolutionary platform is formally defined using the notion of a *chemostat*; a model which is widely used in biology and chemistry to conduct experiments with finite systems. Since the default platform is extended and used in a wide range of experiments, the implementation is modular with many built-in parameters. The genotype instruction-set is also made small and as robust as possible under mutation.

In the first set of experiments, basic parameters such as mutation rate, population size and offspring dispersal were investigated in the evolution of pre-defined logical functions by point-mutation. It was found that evolutionary complexity decreases in the evolution of the logical functions.

Thereafter, *genetic recombination* was added in the second set of experiments to be compared with point-mutation for evolving the logical functions. Recombination was found to be more efficient than mutation, for smooth fitness landscapes and for bridging gaps in rugged fitness landscapes.

In order to try to increase evolutionary complexity, interdependence between the (biotic or abiotic) environment and the evolving population was introduced in the third and fourth set of experiments. In the third set of experiments, *symbiosis* was added to the model. This was done by (i) layering the model with chemostat layers and (ii)

connecting the layers with a transfer of the resources that are used for merit. By doing so, sufficient conditions were found for symbiosis, e.g. by using a novel correlation measure called vicinity difference. It was also found that symbiosis gave increased evolutionary complexity in terms of increased phenotype diversity and genotype size. In the fourth set of experiments, a new merit function was defined for open-ended evolution in a theoretical *Red-Queen scenario*. We found that genetic drift limited the genotype growth earlier than the error threshold, and that the evolved genotypes were strongly dependent on the population size. Theoretically, this model should give unbounded complexity by indefinite genotype growth and accumulative genotype diversity, although genetic drift was found to limit such evolution.

To conclude, we found that recombination was more efficient than point-mutation for certain fitness landscapes, and that interdependencies between environment and evolving population—e.g. by symbiosis or Red-Queen evolution—increased our evolutionary complexity measure.

Tables

3.1 Default self-replicating organism. 31

3.2 Experimentally smallest self-replicating organism. 31

3.3 Self-replicating OR-computing program. 33

4.1 Default merit and merit with gap. 41

4.2 Mutation rate, average merit per organism, average genome length, average genotype diversity, and evolutionary complexity. 43

5.1 NOT and NAND computing self-replicating organisms. 54

6.1 The 4 possible settings for dimensional difference and resource relations. 62

7.1 Experimental and theoretical genome length. 81

8.1 Summary of experiments. 86

Figures

2.1 Point-mutations: substitution, insertion and deletion. 16
2.2 Genetic recombination: merge and split. 17
2.3 Fitness landscape with peaks and valleys. 19
3.1 Program word. 25
3.2 Illustrates how pattern-based addressing can be used. 26
3.3 Illustrates how the instruction `SET` can be used. 28
3.4 Illustrates how the instruction `COPY` can be used. 29
3.5 One cycle of self-replication. 34
4.1 Evolution in terms of average population merit as a function of mutation rate. 42
4.2 Average merit and genome length for different mutation rates. 42
4.3 Average merit as a function of population size. 44
4.4 Average abundance of evolved logical functions in 0-D and 2-D topologies. 45
4.5 Genotype diversity plotted against evolutionary progress for 2-D and 0-D topologies. 46
5.1 Self-replication and genetic recombination (self-replicating loops are striped). 50
5.2 Equivalence function possibly evolved through genetic recombination. . 51
5.3 Merit for a single run and average merit over all successful populations finally evolving EQU. 53
5.4 Merit for a single run and average merit over all successful populations finally evolving EQU with mutation or recombination. 55
5.5 Number of populations that evolved EQU for different rates of genetic recombination. 56
5.6 Average genotype length and genotype diversity with mutation only and with mutation and recombination. 56

6.1 Modified model with 2 layers. 60
6.2 Illustration of accumulative graphs. 62
6.3 Accumulative phenotype abundance and average genotype length, without resource relations or layers. 64
6.4 Accumulative phenotype abundance and average genotype length, with resource relations but without layers. 65
6.5 Accumulative phenotype abundance, without resource relations but with layers. 66
6.6 Average genotype length, without resource relations but with layers. . . 67
6.7 Accumulative phenotype abundance, with resource relations and layers. 68
6.8 Average genotype length, with resource relations and layers. 69
6.9 Low mutational pressure during evolution of two populations in symbiosis. 71
6.10 High mutational pressure during evolution of two populations in symbiosis. 72
7.1 Average genotype length with local offspring dispersal, evolved with different population sizes and merits 79
7.2 Average genotype length with non-local offspring dispersal, evolved with different population sizes and merits 80

Chapter 1

Introduction to Artificial Life

Artificial Life (AL) is an interdisciplinary research field that tries to abstract from life *as-we-know-it*, to find conditions for its origin and dynamics, and possibly enable instantiation of life *as-it-could-be* [Lan88]. Recently, the availability of powerful computers has made it possible to simulate artificial life in numerous different computational "artificial chemistries". Artificial Chemistry (AC) is an important subfield of Artificial Life, that tries to abstract from molecular processes to investigate the emergent dynamics of these complex systems [DZB01]. But the history of computational research in Artificial Life is much older. As early as in the 1940's, was von Neumann interested in the logic of self-replication, and constructed the first universal self-replicator by hand in a cellular automaton [vN66, Sip98]. His construction distinguished between "genotype" and "phenotype", which has been claimed to be the true contribution of von Neumann's work (about five years before DNA was actually discovered) [McM00b]. Today, diverse areas of Artificial Life, such as the origin of life, symbiosis, autopoiesis (self-production), self-organization and evolutionary novelty, are of particular interest. Artificial Life has branched over time, and we may today identify a number of related research fields: Artificial Chemistry (AC), Artificial Intelligence (AI), Genetic Algorithms (GA), Artificial Neural Networks (ANN) and Robotics.

Overview This chapter will first give an introduction to the relatively unknown and interdisciplinary field of Artificial Life and its history, and then outline this thesis. The first section will discuss Artificial Life as a research area, in particular problems associated with evolution and life. The second section will give a historical overview of

related research areas, and how they connect to this work. The third section will present the motivation, and the main thesis for this work. Finally, the last section will outline the structure of this thesis in a chapter-by-chapter overview.

1.1 Life is hard to study

Life is a hard subject to study, and proposed fundamental definitions, conditions and properties are still questioned. We only witness one instance of life; developed on the foundations of natural chemistry, and coded into carbon here on earth. Artificial Life is not restricted to one kind of chemistry, instead any (artificial) chemistry is considered. In contrast to other areas of science, some aspects of life does not seem to be decomposable, but *holistic*. Analytical approaches to investigate these aspects of life immediately fail; some of its properties come from its whole, not from its individual parts. Biological life is, for example, hierarchically ordered at least in five levels of interaction: molecular level, cellular level, organism level, population level and ecosystem level [TJ95]. Some definitions of life (see below) are also directly defined by global properties, e.g. Langton defines life as [Lan88]:

> [...] a property of the organization of the matter, rather than a property of the matter itself.

Life is also an emergent phenomenon, e.g. biological life has organized itself to very high complexity over the last 2 billion years [Ada98, DZB01, Sip98]. So, instead of analyzing life, Artificial Life tries to synthesize it. By synthesizing artificial life, researchers try to: understand the dynamics of evolution, find sufficient conditions for self-organization, study replication, find the origin of life, and more. Again, many problems in the study of Artificial Life are connected to the very notion of *life*, e.g. studying (self-)replication demands a sound definition of an offspring.

1.1.1 Definitions of life

A fundamental, but important problem is to define life. Since Artificial Life is looking at phenomena arising in "living matter", a definition of "living" is necessary. In biology, typically there are no problem in identifying life. But there are exceptions, e.g. scientists are still arguing whether viruses are alive [Rya02]. Viruses do not reproduce by

themselves, instead they replicate by inserting their genome into a replicating host cell. At the same time viruses do evolve, although through close co-evolution with its hosts. Several broad definitions of life have been proposed [Ada98]:

Physiological definitions focus on observable functions of an entity; such as moving, reacting to stimulation, and so on.

Metabolic definitions look at resource exchange between an entity and its environment.

Biochemical definitions center around the capability of storing hereditary information in nucleic acids.

Genetic definitions use the process of evolution (replication, variation and selection) as definition of life, but do not specify how hereditary material is encoded.

Thermodynamic definitions are very general, and focus on the ability of life to maintain low levels of entropy (disorder).

These definitions may be modified and/or combined to cover a desired definition of life. In this work we adopt the genetic definition above. This is a very pragmatic definition that includes entities which intuitively are not alive, e.g. genetic algorithm solutions, rumors or other spoken messages (see Section 2.2).

1.1.2 Weak or Strong Artificial Life

From a philosophical perspective, there are two schools of Artificial Life: Weak AL and Strong AL [Tay99]. Researchers belonging to the school of Weak AL, view life synthesized by artificial means as merely *simulations* or imitations of life. This is in contrast to the school of Strong AL, which claims that Artificial Life is (potentially) *realizations* of life, i.e. true instantiations of life created by artificial means. This work is in agreement of the latter view of Artificial Life, solely for the sake of argument (recall our definition of life from above). By defining life in terms of evolution, we could more precisely discuss about *Artificial Evolution* instead of Artificial Life.

1.1.3 Complexity of life

A puzzling, but intriguing aspect of life is its *complexity*. In biology, complexity of life may intuitively be defined by species diversity and behavior, where the latter may be

concrete-sized e.g. by the number of different organs. Darwin explained the great diversity of different phenotypes as a direct result of rich quantities of different ecological niches, which new species were adapted to by natural selection [Dar59, May02]. Gould has argued in neo-Darwinian spirit that geographical locality and gradual genetic change is the factor behind new species and novel features (phenotype) [Gou02]. Researchers have also emphasized the importance of genetic recombination and symbiosis as major factors in the evolution of complex life [Hux59, Rid00, Mar81, Rya02]. Among other hypotheses for the complexity of life, in particular for the evolution of sexual reproduction, is the *Red-Queen* hypothesis [vV73, Wil03]. Throughout this work, complexity will be defined in terms of diversity and average genotype size, in a finite population of self-replicating organisms with genotype/phenotype distinction; see Section 1.3.

1.2 History and background

Evolution and life have always amazed people. What is the origin of life? What was the first kind of life? What phases did evolution go through? How did such "perfect" organs as the eye evolve? How can such complicated organisms as mammals come about? Evolution has driven biological life to increasingly higher complexity, many times to extinction [May02]; continuously inventing new functions and mechanisms for reproduction and survival. At the same time it is important to understand that (neo-Darwinian) evolution is not *teleological*; evolution is not goal-oriented. Still, even though there is no mastermind behind, life—in the evolutionary theory—has created order where there was chaos [McM00a].

Viewing life as a mechanical process was an unorthodox thought in the mid of the nineteenth century. In France around 1748, Julien Offray de La Mettrice argued in "L'Homme Machine" against Descartes about the dualistic view of life, i.e. that life is the product of spirit and matter [Hel98]. Later, Darwin's work of evolution [Dar59] influenced Samuel Butler (1872) to argue for the possible evolution of machine-like life. In 1948, proposed Norbert Wiener, founder of cybernetics, that organisms and machines are essentially kindred entities of information processing devices. Around the same time started von Neumann to develop the first self-reproducing and universally computing cellular automaton, influenced by Turing's universal Automata Theory. This work by

von Neumann laid the foundation for *software oriented* or *computational* Artificial Life. A few years later (1956) was the field of Artificial Intelligence (AI) born. Around 20 years later (1987), the term "Artificial Life" was first used by Chris Langton, to denote his experiments of how computers can be used to model biological systems. Finally, at the MIT conference 1994, Artificial Life was established as a new research field by Langton, Ray and others. In the mission statement by Santa Fe Institute, the purpose of Artificial Life is summarized [Ins94, p.38]:

> Artificial Life ("AL" or "ALife") studies "natural" life by attempting to recreate biological phenomena from first principles within computers and other "artificial" media. ALife complements the analytic approach of traditional biology with a synthetic approach in which, rather than studying biological phenomena by taking apart living organisms to see how they work, researchers attempt to put together systems that behave like living organisms. Artificial Life amounts to the practice of "synthetic biology".

1.2.1 Cellular automata

Any software-oriented approach in Artificial Life is based on the pioneering work with the first universal self-replicator by John von Neumann [Ada98, vN66, McM00a]. Von Neumann's inspiration came from Turing's Automata Theory, but was extended into the logic of universal self-replication. Turing's Automata Theory defines a universal automaton as a finite state-space with finitely many transition rules, and a tape for Input/Output (I/O) operations. The states (actions) consist of reading from the tape, and moving and/or writing to it. This implies that the finite set of transition rules can be encoded, and written as a mapping between n-tuples of e.g. natural numbers.

Von Neumann partly constructed the universal self-reproducing automaton as a 2-dimensional (2-D) *cellular automata (CA)* with a five-cell neighborhood (Neumann neighborhood), and a cell space of 29 states. The state of each cell at time $t + 1$ is defined by the state of neighboring cells at time t under the transition rules. In total, his universal constructor has been approximated to consist of about $50,000$ to $200,000$ cells, and the number of transition rule is enormous [Sip98]. The purpose of von Neumann's automaton was to create a self-replicating entity, such that—given enough material—it can create an exact copy if itself. His construction separated the *description* (genotype)

from the actual machine (phenotype), and replication respected this distinction and reproduced both. His architecture was first formulated in 1948, about five years *before* DNA was discovered (1953) by Watson and Crick. Von Neumann also realized that such a device possibly could construct machines with higher "complexity", if given a description (genotype) of such a machine. Although he did not suggest from where such increasing complexity (in the description) would come from, or how complexity could be measured [McM00a, MTvK01, McM00b].

1.2.2 Assembler automata

Instead of using cellular automata to implement self-replicators, the so-called *assembler automata* were invented. Assembler automata, i.e. assembler programs, formulate a more powerful way to represent programs than cellular automata. But experiences from genetic programming and Artificial Life showed that ordinary assembler programs were too brittle to function under mutation. In steps, a new and more robust assembler took shape to combat the problems introduced with program perturbation.

- The error tolerance was increased; e.g. by treating an error state as void, i.e. the error state is ignored and execution is resumed, possibly with some side-effects.
- The instruction set was minimized in size, to make the instruction space smaller, and the executable behavior easier to evolve.
- The instruction parameters where made less sensitive to mutation, by introducing special symbolic arguments that indirectly manipulate data in registers or memory.
- Jumps where made less sensitive to variable program length, e.g. by using *patterns* to reference program locations (similar patterns in DNA for start/stop are encoded by so-called codons).

The idea of using assembler-like programs for evolution, came initially from a computer game called *Core War* [Dew84] in which hand-written computer programs compete in a limited cyclic memory space. Among the successful strategies in Core War, are self-replication together with different techniques for terminating the execution of competing programs. Today, this family of robust assembler automata are used to study different evolutionary aspects; such as adaptation, symbiosis and self-organization in finite populations of self-replicators. Note that the assembler automata are typically constructed

with an instruction set that facilitates self-copying. Below follows some milestones in the development of the assembler automata.

Coreworld

After the popularity of Core War (1990), the game was transformed into an evolutionary platform called *Coreworld* by Rasmussen et al. [RKFH90]. They kept the cyclic memory space (the 1-dimensional, or 1-D, "core"), and used a similar instruction set of 10 assembler-like instructions; each instruction with 2 parameters, and with a number of different addressing modes. Rasmussen *et al.* were interested in open-ended, and emergent evolution, including self-organization. By letting the `MOV` instruction mutate the copied instructions with a certain probability, they introduced perturbation into the system. Since the instruction set was designed for replication, and the population space (topology) was limited, Darwinian evolution inevitably took place. To control the number of possible instructions to execute for each organism-update, they used an energy parameter, set to either "desert" (few instruction executions) or "jungle" (many instruction executions). They also parameterized the locality of references, e.g. for jumps, to force local or global interactions between entities. In their experiments, they used a random core with or without seeding hand-written self-replicators.

For a number of different parameter settings, they started to investigate the emergent evolutionary dynamics of the core. Rasmussen *et al.* discovered complex emergent phenomena with the "jungle" setting. Typically, the core showed patterns of self-replicating structures that looped, but the fixed-points of the system were rather disappointing. Over time, either most executions died-out, or simple loops dominated. Instead of open-ended evolution with increasing complexity, Coreworld showed decreasing diversity and decreasing average genotype size (see Section 1.3). Essentially, what lacked was a better suited instruction set and a potential environment that the core could adapt to. But their approach was novel, and made way for an ongoing trend in Artificial Life.

Tierra

Thomas S. Ray, a current professor in zoology *and* computer science at University of Oklahoma, was inspired by Coreworld and decided to develop a similar, but improved evolutionary system called *Tierra*, for the emergence of self-organization [Ray91,

Ray95, Ray03]. Tierra also has a 1-D memory topology like Coreworld, where computer programs live, compete and die. A novel feature of the Tierra assembler, which the organisms consisted of, is pattern-based addressing (see page 26). In Tierra, the organisms replicate by self-copying, but are also able to address other genomes, for reading and/or execution. Ray perturbed the system with mutation, and observed how ecologies of "parasites", "hyper-parasites", "social hyper-parasites", "cheaters" and "symbionts" were discovered in his artificial medium. His purpose of finding the emergence of self-organization, and ecologies was in-a-way achieved. In some aspects Tierra did create emergent self-organization, but the implicit fitness landscape still did not offer much of evolutionary complexity to discover.

Avida

Avida [Ada98] was built on inspiration from the success with Tierra, but with some new design choices to further add evolutionary dynamics. Instead of letting the topology be in 1-dimension, Avida uses a 2-D grid on which 1 organism exists in each position (x, y). By using a grid, Avida exhibits locality, such that information is propagated by self-replication, geometrically in the topology. To avoid parasites, as in Tierra, Avida do not allow organisms to access each others genomes (by default). Instead, to add evolutionary dynamics to the system, an explicit merit function can be defined. The explicit merit function can be used to build complicated fitness landscapes, for evolution under mutation. In particular, the merit function was defined for addition and bitwise logical computation, to create "complex evolution" [AB94]. Avida has been used to show the dynamics of the quasi-species model and to give support for neo-Darwinian evolution [WWO$^+$01, LOPA03].

1.3 Evolutionary complexity

In the core of this work, is the model with which we simulate artificial life. This evolutionary model is based on the genotype/phenotype distinction, first implemented by artificial means by von Neumann, but here expressed in a version of the assembler automata presented above. We assume that the genotype can be generalized to a sequence of symbols, and the phenotype can be categorized to a set of "computations". The

evolutionary model is formally defined in Chapter 2, and implemented in Chapter 3; particular variations of the model follow in subsequent chapters.

Using this model, we will be concerned with evolutionary factors that create complexity in the framework of a finite population of self-replicating organisms, with a clear genotype/phenotype distinction. *The thesis is that mutual relations between organisms and the environment (biotic and abiotic) are a major source of evolutionary complexity.* The model will be enriched with evolutionary factors such as symbiosis and the Red-Queen hypothesis (for open-endedness) to study this mutual relation between organisms and the environment, for complexity in evolution.

Throughout this work, *evolutionary complexity* in a certain population will be defined as *the product of diversity and average genotype size.* Since we distinguish between genotype and phenotype, diversity can be measured with respect to both.

genotype diversity is simply the number of equivalence classes the syntactic equivalence relation $\doteq$ (see Section 2.3) defines in the population. This diversity measure ranges between 1 (all organisms in the population are syntactically identical) and s (all organisms in the population of size s are syntactically different).

phenotype diversity is the number of computational groups, e.g. if we consider the logical computation of the 9 logical functions on page 30, we may have this diversity measure ranging between 0 and 9.

Note that there is no *a priori* implication between genotype and phenotype; e.g. there may be many genotypes which all express the same phenotype, due to quasi-neutral instructions in our chemistry. Average genotype size is simply the average number of genotype symbols, for each organism in the population. Other more formal complexity measures are Shannon's information theory and Kolmogorov's complexity measure [Sta03], but they are not discussed here.

It is interesting to note that diversity, and average genotype size are in conflict, *unless* the environment forces them to coincide. Assume, for example, a self-replicating genotype that replicates with perturbation in a finite but void environment, i.e. additional behavior, except replication, is not selected for. If we increase the perturbation rate, (genotype) diversity increases, but the average genotype size decreases. On the contrary, if the perturbation rate is decreased, (genotype) diversity decreases, but the

size of the average genotype increases [WWO+01]; see also Section 4.2.1, and Table 4.2 in particular.

To summarize, the motivation for this thesis is to find factors that create the above notion of evolutionary complexity in a finite population, despite its natural tendency to be low. If we successfully identifies evolutionary properties that increase the above notion of complexity, by introducing a relation between fitness and environment, we could argue in favor of such a relation as an important factor for evolutionary complexity in (artificial) evolution.

1.4 Thesis outline

Below follows a chapter-by-chapter overview of this thesis. We first develop an evolutionary platform called SALTA, with inspiration from earlier work with assembler automata. Then, we conduct experiments with focus on conditions that create evolutionary complexity—as defined above.

- **Chapter 2** gives a formal definition of the evolutionary model called *chemostat*, presented after an example of non-biological evolution, and an informal discussion of biological evolution. The important concept of the merit function, and its descriptive fitness landscape are also presented.

- **Chapter 3** presents the micro and macro-level of the implementation of SALTA which is formally defined by the chemostat in Chapter 2. SALTA is developed for studying evolution of assembler automata, and is used in a number of experiments reported in this work, with the main focus on evolutionary complexity.

- **Chapter 4** will start the experiments of evolutionary complexity by looking at how point-mutation can evolve increasingly difficult logical functions, in a relatively smooth fitness landscape. In particular, the platform parameters: mutation rate, population size and offspring dispersal are investigated with their effect on evolution and evolutionary complexity.

- **Chapter 5** introduces genetic recombination as a powerful operator for genetic variability. Recombination is compared with point-mutation, for the evolution of

logical functions, called "complex features". Genetic recombination is also shown to create evolutionary complexity in the platform.

- **Chapter 6** modifies the default evolutionary model, by adding resource relations between sub-populations, to capture *biotic* interdependencies between organisms (ecologies). The focus is on sufficient conditions for symbiosis, and the emergence of co-evolution, but also what effect symbiosis has on evolutionary complexity.

- **Chapter 7** tries to approach the important relation between *abiotic* environment and the population, by introducing an abstract function that metabolizes resources from "food" to "waste". This relation will be shown to lead to an ever-changing fitness landscape which the population continuously has to adapt to, to stay fit (called the *Red-Queen hypothesis*), creating indefinite genotype growth (in theory). Unfortunately, genetic drift limits the evolution, and is subsequently investigated with respect to the platform parameters.

- **Chapter 8** ends by concluding the work with summary, contributions, discussion, critique and future work.

- **Appendix A** describes the platform settings, and the format of the initialization and the output files.

- **Appendix B** is a glossary over biological definitions, and how they are used in the context of this thesis.

Chapter 2

Evolution

Today, there is enough evidence to say that evolution is a *theory* of how our planet Earth has been shaped through time [May02]. We witness one type of evolution, with one instance of biological life as its product. All organisms (that we know of) have a clear genotype/phenotype distinction with genotype encoded in nucleic acids, and phenotype expressed in natural chemistry. Evolution is a continuous process over time, and even though its product—life—is complex in constitution and behavior, it has been claimed that its conditions are remarkably simple; as Daniel Dennett puts it [Pag02]:

> [...] evolution will occur whenever and wherever three conditions are met: replication, variation (mutation), and differential fitness (competition).

Even though we witness only one instance of biological evolution, there is nothing restraining us from creating abstract models from it. By doing so, we get a framework for evolution which can be instantiated to create any form of *artificial life*. In this thesis, we narrow the discussion of evolution by adopting the distinction between genotype and phenotype. Genotype, in this artificial setting, will denote inheritable information in an organism that is encoded as a sequence of symbols, and expressed as a phenotype.

Overview In this chapter, we will continue the introduction with describing the process of *evolution*. In the first section, evolution will be presented as a theory in biology and in the second section, as a formal framework in Artificial Life. In the third section, we illustrate non-biological evolution as an example. A particular abstract evolutionary platform called *chemostat* is presented in the fourth section, which will be used as our

artificial chemistry in the next chapter. Finally, the last section presents the useful notion of *fitness landscapes*, that will be used throughout this work to illustrate evolution as a fitness-optimizing process.

2.1 Biological evolution

Today, biological evolution is a theory about replication, genetic variation and natural selection (induced by competition) [May02].

Replication is the process in biology (used synonymously with reproduction) in which organisms create offspring to maintain its genetic continuation over generations [May02]. Most high-order organisms such as mammals, reproduce through sexual reproduction (meiosis), which has been argued to combat many genetic problems, and create high variability for evolutionary success [Rid00]. For low-order organisms such as prokaryotes, sexual reproduction is too difficult, and instead they replicate by binary fission or *cloning*. Cloning does not result in equally low error rate as sexual reproduction, but the typically small genotypes can still maintain themselves efficiently through the generations. Compared to sexual reproduction, cloning does not create equally high genetic variation, instead prokaryotes share genetic material through *lateral gene transfer* [MS02, MMP97].

Genetic variation is necessary in a population to adapt itself to changing environments. Genetic variation is typically achieved by mutation and/or recombination (sexual reproduction or lateral gene transfer) [May02], but the effective variation in a population is essentially a result of two forces: *natural selection* (see below) and *genetic drift* (caused by chance; see Section 2.5.1). A certain amount of genetic variation is necessary in a population, e.g. to avoid harmful genetic drift which may lead to lowered fitness.

The effect of mutation and recombination in populations can be seen as a continuously changing pool of genetic material, from which organisms may develop phenotypes to stay adapted. The pressure to stay adapted in a continuously changing environment, has often been described as the "Red-Queen" hypothesis [vV73]. This analogy between the Red Queen and continuous adaptation to changing environment, is based on Lewis Carroll's story, "Through the Looking Glass"—in which the Queen has to run

continuously, just to remain where she is.

Natural selection can be viewed as an elimination process, in which those organisms in a population with less adapted phenotypes are eliminated over generations [May02]. Disappearance of organisms over generations with less adapted phenotypes, is a direct result of continuously (in each generation) fitting an excessive population to its limited habitat (topology). This selection process was first stated by Charles Darwin in his "The Origin of Species" [Dar59], and was based on the observation that the species typically produce more offspring than can survive. Since the fitter offspring survive to reproduce, these "stronger" genes are retained and inherited by future generations. While genes of less desirable traits become eliminated. It is important to note that selection acts on the phenotype, encoded-for by the genotype, in reaction with the environment. Hence, evolution acts implicitly on the genotype by limiting its transfer over generations via the phenotype.

If we look at the genetic diversity in a population, replication is a force that decreases diversity; in contrast to variation that do the opposite, whereas natural selection may do both [ES79, May02, Rid00]. If we assume differential fitness, replication decreases diversity since the most adapted organism, with the highest fecundity, increases in concentration over time. In contrast, variation increases diversity by directly modifying single genotypes, to newer versions in the population, over the generations. Natural selection may do both. Given a single ecological niche, selection will force all organisms in the population to adapt to that niche. But, given potentially infinity number of connected niches, variation will branch the population into the different niches to induce diversity.

2.2 An example of evolution

We can use the evolution of a spoken message to illustrate replication, variation and selection, by an example from [Rid00]. Consider a lethal version of the whispering game, where a spoken message is transferred between children. Assume that the message is "IF A TIGER COMES, FREEZE", and that the children are in a group, in a jungle full of tigers. Picture that the leader starts spreading this message in a situation of approaching tigers, which becomes repeated between the children (replication). Unavoidably,

the message becomes scrambled during communication between the children. Instead of receiving the original message, versions like: "IF A TIGER COMES, SQUEEZE" or "IF A TIGER COMES, SNEEZE", appear in the branching of communication (variation). Those children who get the scrambled message are likely to be eaten by the hypothetical tigers in this scenario, and that branch of information is broken (selection). This example shows, in addition to evolution, how information is correlated to the environment, and how low correlation results in elimination. This connection between evolution and information theory can be understood be viewing evolution as an information-gathering process, where information about how to best survive in a particular environment is gradually transfered from that particular environment to the genotype [Ada98].

2.3 Artificial evolution

In this work we will mainly look at the evolution of assembler automata. In the following definitions of replication, variation and natural selection, derived from a generalization of above biological ones, we restrict ourselves to the domain of assembler automata $\mathbb{A}$, i.e. sequences of assembler instructions. We define the syntactic equivalence relation ($\doteq$) over $\mathbb{A}$, such that for two programs $p_1, p_2 \in \mathbb{A}$, $p_1 \doteq p_2$ holds, if and only if p_1 and p_2 are equally long sequences of the same assembler instructions; in the same order.

2.3.1 Replication

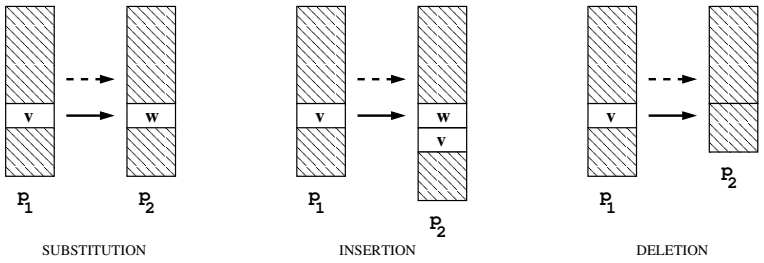
Replication is the function of one parent $p \in \mathbb{A}$, to produce one offspring $c \in \mathbb{A}$. Without any perturbation during replication, c is a *clone* of p, that is $c \doteq p$. We define replication as a function from parent to offspring, $r : \mathbb{A} \to \mathbb{A}$. Assuming no perturbation we have

$$r(p) = c, \ \textit{where } p \doteq c \tag{2.1}$$

2.3.2 Variation

Variation is the syntactic difference between any two programs $p, c \in \mathbb{A}$, induced by mutation or genetic recombination. Mutation is introduced by perturbing replication by point-mutations: insertion, deletion and substitution (see Figure 2.1). Genetic recombination may merge or split neighboring genomes [MMP97, KS88, KPP94] (see Section 5).

Figure 2.1 Point-mutations: substitution, insertion and deletion.



The figure shows 3 types of implemented point-mutation, that perturb the transfer of genotype during self-replication.

Variation $\simeq$ between two programs $p_1, p_2 \in \mathbb{A}$ through single step point-mutation (substitution, insertion and deletion), can be defined as follows.

Definition 1 ***Substitution*** *($\simeq_S$):*

$$p_1 \simeq_S p_2 \quad \Leftrightarrow \quad \|p_1\| = \|p_2\| = n \wedge \\ \exists i : \forall j \neq i : (p_1^i \neq p_2^i \wedge p_1^j = p_2^j) \tag{2.2}$$

Where p^i is the i'th instruction in the assembler program p with length $\|p\|$ i.e. number of instructions n.

Definition 2 ***Insertion*** *($\simeq_I$):*

$$p_1 \simeq_I p_2 \quad \Leftrightarrow \quad \|p_1\| = \|p_2\| - 1 = n \wedge \\ \exists ij : (hd_i(p_1) \doteq hd_i(p_2) \wedge tl_j(p_1) \doteq tl_j(p_2) \wedge i + j = n) \tag{2.3}$$

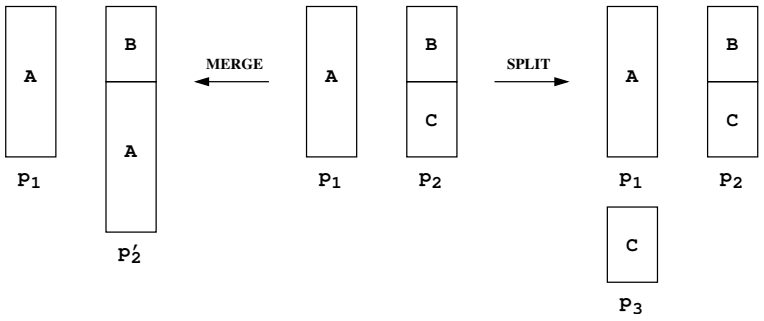
Where hd_i and tl_j are the functions head and tail of length i and j, respectively.

Definition 3 ***Deletion*** *($\simeq_D$):*

$$p_1 \simeq_D p_2 \quad \Leftrightarrow \quad \|p_1\| = \|p_2\| + 1 = n \wedge \\ \exists ij : (hd_i(p_1) \doteq hd_i(p_2) \wedge tl_j(p_1) \doteq tl_j(p_2) \wedge i + j = n - 1) \tag{2.4}$$

Variation through genetic recombination (merge and split) can be defined as a relation between three programs $p_1, p_2, p_3 \in \mathbb{A}$ as follows.

Figure 2.2 Genetic recombination: merge and split.



Illustrates 2 possibilities of recombination; merge and split.

Definition 4 ***Merge*** *($\simeq_M$):*

$$(p_1, p_2) \simeq_M p_3 \quad \Leftrightarrow \quad \exists ij : (hd_i(p_2) \doteq hd_i(p_3) \wedge tl_j(p_3) \doteq p_1 \wedge i + j = \|p_3\|) \tag{2.5}$$

Definition 5 ***Split*** *($\simeq_{Sp}$):*

$$(p_1, p_2) \simeq_{Sp} p_3 \quad \Leftrightarrow \quad \exists i : tl_i(p_2) \doteq p_3 \tag{2.6}$$

See also Figure 2.2, where $p_3 = p_2'$ for the merge case.

2.3.3 Selection

Natural selection can be defined as the elimination function (S) of fitting an excessive population of organisms (P) to a finite topology (T) holding n organisms.

Definition 6 ***Selection*** *($S : \mathbb{A}^{\mathbb{N}} \to \mathbb{A}^{\mathbb{N}}$):*

$$S(P) = P', \ \textit{where} \ P' \subseteq P \wedge \|P'\| = n \wedge \forall p \in P \setminus P' : \forall p' \in P' : f(p) \leq f(p') \tag{2.7}$$

Where $f : \mathbb{A} \to \mathbb{N}$ is a well-defined *merit function* that maps organisms to some merit in $\mathbb{N}$ which gives rise to a measure of fitness (see Section 2.5).

2.4 Chemostat

Evolution in a limited population of organisms can be studied efficiently using a so called "chemostat" (or reactor vessel) in microbiology [SW95], and recently in Artificial Life as well [FB94, FWB94, TJ95, YB01]. We may define a chemostat, in general terms,

as a finite topology holding a limited ensemble of reacting organisms which ultimately compete for the limited amount of space, and

(i) entities react either in a well defined neighborhood (e.g. n-Moore neighborhood in 2-D; see below) or without space constraints (mass reaction).

(ii) since replication makes the ensemble grow, a dilution factor (death) constrains the chemostat by flushing out entities in proportion to the replication rate.

(iii) due to the presence of dilution, entities must find some pathways for their genotype to be maintained in the system.

n-Moore neighborhood in 2-D is the set of $(2n+1)^2$ surrounding squares at range n, e.g. at range 1 there are 9 neighboring squares (including the center square). There is also a n-von Neumann neighborhood, which is the set of $2n(n+1)+1$ surrounding squares in a diamond-shape, e.g. at range 1 there are 5 neighboring squares (including the center square). As topology, a toroidal[1] 2-D-space is used in our chemostat.

This general chemostat-model can be instantiated with replication (r), variation ($\simeq$) and selection (S) over the domain of assembler automata ($\mathbb{A}$) as follows. Let $T[i,j]$ be a 2-D array of size s^2, where each position (i,j) in T (we impose the Euclidean space $\mathbf{R}^2$ on T) holds an organism $a \in \mathbb{A}$ (denoted a_{ij}). We define a fair and random scheduler α over T as follows. Let $r^{\simeq^R}(p)$ be the replication function r with variation $\simeq^R$ and fidelity $F \in [0,1]$ ($F = (1-R)^l$, where l is the genome length of p in number of words, and R is error rate per replicated instruction/word):

$$r^{\simeq^R}(p) = \begin{cases} p, & \textit{with probability } F \\ \textit{some } c \textit{ such that } p \simeq^R c, & \textit{with probability } (1-F) \end{cases}, \tag{2.8}$$

where $p, c \in \mathbb{A}$. Then the relation $\sim_{ij}$ between the current state of T, and the following state T' after replication of organism a_{ij}, can be formulated as

$$\begin{aligned} T \rightsquigarrow_{ij} T' \quad &\Leftrightarrow \quad \forall (m,n) \in M_{ij}(S(a_{ij})) : T'[m,n] = r^{\simeq^R}(a_{ij}) \\ &\wedge \quad \forall x,y \in \{1,\ldots,s\} : (x,y) \notin M_{ij}(S(a_{ij})) \Rightarrow T[x,y] = T'[x,y] \end{aligned} \tag{2.9}$$

[1] A torus can be visualized as a rectangle with end edges put together to form a donut.

Figure 2.3 Fitness landscape with peaks and valleys.

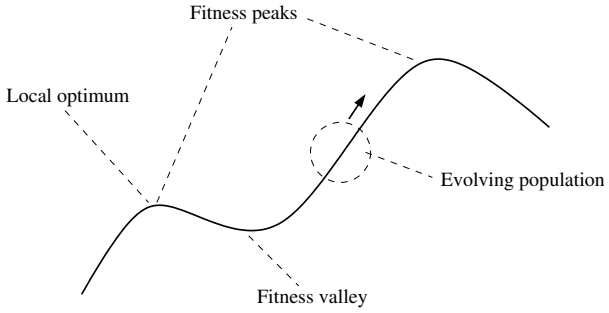


Illustration of 2 fitness peaks, and an evolving population that traverses the landscape.

where $S(a_{ij}) \in \mathbb{N}$ is the merit function for the organism a_{ij} and $M_{ij}(e)$ is the set of e positions (m, n) in some order in the 1-Moore neighborhood around the position (i, j) (excluding the parent position (i, j)). Then the scheduler α is the iterated application of $\rightsquigarrow_{ij}$ over all positions (i, j) in T:

$$T \rightsquigarrow_{\alpha} T' : \forall i, j \in \{1, \ldots, s\} : T \rightsquigarrow_{ij} T'. \tag{2.10}$$

Let $CHEM(r, \simeq, R, S, \alpha, s, T)$ be such a chemostat.

2.5 Merit function and fitness landscapes

Fitness is the success of an organism, or a particular species, to produce offspring, hence it is a *measure* of evolutionary success. Note that fitness will be used as a descriptive measure, secondary to merit. In concrete terms, we may define a merit function as a mapping from genotypes to merit, by evaluating the corresponding phenotypes in given environment. This mapping can be visualized as a descriptive fitness landscape with peaks and valleys [CAW01, LW04, WP02], representing fitness optimum and minimum (see Figure 2.3). Evolving populations climb peaks in the fitness landscape, sometimes crossing valleys (saddles), to maximize merit (and fitness)—globally or locally. The granularity of the climb is the amount of genetic variation in each generation of the population. In the most fine-grained case, the landscape is traversed by one-step point-mutations. In the more coarse-grained case, the landscape is traversed by recombination. Beneficial genetic variation moves the population upwards in the landscape, while deleterious mutations (variation) move the population downwards.

This view of evolution as a fitness climb turns it into an optimization problem over the merit function. Populations try to reach higher peaks, but may get stuck in local optima by blocking fitness valleys. The larger the population, the more solutions are simultaneously represented to lower the chance of getting stuck in particular sub-optimal solutions. Even though a fitness peak has been reached, deleterious mutations may accumulate to eventually drive the population away to lower fitness (genetic drift, see below). It may also be the case that the mutation rate is too high to climb a very sharp fitness peak, or that the mutation rate is too low for bridging fitness valleys in the landscape.

In biology, fitness landscapes are not smooth, or fixed, but are continuously changing. Firstly, epistasis cause non-linear interactions between genes, creating a very rugged landscape such that small one-step mutations may totally displace the phenotype. Secondly, since fitness is a product of both phenotype *and* environment, environmental changes related to the evolving phenotypes modifies the very fitness landscape during traversal. Hence, there is a cyclic dependency between environment, fitness and evolving phenotypes, creating dynamic and complex fitness landscapes. Note that the notion of environment includes both abiotic and biotic entities. Thirdly, the abiotic environment may drastically change due to external factors, e.g. through changing temperature (ice age, volcanic activity), changing chemical conditions (gases, catalysts), or topological changes (continental drift, erosion). In this work we will look at dynamic fitness landscapes due to biotic (symbiosis) and abiotic dependencies (Red-Queen hypothesis). We will also look at both gradual fitness climbs through point-mutation, and more dramatic climbs through genetic recombination.

2.5.1 Genetic drift

Genetic drift is an opposite force to natural selection that may cause random genetic changes in a finite population. Although, genetic drift exists in any finite population, it may become the major driving force in modifying the allele frequency in smaller populations. In contrast to natural selection, genetic drift is a random force that is based on differential reproductive success (like natural selection), but where the reproductive success in some way is unrelated to phenotype (and genotype). This may be the case, from one generation to the next if random selection of the reproductive entities is not evenly

distributed in the population. The effect becomes especially significant in small populations when unevenly distributed samples accumulate over many generations. Imagine a coin that is being tossed 10 times. For this small number of tosses, it is not unlikely to get 7 heads in a row (chance of $1/2^7 = 1/128$). But if we would toss the coin $1,000$ times instead, the chance of getting 700 heads in a row is extremely small ($1/2^{700}$). Hence, a small number of events has a higher chance of an unevenly distributed outcome than a large.

2.5.2 Müller's ratchet

Since genetic drift is random and irrespective of natural selection, evolving populations may be dislocated in their fitness landscapes. In particular, genetic drift may force a population of organisms away from an optimal fitness peak. Such harmful genetic drift may occur when deleterious mutations accumulate at a higher rate than natural selection can remove them. Since lowered fitness increases the genetic drift, this may lead to *genetic melt-down* [Rid00], i.e. population extinction—a hypothesis called Müllers ratchet [Mul64]. This happens when populations are driven into a negative spiral of genetic drift, and reduced fitness (inducing each other) [Mul64]. Note that the effect of Müllers ratchet is high in small and asexual populations under high mutation rate, and without any form of recombination. Larger and sexually reproducing populations can accommodate more variety to better combat drift by chance, and cancel-out deleterious mutations by meiosis. It has also been argued that small population sizes, e.g. as a result of geographic dislocation, or swift environmental changes (inducing decreased fitness and population size), is the major source of *inter-population variation*, and speciation by genetic drift [Gou02].

Chapter 3

A New Artificial Chemistry—SALTA

This chapter describes an evolutionary platform called *SALTA*, that we developed with inspiration from earlier work with assembler automata, e.g. Tierra and Avida [Ray91, Ada98], to simulate artificial organisms that consists of a virtual CPU and a mutating assembler program (genome). A population of these organisms is made to evolve to a given fitness landscape, via a user-defined merit function, similar to the platform Avida. In line with von Neumann's work (see Section 1.2.1), the platform respects the distinction between genotype and phenotype. SALTA corresponds on the macro-level to the earlier defined chemostat ($CHEM(r, \simeq, R, S, \alpha, s, T)$, see Section 2.4), which has been widely used in "wetware", for experiments using microorganisms (e.g. prokaryotes) [YB01].

We will use this artificial chemostat SALTA, and appropriate variations of it, to effectively conduct a number of experiments with the motivation to study properties that create complexity in evolution (see Section 1.3). The default model of SALTA, presented in this chapter, is subsequently extended in Chapter 5, Chapter 6 and Chapter 7 with both symbiosis, and open-ended evolution to find complexity in the finite population. In addition to mutation, we have also implemented genetic recombination (see Chapter 5) to compare both types of variation (mutation and recombination), for evolution in complicated fitness landscapes. The major limitation for this model is the explicitly defined fitness landscape which implicitly characterizes the environment, although this approach will be slightly modified in Chapter 7.

SALTA has been implemented such that the platform can easily be extended with

different merit functions, topologies and resources to study different evolutionary aspects of asexual organisms in the chemostat. The assembler can easily be extended, and modified to suit different phenotypes. From an implementation perspective, the evolutionary platform can be divided into micro and macro-level architecture (presented below) to ease further modifications and extensions.

Overview Our artificial chemistry can be sub-divided into micro and macro-level. The micro-level covers the composition of the organisms and their behavior (complex computation and self-replication/recombination), and is presented in the first section. At the macro-level, populations can be observed to compete and gradually adapt to the environment (fitness landscape), and is explained in the second section.

3.1 Micro-level: Virtual CPU and program

Each organism consists of a *virtual CPU* (Central Processing Unit) on which execution of its *assembler program* (genome) takes place. State changes in the CPU correspond loosely to state changes in chemical/biological systems. We will use state changes to represent behavior, i.e. *phenotype*. The virtual CPU and the instruction set have been made as simple as possible for clarity, but also powerful enough for self-replication and arithmetic/logic computation. Along with the CPU and the program is also a set of flags and variables, necessary for simulation of each organism.

3.1.1 CPU

The virtual CPU consists of three basic components: general purpose registers, pointers (such as instruction pointer, read pointer and write pointer), and the I/O (input/output) Buffer.

General purpose registers

There are three general purpose registers (R_0, R_1 and R_2) mainly for mathematical (arithmetic and logic) computation, but also for conditionals. Each register can hold a 32-bit word representing either a number, a logic bit-string, or a program word (instruction and argument(s)). Before any of the registers can be used as argument for an instruction, they have to be initialized by executing appropriate instructions.

Pointers

Addressing the genome (e.g. with `SET`, see page 27), is done through any of the three pointers; instruction pointer (`IP`), read pointer (`RD`) and write pointer (`WR`). The read and write pointers do not differ in semantics, but simply have different names out of convenience. This is not the case for the instruction pointer which slightly differs in semantics. By default, the instruction pointer cannot be set directly with `SET`, instead the instruction pointer is updated in the execution cycle to the next program word, or it may reference patterns in the genome by the `JUMPNEQ` instruction. The read and write pointer have to be initialized before they can be used, but `IP` is considered initialized at all times. Note that addressing is pattern-based (see page 26).

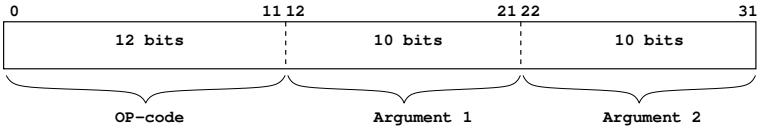
I/O Buffer

I/O operations are handled by the so-called I/O Buffer. There is 1 autonomous I/O Buffer for each organism. It is from this data structure that values are read from, and computed values are written to, by the `INPUT` and `OUTPUT` instructions, respectively (see page 29). For each new *genome execution* (see page 25), the I/O Buffer is typically initialized to a few new values (default 2), and newly computed values are added to it during execution. Thus, reading from the I/O Buffer may return new values (initialization values), or previously computed values (if any). The `INPUT` instruction uses a hidden second argument, for indexing the I/O Buffer. This second argument gives deterministic input values for identical genome executions, but possibly different input values for different genome executions. We may view the I/O Buffer as a set that is initialized to a few values for each new genome execution, and where new values are added (if unique) or read (non-destructively) with an index.

3.1.2 Program

Each organism executes its own local assembler program, represented as a cyclic string of program words, coding both instruction(s) and argument(s). It is cyclic in the sense that executing, or reading from the last instruction is continued with the first instruction, in a loop-back fashion. Writing *past* the last instruction does not cause a wrap to the beginning, instead copied program words are added to the end of the program to make it longer. Note that setting the pointer to the end of the program, and then writing to it,

Figure 3.1 Program word.



The illustration shows the bit-layout of the program word, that codes both for the instruction and the arguments.

does not add instructions. Instead a new empty program is automatically allocated as offspring (see Section 3.2.1), and the write pointer is set to its beginning. Program words are represented as strings of 32 bits. The first 12 bits code for instructions (via an OP-code), and the second and third sequence of 10 bits each code for argument(s); giving a maximum of $4,096$ possible instructions, and an argument range of $1,024$ possible values (see Figure 3.1). An instruction may have 1 or 2 arguments, typed by the instruction as either (i) register, (ii) pointer or (iii) pattern. Note that there is *epistasis* between program words in the program (genome). In particular, there are data and control dependencies between the assembler statements.

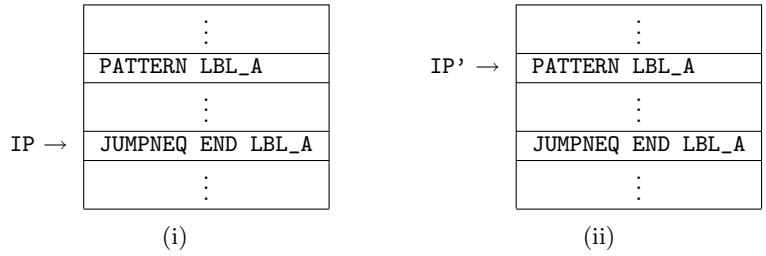
Execution

Programs are executed by continuous interpretation of each program word in sequence (except for jumps), until the last instruction loops back to the first, and completing one *genome execution*. For each new genome execution, the virtual CPU is cleared such that registers, read and write pointers are uninitialized, and the I/O Buffer is re-initialized. If any of an instruction's arguments is found uninitialized during execution it is deemed as failed, i.e. that instruction is simply skipped (void execution). For a few instructions, such as `SET` and `COPY` (see page 27), the whole genome execution may be terminated prematurely in case of an addressing error. Interpretation of each program word is done in a *fetch-decode-execute* cycle as follows.

(i) **Fetch:** Mask out the instruction word (12 bits) and argument words ($2 * 10$ bits) from current program word referenced by `IP`.

(ii) **Decode:** Given the instruction and argument words, look up corresponding instruction and arguments in a dynamic *mapping table*. The arguments are typed with respect to the particular instruction.

Figure 3.2 Illustrates how pattern-based addressing can be used.



Illustrates how pattern-based addressing can be used. The first pattern instruction (`PATTERN LBL_A`) identifies a program location (case (i)), that can be jumped to (case (ii)) by the conditional jump instruction `JUMPNEQ END LBL_A`, that jumps backwards to the pattern `LBL_A` if not register R_0 holds the pattern instruction `PATTERN END`

(iii) **Execute:** The decoded instruction is interpreted with its arguments. Note that argument values are retrieved from registers, except for patterns which are found in the program. Execution can become void, or terminated prematurely when given invalid arguments. If an instruction is successfully executed, any result is written to the register, or to the program as a program word. `IP` is always updated to the next program word.

Pattern-based addressing

Instead of referencing program locations with an absolute or relative address, pattern-based addressing, similar to the infamous `goto 'label'`, is used [Ray91]. A particular advantage with pattern-based addressing in evolving programs is its robustness under mutation—changes in the program size by insertions or deletions does not affect the addresses. An obvious disadvantage with pattern-based addressing is that for each reference to a particular pattern, a complete search of the program for that pattern may be necessary. For an illustration of patterns and jumps, see Example 3.2

Instruction set

An organism's behavior (phenotype) is represented in the execution of its program (genotype). Similar instruction sets have previously been designed for the study of artificial life [RKFH90, Ray91, Ada98], typically having support for self-replication and some type of computation. The following instruction set is made as small as possible, but it is also designed to be as robust as possible under genetic variation.

- Pattern-based addressing is used to improve program referencing under muta-

tion [Ray91].

- There are no special instructions for replication such as offspring allocation and parent-offspring division. Instead, allocation and division is automatically handled by the system.

- Each instruction has argument(s) encoded in its representing program word, and are dynamically typed by the instruction type (see below).

- Quasi-neutral instructions (e.g. no-ops) are represented by pattern instructions, and are important for evolutionary plasticity [YB01]. Particular patterns are given as user-defined arguments (see below).

Based on usage, the instruction set can be divided into two: necessary *instructions for replication*, and auxiliary *instructions for computation*. In contrast to other similar chemistries [RKFH90, Ray91, Ada98], this chemistry does not have special instructions for replication, such as allocation of new space or offspring-parent division. Instead, this is automatically handled by the system to keep the instruction set to a minimum. Below follows a description of each implemented instruction, for (i) self-replication, (ii) logic computation and (iii) arithmetic computation.

(i) Self-replication The following 4 instructions are necessary for self-replication.

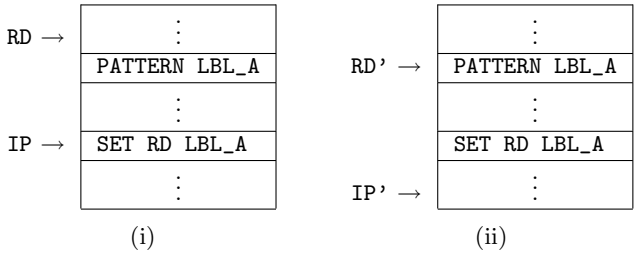
- [`PATTERN` *patt*]

  Encodes a virtual program address *patt* which can be used e.g. as argument for jumps or pointer references (`SET`). Pattern-based addressing is very robust under variation, such as mutation, since changes in the program size does not affect the address of a particular pattern. A negative aspect of patterns is that, since the programs are interpreted, a reference may result in a complete program search. Pattern instructions are also called "quasi-neutral" [YB01] since their execution does not change the state of the virtual CPU, and are similar to `no-ops` in other assembler automata [Ada98]. The pattern instruction never fails to execute. See also Example 3.2.

- [`SET` *ptr patt*]

  Sets the pointer *ptr* to where the first pattern instruction with argument *patt* can

Figure 3.3 Illustrates how the instruction `SET` can be used.



Illustrates how the instruction `SET` can be used. The pattern instruction `PATTERN LBL_A` identifies a program location (case (i)). At execution of the instruction `SET`, the given pointer (here the read pointer `RD`) is set to the first occurrence of the given pattern (here `LBL_A`) found searching forward (if any), wrapping from end to beginning (case (ii)).

be found; searching forward from current instruction pointer (`IP`), and wrapping around from end to beginning. Typical usage is to set the read pointer to the beginning of the program, and the write pointer to the end of the program, to initiate self-replication with subsequent copy instructions. For genetic recombination, it is possible for the `SET` instruction to reference a neighboring program in the pattern search. The instruction fails to execute if the pattern could not be found in the referenced program. However, the instruction does not immediately fail when searching without success in a neighboring program; instead the search is shifted to the executing program. See also Example 3.3.

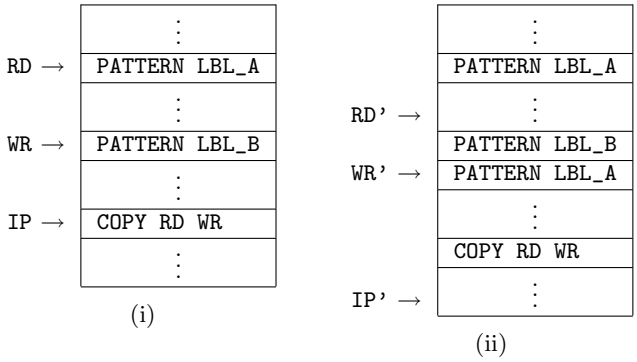
- [`COPY` ptr_R ptr_W]

  Copies one program word from pointer ptr_R to pointer ptr_W, and updates both pointers. To avoid over-writing patterns that are referenced by the `SET` instruction, the word is first read from ptr_R and thereafter is ptr_R updated to point at the next word, while ptr_W is first updated to the next word and then written to (e.g. see self-replication in Figure 3.5). As a side effect the copied program word is stored in register R_0. If the write pointer initially references the end of the program, a new empty program is allocated (offspring born), and the write pointer is set to its beginning. The instruction fails if either the read pointer ptr_R or the write pointer ptr_W has not been initialized (by `SET`). If the arguments are identical (same pointer), or either of them is `IP`, the instruction is simply skipped. See also Example 3.4.

- [`JUMPNEQ` $patt_C$ $patt_J$]

  Jumps by setting the instruction pointer (`IP`) to the first occurrence of pattern

Figure 3.4 Illustrates how the instruction `COPY` can be used.



Illustrates how the instruction `COPY` can be used. The pattern instructions `PATTERN LBL_A` and `PATTERN LBL_B` identify two program locations, which are referenced by the read pointer (`RD`) and the write pointer (`WR`), respectively (case (i)). At execution of the instruction `COPY`, the read pointer (`RD`) is read from and then updated, while the write pointer (`WR`) is first updated and then written to (case (ii)).

$patt_J$ found backwards from current `IP`, only if the pattern instruction with argument $patt_C$ is not in register R_0. The search loops back to the end of the genome if the program start is reached. The instruction fails if register R_0 has not been initialized, or the pattern $patt_J$ could not be found. See also Example 3.2.

(ii) Logic For logical computation, the following instructions have been implemented.

- [`INPUT` *reg*]

  Reads a value from the I/O Buffer, and stores it in register *reg*. For deterministic execution, the same value is read from the I/O Buffer for the same `INPUT` instruction and identical genome execution history. The particular value from the I/O Buffer is chosen by a hidden and fixed second argument, taken modulo the size of the I/O Buffer. Note that the hidden argument and the size of the I/O Buffer may vary under mutation. The instruction can never fail assuming the I/O Buffer is non-empty at all times (which it is since values are non-destructively read, and the buffer is initialized to some value(s)).

- [`OUTPUT` *reg*]

  Adds the value in register *reg* to the I/O Buffer (if unique). The instruction fails if register *reg* has not been initialized. If a unique value is added, the I/O Buffer increases in size by 1, but is cleared and re-initialized after the whole genome has executed.

- [`SWAP` reg_A reg_B]

  Swaps the values in registers reg_A and reg_B. The instruction fails if either register has not been initialized.

- [`NAND` reg_A reg_B]

  Computes bitwise logical NAND of registers reg_A and reg_B, and stores the result in register reg_A. The instruction fails if either register has not been initialized. Note that reg_A `NAND` $reg_B \equiv \neg(reg_A \wedge reg_B)$, where $\wedge$ and $\neg$ is bitwise conjunction and negation, respectively.

(iii) Arithmetic To extend computation with arithmetic, the following instructions have been implemented (`INC` is e.g. used in Chapter 7).

- [`INC` reg]

  Increases the value in register reg by 1 modulo program word size. The instruction fails if register reg has not been initialized.

- [`DEC` reg]

  Decreases the value in register reg by 1 modulo program word size. The instruction fails if register reg has not been initialized.

- [`ADD` reg_A reg_B]

  Adds together the values in register reg_A and reg_B (modulo program word size), and stores the result in register reg_A. The instruction fails if either register has not been initialized.

- [`SUB` reg_A reg_B]

  Subtracts the value in register reg_B from the value in register reg_A (modulo program word size), and stores the result in register reg_A (i.e. $reg'_A \leftarrow (reg_A - reg_B)$). The instruction fails if either register has not been initialized.

Self-replication and Computation

Basically, organisms have two types of behavior: (i) self-replication is necessary to maintain the genotype in a competitive evolutionary system, (ii) computation can be subdivided into arithmetic and bitwise logic computation, and is necessary for earning merit to increase in fitness (see Section 3.2.3).

Table 3.1 Default self-replicating organism.

Program Word	Semantics
`PATTERN START`	Pattern instruction for start of genome.
⋮	Computational instructions.
`PATTERN END`	Pattern instruction for start of replicating gene.
`SET RD START`	Set read pointer to pattern `START` (search looping back to start).
`SET WR END`	Set write pointer to pattern `END`.
`PATTERN LOOP`	Pattern instruction for the loop.
`COPY RD WR`	Copy word from read pointer to write pointer and put copied word in register R_0, update pointers.
`JUMPNEQ LEND LOOP`	Jump backwards to pattern `LOOP` if not R_0 contains instruction `PATTERN` with argument `LEND`.
`COPY RD WR`	Copy last word.
`PATTERN LEND`	Pattern instruction for ending loop.
`PATTERN END`	Pattern instruction for end of genome.

The table shows the default self-replicating organism together with semantics for each instruction.

Table 3.2 Experimentally smallest self-replicating organism.

`PATTERN START`
`SET RD START`
`SET WR LEND`
`PATTERN LOOP`
`COPY RD WR`
`JUMPNEQ LEND LOOP`
`PATTERN LEND`

The table shows the experimentally smallest self-replicating organisms, that the system evolved by seeding it with the default self-replicator.

(i) Self-replication The default self-replicator in Table 3.1 self-replicates through a self-copying loop (see Section 3.2.1). It is not the shortest self-replicator, e.g. the second instruction `PATTERN END` is not necessary for successful self-replication, but used in the case of genetic recombination. Table 3.2 shows the experimentally smallest self-replicator (see Section 4.1.2).

(ii) Computation Computation of a function $f(i_0, ..., i_n)$ over input values $i_0, ..., i_n$ (read with `INPUT`), is achieved when there is an output t (written by `OUTPUT`) in a genome execution such that $t = f(i_0, ..., i_n)$. Consider, for example, the following 9 (bitwise) logical functions which can all be computed with the universal logical function `NAND` [LOPA03]: `NOT`, `NAND`, `AND`, `OR_N`, `OR`, `AND_N`, `NOR`, `XOR` and `EQU`. Assume that the

organism has taken 2 inputs and made 1 output as follows:

$$\begin{aligned} i_0 &= \langle 00000000\ 10101010\ 00001111\ 00000000 \rangle \\ i_1 &= \langle 00000000\ 00000000\ 10101010\ 11110000 \rangle \\ t &= \langle 00000000\ 10101010\ 10101111\ 11110000 \rangle \end{aligned}$$

Then the organism has computed the bitwise `OR`-function. Any of the 9 functions above can be computed as a sequence of `INPUT`, `OUTPUT` and the universal logical function `NAND` (as above); e.g. the `OR`-function can be computed with three `NAND`'s:

$$\begin{aligned} x \texttt{ OR } y &\equiv (x \texttt{ NAND } x) \texttt{ NAND } (y \texttt{ NAND } y) \\ &= \neg(\neg(x \wedge x) \wedge \neg(y \wedge y)) \\ &= (x \wedge x) \vee (y \wedge y) \\ &= x \vee y \end{aligned}$$

An example of a program that can compute the logical function `OR` is given in Table 3.3. Note that there is a small possibility that the program will not compute the `OR`-function if the two `INPUT` functions read the same value from the I/O Buffer (the same value is read if the hidden second argument modulo the I/O Buffer size is the same; see the `INPUT` instruction on page 29). The computation of a binary function with identical arguments is avoided in the merit function. Without this constraint, there would be no computational difference between the echo-function of a value x, and x `OR` x.

3.2 Macro-level: Dynamics

Any evolutionary system is based on a notion of *fitness*, i.e. differential reproductive success over generations. By viewing the set of organisms at the macro-level, we can implement fitness by *differential execution speed*. Based on an organism's merit m (see Section 3.2.3), the organism is allowed to execute m genome executions when selected by the scheduler, and typically resulting in as many offspring (i.e. fecundity of m). For this, the macro-level of the evolutionary system handles scheduling, self-replication, mutation, merit and death.

Table 3.3 Self-replicating OR-computing program.

```
PATTERN START
INPUT R0
INPUT R1
NAND R0 R0
NAND R1 R1
NAND R0 R1
OUTPUT R0
PATTERN END
SET RD START
SET WR END
PATTERN LOOP
COPY RD WR
JUMPNEQ LEND LOOP
COPY RD WR
PATTERN LEND
PATTERN END
```

The table shows a self-replicating and OR-computing organism.

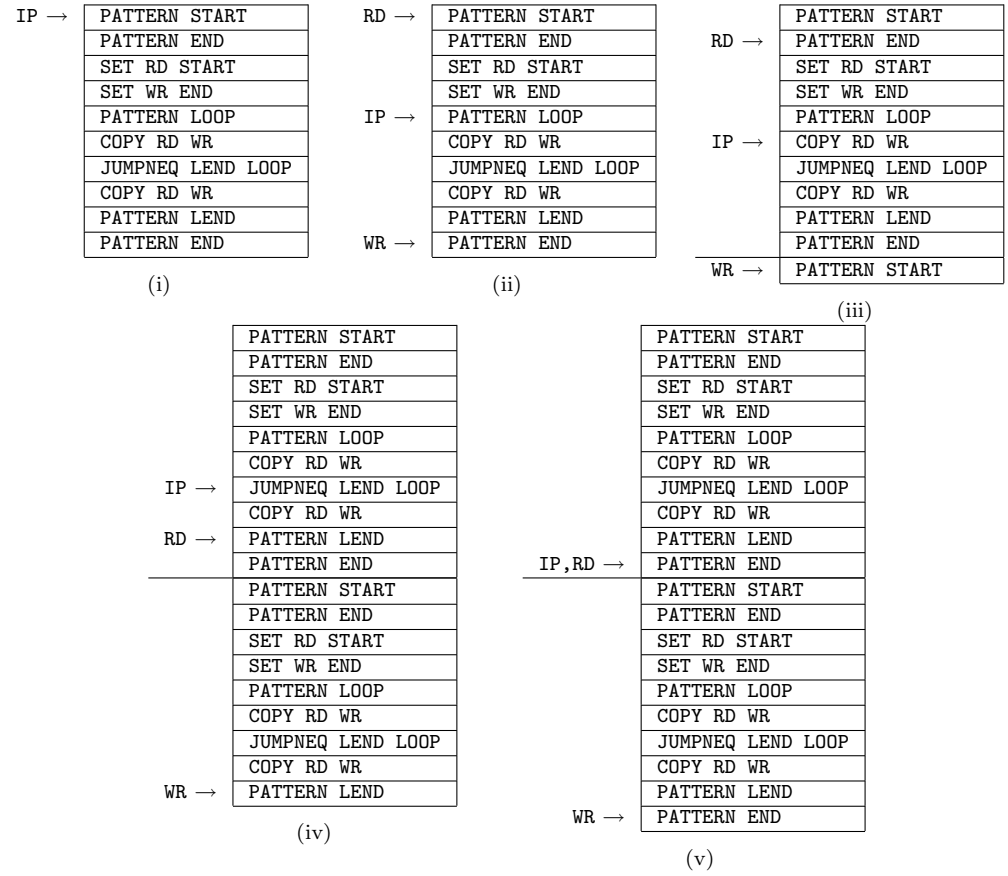
Observe that SALTA implements $CHEM(r, \simeq, R, S, \alpha, s, T)$, by self-replication ($r$), point-mutation or recombination ($\simeq$), mutation/recombination rate (R), merit function (S), parallel scheduler (α), topology size (s) and topology (T).

3.2.1 Self-replication and death (r)

Organisms replicate through asexual self-replication (like prokaryotes) [MMP97], which is implemented by a self-copying loop (see also page 30). The instruction set does not contain any specific instructions for replication, instead the system handles allocation of new offspring, their placement and detachment from parent (division). New offspring is placed in the parents 1-Moore neighborhood (causing death) without regard to copying success. Example 3.5 shows one whole replication cycle.

Death is necessary to keep the system within the limit of a finite topology. In the chemostat, death works through dilution at the rate of replication. Death is implemented in the system by new offspring overwriting a neighbor in the following order of priority. If any unscheduled neighbor is found, it is replaced with the offspring randomly. If this is not the case, the oldest neighbor is replaced (each born organism has a unique age), or if all organisms have the same age, i.e. they are all seeding organisms, a random neighbor is replaced. By overwriting unscheduled organisms, the system cleans up itself from unsuccessful replications. Overwriting the oldest organism has been shown to propagate new information faster in the population [Ada98].

Figure 3.5 One cycle of self-replication.



The example shows self-replication of the default self-replicator in 5 steps. The extended horizontal line illustrates the automatic separation between parent (upper) and offspring (lower).

3.2.2 Mutation ($\simeq$)

Genetic variation by mutation is captured by perturbing the copying instruction (`COPY`), that copies program words from parent to offspring, with point-mutations such as substitution, deletion and insertion (see also recombination in Chapter 5).

Substitution: modify the copied program word by some random bit flips (default 32), randomly distributed over the whole word to affect both instruction and arguments.

Deletion: do not copy the word.

Insertion: copy the current word and add an extra random program word before.

Typically, the rate of substitution (μ_s) is about 1.5 the rate of deletion (μ_d) and insertion (μ_i), i.e. $\mu_s = 1.5 * \mu_d = 1.5 * \mu_i$. Since $\mu_d = \mu_i$, program length increases or decreases with the same rate, without regard to other selective factors. When overwriting instruc-

tions during recombination, substitution is the only effective point-mutation. Mutation rates are given as a parameter R to the system.

3.2.3 Merit (S)

To create competition as basis for natural selection, differential reproductive success is implemented in the system via a merit function. Organisms can earn merit to increase their relative execution speed by computing pre-defined functions. Typically, more computationally difficult functions give higher merit, but this relation is user-defined. A merit m is given to an organism at the end of the genome execution, based on the highest computed function (with respect to merit) as follows. For any output t in the execution, all previous input values $i_0, ..., i_n$ are tested against all given functions f_j for a match $t = f_j(i_0, ..., i_n)$. If such a match is found, a temporary variable is updated with f_j's associated merit m_j, if m_j is higher than the current value of the temporary variable. An organism can only get merit for a particular function a constant number of times, to increase the pressure for novelty [Ada98]. By the end of the genome execution, the organism is given the value of the temporary variable, i.e. merit for the highest computed function. Since each genome execution may result in new merit (or none), they are accumulated in a local *merit queue*. Each time an organism is scheduled, a new merit is drawn from the organism's local merit queue to decide the execution speed of that organism. High merit gives high execution speed, and generally high fecundity (high number of offspring), see below.

3.2.4 Scheduling (α)

The set of organisms that makes up the population is scheduled by a scheduler (α). When a parent creates an offspring, that offspring is directly placed in its vicinity, but is only dispatched to the scheduler after the parent has executed one complete genome execution. By default, the scheduler executes each dispatched organism non-preemptively, randomly but fairly, as many genome executions as the next drawn merit (m) in the merit queue allows. Thus, the scheduler imposes differential execution speed fairly among the organisms based on computed merit (m), resulting in differential reproductive success (fecundity) and fitness. If a dispatched organism has an empty merit queue, it is only allowed to execute 1 genome execution. This basic execution speed is necessary to keep

evolution going, by mutating the produced offspring to search for new evolutionary paths to higher merit.

Chapter 4

Mutational Evolution

There has been a large amount of work in studying how point-mutations can be used to evolve so-called "digital organisms" [AB94, BP03, HJF97, RKFH90, Ray91, Sip98, Tay02, LW04, YB01, Wil03]. Some approaches use explicit merit functions, giving rise to static fitness landscapes [AB94], while others use emerging fitness landscapes through ecologies [RKFH90, Ray91] or other forms of interaction [Tay02], but in most of them is random mutation the force of variation. We will, in this chapter, look at mutational evolution in a fixed fitness landscape.

Overview This chapter will start by looking at how simple point-mutations can evolve a population to an explicitly defined merit function, without any interaction with the environment (other than death; see Section 3.2.1). However, complexity will not be of great concern in this chapter since the focus is more on the basic dynamics of simple mutational evolution, it will be shown that

(i) populations always lower the average genotype size without environmental fitness pressure (Section 4.1.2),

(ii) since the merit function creates a fixed fitness landscape with finite high peaks, evolution will always produce populations of limited average genotype size and lowest possible diversity (as a function of mutation rate)(Section 4.2.3), and

(iii) genotype diversity is higher in a 2-D topology than in a 0-D topology (Section 4.2.3).

First, a section about Shannon's information theory and Eigen's hypothesis about quasi-species are presented. Basic understanding of information theory and the quasi-species hypothesis will be useful in the discussion of reported experiments. The strive for a small genotype is also illustrated in an experiment over a flat fitness landscape. Second, we define a fitness landscape, by a particular fixed merit function, over a set of logical functions. Using this fixed merit function, parameters such as mutation rate, population size and topology are evaluated with respect to their effect on evolution.

4.1 Information theory and quasi-species

Evolution can be seen as a continuous transfer of *information* from parent to offspring, and refined over generations to better adapt each population to its environment (niche). Information is encoded into the genotype of each organism (assuming genotype/phenotype distinction), and is made hereditary by genetic transfer (asexually or sexually) to the offspring. The genotype holds information in the sense that it is the evolved "blueprint", for how an organism can best survive and reproduce in its particular niche. Formally, Shannon defined information as the amount of correlation between two systems. In evolution, the two systems are genotype (via the phenotype) and environment, and they are correlated to each other in terms of entropy [Sha48, Ada98, Ada04].

4.1.1 Eigen's quasi-species

Quasi-species is a Darwinian model, for evolution of self-replicating entities (molecules or simple organisms in our case), put forward by Manfred Eigen and Peter Schuster [Eig71, ES79]. It is applicable to our abstract chemostat $CHEM(r, \simeq, R, S, \alpha, s, T)$, defined earlier (see Section 2.4), hence we may use it to reason about the dynamics.

The term *quasi-species* was defined by Eigen as the most abundant genotype with *optimal superiority parameter (σ_{opt})* defined as

$$\sigma_{opt} = \frac{S(a_{opt})}{\bar{r}}, \tag{4.1}$$

where $S(a_{opt})$ is the merit for the optimally fit genotype/organism (a_{opt}), and $\bar{r}$ is the removal rate of each genotype (rate of death), i.e. σ_{opt} is the effective propagation rate of the most fit genotype. His idea was that learned information (about the environment)

is stored by the majority of the population, i.e. those entities that replicate most successfully (σ_{opt}). In addition, these optimal genotypes (a_{opt}) must replicate with a rate above zero. Hence the following constraint must be satisfied

$$\begin{aligned} FS(a_{opt}) - \bar{r} > 0 &\Leftrightarrow FS(a_{opt}) > \bar{r} \\ &\Leftrightarrow \frac{FS(a_{opt})}{\bar{r}} > \frac{\bar{r}}{\bar{r}} \\ &\Leftrightarrow F\sigma_{opt} > 1. \end{aligned} \tag{4.2}$$

This equation can be reformulated in terms of the error rate R and optimal genotype length l_{opt}. First we rewrite F as follows ($F = (1-R)^l$, see Section 2.4)

$$\begin{aligned} F &= (1-R)^{l_{opt}} \\ &= e^{\ln(1-R)^{l_{opt}}} \\ &= e^{l_{opt}\ln(1-R)} \\ &\approx e^{-l_{opt}\ln(1-(1-R))} \\ &= e^{-l_{opt}\ln R}. \end{aligned} \tag{4.3}$$

This new formulation together with Equation 4.2 gives

$$\begin{aligned} F\sigma_{opt} > 1 &\Leftrightarrow e^{-l_{opt}\ln R}\sigma_{opt} > 1 \\ &\Leftrightarrow e^{-l_{opt}\ln R} > \sigma_{opt}^{-1} \\ &\Leftrightarrow \ln e^{-l_{opt}\ln R} > \ln \sigma_{opt}^{-1} \\ &\Leftrightarrow -l_{opt}\ln R > \ln \sigma_{opt}^{-1} \\ &\Leftrightarrow l_{opt}\ln R < \ln \sigma_{opt} \\ &\Leftrightarrow l_{opt} < \ln \sigma_{opt}\frac{1}{\ln R} \\ &\Leftrightarrow l_{opt} < \ln \sigma_{opt}\frac{1}{R}. \end{aligned} \tag{4.4}$$

From Equation 4.4, it is easy to see that the error rate R puts an upper limit to the genotype length l_{opt}. This is called Eigen's *error threshold*, and is reached whenever $Rl = \ln(\sigma_{opt})$, above which the information cannot be conserved by the system.

Based on this, the error threshold has been suggested to act as an evolutionary gradi-

ent [BP03, Ada95, CAW01, WWO+01]. For evolution to continue beyond this threshold, error-correction has to be invented. Many different hypotheses about biological error-correction have been put forward, e.g. the very powerful innovation of diploidity and sex [Rid00]. Asexual organisms do not have such fancy error-correction mechanisms, and therefore have smaller genotype.

4.1.2 Strive for the smallest

In the absence of any interesting environment (merit function) to adapt to, asexual populations will gradually minimize their genome. This has been verified in "wet-ware" [MPS67] as well as in software [AB94]. There are two reasons for this: (i) smaller genotypes are faster to copy, and (ii) smaller genotypes can be copied with lower error rate (recall that replication fidelity decreases with the length l; $F = (1 - R)^l$).

In this experiment we used a constant merit function giving rise to a flat fitness landscape, to evolve the experimentally smallest self-replicator. SALTA was seeded with the default self-replicator in Table 3.1, and after around $1 * 10^9$ genome executions, the chemostat consisted of more than 90% of the self-replicator in Table 3.2. The initial self-replicator was 10 instructions long with extra instructions for recombination, but the final experimentally smallest self-replicator had only 7 instructions. Evolutionary complexity is therefore at a minimum; both diversity and average genotype size decreases to an existential minimum. Note that this drive for smallness is strictly based on reason (ii) above, since the replication speed in SALTA is insensitive to genome size.

4.2 Adaptive landscape—an example

We may define a particular fitness landscape based on point-mutations (see Section 2.3.2) and computational phenotypes defined over a set of bitwise logical functions (see page 30). The resulting landscape, defined by default merit (m_{def}) in Table 4.1, has 6 levels of increasingly higher peaks, and interconnected by infinitely many paths. Since increasingly more complex logical functions (based on number of needed NAND operators) can be computed by adding new NAND instructions (by point-mutation), there is always a non-deleterious mutational path from one peak to another. The highest peaks are the EQU-peaks, from which there are no ascending climb. Note that there is epistasis

Table 4.1 Default merit and merit with gap.

Function	Merit	
	m_{def}	m_{gap}
NOT	2	2
NAND	3	3
AND	4	1
OR_N	4	1
OR	5	1
AND_N	5	1
NOR	6	1
XOR	6	1
EQU	8	8

The table shows both default merit (m_{def}) that increases for more complex logical functions (with respect to the number of NANDs needed for their computation), and merit with gap (m_{gap}) that creates a U-shaped fitness landscape.

between instructions which creates some ruggedness. Also note that the existence of quasi-neutral instructions in the default instruction set, gives indefinitely many peaks of equal height.

In the following subsections we performed 3 different experiments to identify the following fundamental parameters in our chemistry: mutation rate, population size and topology. In all experiments we used the default merit in Table 4.1.

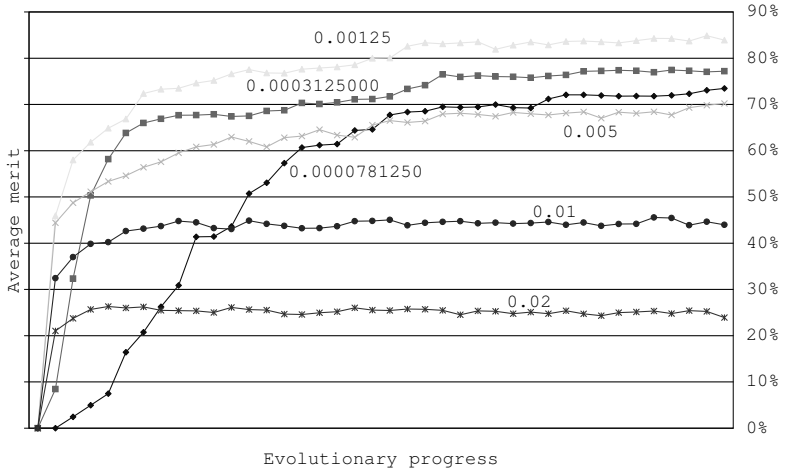
4.2.1 Evolution, as a function of mutation rate

There is a significant amount of work regarding the relationship between mutation rate and evolution [AB94, Ada95, DF98, BB98, BP03, Eig71, ES79, CAW01, YB01, WWO$^+$01]. It is illustrative to view mutation and selection as two opposite forces in evolution; the former creates variation in the population to find new evolutionary paths, the latter lowers variation to slowly push the population in an evolutionary direction.

To test how the mutation rate affected adaptation in our chemistry, we evolved 10 populations in a 40^2 topology seeded with default self-replicator (Table 3.1) with extra quasi-neutral patterns, for different mutation rates (R in *CHEM*), over $6 * 10^6$ genome executions.

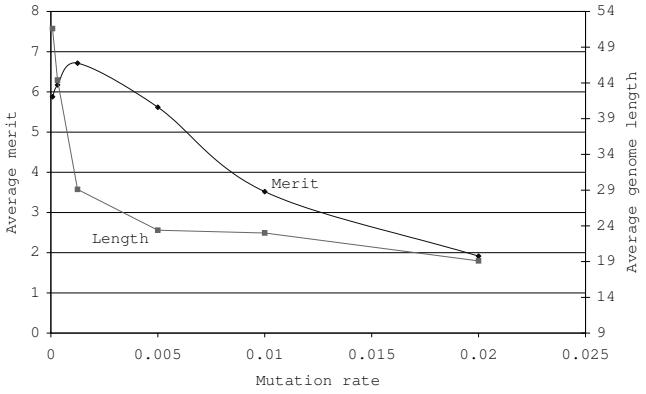
It was found that there was an optimal mutation rate of 0.00125 for substitution (rate of deletion and insertion is 1/2.5 the rate of substitution), for which the populations adapted very fast to the experimentally highest average fitness of about 85%; see Figure 4.1. For lower mutation rates the adaptation was slower, but for higher rates the mutation rate induced an error threshold above which organisms cannot replicate

Figure 4.1 Evolution in terms of average population merit as a function of mutation rate.



The figure shows the average population merit during evolution, for a number of different mutation rates.

Figure 4.2 Average merit and genome length for different mutation rates.



The figure shows average merit and average genome length for different mutation rates.

with enough precision, for maintaining their phenotypes. Table 4.2 and Figure 4.2 show the different mutation rates together with merit and genome length, for the average organism after the total number of genome executions. The inverse relation between genome length and mutation rate R respects Eigen's error threshold (Equation 4.4), by creating an effective upper limit on genotype length. Note also in Table 4.2 that average genotype size and average genotype diversity are in opposite relation (in conflict) to each other; average genotype size increases but average genotype diversity decreases, for higher mutation rates.

Table 4.2 Mutation rate, average merit per organism, average genome length, average genotype diversity, and evolutionary complexity.

Mutation rate		**Merit**	**Length**	**Diversity**	**Evol.Compl.**
Subs	**Ins/Del**				(Length ∗ Diversity)
0.000078125	0.00003125	5.8783125	52	40	2,080
0.0003125	0.000125	6.1749375	44	102	4,488
0.00125	0.0005	6.7139375	29	203	5,887
0.005	0.002	5.6174375	23	475	10,925
0.01	0.004	3.5178125	23	789	18,147
0.02	0.008	1.9138125	19	1037	19,703

The table shows the mutation rates together with measured average merit, average genome length, average genotype diversity and evolutionary complexity, per organism, for evolution with different mutational pressure.

4.2.2 Evolution, as a function of population size

An evolving population may be seen as a *genetic pool* of adaptive approximations. If the pool is not sufficiently large, there may be divergence in the adaptive process called *genetic drift* (see Section 2.5.1). It has been shown that larger populations create a larger "learning window", for evolution under different mutation rates [AB94]. Researchers have also proposed that large products of population size and mutation rates creates "clonal inference" for asexual populations, lowering the chance of fixation of beneficial mutations [GL98].

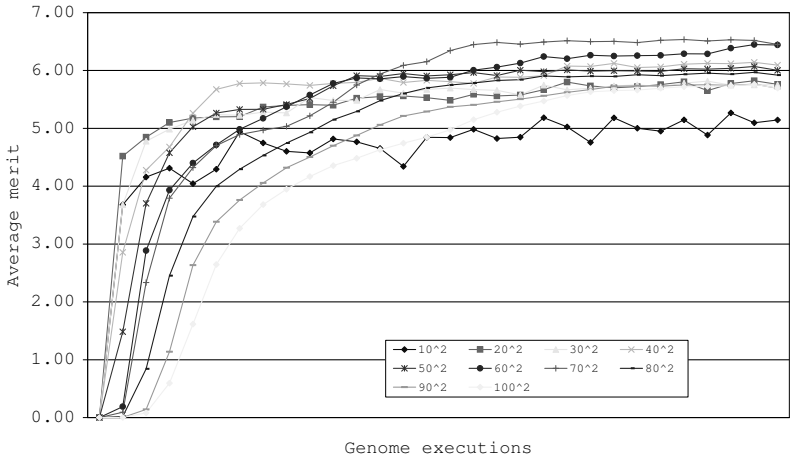
To test how the population size (s in *CHEM*) affected adaptation in our chemistry, we evolved 10 populations in different sizes of the 2-D topology, seeded with default self-replicator (Table 3.1) with extra quasi-neutral patterns, for $6 * 10^6$ genome executions. Two mutation rates of 0.00125 and 0.000625 for substitution (rate of deletion and insertion is 1/2.5 the rate of substitution) were tested, both with similar results.

Figure 4.3 shows the average merit, for the different population sizes $s^2 = 10^2, 20^2, ..., 100^2$, and over the total number of genome executions. First, note that the total of $6 * 10^6$ genome executions are distributed over the population, resulting in much lower ratio of genome executions per organism, for larger populations than for smaller. We may view the genome executions as *energy* that we invest in different population sizes, for evolutionary success.

A number of observations can be made:

- Higher population size gives slower learning rate compared to lower population size. This is not surprising since larger populations distribute less energy per organism along the x-axis.

Figure 4.3 Average merit as a function of population size.



The figure shows average population merit during evolution, for a number of different population sizes.

- Very low population size has a very rugged learning curve compared to higher population sizes. This may be explained by the relatively narrow search performed by small populations (larger populations hold more approximations to widen the search).

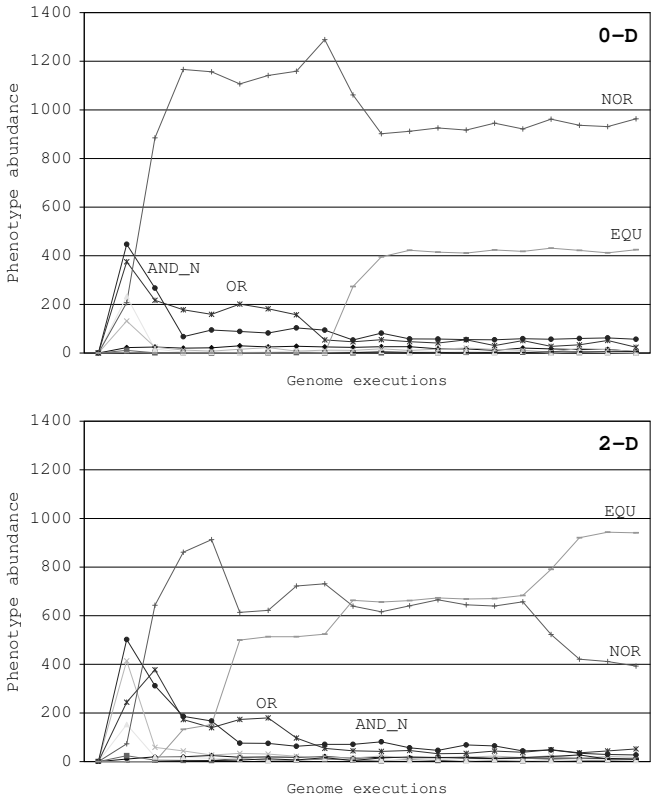
- There is an optimal population size (70^2), for the given mutation rates. This is a bit surprising, but can be explained by the narrow search, and possibly also by genetic drift, for the small populations, and clonal inference, for the large populations. Also, for large populations, the distributed energy to each organism is less than for small populations; when the population size goes to infinity, the distributed energy goes towards zero.

It was also noticed that the above observations hold for other mutation rates, but less obvious for lower rates. Naturally, evolutionary complexity is higher in larger populations since they can accommodate a larger number of different genotypes. In terms of genotype size, this is not necessarily the case since small populations can drift, and dramatically increase the average genome size with quasi-neutral instructions.

4.2.3 Evolution, as a function of topology

It has been argued that the topology is the main factor behind evolutionary diversity (punctuated equilibrium [Gou02]). Apart from that, topology specifies the locality of competition and co-operation (see e.g. symbiosis in Section 6). In particular, we may

Figure 4.4 Average abundance of evolved logical functions in 0-D and 2-D topologies.

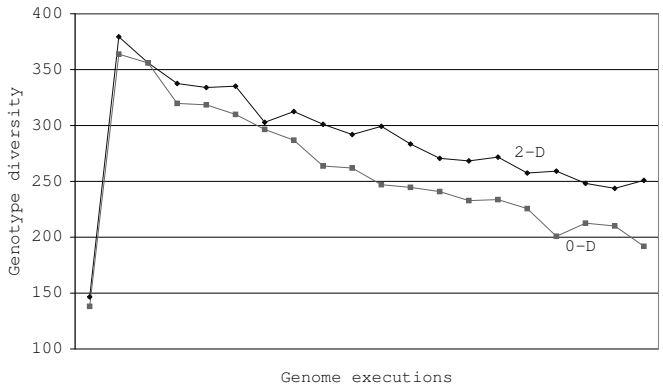


The figure shows the average abundance of each logical function in 0-D topology (upper) and 2-D topology (lower) of size 40^2.

study evolution without locality, i.e. 0-D (mass-reaction), and compare that with evolution in a 2-D topology with locality. In SALTA, a 0-D topology (mass-reaction) was achieved by randomly dispersing new offspring in T, instead of as in 2-D, in the 1-Moore neighborhood.

We tested adaptation and *genotype diversity*, for the same fitness landscape of logical computation as above, and for both 2-D and 0-D topologies. Except for the dispersal settings (0-D or 2-D), the experiment was conducted as in Section 4.2.2, but with a fixed population size of 40^2 organisms. Figure 4.4 shows adaptation in 0-D and 2-D. In the 0-D topology, evolution often got stuck in a local optimum (the NOR function), but evolved this optimum quite fast and for a large percentage of the populations. In contrast, the 2-D topology did not get stuck in local optimum as frequently. Instead, the global optimum (the EQU function) evolved slowly but in a higher number of populations; 7 populations in 2-D, compared to 3 populations in 0-D. It is also interesting to note that the genotype diversity was notably higher in 2-D than in 0-D. Most likely since a 2-D

Figure 4.5 Genotype diversity plotted against evolutionary progress for 2-D and 0-D topologies.



The figure shows average genotype diversity plotted against evolutionary progress for 2-D and 0-D topologies.

topology creates higher locality in the competition for space than 0-D. Although, the genotype diversity in either setting decreased continuously from initially around 375, for both 0-D and 2-D, to around 190 for 0-D, and 250 for 2-D, see Figure 4.5.

We may observe that the locality in the 2-D topology gives better evolutionary success than for mass-reaction in 0-D. In the 2-D topology, we have higher diversity of potential genotypes, which broadens the search process. This is compared to mass-reaction, which exhibits high homogeneity and leads to fast evolution, but more often to local and not global optimum. It is also worth noting that during evolution the genotype length constantly decreases; from initially 50 instructions to 36 instructions (see also Figure 5.6). The decrease is due to the finite amount of information in the merit function, and the initial amount of useless quasi-neutral instructions that are shed by natural selection. In terms of evolutionary complexity, 2-D topology gives higher diversity than 0-D and therefore higher complexity, even though the merit function can only support limited genotype growth.

Chapter 5

Genetic Recombination

Having studied evolution with mutation as the single factor for variation in the previous chapter, we now add *genetic recombination* to the platform to compare them in-between for evolving "complex features" (i.e. logical functions). Genetic recombination is the biological process where parallel lineages of genetic material are merged into one genotype [KPP94, KS88, MMP97]. For sexual organisms, genetic recombination is part of meiosis for variation in the gametes (germ cells). For asexual organisms, such as the prokaryotes, genetic variation is preceded by conjugation, transduction or transformation. There is also evidence that so-called transposons that multiplies at the gene-level, are a major factor for genetic recombination, for both sexual and asexual organisms. In this chapter, we will look at recombination through *lateral gene transfer* by conjugation, i.e. parallel lineages of organisms that transfer genetic material in-between to combine their genotypes. Prokaryotes, e.g. bacteria, are known to conjugate and transfer genetic elements from donor to recipient, and to establish a relatively transparent pool of genes.

Overview This chapter presents genetic variation through recombination and compares its dynamics with mutation for the evolution of "complex features" (i.e. logical functions as in previous chapter). It will be shown that recombination is less sensitive to the fitness landscape, more efficient, and can even drive evolution without mutation (given sufficiently initial genetic material). In terms of evolutionary complexity it will also be shown that recombination increases both the genotype diversity and the average genotype size. In the first section, a short discussion about composability is held. The second section presents the modified version of SALTA that includes genetic recombina-

tion. In the third and ending section, we reproduce the results of evolution of "complex features", published in *Nature* by Lenski *et al.* [LOPA03], and compare these results with recombination. Note the distinction between the evolution of "complex features" in [LOPA03], and evolutionary complexity denoting the product of diversity and average genotype size in a finite population (defined in Section 1.3).

5.1 Composable evolution

From a theoretical perspective of viewing evolution as optimization over a set of variables, there are 3 classes of variable dependencies that may be identified; *arbitrary interdependencies*, *modular interdependencies* and *weak interdependencies* [WP02, Chapter 1].

- If we assume arbitrary interdependencies between the variables (alleles/genes), the resulting optimization landscape (fitness landscape) is very rugged. Such dependencies are the result of strong and arbitrary functional dependencies between the variables (high level of epistasis). Solving such optimization problems require typically an exhaustive search with exponential complexity as a result.

- At the other extreme is the weak, or non-existing interdependencies that create a smooth landscape with weakly dependent variables, which can be optimized independently. For such problems, a linear search is typically enough.

- Finally, in the middle of these two extreme cases we may put what Watson calls *modular interdependencies*. In this class of variable dependencies, there are subsets of variables with strong intra-dependencies, but with only weak inter-dependencies (between subsets). This means that subsets (modules) can be optimized independently, and subsequently put together and optimized as a whole.

In computer science we call the last class of optimization problems for "Divide & Conquer"-problems, typically with complexity exponential to $n/2$, where n is the number of variables, instead of exponential to n as in the case of arbitrary interdependencies.

In biology, this view of *composable evolution* has be put forward by Margulis [Mar81], in her Serial Endosymbiosis Theory (SET), to explain the origin of the eukaryote cell by mergers of parallel lineages of organisms. In Artificial Life, Holland proposed this evo-

lutionary hypothesis, which he named *the Building Block Hypothesis* [Hol75], as early as 1975. Holland also introduced composable evolution by genetic crossover (meiosis/sex) in the field of Genetic Algorithms (GA). In this chapter we will recombine genomes (merge/split), for composable evolution of logical functions.

5.2 Extending SALTA with recombination

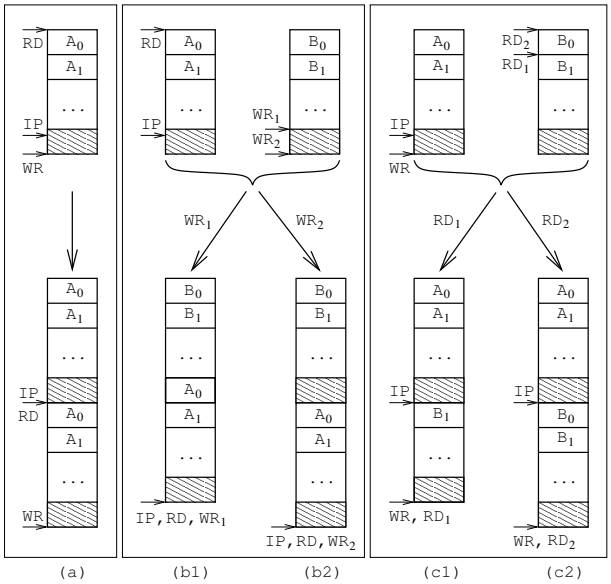
Recombination opens up the possibility of copying parts of an organism's program to another organism to allow for so-called *lateral gene transfer*. Instead of copying program words from parent to offspring in the self-copying loop, the `SET` instruction may reference another organism, and allow for instructions to be read from, or written to it. The neighboring such organism is chosen randomly from the 1-Moore neighborhood (by default), but the success of recombination is dependent on the existence of matching patterns in the target program.

In our chemistry, it is made possible, for the self-replicator in Table 3.1, to transfer instructions to another program by setting the pointer in either, but not both of the `SET` instructions to a neighboring program. By doing so, it is possible for the search of the `START` or `END` pattern to reference into another program to allow for transfer of instructions between existing organisms. If the search fails in the neighboring program, the search is relocated to the local program instead. Figure 5.1 illustrates the 5 possible cases of replication and recombination in SALTA.

5.2.1 Recombination for computation

Genetic recombination can effectively chain up sequences of instructions from different organisms to compute increasingly complex functions. The I/O Buffer makes it possible for outputs computed earlier to become inputs to another instruction later in the program. Higher complex functions can be formed by merging instruction sequences from different programs, assuming that the complex function can be computed by "Divide & Conquer" (see above). Complex logical functions, for example, can be computed by combining simpler logical functions, e.g. (equivalence) $x \Leftrightarrow y \equiv (x \wedge y) \vee (\neg x \wedge \neg y)$. Note that split (case c1 in Figure 5.1) works in the opposite direction by shedding parts of a program. See Figure 5.2 for an illustration of a recombined program, computing

Figure 5.1 Self-replication and genetic recombination (self-replicating loops are striped).



Self-replication and genetic recombination (self-replicating loops are striped). Cases: (a) self-replication, (b1) genetic merge (write pointer `WR`$_1$ points to neighbor, but not to its last `END` pattern), (b2) self-replication (write pointer `WR`$_2$ points to last `END` pattern of neighbor), (c1) split (read pointer `RD`$_1$ points to neighbor, but not to its first `START` pattern), (c2) replication (read pointer `RD`$_2$ points to first `START` pattern of neighbor).

the bitwise equivalence function `EQU` in steps.

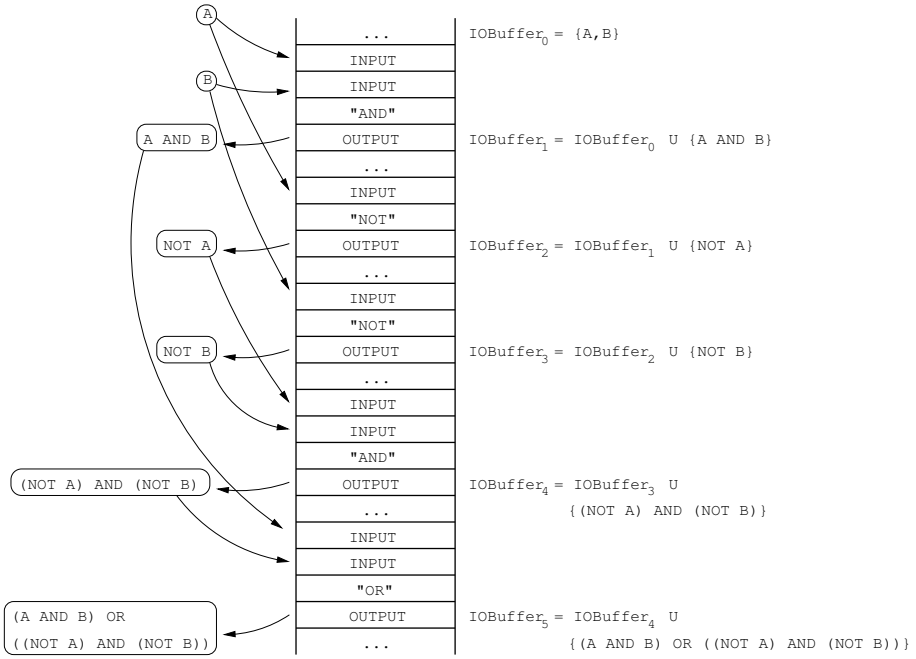
5.2.2 Negative effects of recombination

Even though recombination can be used in our chemistry, for effectively evolve logical functions, it is strictly not a favorable feature. Recombination may merge genomes to lengthen the target genome very fast, increasing the chance of fatal mutations, or split genomes to shed the target genome length, and possibly lower its functionality (case b1 and c1 in Figure 5.1, respectively). Furthermore, recombination lowers the reacting organism's fitness by spending valuable genome executions to merge or split genomes, instead of producing new offspring.

5.3 Experiment: recombination versus mutation

In order to show the evolutionary effect of genetic recombination versus ordinary point-mutation, and how complexity in terms of diversity and average genotype size are affected, we have conducted a series of experiments with inspiration taken from the work by Lenski *et al.* [LOPA03] as a case study.

Figure 5.2 Equivalence function possibly evolved through genetic recombination.



Equivalence function possibly evolved through genetic recombination. The illustration shows a scenario where possibly simpler logical functions have independently evolved and then merged together to compute the `EQU` function in steps, via the I/O Buffer. The accumulating I/O Buffer is shown on the right side of the genome during execution. On the left side, the input and output values from the I/O Buffer are illustrated as they are used during computation. Patterns are not shown.

Recently, Lenski *et al.* used the 9 different logical functions (on page 30) to show support for gradual evolution of "complex features" by random mutation [LOPA03]. They used a platform named Avida [Ada98], and evolved 50 populations to see how many of these which evolved the most complex logical function—equivalence (EQU). Instead of having a linearly increasing merit function (as in Table 4.1), they used an exponentially increasing merit function; based on the number of NAND operations needed for computing them. Lenski *et al.* found that 23 out of 50 populations evolved the EQU function after around $10.8 * 10^9$ executed instructions. They also showed that removing single, or pairwise logical functions from the merit table did not affect the evolution significantly.

5.3.1 Complex evolution with mutation

We reproduced gradual evolution by letting point-mutations evolve populations of non-computing self-replicators (Table 3.1) to a the pre-defined linearly increasing merit function, defined by the given logical functions (default merit in Table 4.1). A quasi-neutral

sequence of instructions acted as genetic raw material—as in [LOPA03]. Similar parameters as in the original work were used; evolution continued for $6*10^6$ genome executions, point-mutation rate per copied word was 0.00125 for substitution, and 0.0005 for insertion and deletion, respectively. After evolution, 17 out of 40 populations had evolved the most complex logical function EQU. Each such population had almost 90% of the maximum merit on average; see Figure 5.3.

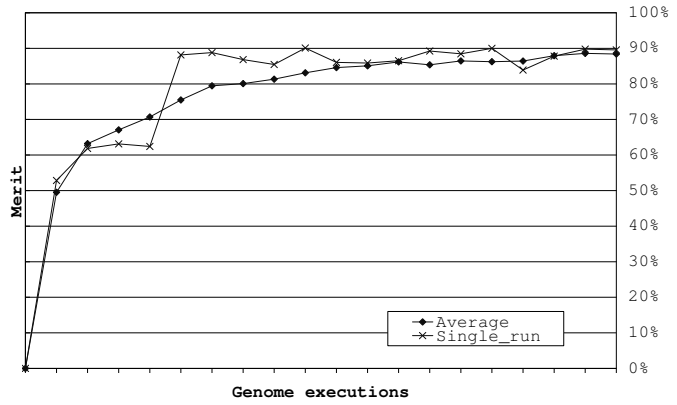
There are strong similarities between our setting and [LOPA03]. We evolved the same logical functions (computed by NANDs) in the domain of assembler automata, with comparable execution duration ($3 * 10^6$ instructions on average per organism in [LOPA03] compared to $562,500$ here, note that our instruction set is about 3 times smaller), and similar macro-dynamics (self-copying with random mutation in a finite population of about the same size). We also have the following indications that this experiment reproduces the results in [LOPA03].

- The ratio 17/40 of evolved populations computing EQU corresponds well to the success rate of 23/50 in [LOPA03].
- The evolved EQU-computing organisms were similar to the ones in [LOPA03]. Our EQU-computing populations had an average genome length of 35 instructions, and looked similar to the organism in Table 3.1, but with extra instructions, e.g. to compute EQU; compared to the median of 28 instructions (excluding instructions needed for self-replication) reported in [LOPA03], for those organisms finally computing EQU. Minimum hand-coded EQU-organism is 18 instructions long in our setting, compared to 34 in [LOPA03].

5.3.2 Complex evolution with recombination

Point-mutation is not the only way for genetic variation, as explained above, genetic recombination can also be a factor for variation in evolution. In the following 3 subsections, we will look at (i) how recombination can drive evolution even without mutation (given enough genetic material), (ii) how recombination may speed up evolution with mutation, and (iii) how recombination may be crucial for bridging evolutionary gaps. Hence, $\simeq$ in *CHEM* is extended with recombination besides point-mutations.

Figure 5.3 Merit for a single run and average merit over all successful populations finally evolving EQU.



The figure shows merit for a single run and average merit over all successful populations finally evolving EQU.

(i) Recombination or mutation

First, we investigated the effectiveness of genetic recombination versus mutation with initial populations of organisms that compute the simplest logical functions NOT and NAND. The populations evolved for $6 * 10^6$ genome executions in a 40^2 topology, and seeded with equal amount of NOT and NAND-computing organisms (see Table 5.1). The rate of point-mutation per copied word was 0.00125 for substitution, and 0.0005 for insertion and deletion, respectively. For recombination, there was a probability of 0.01 per replication, for either merging or splitting the target genome. It was found that 38 of the 40 populations evolved the EQU-function with recombination, but only 3 populations did so with mutation only. In comparison with the experiment in Section 5.3.1, the low success rate of mutation (3/40 compared to 17/40) is due to the lack of quasi-neutral instructions in the NOT and NAND-organisms, functioning as evolutionary raw material.

A comparison of the evolutionary dynamics of genetic recombination and mutation shows that recombination is much more efficient than mutation. Figure 5.4 shows the results from the results from the above experiment with NOT and NAND-computing organisms. Note that the quasi-neutral instructions in Section 5.3.1 give a much smoother and faster learning curve than without in the case of mutation only. By recombining pre-existing genetic material (here NOT and NAND), EQU-computing organisms evolved almost immediately, and reaches a high concentration much faster than by mutation

Table 5.1 NOT and NAND computing self-replicating organisms.

NOT-organism	**NAND-organism**
PATTERN START	PATTERN START
INPUT R0	INPUT R0
NAND R0 R0	INPUT R1
OUTPUT R0	NAND R0 R1
PATTERN END	OUTPUT R0
SET RD START	PATTERN END
SET WR END	SET RD START
PATTERN LOOP	SET WR END
COPY RD WR	PATTERN LOOP
JUMPNEQ LEND LOOP	COPY RD WR
COPY RD WR	JUMPNEQ LEND LOOP
PATTERN LEND	COPY RD WR
PATTERN END	PATTERN LEND
	PATTERN END

The table shows NOT and NAND computing self-replicating organisms.

only. Mutation evolved relatively few EQU-computing populations with comparatively low concentration after a considerable amount of time.

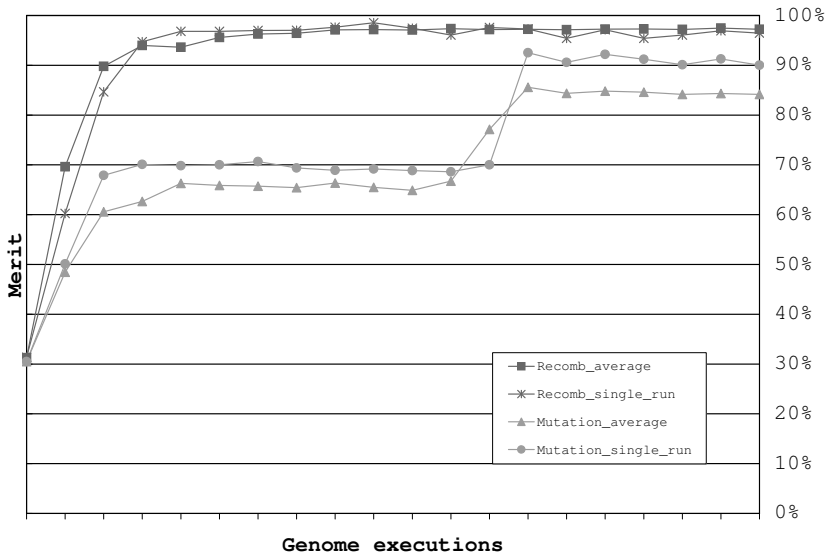
(ii) Recombination and point-mutation

In order to find out how genetic recombination affected the evolution of mutating populations of non-computing self-replicators, we varied the recombination rate (λ) over a given range, for each population, and for a fixed period of $1 * 10^6$ genome executions in a 40^2 topology. Without recombination ($\lambda = 0$), only 5 populations evolved the EQU-function. With genetic recombination, all populations evolved the highest logical function EQU for some λ, and as many as 25 of the 40 populations evolved EQU for $\lambda = 0.1$ (see Merit_def in Figure 5.5)

(iii) Recombination for bridging merit gaps

Finally, we repeated the above experiment with a non-monotonic merit function giving rise to a U-shaped fitness landscape (see merit m_{gap} in Table 4.1). In particular, we only gave increasing merit to NOT, NAND and EQU-computing organisms, and found that none of the populations evolved EQU with mutation only. Fortunately, genetic recombination made the organisms bridge the evolutionary gap, and all of the populations evolved EQU for some λ. It was also found that 31 of the 40 populations evolved EQU for $\lambda = 0.0729$ (see Merit_gap in Figure 5.5).

Figure 5.4 Merit for a single run and average merit over all successful populations finally evolving EQU with mutation or recombination.

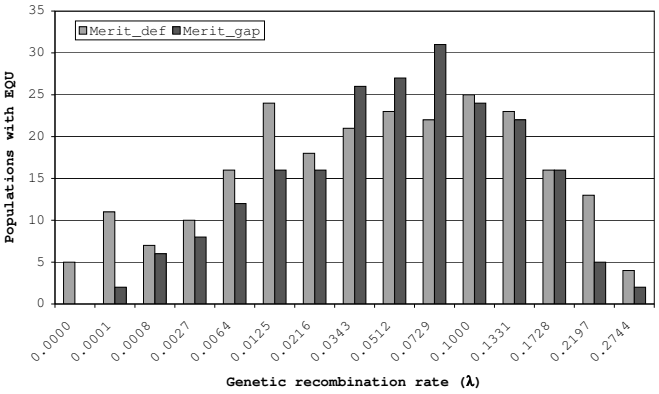


The figure shows the evolved merit for a single run and the average merit over all successful populations finally evolving EQU, with recombination or mutation only of NOT and NAND-computing organisms.

5.3.3 Genotype length and diversity

In Figure 5.6 we compared genotype diversity and average genotype length, for the different dynamics of mutation only, and mutation with optimal recombination rate of $\lambda = 0.0729$. Initially genotype diversity and genotype length were much higher, for the runs with recombination. This is expected since recombination adds extra noise to the system which increases the diversity. Recombination also increases genotype length by merging genomes. During evolution, genotype diversity and genotype length strictly decrease, with or without recombination, to around half the initial values. The decrease in genotype diversity and genotype length is also expected, and can be explained by the finite amount of environmental information (merit) and by the drive for increased fidelity by shedding length. Although, note that throughout the run, recombination creates higher average genotype length and genotype diversity, i.e. higher evolutionary complexity.

Figure 5.5 Number of populations that evolved EQU for different rates of genetic recombination.



Number of populations that evolved EQU for different rates of genetic recombination (λ), for default merit (Merit_def) and merit with gaps (Merit_gap) in Table 4.1.

Figure 5.6 Average genotype length and genotype diversity with mutation only and with mutation and recombination.

The figure shows average genotype length and genotype diversity with mutation only (GenLenMut and DivMut), and with mutation and with optimal recombination rate $\lambda = 0.0729$ (GenLenRecomb and DivRecomb).

Chapter 6

Symbiosis and co-evolution

So far, our platform has not incorporated any relations to the environment in which the population evolves. After studying how mutation and recombination may drive the evolution in a fixed fitness landscape, in Chapter 4 and 5, we now add a form of *symbiosis* to form interdependencies between the evolving populations. In this way, both populations act as (biotic) environment to each other. Symbiosis was first defined by Anton de Bary (1879), as the living together of organisms from different species [DGSR96]. It is a biological phenomenon which challenges a naive version of Darwin's "survival of the fittest" since a wide range of species may survive in a habitat by forming symbiotic relations. In particular, ecologies are established in biology by symbiosis, where different species create inhabitable niches for each other. Instead of single fitness climbs, symbiosis creates very complex fitness landscapes with interdependencies between evolving symbionts [WP01, WP99].

There are many levels of symbiosis both in space and in behavior. Endosymbiosis, is a very close symbiotic association between typically a small and a large organism, where the small is *inside* the large organism. Such intimate associations have been argued to cause lateral gene transfer and genetic recombination, e.g. in the case of the mitochondria [Mar81, MS02]. The opposite association is called exosymbiosis, where two organisms are anatomically separate, although in some cases physically attached. Endosymbiotic relationships are generally established to form crucial resource dependencies, while exosymbiosis are established both to form behavioral or resource dependencies.

Overview In this chapter, a simple model of exosymbiosis is presented together with a discussion of its sufficient conditions to possibly increase the evolutionary complexity in our framework. It will be shown that two conditions have to be met for symbiosis in our model. The model extends the simple notion of the default chemostat by adding a third dimension to the space. Symbiotic relations are identified and evaluated using *accumulative phenotype*, *evolutionary activity* and a topology measure we call *vicinity difference*. It will be shown that symbiosis can be a source of complexity by increasing the phenotype diversity in experiment (d); Section 6.4.4. The first section clarifies the distinction between cost-benefit relations and symbiosis. The second section gives a summary of earlier work with symbiosis in Artificial Life. The third section presents an extended version of SALTA which models symbiosis. The fourth section gives an experimental discussion of sufficient conditions for symbiosis. In the last section, we show an example of mutational pressure in the symbiotic model visualized with *evolutionary activity* [BSP98, BB99].

6.1 Cost-benefit relations and symbiosis

Symbiosis is often discussed in association with cost-benefit relations such as *parasitism*, *mutualism* and *commensalism* [IK90, BF95, Mar81, MS02], but such relations are strictly not necessary, although they may be sufficient [DGSR96].

Parasitism is a symbiotic relationship between a host and a parasite such that the parasite gains fitness but the host loses fitness.

Mutualism is a symbiotic relationship between two organisms such that both benefit in fitness.

Commensalism is a symbiotic relationship between a host and another organism such that the host is not affected (fitness unchanged) but the other organism gains fitness.

Symbiosis has often been used synonymously to mutualism since such a cost-benefit relation induces high fitness for both symbionts. We will not be concerned about these cost-benefit relations; instead the focus will be on the two physical conditions *dimensional difference* and *resource relations*.

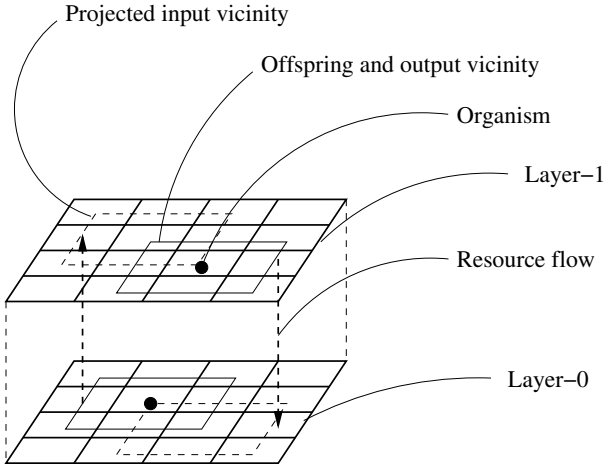
6.2 Symbiosis and co-evolution in Artificial Life

Co-evolution implied as a consequence of symbiosis has been presented and discussed by a number of researchers [BF95, IK90, KJ92, PTJ02, WP01]. Kauffman and Johnsen extended the NK-fitness model, i.e. a fitness model for genomes of N genes/alleles with K intra-genome dependencies, to include inter-species epistasis (NKC-fitness) [KJ92]. They showed that the level of intra-genome epistasis has to be higher than the level of inter-species epistasis for efficient co-evolution, and that there may be evolutionary pressure for such a relation. The NKC-fitness model was also used by Bull and Fogerty, to simulate endosymbiosis and symbiogenesis in a multi-agent system, with recombination by genetic cross-over [BF95]. Both Watson *et al.* and Ikegami *et al.* have looked at parasitic emergence under differential mutation rates [IK90, WRP00]. Symbiosis and co-evolution has also been investigated as a mechanism of pre-adaptation, followed by relatively slow mutational adaptation by Watson [WP99, WP01, WP02]. Different kinds of cost-benefit relations, such as mutualism and ecological relations (predator/prey), have also been studied [PTJ02, HJF97]. From the abstract view of fitness landscapes, co-evolution *couples* two fitness landscapes into an inter-dependent landscape, e.g. increasing fitness for one species may decrease the fitness for another (parasitism). This means that even though the implemented merit function is fixed, the coupled fitness landscapes are constantly changing as a function of each other.

6.3 Extending SALTA with symbiosis

We will extend SALTA to look at sufficient conditions for symbiosis and co-evolution. Basically, the claim is that *dimensional difference* and *resource relations* are both sufficient in our evolutionary model to exhibit co-evolution and symbiosis. Figure 6.1 illustrates the modified model for symbiosis with two chemostat layers. It will be shown that a single layer is not sufficient in our model for exhibiting symbiosis and co-evolution due to the so-called *Competitive Exclusion Principle*. This principle, believed to have originated from the mathematical model by Lotka and Volterra in the 1920's, states that for co-existence of more than one species to last, the intra-specific competition for each species must be greater than the inter-specific competition. Another researcher named May (1981), defined the same principle slightly differently as the conditions under which

Figure 6.1 Modified model with 2 layers.



Modified model with 2 layers. The figure illustrates two chemostat layers (shown as grids) that are superimposed on-top of each other, and interconnected by input and output relations between the layers (shown as arrows). Each organism (shown as a black dot) disperses resources and offspring in the vicinity at the same layer (shown as a solid-line square), but takes inputs from the projected vicinity of the immediately lower layer (shown as dashed-line square) in a circular fashion, where organisms at the lowest layer takes from the highest.

more than one species make their livings in identical ways being unable to exist in a stable fashion [May81]. In other words, more than one species cannot share the same niche. This is indeed the case in our model, since there is only one merit function for the population(s) to evolve to.

Dimensional difference means that distinct populations are superimposed on-top of each other but replicate separately. In the chemostat model, this is achieved by putting one chemostat layer over the other, creating a 3-D space where offspring dispersal is local to each 2-D layer (see also Figure 6.1). Note that in this model, waves of new "species" can co-evolve since they propagate in parallel dimensions.

Resource relations are introduced via the I/O Buffer by mapping inputs and outputs to the topology as follows. Outputs are dispersed in the 1-Moore neighborhood of the local topology layer, while inputs are taken from the projected 1-Moore neighborhood of the immediately lower layer (the lowest layer projects to the highest, creating a cycle). Hence, T_i in *CHEM* is modified to hold resources as well as organisms.

6.3.1 I/O Buffer

Two extra parameters that represent positions in the neighborhood are added to the organism's I/O Buffer, each indicating the input or output preference for the `INPUT` and `OUTPUT` instruction, respectively. In particular, the parameters are 2-tuples of numbers between 1 and 9, indicating unique positions in the 1-Moore neighborhood. 2-tuples are enough since only 2 different inputs are needed to compute the binary logical functions. Both parameters are under mutation, but the output parameter mutates less often than the input parameter. The difference in mutation rate is to increase the possibility of the input preferences of one population to adapt to the output preferences of another. By adapting their input/output preferences, the sub-populations can communicate resources effectively, and possibly establish symbiotic relationships.

6.3.2 Resources

Resources are still bit-strings but since logical functions may be computed in steps and communicated between the layers, extra information is stored to track the stepwise computation. In each resource we store the initial bit-string(s) and the last computed value. For example, an organism may input the resources `NOT A` and `NOT B`, and subsequently compute the logical function `A OR B` by computing `(NOT A) NAND (NOT B)`. In this computation, the intermediate `NOT`-values are used, but the final `OR`-value is with respect to the initial values `A` and `B`. Thus, for each new logical computation we check that it is a computation based on any of the initial bit-string(s). Resources are depletable and finite in the topology, hence it has to be continuously seeded with new resources. The seeding rate of new resources, performed for each new scheduling of the population, is given as a parameter to the experiments.

6.4 Experiment: Sufficient conditions for symbiosis

To validate the claim that dimensional difference and resource relations are both sufficient for symbiosis in our model, the 4 possible settings in Table 6.1 were independently tested. For each setting (a–d), two evolutionary measures were used to characterize the adaptive behavior. Note that each evolutionary measure is based on data sampling with parameterized granularity.

Table 6.1 The 4 possible settings for dimensional difference and resource relations.

Resource Relations	**Layers**	
	Single	*Multiple*
No	(a)	(c)
Yes	(b)	(d)

The table shows the 4 possible settings for dimensional difference and resource relations.

Figure 6.2 Illustration of accumulative graphs.

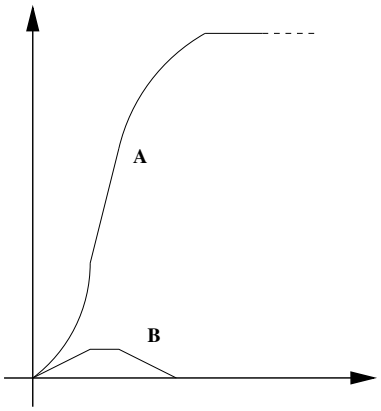


Illustration of an accumulative function. Function A shows the accumulative quantity of function B. Note that function A is derived from the area covered by function B.

Accumulative phenotype is an *accumulative* measure of the highest computed logical functions (based on merit) in the population. To illustrate, Figure 6.2 shows two functions; function A shows the accumulative quantity of function B. The motivation behind accumulative quantities instead of absolute, is to increase the illustrative clarity by plotting the data to strictly higher values in both axis of the diagram.

Vicinity difference is a relative measure of the difference in positions between two populations' k most frequent genotypes ($k = 1$ below). Again, the syntactic identity relation ($\doteq$) is used. Algorithmically, vicinity difference between two populations A and B is computed as follows. First, for each population A and B, construct a matrix M of the same size as the topology T, with 1s in the positions $M(i,j)$ of the most frequent genotype g (when $k = 1$):

$$\forall i,j : M(i,j) \leftarrow \begin{cases} 1 & , \ if \ T(i,j) \doteq g \\ 0 & , \ otherwise \end{cases} . \tag{6.1}$$

Second, for each position with a 1-digit, add one for each position in the 2-Moore

neighborhood with a non-zero value, i.e. position of most frequent genotype. Hence, each position in the computed matrix M will now be a number between 0 and 25:

$$\forall i,j : M(i,j) = 1 \Rightarrow M(i,j) \leftarrow \sum_{\forall x,y \in 2-Moore(i,j)} \begin{cases} 1 & , \ if \ M(x,y) > 0 \\ 0 & , \ otherwise \end{cases}, \quad (6.2)$$

where $2 - Moore(i,j)$ is the set of positions (x,y) in the 2-Moore neighborhood around the position (i,j). Let M_A and M_B be the final matrices for populations A and B, respectively. Third, we sum up the absolute difference for each position in the matrices:

$$VD = \sum_{\forall i,j} \|M_A(i,j) - M_B(i,j)\|. \quad (6.3)$$

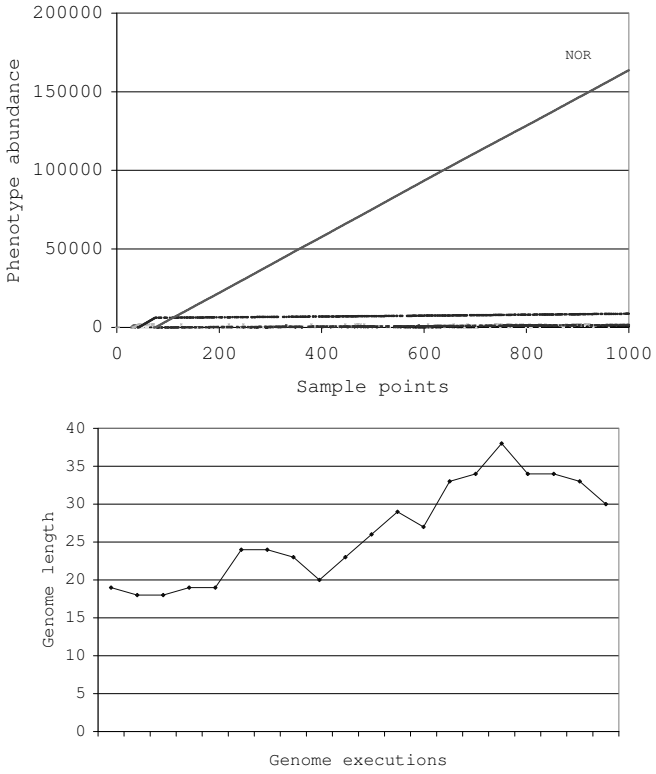
This sum is the *vicinity difference* (VD) between populations A and B. It is a measure of how correlated the most frequent genotypes are in the "resource dimension".

6.4.1 (a) Single layer without resource relations

First, we looked at what dynamics a single layer of chemostat (T_0) exhibits without any resource relations. Instead of linking resources between organisms, via the topology and the I/O instructions, fresh random resources were given for each executed input instruction. The topology was a single layer of size 28^2 (to later be compared with two layers of size 20^2; $28^2 \approx 2 * 20^2$, see case (c) below), seeded with the default non-computing self-replicator with a sequence of quasi-neutral pattern instructions. The mutation rate was 0.000625 and 0.00025, for substitution and insertion/deletion, respectively. For this and below experiments (b–d) the default merit function in Table 4.1 was used.

Figure 6.3 shows the accumulative phenotype activity and the average genotype length during evolution without resource relations or layers, based on one random run. It is easy to see that the phenotypes very soon converge to a semi-optimal solution (phenotype NOR). The genotype length fluctuates but increases slightly from 20 to finally 30 instructions. Since evolution is stuck in the phenotype NOR, the accumulation of extra instructions is probably a random event (genetic drift) due to the relatively low mutation rate.

Figure 6.3 Accumulative phenotype abundance and average genotype length, without resource relations or layers.



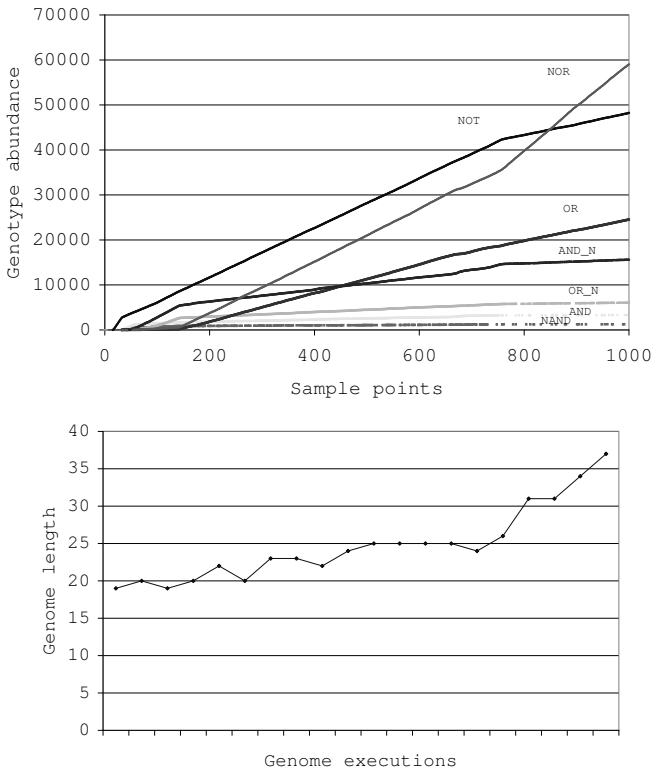
The figure shows the accumulative phenotype abundance and average genotype length, evolved in the model without resource relations or layers.

6.4.2 (b) Single layer with resource relations

Second, using the same settings as in the above experiment (a) we looked at how the dynamics changed by adding resource relations between organisms. Instead of communicating the resources between the layers as below, resources were taken (input) and given (output), in the same layer and in the 1-Moore neighborhood. The seeding rate was 1.0, i.e. the topology was fully seeded each time the population was completely scheduled, i.e. after each organism had executed once. The preference mutation rate per replication was 0.02.

Figure 6.4 shows the accumulative phenotype activity and average genotype length during evolution with resource relations, but without layers. Compared to the above case without resource relations, resource relations give a richer phenotype variance. With resource relations, it was found that the genotypes had a growing number of INPUT and NAND instructions; illustrated in the second panel of Figure 6.4. The extra instructions accumulate probably due to the (highly) changing input and output

Figure 6.4 Accumulative phenotype abundance and average genotype length, with resource relations but without layers.



The figure shows the accumulative phenotype abundance and average genotype length, evolved in the model with resource relations but without layers.

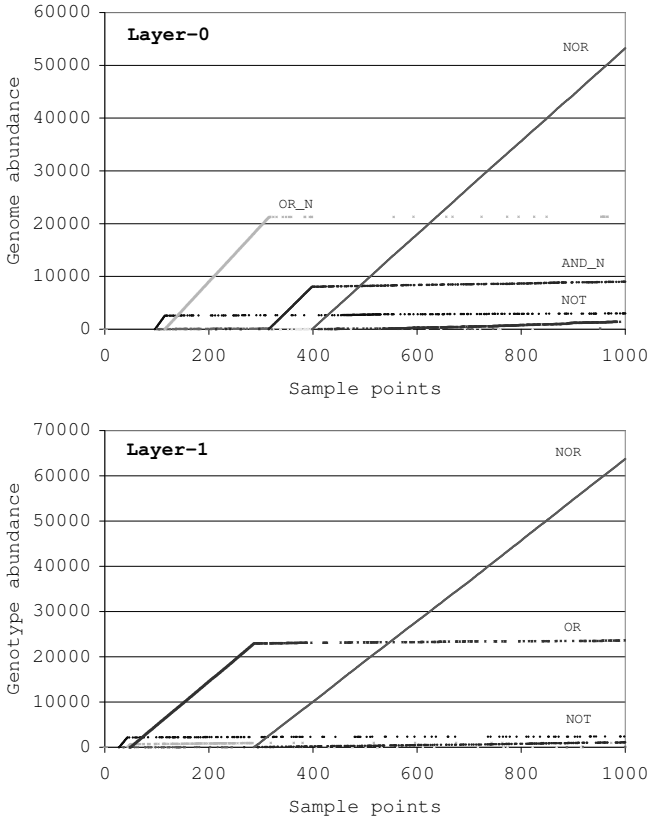
preferences that destroy evolved merit-giving INPUT-NAND-OUTPUT sequences such that new such sequences have to evolve. Note that there is no sign of symbiosis since the chemostat converges to the secondary most fit phenotype NOR (Competitive Exclusion Principle). Note also the exclusive relations between the evolving phenotypes, as soon as a new phenotype increases in abundance the others decrease; see e.g. NOT and NOR in Figure 6.4.

6.4.3 (c) Multiple layers without resource relations

Third, we added a second chemostat layer (T_1) without resource relations as an intermediate step towards symbiosis and co-evolution. Each layer (T_0, T_1) was a topology of size 20^2 with the same mutation rates and number of genome executions as above experiments (a) and (b).

Figure 6.5 and Figure 6.6 shows the accumulative phenotype activity and average genotype length (respectively) during evolution without resource relations, but with

Figure 6.5 Accumulative phenotype abundance, without resource relations but with layers.



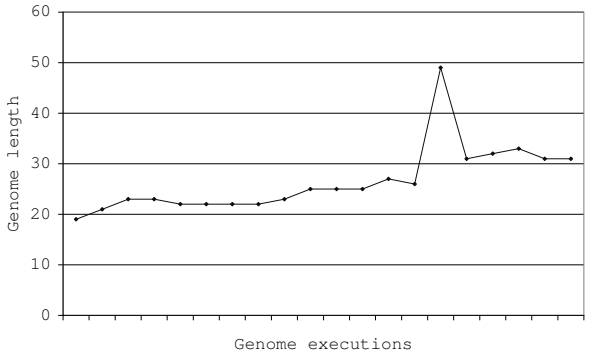
The figure shows the accumulative phenotype abundance for Layer 0 and Layer 1, evolved in the model without resource relations but with layers.

layers. Similar observations as in case (a) can be made. Average genotype length (in both layers) evolves in about the same way as in case (a). Note that the total number of organisms is about the same in the layered and the non-layered case ($28^2 \approx 2 * 20^2$). Note also that there are no signs of symbiosis since both layers evolve the same semi-optimal solutions (phenotype NOR) without any visible co-evolution in the phenotype activity.

6.4.4 (d) Multiple layers with resource relations

Finally, we included both resource relations and multiple layers (T_0 and T_1) to show the dynamics of symbiosis and co-evolution. Apart from the layers, the settings were the same as above. The seeding rate was 0.1, and the mutation rate of the preference parameters was 0.001 per replication. Note that since the resources are depletable, the seeding rate is important to force the populations to adapt their resources to each other.

Figure 6.6 Average genotype length, without resource relations but with layers.



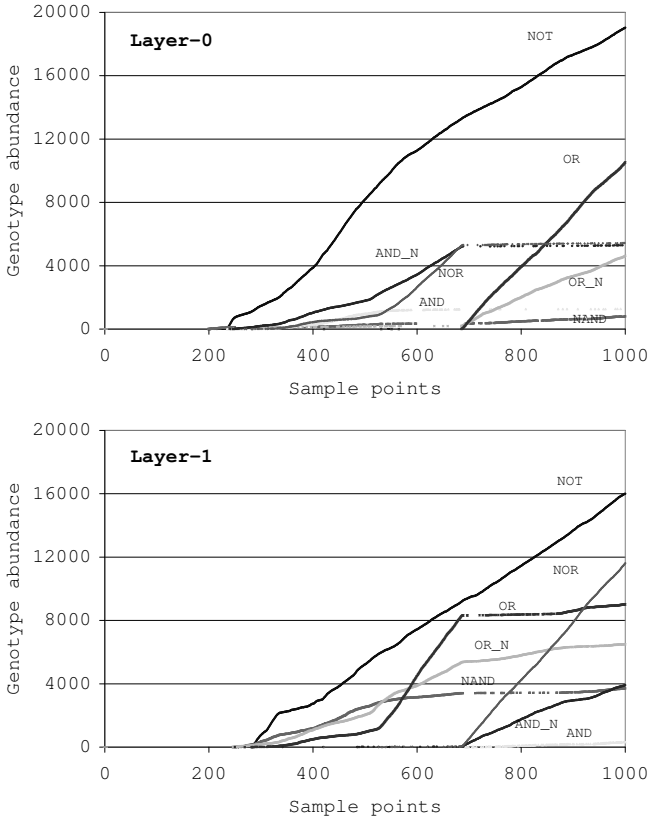
The figure shows the average genotype length, evolved in the model without resource relations but with layers.

Too high seeding rate (~ 1) minimizes the populations resource interdependence, and too low rate (~ 0) creates a very fluctuating environment to adapt to. The mutation rate of the preference parameters is also very important since it controls the level of resource noise between the layers.

Figure 6.7 and Figure 6.8 shows the phenotype activity and average genotype length (respectively) during evolution with resource relations and layers. Even though the accumulative phenotypes are in much lower quantity than in case (c), probably due to the differences in the availability of resources, it is easy to see that the phenotype diversity of logical computation is higher, i.e. there are more phenotypes with positive accumulative growth. The genotype length is relatively low compared to earlier cases (a–c), probably because the organisms can link resources between the layers, and together compute the logical functions (semi-optimally in this case). Although, it was observed that with symbiosis, a very high average genotype length evolved in the majority of the cases (over 10 differently evolved populations). This was the case when the two layers adapted well to each other, e.g. in terms of vicinity difference (see below). This genotype increase can be explained by the continuously changing input and output preferences together with the changing resource relations between the layers (see also case b above).

It is interesting to note that in Figure 6.7 there are indeed evidence for symbiosis and co-evolution. For example, at $x = 700$, the phenotypes OR and OR_N in layer-0, and the phenotypes NOR and AND_N in layer-1 co-evolve; both pairs of phenotypes grow with about the same rate relative to each other. At a closer look, we see that it is the logical functions OR and OR_N in the first layer that is used for computing NOR

Figure 6.7 Accumulative phenotype abundance, with resource relations and layers.



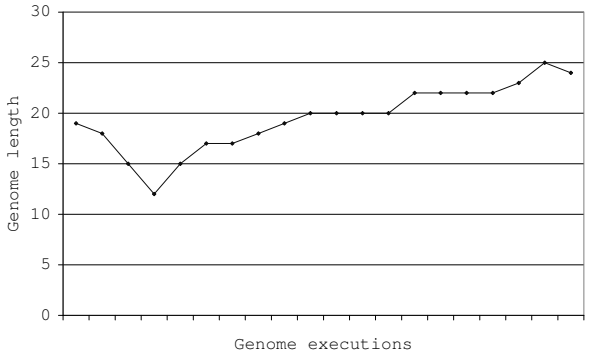
The figure shows the accumulative phenotype abundance in Layer 0 and Layer 1, evolved in the model with resource relations and layers.

and AND_N in the second layer; NOR and AND_N are computed by applying NAND to OR and OR_N, respectively. Hence, since symbiosis creates higher (phenotype) diversity and increasing genotype size by mutual adaptation than without symbiosis, evolutionary complexity is increased in our model.

Vicinity difference and the shadow model

To measure the topological correlation between two co-evolving populations in case (d), we used the earlier discussed vicinity difference measure. For this, 10 random populations evolved, and were compared to a resource-neutral *shadow model* as follows. For each run, two pairs of independent chemostat layers were evolved; one pair was the actual resource-interconnected model and the other was the shadow model. The shadow model evolved under the same conditions as for the actual model, but with a different random seed. Continuously, the vicinity difference between resource-interconnected layers in the actual model (VD_a), and between the first layers in the actual model and

Figure 6.8 Average genotype length, with resource relations and layers.



The figure shows the average genotype length, evolved in the model with resource relations and layers.

the shadow model (VD_h), were both computed with parameterized sampling rate. The seeding rate was set to 0.1, and the mutation rate of the preference parameters was set to 0.001. As already discussed, symbiosis and co-evolution is balanced by both the seeding rate and the mutation rate of the preference parameters.

It was found that 8 out of the 10 evolved populations had higher topological correlation between resource-interconnected layers in the actual model, than between non-interconnected layers between the actual model and the shadow model; i.e. $VD_a < VD_h$. On average, the resource-interconnected layers had 90% of the vicinity difference between the actual and the shadow model. Note that the sampling resolution is a major bottleneck in the vicinity measure; higher sampling rate will give higher difference between VD_a and VD_h. Higher sampling resolution was not used due to impractically long execution time.

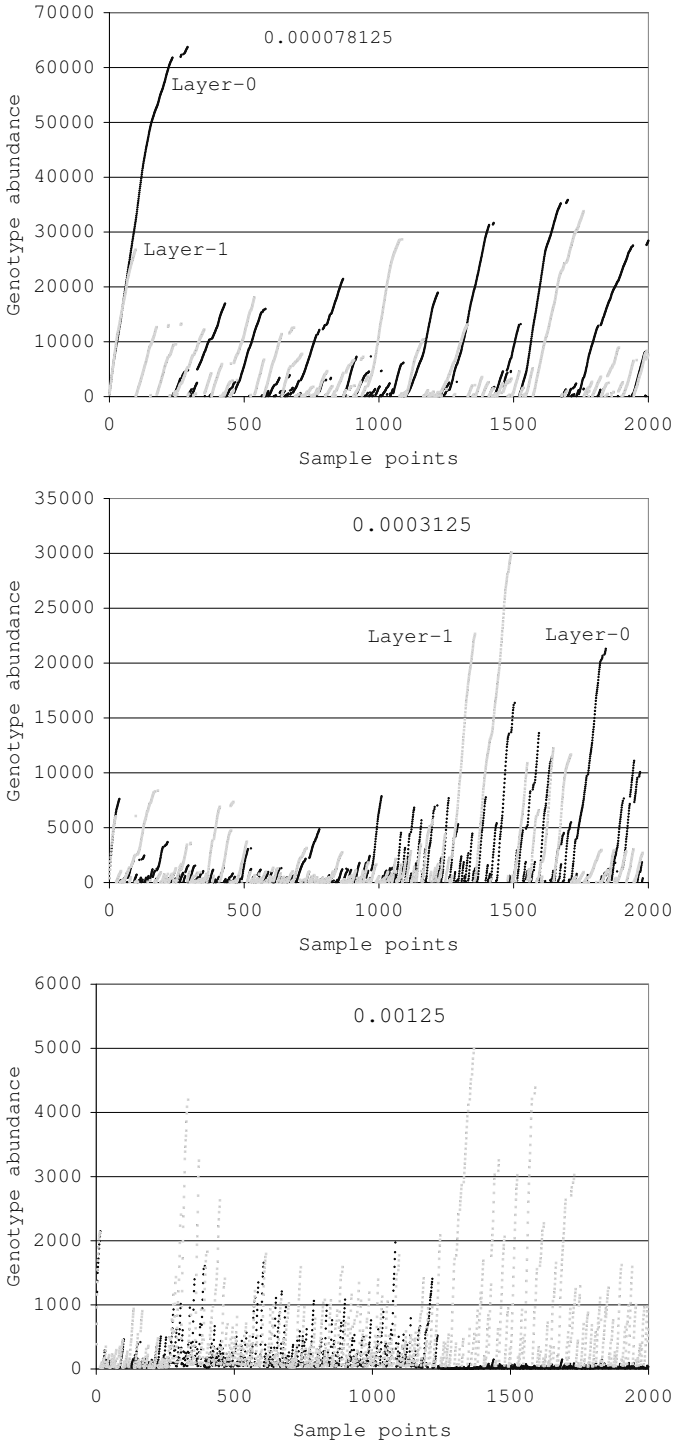
6.5 Mutational pressure with symbiosis

By varying the mutation rate, we studied the dynamics of two resource-interconnected populations by using an *evolutionary activity* measure. Evolutionary activity is an accumulative measure of the k most frequent genotypes [BSP98, BB99]. Here we looked at the most frequent genotype ($k = 1$). The syntactic identity relation $\doteq$, defined in Section 2.3, was used to classify genotypes in each population to their equivalence groups.

Figure 6.9 and Figure 6.10 show the mutational pressure on two populations in symbiosis with 5 different mutational rates: $7.8125 * 10^{-5}$, $3.125 * 10^{-4}$, $1.25 * 1-^{-3}$,

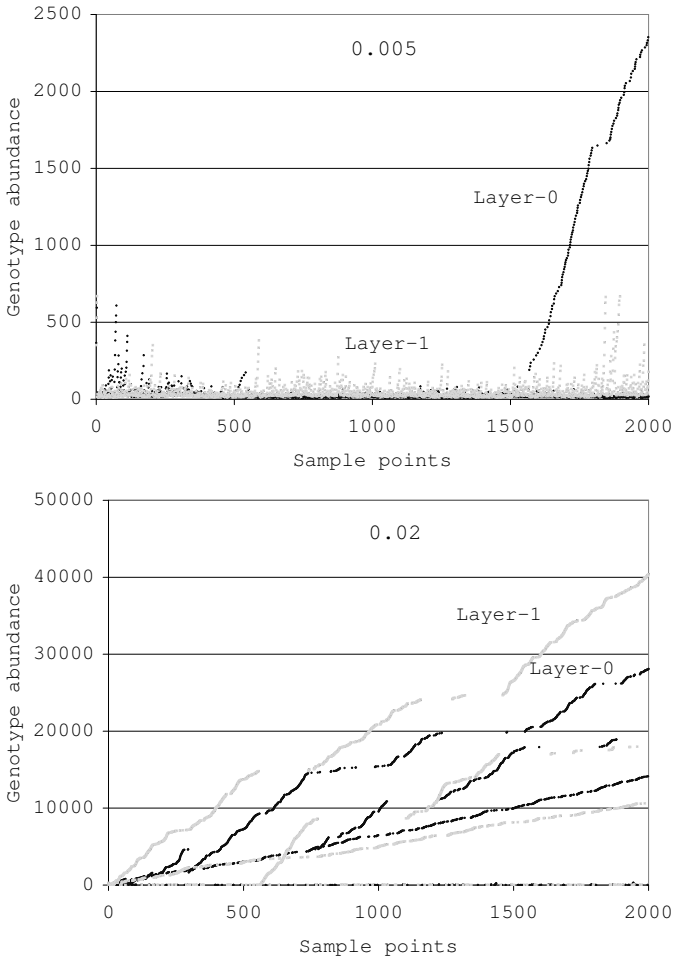
$5 * 10^{-3}$ and $2 * 10^{-2}$. For the lowest mutation rate of $7.8125 * 10^{-5}$, co-evolution is illustrated in the growth of the most frequent genotypes. For example, there are two such illustrative cases between the sample points $x = 1000$ and $x = 1500$, where the most frequent genotype in each layer occur at the same time, and grows with about the same rate. When the mutation rate increases, the genotype correlation becomes less obvious. At the mutation rate of 0.005, the mutation rate is so high that initially there are no continuity. At the sample point $x = 1500$, there is one successful genotype in one of the layers. At a closer look it is a simple self-replicator without any logical computation at all. Finally, at the highest mutation rate of 0.02, each layer is governed by tiny self-replicators that continue to persist, although with minor gaps. These gaps are formed by point-mutations that mutates back and forth among a few similar genotypes (shown as almost parallel lines). It can also be observed that there are 3 different versions of these tiny self-replicators in each layer that are mutated in-between.

Figure 6.9 Low mutational pressure during evolution of two populations in symbiosis.



The figure shows evolutionary activity with low mutational pressure (substitution rate is $7.8125 * 10^{-5}$, $3.125 * 10^{-4}$ and $1.25 * 1-^{-3}$, top-down) during evolution of two populations in symbiosis (i.e. with resource relations and with layers).

Figure 6.10 High mutational pressure during evolution of two populations in symbiosis.



The figure shows evolutionary activity with high mutational pressure (substitution rate is $5 * 10^{-3}$ and $2 * 10^{-2}$, top-down) during evolution of two populations in symbiosis (i.e. with resource relations and with layers).

Chapter 7

Red-Queen Evolution

After modeling the relation between evolving populations and biotic environment in the previous chapter, we here introduce abiotic environmental relations in what-we-call *Red-Queen evolution*, to possibly create *open-ended evolution*. In efforts to synthesize what has been called open-ended evolution, researchers have emphasized important relationships between organisms and their environment (biotic and abiotic) [HJF97, Ray95, Tay02]. Open-ended evolution may be found by asking under what conditions will an evolutionary system continue to produce novel forms with "increasing complexity"? [Sta03]. Most platforms have approached open-ended evolution by presenting different kinds of optimization problems for the system to solve via an explicit merit function. These approaches have been shown to asymptotically reach a limited level of complexity in whatever way complexity has been defined [Tay02]. Other approaches like Tierra, do not define any explicit merit function, but instead complexity arise through emerging ecologies of interacting organisms. Similarly, these systems have been shown to converge to fairly low complexity (genotype size and diversity) [Sta03].

Overview This chapter will focus on the relationship between an evolving population and its abiotic environment, to possibly increase the evolutionary complexity. In a simple model, it will be shown that a cyclic fitness-dependency between environment and population can (theoretically) create "open-ended" evolution to indefinite "complexity". Our model illustrates a Red-Queen scenario where the average genotype size and accumulative genotype diversity may grow indefinitely. In the first section, a discussion about open-ended evolution is held. In the second section, the Red-Queen hypothesis is

explained in the light of open-ended evolution in a simple abstract model. In the third section, a modified version of SALTA is presented that implements the abstract model for open-ended evolution. And finally in the last section, the experimental results are presented.

7.1 Open-ended evolution

It has been claimed that what separates evolution in Artificial Life from biological evolution is the characteristics of *open-endedness*. From a biological perspective, open-ended evolution means *unbounded diversity* and *continuous novelty/complexity growth*. Such open-ended evolution with respect to these measures have been observed in the Phanerozoic fossil record, but never in Artificial Life [BSP98]. It is interesting to note that it may not necessarily be the case that biological evolution is open-ended, i.e. open-endedness may not be a necessary condition for complex evolution. Observations both in Artificial Life and in the Phanerozoic fossil record, are categorized as "long-term" observations of growing diversity, but this is simply an inductive generalization. For example, the Phanerozoic biosphere may not exhibit growing diversity forever; implying bounded diversity, and bounded evolution.

Previously, in Chapter 4 and Chapter 6, we measured evolutionary complexity based on both genotype/phenotype diversity and average genotype size during adaptation to a pre-defined merit function, for logical computation. In this chapter, we will focus only on average genotype size for the notion of complexity since diversity will always be limited in a finite topology.

7.2 Red-Queen hypothesis and evolution

The Red-Queen hypothesis, first stated by von Valen [vV73], got its name from Lewis Carrol's novel "Through the Looking Glass". In a section of the novel, Carrol describes a conversation between Alice and the Red Queen in which the Red Queen tells Alice that she has to run, just to stay in place. This scenario has become popular in describing how co-evolution may force two symbionts to evolve, just to keep their fitness. This hypothesis has successfully been applied to artificial co-evolution between populations of symbiotic organisms (de Bary's definition of symbiosis) [Wil03]. The hypothesis may

as well be applied to a relation with an ever-changing *abiotic* environment. In particular, we will look at the Red-Queen hypothesis applied to populations that affect their own abiotic environment, and in turn their own fitness.

7.2.1 Model for evolution in an ever-changing environment

To create an open-ended evolution by a Red-Queen scenario is to model a cyclic fitness-dependency between the environment and the evolving population. This is accomplished by defining a merit function that forces the population to evolve to a continuously changing abiotic environment, which changes as a result of the activity performed by the populations to receive merit and increase in fitness.

Assume that we give extra merit to organisms that can *metabolize* resources in the environment by a metabolizing function (τ). Let $R = \{r_0, r_1, \ldots, r_n\}$ be a strict partially ordered set of resources (under $<$) that can be taken from the environment, metabolized for merit and returned to the environment. Let $\omega : \mathbb{A} \to R$, be a preference function that maps genotypes to metabolizable resources. Then we may define a metabolizing function as follows.

Definition 7 ***Metabolizing function*** *$\tau : R \to R$, is a function from "food" to "waste" such that*

$$\tau(r) = r' \Leftrightarrow r < r' \wedge r = \omega(g), \tag{7.1}$$

where $r, r' \in R$ are resources and $g \in \mathbb{A}$ is a genotype.

Further, if we assume that the organisms need longer genotypes for metabolizing larger resources (r_i is larger than r_j iff $r_j < r_i$), evolution will drive the population to longer genotypes, and increasingly larger resources in the environment.

7.3 Modifying SALTA

Some minor changes are made in SALTA to implement the abstract model in Section 7.2.1, in particular how resources are communicated with the environment and how they give merit. Also, the topology (T in *CHEM*) is extended to hold resources as well as organisms, similar to the symbiotic model; see Section 6.3.2.

7.3.1 Resources

Similar to the previous versions of SALTA, resources here are used by the organisms to possibly reward themselves with merit. Although in this version of the platform, we are not considering logical computation, instead we use an abstract type of resources that can be used with the metabolizing function τ defined above. To make things concrete, resources are implemented as natural numbers ($R = \mathbb{N}$), i.e. the bit-strings are made to represent numbers. Resources are modified by the incremental instruction (`INC`), and are ordered by the less-than relation ($<$) over the natural numbers (gives a strict partially ordered set) to satisfy the Equation 7.1 for the metabolizing function.

7.3.2 I/O Buffer

Compared to the default version in Chapter 3, the I/O Buffer is here slightly modified. It is not initialized to any values (resources) but holds a list of values taken during the genome executions. The I/O Buffer is also modified such that resources are communicated to the environment at the very position of the organism, by the `INPUT` and `OUTPUT` instructions (0-Moore neighborhood), in contrast to the 1-Moore neighborhood in Chapter 6. Outputs are also only returned to the environment as "waste" if successfully metabolized by the metabolizing function (τ).

7.3.3 Merit function

Instead of giving a predefined merit for logical computation, a merit (higher than 1) is given to those organisms that metabolize resources with the τ function above. Note that there is only one rewarded merit that is higher than the default merit of 1, since there is only one metabolizing function (τ). The τ function, that is used for the experiments below, is implemented as follows.

- Patterns are used to encode metabolizable resources in the genome. In the below experiments, unary encoding of natural numbers by sequences of a special pattern was the evolutionary easiest approach. For example, a genome with k number of the special pattern can metabolize the resource represented by the natural number k. Binary encoding can also be used, but is harder to evolve. The function ω maps sequences of patterns to such metabolizable resources.

- In order to metabolize, and be rewarded a merit higher than the default, the output w ("waste") has to be one number greater than the input f ("food"), i.e. $w = f + 1$ (see $<$ in the definition of τ in Equation 7.1). Hence, organisms either replicate with default merit of 1, or with a higher parameterized merit that is rewarded for metabolizing the resources.

7.4 Experiment: open-ended evolution

We used this modified version of SALTA to experiment with the possibility of "open-ended" evolution. It is easy to see that the populations will continuously adapt to the ever-changing environment of natural numbers. By initially seeding with 1's (bottom element in the order), the population will "count numbers" indefinitely (theoretically at least, as we shall see). This type of open-ended evolution is perhaps not very interesting, although it may be classified as open-ended if we accept that growing genotypes in general signifies growing complexity, i.e. for the present we ignore diversity. Note that diversity may be said to increase indefinitely as well (theoretically), if we look at the accumulative number of different evolving genotypes.

7.4.1 Experimental observations

Even though the system has the theoretic possibility to evolve indefinitely large genotype sizes, Eigen's error threshold will limit the evolution. For a certain mutation rate, Eigen's error threshold predicts an effective upper limit to the genotype length (see Equation 4.4).

When running the system as defined above—to "count numbers"—it was soon realized that the evolving genotype was limited to a smaller genotype size than predicted by Eigen's error threshold. It was also found that the evolved genotype size was dependent on the population size. What actually was observed was that the genotype evolved up to a certain length, and then sharply started to decrease. It was obvious that evolution traversed the fitness landscape up to a point where genetic drift (see Section 2.5.1) dislocates the population indefinitely. To find out what affected this unexpectedly early dislocation, we conducted a number of experiment with different settings of population size (s), merit (m) and offspring dispersal. Note that *Eigen's error threshold is not*

related to population size.

Settings In all runs, the mutation rates of 0.000625 for substitution, and 0.00025 for insertion/deletion were used. A total of 10 populations evolved, for each combination of the parameters; population size ($s \in \{10, 20, 30, 40\}$), merit $m \in \{2, 4, 6, 8\}$ and offspring dispersal (in local or non-local neighborhood). The seeding self-replicator had 9 quasi-neutral pattern instructions (different from the unary-encoding pattern), and was from the beginning not able to metabolize any type of resources. An extra pattern was given to the system, for the unary encoding of metabolizable resources (`PATTERN ONE`) together with an extra instruction for incrementing the resources (`INC`). Each population evolved for $1 * 10^7$ genome executions.

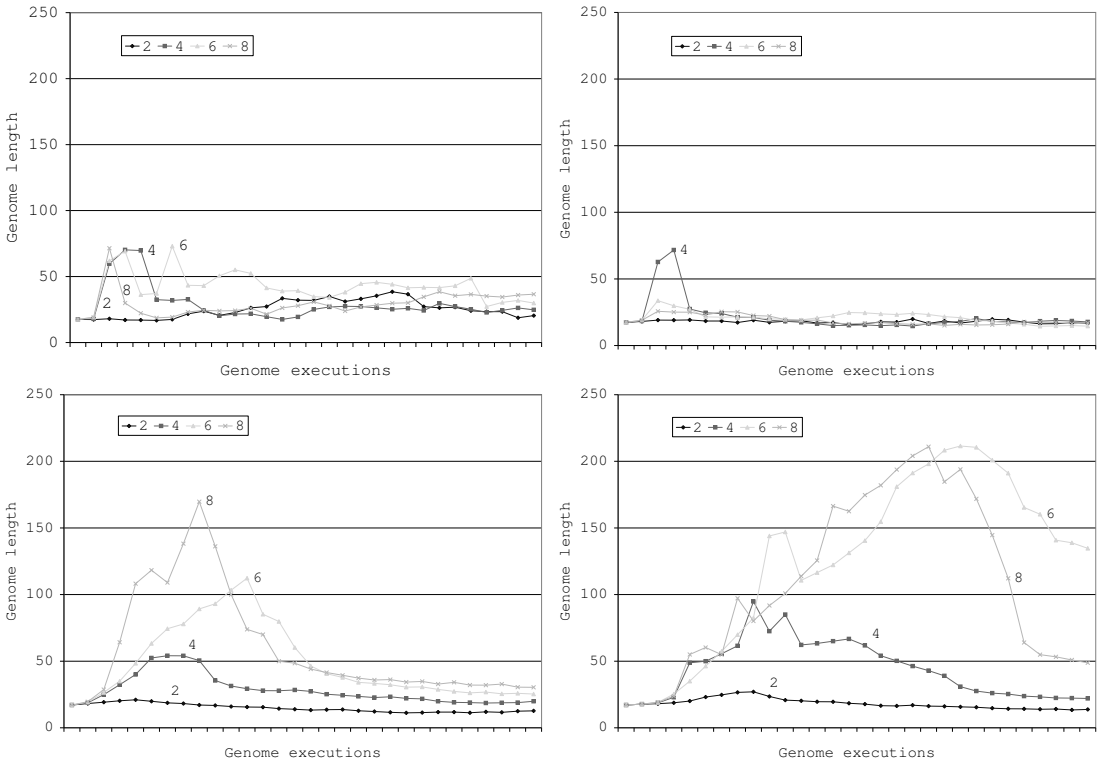
Local offspring dispersal

The first set of runs investigates how open-ended evolution is affected by genetic drift by local dispersal (2-D topology) of offspring in the 1-Moore neighborhood. Figure 7.1 shows the average genome length of continuous evolution in the ever-changing environment, for the different population sizes and merits.

Observations Some obvious observation can be made:

- For the two smallest populations, of size 10 and 20, the effect of genetic drift is stronger than the effect of natural selection, and resulting in almost no evolution.
- Larger population sizes creates a significantly higher defense against genetic drift due to higher diversity, and the effect is obvious for all merit settings. Larger population sizes also naturally delays adaptation since larger populations consume more genome executions over time.
- In general, higher merit gives larger genotypes, and seems to combat genetic drift better than lower merit. This is most likely a result of stronger natural selection due to increased fecundity by increased merit.
- Higher merit generally results in faster adaptation, but also faster (earlier and steeper) genetic drift, especially when comparing the merit of 6 and 8 in the 2 largest populations (note that high merit typically results in high fecundity).

Figure 7.1 Average genotype length with local offspring dispersal, evolved with different population sizes and merits



The figure shows the evolved average genotype length with local offspring dispersal, evolved in population sizes of 10^2 (upper-left panel), 20^2 (upper-right panel), 30^2 (lower-left panel) and 40^2 (lower-right panel), and with merits 2, 4, 6 or 8.

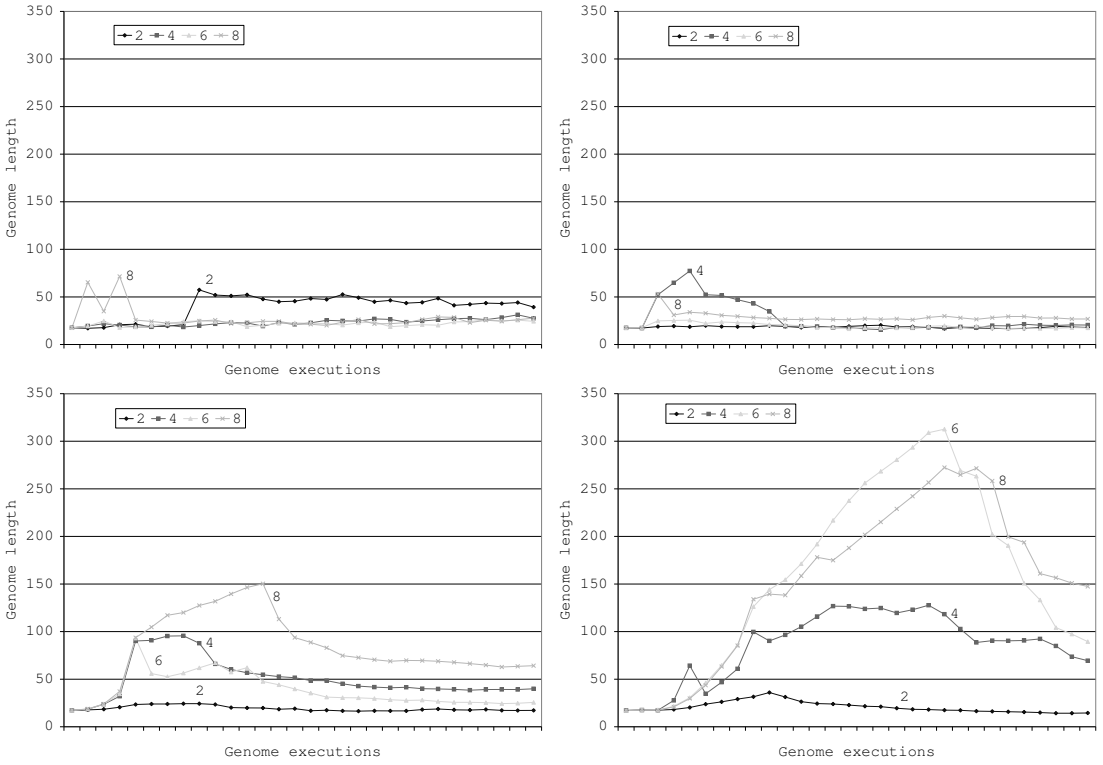
Higher merit gives faster propagation of novelty, but when genetic drift comes in, accumulation of deleterious mutation also seems to become more rapid.

Non-local offspring dispersal

The second set of runs looks at what difference *non-local* dispersal (0-D topology) of offspring makes to open-ended evolution, and genetic drift compared to the first set. Table 7.2 shows the results, for the same settings as in the first set of runs except for the type of offspring dispersal.

Observations When comparing the effect of non-local offspring dispersal to local dispersal, it can be seen that in all settings the effect of genetic drift is weaker for the former, i.e. the genotype length can grow significantly longer with non-local dispersal (most obvious in the population of size 40^2). This indicates that the effect of selection is comparatively stronger than genetic drift in populations of non-local dispersal; compared

Figure 7.2 Average genotype length with non-local offspring dispersal, evolved with different population sizes and merits



The figure shows the evolved average genotype length with non-local offspring dispersal, evolved in population sizes of 10^2 (upper-left panel), 20^2 (upper-right panel), 30^2 (lower-left panel) and 40^2 (lower-right panel), and with merits 2, 4, 6 or 8.

to local such. One reason for this may be due to increased genetic drift caused by locality in the case of local dispersal. The same observations as for the first set of runs also hold for the second set.

Error threshold and genetic drift

Note that the error threshold induced by the mutation rate does limit the genome length, although genetic drift (see Section 2.1) puts a limit to the evolution much earlier. Equation 4.4 gives the theoretical values for the error threshold, given the merit used and mutation rates. Table 7.1 gives the theoretical threshold by Equation 4.4 (Threshold) and the maximum genome length, for both sets of the experiments (MaxGenLen_1 and MaxGenLen_2), and for the different merits (Merit) in the population of size 40^2.

Thus, even though the model exhibits the theoretic possibility of open-ended genotype growth, genetic drift and the error threshold puts an effective limit to evolution. In terms of evolutionary complexity, we can initially observe increasing complexity through

Table 7.1 Experimental and theoretical genome length.

Merit	**MaxGenLen$_1$**	**MaxGenLen$_2$**	**Threshold**
2	27	36	616
4	95	128	1232
6	212	313	1592
8	211	272	1848

Experimental and theoretical genome length for the population of size 40^2, with local and non-local offspring dispersal (MaxGenLen$_1$ and MaxGenLen$_2$, respectively).

continuously growing genotype, but only up to a certain level. For the organisms to surpass this limit that prevents further evolution, mechanisms to lower the error rate such as meiosis have to evolve, and/or mechanisms to increase the variation.

Chapter 8

Conclusion

8.1 Summary

In the last four chapters, we have presented experiments to explore properties for evolutionary complexity. Each experiment was set up with its own version of the new evolutionary platform SALTA to investigate a certain aspect of artificial life with inspiration from biology.

As a starter, the first experiment (Chapter 4) investigated the simplest form of evolution with a pre-defined merit function, and variation through point-mutation only. We tested how mutation rate, population size and topology type (offspring dispersal in 0-D or 2-D) affected the evolution. We also reproduced published results concerning "complex features" by Lenski *et al.* It was found that all the 3 parameters have a significant impact on evolution. It was also found that there was an optimal population size for the given mutation rates. In terms of evolutionary complexity, it was noticed that both (phenotype and genotype) diversity and average genotype size decreased to a lower bound; see e.g. Figure 4.5 and Figure 5.6.

In the second experiment (Chapter 5), we introduced genetic recombination as a second factor to mutation for genetic variation, to compare them for the evolution of "complex features" (logical functions). Recombination combines existing genes/alleles in contrast to random mutation, for variation in the evolving population. It was found that recombination increased the rate of adaptation, made it possible to bridge evolutionary gaps, and could even drive evolution without mutation (given enough initial genetic material). In terms of evolutionary complexity, recombination increases both genotype

size and (genotype) diversity, the former due to genome mergers, and the latter due to the added noise recombination; see also Figure 5.6.

In the third experiment (Chapter 6), we looked at sufficient conditions for symbiosis. The model was extended in 4 steps, starting with 1 chemostat layer without any resource relations, and ending with 2 chemostat layers and resource relations. Different settings of mutation rate were also tested to visualize co-evolution with evolutionary activity. By using other measures, evidence of co-evolution and symbiosis was only found in the model with multiple layers (dimensional difference) and resource relations. Since the merit function is finite, there is no possibility of continuously growing diversity, although the (phenotype) diversity is higher with symbiosis than without; see Figure 6.7. It can also be seen that the average genotype size increases, e.g. in Figure 6.8, due to the accumulation of `INPUT` and `NAND` instructions as a result of the continuous adaptation between the populations and the resource relations.

In the final experiment (Chapter 7), we tried to create open-ended evolution by modifying the merit function and the resources to achieve continuous genotype growth. The scenario is based on the Red-Queen hypothesis, where the population and the abiotic environment evolve together to more complex resources (with respect to the metabolize function), and longer average genotypes in order to stay fit. In the implementation of the proposed framework for open-ended evolution, it was found that genetic drift limits the possibly indefinite genotype growth. We investigated how population size, merit (fecundity) and offspring dispersal are related to genetic drift. It was found that larger populations, higher merit and non-local offspring dispersal create a higher threshold for genetic drift. Since there is no possibility for sex or recombination here, genetic drift unavoidably limits indefinite genotype growth and open-ended evolution. Also, since the average genotype size grows until reaching the limit, we may characterize the accumulative diversity as increasing as well (until dislocation).

To summarize, Table 8.1 presents the different experiments (Exp) and their purpose (Purpose). Together with obtained results (Results), i.e. primary results, secondary results and results for the two complexity measures; diversity (Div) and average genotype length (Len).

8.2 Contributions

We reproduced gradual evolution of logical functions (so-called "complex features"), published by Lenski *et al.* [LOPA03], in a similar evolutionary platform that we called SALTA. Thereafter this model was enriched with genetic recombination, which was shown to be more efficient than mutation, for evolving the "complex features".

SALTA was also extended to capture interdependencies between the environment and the evolving population, and to create *evolutionary complexity* in a finite population. We defined evolutionary complexity as the product of (genotype or phenotype) diversity and average genotype size, and showed that symbiosis and Red-Queen evolution creates such evolutionary complexity, for biotic and abiotic interdependencies, respectively.

8.3 Discussion

It has been claimed that replication, variation and selection are enough for evolution. Still, there has been no success in creating any really interesting evolution in Artificial Life. The basic problem seems to be in how to implement fitness, since natural selection is what really drives evolution. By defining an explicit merit function to deduce fitness from, evolution will never create anything more interesting than what is loaded into the merit function. So where does *emergent* and interesting evolution in biology come from?

We would like to argue that it is circular relations between the evolving population and its environment that create evolutionary complexity. Since organism's phenotypes are evaluated in their respective environment, to give a measure of fitness, and the environment is a result of the organisms reactions for fitness, there is an effective relation between them. It is this type of interdependencies that we have tried to introduce in an artificial setting, both between organisms (symbiosis "in layers") and between organisms and their abiotic environment (Red-Queen evolution).

8.4 Critique

Research in Artificial Life is extremely diverse, and there are many difficulties in this research area. Fundamental definitions are still very much in question, e.g. what is life? If we do not know then how can life be studied? In this thesis, life is defined as any

entity which replicates, varies and is under natural selection. In particular, we looked at assembler programs, or so-called "digital organisms" [Ada98], that self-copied, mutated (and recombined) and were under natural selection via an explicit merit function. Is this how we intuitively would characterize life? We propose to narrow the scope of analogous studies, to more precisely reason about *Artificial Evolution* instead of Artificial Life.

As has also been discussed earlier, there are obvious weaknesses with an explicitly defined merit function for studying evolution. Defining such a merit function and consequently using it for imposing fitness, does not render possible emergent evolution. Hence, evolution is reduced to an optimization process in a fixed domain. We believe that fitness should not be explicitly defined, instead it should be a product of the system itself (e.g. see Tierra [Ray95]).

8.5 Future work

As a continuation of trying to find evolutionary complexity, it would be interesting to try to merge abiotic and biotic relations into one coherent framework, e.g. exhibiting both symbiosis and Red-Queen evolution.

Throughout this work, we have observed the presence of genetic drift as an opposite force to natural selection. It would also be interesting to elaborate with techniques to combat such an important and strong genetic force, e.g. by genetic recombination or sexual reproduction.

Table 8.1 Summary of experiments.

Exp	**Purpose**	**Results**	
		Primary	*Div*
		Secondary	*Len*
Mutation (Chapter 4)	To reproduce results of complex evolution by Lenski et al., with point-mutations and a finite merit function.	Complex evolution of logical functions was reproduced, with comparable results and conditions as reported by Lenski *et al.*	
		Evolution was found to be a function of mutation rate, population size and topology. It was also found that small populations gave high genetic drift, and that there was an optimal population size for the given mutation rates.	
Genetic recombination (Chapter 5)	To show that recombination is more efficient than point-mutations to evolve a finite merit function.	Recombination together with point-mutation was shown to be more efficient than point-mutation only, for the particular evolutionary landscape.	
		Recombination may be necessary for bridging evolutionary gaps, and may even drive evolution single-handedly.	
Symbiosis (Chapter 6)	To show sufficient conditions for symbiosis and evidence of co-evolution.	Three different measures were used to show co-evolution and symbiosis under the sufficient conditions of (i) dimensional difference and (ii) resource relations.	
		Evidence of co-evolution was found, as a function of seeding rate, preference mutation rate and mutation rate.	
Red-Queen evolution (Chapter 7)	To show an approach to open-ended evolution with increasing complexity in an ever-changing environment (Red-Queen hypothesis).	Formal model for open-ended evolution was implemented. The possibility of indefinite increase in genotype length was limited by genetic drift.	
		Genetic drift is strongly related to population size. Larger populations, higher merit and non-local dispersal create a higher threshold for genetic drift.	

Bibliography

[AB94] Chris Adami and C. Titus Brown. Evolutionary learning in the 2D artificial life system Avida. In Rodney A. Brooks and Pattie Maes, editors, *Proceedings of the 4th International Workshop on the Synthesis and Simulation of Living Systems (Artificial Life IV)*, pages 377–381, Cambridge, MA, USA, July 1994. MIT Press.

[Ada95] Christoph Adami. Self-organized criticality in living systems. *Physics Letters A*, 203:29–32, 1995.

[Ada98] Christoph Adami. *Introduction to Artificial Life*. Springer-Verlag, Berlin, 1998.

[Ada04] Christoph Adami. Information theory in molecular biology. *Physics of Life Reviews*, 1(1):3–22, April 2004.

[BB98] F. Bagnoli and M. Bezzi. Eigen's error threshold and mutational meltdown in a quasispecies model. *Int. Journ. Mod. Phys. C*, 9(999), 1998.

[BB99] Mark A. Bedau and C. Titus Brown. Visualizing evolutionary activity of genotypes. *Artificial Life*, 5(1):17–35, 1999.

[BF95] Lawrence Bull and Terence C. Fogarty. Artificial symbiogenesis. *Artificial Life*, 2(3):269–292, 1995.

[BP03] M. A. Bedau and N. H. Packard. Evolution of evolvability via adaptation of mutation rates. *Biosystems*, 69(2–3):143–162, May 2003.

[BSP98] Mark A. Bedau, Emile Snyder, and Norman H. Packard. A classification of long-term evolutionary dynamics. In *Proceedings of the sixth international conference on Artificial Life*, pages 228–237. MIT Press, 1998.

[CAW01] Paulo R. A. Campos, Christoph Adami, and Claus O. Wilke. Optimal adaptive performance and delocalization in NK fitness landscapes. *Physics*, 0109020, 2001.

[Dar59] Charles Darwin. *On the Origin of Species by Means of Natural Selection*. Murray, London, 1859.

[Dew84] A. K. Dewdney. In the game called Core War hostile programs engage in a battle of bits. *Scientific American*, pages 14–21, May 1984.

[DF98] Alves Domingos and J. F. Fontanari. Error threshold in finite populations. *Phys. Rev. E*, 57(6), june 1998.

[DGSR96] J. M. Daida, C. S. Grasso, S. A. Stanhope, and S. J. Ross. Symbionticism and complex adaptive systems I: Implications of having symbiosis occur in nature. In Lawrence J. Fogel, Peter J. Angeline, and T Baeck, editors, *Evolutionary Programming V: Proceedings of the Fifth Annual Conference on Evolutionary Programming*, pages 177–186. MIT Press, 1996.

[DZB01] Peter Dittrich, Jens Ziegler, and Wolfgang Banzhaf. Artificial chemistries—a review. *Artificial Life*, 7(3):225–275, 2001.

[Eig71] M. Eigen. Selforganization of matter and the evolution of biological macromolecules. *Naturwissenschaften*, 58:465–523, 1971.

[ES79] M. Eigen and P. Schuster. *The Hypercycle—A Principle of Natural Self-organization*. Springer, Berlin, 1979.

[FB94] Walter Fontana and Leo W. Buss. Arrival of the fittest: Toward a theory of biological organization. *Bulletin of Mathematical Biology*, 56:1–64, 1994.

[FWB94] W. Fontana, G. Wagner, and L. Buss. Beyond digital naturalism. *Artificial Life*, 1(1):211–227, 1994.

[GL98] Philip J. Gerrish and Richard E. Lenski. The fate of competing beneficial mutations in an asexual population. *Genetica*, 102–103:127–144, 1998.

[Gou02] Stephen Jay Gould. *The Structure of Evolutionary Theory*. Harvard University Press, 2002.

[Hel98] Stefan Helmreich. *Silicon Second Nature*. University of California Press, California, 1998.

[HJF97] Peter T. Hraber, Terry Jones, and Stephanie Forrest. The ecology of echo. *Artificial Life*, 3(3):165–190, 1997.

[Hol75] J. H. Holland. Adaptation in natural artificial systems. *International Journal of Computational Intelligence and Applications*, 1975.

[Hux59] T. H. Huxley. Letter to Charles Darwin. *The Correspondence of Charles Darwin*, 7:1858–1859, 1859.

[IK90] Takashi Ikegami and Kunihiko Kaneko. Computer symbiosis—emergence of symbiotic behavior through evolution. *Physica D*, 42:235–243, 1990.

[Ins94] Santa Fe Institute. Annual report on scientific programs, 1994.

[KJ92] Stuart A. Kauffman and Sonke Johnson. Co-evolution to the edge of chaos: Coupled fitness landscapes, poised states, and co-evolutionary avalanches. *Artificial Life II*, pages 325–368, 1992.

[KPP94] A. B. Korol, I.A. Preygel, and S.I. Preygel. *Recombination, Variability and Evolution*. Chapman and Hall, 2–6 Boundrary Row, London SE1 8HN, UK, 1994.

[KS88] Raju I. Kucherlapati and Gerard R. Smith. *Genetic Recombination*. American Society for Microbiology, 1913 I Street, N.W., Washington, DC 20006, 1988.

[Lan88] Chris Langton. Toward artificial life. *Whole Earth Review*, 58:74–79, 1988.

[LOPA03] Richard E. Lenski, Charles Ofria, Robert Pennock, and Christoph Adami. The evolutionary origin of complex featues. *Nature*, 423, 5 2003.

[LW04] Ye Li and Claus O. Wilke. Digital evolution in time-dependent fitness landscapes. *Artificial Life*, 10(2):123–134, 2004.

[Mar81] Lynn Margulis. *Symbiosis in Cell Evolution*. W. H. Freeman and Company, San Francisco, 1981.

[May81] R.M. May. *Theoretical Ecology Principles and applications*. Sinauer Associates, Oxford, 1981.

[May02] Ernst Mayr. *What Evolution Is*. Basic Books, 2002.

[McM00a] Barry McMullin. John von Neumann and the evolutionary growth of complexity: Looking backward, looking forward... *Artificial Life*, 6(4):347–361, 2000.

[McM00b] Barry McMullin. The von Neumann self-reproducing architecture, genetic relativism and evolvability. In *Evolvability Workshop at Artificial Life VII: Proceedings of the Seventh International Conference on Artificial Life*, 2000.

[MMP97] Michael T. Madigan, John M. Martinko, and Jack Parker. *Brock Biology of Microorganisms, Eight Edition*. Prentice-Hall, New Jersey, 1997.

[MPS67] D. R. Mills, R. L. Peterson, and S. Spiegelman. An extracellular darwinian experiment with a self-duplicating nucleic acid molecule. *Proc. Natl. Acad. Sci. USA*, 58:217–224, 1967.

[MS02] Lynn Margulis and Dorian Sagan. *Acquiring Genomes: A Theory of the Origin of Species*. Basic Books, New York, 2002.

[MTvK01] Barry McMullin, Tim Taylor, and Axel von Kamp. Who needs genomes? In *Atlantic Symposium on Computational Biology, Genome Information Systems and Technology (CBGI)*, pages 371–408, Duke University, USA, March 2001.

[Mul64] H. J. Muller. The relation of recombination to mutation advantage. *Mutation Research*, 1:2–9, 1964.

[Pag02] Mark Pagel. *Encyclopedia of Evolution*. Oxford University Press, New York, April 2002.

[PTJ02] Elizaveta Pachepski, Tim Taylor, and Stephen Jones. Mutualism promotes diversity and stability in a simple artificial ecosystem, 12 2002.

[Ray91] T. S. Ray. An approach to the synthesis of life. In C. G. Langton, C. Taylor, J. D. Farmer, and S. Rasmussen, editors, *Artificial Life II: SFI Studies in*

the Sciences of Complexity, Vol. X, pages 371–408, Redwood City, CA, 1991. Addison-Wesley.

[Ray95] Thomas S. Ray. An evolutionary approach to synthetic biology: zen in the art of creating life. In Christopher G. Langton, editor, *Artificial Life: An Overview*, pages 179–210, Cambridge, Massachusetts, 1995. MIT Press.

[Ray03] Tom Ray. Tierra 6.01, network version, 11 2003.

[Rid00] Mark Ridley. *Mendel's Demon—Gene Justice and the complexity of life.* Weidenfeld & Nicolson, London, 2000.

[RKFH90] Steen Rasmussen, Carsten Knudsen, Pasmus Feldberg, and Morten Hindsholm. The Coreworld: emergence and evolution of cooperative structures in a computational chemistry. In *Proceedings of the Ninth Annual International Conference of the Center for Nonlinear Studies on Self-organizing, Collective, and Cooperative Phenomena in Natural and Artificial Computing Networks on Emergent computation*, pages 111–134. North-Holland Publishing Co., 1990.

[Rya02] Frank Ryan. *Darwin's Blind Spot.* Houghton Mifflin Company, Boston, New York, 2002.

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

[Sip98] Moshe Sipper. Fifty years of research on self-replication: An overview. *Artificial Life*, 4:237–257, 1998.

[Sta03] R. K. Standish. Open-ended artificial evolution. *International Journal of Computational Intelligence and Applications*, 3(167), 2003.

[SW95] Hal L. Smith and Paul Waltman. *The Theory of the Chemostat: Dynamics of Microbial Competition.* Cambridge University Press, Cambridge, February 1995.

[Tay99] T.J. Taylor. *From Artificial Evolution to Artificial Life.* PhD thesis, University of Edinburgh, 1999.

[Tay02] Tim Taylor. Creativity in evolution: individuals, interactions, and environments. In *Creative evolutionary systems*, pages 79–108. Morgan Kaufmann Publishers Inc., 2002.

[TJ95] Charles Taylor and David Jefferson. Artificial life as a tool for biological enquiry. In Christopher G. Langton, editor, *Artificial Life: an overview*, pages 1–14, Cambridge, Massachusetts, 1995. MIT Press.

[vN66] J. von Neumann. *Theory of Self-Reproducing Automata*. University of Illinois Press, Urbana, Illinois, 1966.

[vV73] L. van Valen. A new evolutionary law. *Evol. Theory*, 1:1–30, 1973.

[Wil03] Claus O. Wilke. Does the red queen reign in the kingdom of digital organisms? In Wolfgang Banzhaf, Jens Ziegler, and Thomas Christaller et al., editors, *Advances in Artificial Life*, pages 405–414, Heidelberg, December 2003. Springer-Verlag.

[WP99] Richard A. Watson and Jordan B. Pollack. How symbiosis can guide evolution. In D. Floreano, J. D. Nicoud, and F. Mondada, editors, *Advances in Artificial Life: 5th European Conf., ECAL'99*, pages 29–38, Heidelberg, 1999. Springer-Verlag.

[WP01] Richard A. Watson and Jordan B. Pollack. Symbiotic composition and evolvability. *Lecture Notes in Computer Science*, 2159:480–490, 2001.

[WP02] Richard A. Watson and Jordan B. Pollack. *Compositional evolution: interdisciplinary investigations in evolvability, modularity, and symbiosis*. PhD thesis, Brandeis University, 2002.

[WRP00] Richard A. Watson, Torsten Reil, and Jordan B. Pollack. Mutualism, parasitism, and evolutionary adaptation. *Artificial Life*, 7, 2000.

[WWO+01] Claus O. Wilke, Jia Lan Wang, Charles Ofria, Richard E. Lenski, and Christoph Adami. Evolution of digital organisms at high mutation rate leads to survival of the flattest. *Nature*, 412, 7 2001.

[YB01] Gabriel Yedid and Graham Bell. Microevolution in an electronic microcosm. *The American Naturalist*, 157(5):465–487, 2001.

Appendix A

Settings and Parameters

By parameterizing the evolutionary system, the user can study many different kinds of evolutionary experiments. The platform has 3 ways to input parameters: command-line, initialization file (init-file) and global settings (java class-file). Each run results in a single output file.

A.1 Command line arguments

The most dynamic parameters are given as command line arguments, and together with the init-file these are the major parameters for the system. These parameters are effectively used in scripts, to conduct large scale experiments with possibly many input and output files.

A.1.1 Default SALTA

The default parameters for SALTA are the following:

- `INDATA [file]`. Specifies the init-file with path. Default value is `./initData.txt`.
- `OUTDATA [file]`. Specifies the output data file with path. Default value is `./outData.txt`.
- `SEED [number]`. Specifies the random seed to base the simulation on. Identical seeds will result in identical simulations, everything else being equal. Default value is `4711`.

- `MACROMUT`. If given, macro-mutation (recombination) will operate at given rate (`ICHANCE`). Default value is `false`.

- `ICHANCE [float]`. Specifies the rate of macro-mutation, given as a fraction between 0.0 (never) and 1.0 (always), effective for each `SET` instruction to reference a neighbouring program. Note, that the effective rate (r_i) of the i'th `SET` instruction to reference a neighbour is: $(\prod_{j=1}^{i-1}) * r_i$. Default value is `0.0`.

- `MUTCOPY`. If given, the `COPY` instruction is perturbed by point-mutations: substitution, insertion and deletion, at given rates (below). Default value is `false`.

- `SUBSRATE [float]`. Specifies the point-mutation rate of substitution, given as a fraction between 0.0 and 1.0. Default value is `0.00125`.

- `DELRATE [float]`. Specifies the point-mutation rate of deletion, given as a fraction between 0.0 and 1.0. Default value is `0.0005`.

- `INSRATE [float]`. Specifies the point-mutation rate of insertion, given as a fraction between 0.0 and 1.0. Default value is `0.0005`.

- `NRSTEPS [number]`. Specifies the total number of genome executions for the simulation. Default values is `0`.

- `STEP`. If given, lets the user run the simulation stepwise by executing single instructions in an interactive fashion (used for debugging). Default value is `false.`

- `SAMPLE [number]`. Specifies the sample rate of statistics, as number of times over total period, written to the output file. Default value is `1`.

- `ISTART [float]`. Specifies a start for macro mutation, given as a fraction of remaining genome executions, e.g. 0.8 start macro mutation when 80% of the given genome exections are remaining. Default value is `0.0`.

- `IEND [float]`. Specifies an end for macro mutation, given as for `ISTART`. Default value is `0.0`.

- `RINIT`. If given, creates a randomized initialization population instead of the one specified in the init-file. Default value is `false`.

- `NOT_LOCAL`, specifies if the offspring should be placed in the neighborhood or randomly in the population. Default set to `false`.

A.1.2 SALTA with symbiosis

SALTA was extended from the default model with the following parameters for symbiosis:

- `SEEDRATE [float]` Specifies the seeding rate of new values (bit-strings) randomly placed in the topology, given as a fraction of the topology to seed for each whole scheduling of all organisms; e.g. 0.1 seeds the topology by 10% for each new scheduling phase. Default set to `0.2`.
- `PLACERAD [number]` Specifies the radius for the Moore neighborhood to output values to. Default set to `1`.
- `UPTAKERAD [number]` Specifies the radius for the Moore neighborhood to input values from. Default set to `1`.
- `LINK` If given, links the resources between the layers to possibly induce symbiosis and co-evolution. Default set to `false`.
- `VICDIFF` If given, computes the vicinity difference between all layers in the model, as relative to a "shadow model". Default set to `false`.
- `PMUTRATE [float]` Specifies the mutation rate for the input parameters, note that the output is half this mutation rate to ease adaptation. Default set to `0.02`.
- `NRLAYERS [number]` Specifies the number of layers to evolve in the model. Each layer will be initialized to the given organisms. Default set to `1`.

A.1.3 SALTA with Red-Queen

SALTA was extended from the default model with the following parameters for Red-Queen scenario:

- `PLACERAD [number]` Specifies the radius for the Moore neighborhood to output values to. Default set to `1`.

- `UPTAKERAD [number]` Specifies the radius for the Moore neighborhood to input values from. Default set to `1`.
- `MERIT [number]` Specifies the higher merit, rewarded for metabolizing the resources, as a number between 1 and 8. Overrides the merit in the init-file. Default set to `1`.

A.2 Initialization file

Second most dynamic parameters are given in an initialization file (init-file). The init-file specifies the experimental world; topology, instruction set, merit function and seeding population. Path and filename to the init-file is given as command line argument and the keywords of the file are given below. Comments are written after `//`, and the different types of data can be given in any order after the keywords. All white spaces are treated the same and considered interchangeable. The same init-file is used for all versions of SALTA.

- `MERIT:`, is followed by a sequence of 9 numbers 1 to 8 for giving merit to the logical functions: NOT, NAND, AND, OR_N, OR, AND_N, NOR, XOR and EQU, in the same order.
- `DIM:`, is followed by 1 or 2 for the dimensionality of the topology (1-D or 2-D).
- `SPACE:`, is followed by a single number that specifies the symmetric length of the topology. The topology will have $(\texttt{SPACE})^{(\texttt{DIM})}$ number of cell positions.
- `PATTERNS:`, is followed by a sequence of strings, representing user-defined patterns for the chemistry. Any string is accepted and is not case sensitive in usage.
- `INSTRUCTIONS:`, is followed by a sequence of strings of implemented instructions in the chemistry, typically given as a subset.
- `GENOME NR:`, is followed by a single digit that gives the number of copies of the subsequent genome, directly followed by the keyword '`GENOME:`' (below).
- `GENOME:`, is followed by a sequence of instructions (specified by '`INSTRUCTIONS:`') with their arguments (patterns specified by '`PATTERNS:`').

A simple init-file may look as follows:

```
// Default self-replicator

MERIT:          2  3  4  4  5  5  6  6  8

DIM:            2

SPACE:          10

PATTERNS:       LEND  END  LOOP  START

INSTRUCTIONS:   PATTERN  SET  COPY  JUMPNEQ

GENOME_NR:      0

GENOME:         PATTERN START
                PATTERN END
                SET RD START
                SET WR END
                PATTERN LOOP
                COPY RD WR
                JUMPNEQ LEND LOOP
                COPY RD WR
                PATTERN LEND
                PATTERN END
```

A.3 Global settings

There is also a java class-file called `GlobSet.java` or `ExtGlobSet.java` which defines all global settings for default model and extended such, respectively (the base class is in the `cpu` package). This file also has some variables which may be set, but are not accessible from the command line or the initialization file. We describe some of the most

useful variables here.

- `COPY_BITS_TO_FLIP`, specifies how many program word bits to randomly flip during substitution mutation. Default is 32 bits.
- `MAX_NR_REW`, specifies how many times an organism may get rewarded for the same logical function. Default is 3.
- `NR_OF_GEN_TO_SAMPLE`, specifies how many of the most frequent genomes to sample for each statistical sample point. Default is 3.
- `SAMPLE_GEN_EVOL_ACT`, specifies if genetic evolutionary activity [BSP98, BB99] should be measured (very memory and cpu demanding). Default is `false`.
- `SAMPLE_PHEN_EVOL_ACT`, specifies if phenotype evolutionary activity should be measured. Default is `false`.
- `EVOL_ACT_SAMPLE_RATE`, specifies the evolutionary activity sampling rate as a fraction between 0.0 and 1.0 that gives the percentage of genome executions between each measurement of activity. Default is 0.001, which gives $1,000$ measurements over given period.

A.4 Output file

All data generated by the platform is collected in an output file, specified by the user as an argument (see Section A.1). The output file holds data for studying evolution of the seeded population over time, based on the granularity of the sample rate. Data in the output file is categorized into sections as follows:

1. `GLOBAL SETTINGS` contains the simulation settings used for generating this output data-file.
2. `WORLD SETTINGS` describes the parsed world data from the initialization file.
3. `INPUT DATA` states total number of genome executions to run and the number of samples to take.

4. `STATISTICS` gives statistics of computed logical functions, mutations, average genome length, number of genetically different organisms and so on. One such section is computed and written to the output for each sample point specified.

5. `GENOME EVOLUTIONARY ACTIVITY` holds data for measures of genetic evolutionary activity as tuples of sample point and abundance (optional).

6. `GENOME EVOLUTIONARY ACTIVITY` holds data for measures of phenotype evolutionary activity as tuples of sample point and (optional).

8. `VICINITY DIFFERENCE` holds data for computed vicinity difference between each layer relative a shadow model, as 2-tuples.

`TIME` tells the total simulation time in milliseconds (msec).

Appendix B

Glossary

Adaptation Acquisition of phenotype properties that increase fitness in a given environment (niche).

Allele A particular gene instance, coding for variations of a protein (e.g. eye-color).

Chromosome Strand of genetic material (DNA).

Conjugation Physical transfer of genetic material (chromosomes or single genes) between prokaryotes, and typically followed by genetic recombination to merge transfered genes.

Environment Biotic (anything alive) and abiotic (physical conditions and resources) elements, having an effect on an organism's fitness.

Epistasis Functional interdependencies between genes, e.g. such that the existence of one gene may suppress the effect of other genes.

Fecundity Capacity of produced offspring; measured as a quantity.

Fidelity Accuracy in producing offspring that is identical to its parent.

Fitness An organism's relative success in producing offspring.

Gene Particular hereditary unit of DNA; coding for a certain protein (function).

Genetic recombination Recombination of gene sequences from possibly different parallel sources.

Genetic drift Genetic non-uniform random change that over time may affect a pool of evolving genotypes to possibly lower fitness. Genetic drift may be seen as an opposite force to natural selection, of genetic change by chance in an evolving population.

Genome Total genetic material of an organism.

Genotype Genetic material of an organism that determines its phenotype.

Gradualism (Darwin) Evolution through small genetic changes by beneficial mutations.

Mutation Random genetic change, typically point-mutations (substitution, deletion and insertion), that modify single bases in a sequence during replication. Mutations can be neutral, deleterious or beneficial (in fitness).

Natural Selection (Darwinian selection) The mechanism in a population over generations by which the organisms with lower fitness are removed.

Niche Subspace in an ecology with certain functional properties on an organism or a population.

Phenotype A particular organism's physiological character or behavior/trait, which is coded by its genotype.

Population Ensemble of organisms under evolutionary investigation, and the unit of natural selection.

Protein Molecule of amino acids; coded by possibly many alleles and has biological functionals for the host (ultimately creating the phenotype).

Species A species is typically an interbreeding group of organisms, but since prokaryotes does not interbreed, there are no species in this kingdom of organisms, at least not by this definition.

Symbiont An organism participating in a symbiotic relationship.

Symbiosis (de Bary's definition) Long-term close association between different species.

Symbiogenesis The genesis (birth) of new species through genetic integration of symbionts.

Transduction Insertion of genetic material by viruses into prokaryotes; typically followed by genetic recombination.

Transformation Uptake by prokaryotes, of free genetic material from the environment; typically followed by genetic recombination.

Translation Interpretation of alleles/genes to proteins.

Transposons A sequence of genetic material that can move around in the genome, also called "jumping gene"; typically followed by genetic recombination.