The Watchmaker’s guide to Artificial Life: 
On the Role of Death, Modularity and Physicality in Evolutionary Robotics

Author: Frank Veenstra

Supervisors: Kasper Støy, Sebastian Risi

A thesis submitted in fulfilment of the requirements for the degree of Doctor of Philosophy

in the

Robotics Evolution and Art Laboratory
Declaration of Authorship

I, Frank Veenstra, declare that this thesis titled, “The Watchmaker’s guide to Artificial Life” and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at the IT University of Copenhagen.

- Where any part of this thesis has previously been submitted for a degree or any other qualification at the IT University of Copenhagen or any other institution, this has been clearly stated.

- Where I have consulted the published work of others, this is always clearly attributed.

- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.

- I have acknowledged all main sources of help.

- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date:
May 1, 2018
“Surely if a machine is able to reproduce another machine systematically, we may say that it has
a reproductive system. What is a reproductive system, if it be not a system for reproduction?
And how few of the machines are there which have not been produced systematically by other
machines? But it is man that makes them do so. Yes; but it is not insects that make many of the
plants reproductive, and would not whole families of plants die out if their fertilization was not
affected by a class of agents utterly foreign to themselves? Does anyone say that the red clover
has no reproductive system because the humble bee (and the humble bee only) must aid and abet
it before it can reproduce? No one. The humble is part of the reproductive system of the clover.
Each one of ourselves has sprung from minute animalcules whose entity was entirely distinct
from our own, and which acted after their kind with no thought or heed of what we might think
about it. These little creatures are part of our own reproductive system; then why not we part
of that of the machines? “

Samuel Butler – Erewhon
Abstract

Computer Science

Doctor of Philosophy

The Watchmaker’s guide to Artificial Life

by Frank Veenstra

English Abstract

Optimizing robots through the implementation of evolutionary processes plays a key role in evolutionary robotics and artificial life. Challenges that arise in evolutionary optimization of robots are usually related to an algorithm’s compromise between trying new solutions or improving previously found solutions (the exploration vs exploitation trade-off), and whether to express genomic information directly or generatively (genotype to phenotype mapping). An additional challenge is that there is a discrepancy between simulation and reality (reality gap) where robots ‘evolved’ in simulation environments function differently when transferred to the real-world.

The exploration vs exploitation trade-off is addressed through describing and experimenting with the inclusion of biologically-inspired intrinsic mortality and how this affects the evolvability of populations. The results contribute to our understanding of the relationship between intrinsic mortality and mutation rate. The results further show how it can be utilized to develop algorithms that can outperform state-of-the-art algorithms.

The thesis continues by addressing the challenge of mapping the genotype to phenotype through investigating the influence of generative encodings on the evolution of simulated modular robots. It is investigated how an L-System as a generative encoding can lead to the formation of plant-inspired virtual creatures and shows that movement for tracking a moving light source is not an emerging phenotypic trait. Afterwards, this generative encoding is shown to better evolve modular robots for locomotion compared to a direct encoding. The addition of real-world solar panel modules demonstrate how modular robots can be evolved toward energy autonomy.

The final part of the thesis describes the evolution of embodiment and control of physical robots. As part of this, an automated process for assembly and disassembly of modular robots is demonstrated, which can be used to evaluate evolved individuals in the real world. A viable method for implementing evolution directly is demonstrated through evolving the behavior of a knifefish-inspired physical soft robot. Both approaches represent strategies for addressing the reality gap.

The experimental results of the thesis contribute to the understanding of biological phenomena and elucidate how improvements can be made to existing methods in evolutionary robotics. It shows that we can utilize concepts from evolutionary biology to advance our understanding of evolutionary dynamics, encodings and physical designs that are valuable for the automated design of robots.
Danish Abstract

Optimering af robotter gennem implementering af evolutionære processer spiller en central rolle i evolutionær robotik og kunstigt liv. Udfordringer der opstår i evolutionær optimering af robotter er normalt relateret til en algoritmes kompromis mellem at prøve nye løsninger eller forbedre tidligere fundne løsninger (søge kompromiset) og om at udtrykke genomisk information direkte eller generativt (genetisk kodning). En yderligere udfordring er, at der er en uoverensstemmelse mellem simulering og virkelighed, hvorved robotter udviklet i simuleringssmiljøer fungerer anderledes, når de overføres til virkeligheden (virkelighedsgabet).


Afhandlingen fortsætter med at kigge på udfordringen om genetisk kodning ved at undersøge indflydelsen af generative kodninger på udviklingen af simulerede modulære robotter. Det undersøges, hvordan et L-system, som er en generativ kodning, kan føre til dannelsen af planteinspirerede virtuelle væsner og viser, at bevægelse til sporing af en bevægende lyskilde ikke er et fremtrædende fænotypisk træk. Derefter vises denne generative kodning at være bedre til at udvikle modulære robotter til bevægelse sammenlignet med en direkte kodning. Tilføjelsen af virkelige solpanelmoduler demonstrerer, hvordan modulære robotter kan udvikles mod energi autonomi.


Resultaterne i afhandlingen bidrager til forståelsen af biologiske fænomener og beleyser, hvordan man kan forbedre eksisterende metoder i evolutionær robotik. Afhandlingen viser, at vi kan udnytte koncepter fra evolutionær biologi for at fremme vores forståelse af evolutionær dynamik, kodninger og fysiske design, der også er værdifulde i forbindelse med automatisk design af robot.
Preface

This thesis is about evolution in the context of artificial life. Here, evolutionary computation shapes the progression of usually in silico systems toward user specified goals through the artificial implementation of evolution. Since evolution has led to the many elegant adaptations we see in organisms today, the implementation of evolutionary computation in robotics is done with the aim to bring about artificial complexity using the same evolutionary processes underlying billions of years of adaptation. This is the paradigm of Evolutionary Robotics. As the biological resulting machinery, organisms, have been analogized to the complexity of watches, many have attributed the emergence of organisms and species in this world falsely to a higher deity. The many works of Richard Dawkins assert that the acclaimed watchmaker processes seen in nature are in fact the result of the blind forces of physics. Evolution, as a result of these forces, elicits the works of an apparent blind watchmaker. Since us evolutionary roboticists try to nudge the evolutionary processes to engender effective features in robots, we are the parody of the watchmaker for the creation of artificial life. We incorporate as many biological features shown to be useful in the evolutionary processes exhibited in nature and contemplate about how to thereby create beings of complexity rivaling nature’s emergent ‘design’.

The subtitle of the thesis explains its major constituents. Death is referring to the experiments on the potential evolutionary advantage of programmed death, which has been heavily inspired by a guest lecture from Justin Werfel on altruistic aging. Modularity concerns the implementation of modular robots; subjecting modular robot conglomerates to simulated evolution. This work building on existing work of evolving modular robots done in collaboration with Andrés Faíña. Physicality is discussed by the application of evolutionary computation to physical robots. Where Jonas Jørgenson and I have collaborated to evolve the behavior on a physical knifefish inspired robot. Death, modularity and physicality are discussed in the context of evolutionary robotics; toward a better understanding of (artificial) life.

Acknowledgements

I would like to thank first and foremost, my advisors, Kasper Støy and Sebastian Risi. Not only were you excellent supervisors, but the fun, relaxed and thought provoking atmosphere made the PhD an incredible experience. For this, I feel incredibly fortunate for having been able to pursue my academic interest with you. An incredible source of support and influence for this thesis was Andrés Faíña, you enabled me to develop my own algorithms and robots through your advice and patience and I am incredibly grateful for it. As a whole, I would like to furthermore thank all my friends and colleagues at the IT University of Copenhagen. This includes all of the former and current members of the Robotics Evolution and Art Lab. Many thanks to my advisor abroad, Josh Bongard and all the former and current members of the University of Vermont’s Morphology, Evolution and Cognition Laboratory that I met during my stay abroad. The experience of collaborating with you was incredibly influential on the progress of my research and I feel fortuitous considering all our memorable experiences.

Special thanks to my adorable girlfriend Franziska, whom I would have never met were it not for being able to take a course in Neurobiology for my PhD. Thereby also thanks to all my friends that have supported me during my PhD. And thanks to my entire family, with honorable mentions such as: mom, dad, Jessie, beppe & opa. Without dad, evolution in this thesis would have been brought about by ‘decent’ with modification. Where credit for the awesome drawings on the cover and part sections of this thesis go to my sister, Jessie Veenstra.

The work presented in this thesis received funding from “Flora Robotica”, a European Union’s Horizon 2020 research and innovation program under the FET grant agreement, no. 640959. Computation/simulation for 3 & 9 was supported by the Vermont Advanced Computer Core, University of Vermont. Computation/simulation for Chapters 6 & 7 was supported by the DeIC National HPC Centre, University of Southern Denmark, and the Vermont Advanced Computer Core, University of Vermont.
Special thanks to Sam Kriegman whose input and feedback shaped parts of the Chapter 3. Special thanks to Rodrigo Moreno, Ceyue Liu, Chloé Metayer and Andrés Fañaná for their contribution to the development of the modular robots presented in Chapters 4, 5, 6, 7 and 8. Special thanks to Jonas Jørgenson for his contributions to the work presented in Chapter 9.
Contents

Declaration of Authorship iii
Abstract vii
Preface ix

1 Introduction 1
   1.1 Challenges ................................................. 2
   1.2 Structure and Contributions ............................. 3
   1.3 Evolutionary Robotics ............................... 5
   1.4 Scope and Context ..................................... 7

I Evolutionary Dynamics 9

2 Evolution 11
   2.1 Fundamentals of Evolutionary Theory ..................... 12
      2.1.1 Performance Measure .................................. 13
   2.2 The genome and the Sequence Space ....................... 14
      2.2.1 Selection ........................................... 16
   2.3 Development of the Phenotype ........................... 16
   2.4 Evolutionary Game Theory ............................. 19
   2.5 Evolvability .......................................... 20
   2.6 Evolutionary Computation ................................ 23
      2.6.1 Binary Approach .................................... 25
      2.6.2 continuous single objective functions ............. 27
   2.7 Spatial Models .......................................... 28
   2.8 Concluding Remarks ..................................... 33

3 Evolution and Longevity 35
   3.1 Theory on the Evolution of Senescence ................. 36
      3.1.1 Summary of Theories on Senescence ............... 36
      3.1.2 Programmed vs Non-programmed .................... 38
      3.1.3 Why Mortality Promotes Evolvability ............. 39
   3.2 Mortality and Mutation Rate shape Evolvability ....... 41
      3.2.1 Mortality in the Steady State Genetic Algorithm .. 41
      3.2.2 Spatial Model ....................................... 45
      3.2.3 Comparison to Other Algorithms ................. 49
      Age Fitness Pareto Optimization .................... 49
      Other Conventional Implementations .................. 50
   3.3 Changing the Domain .................................... 52
   3.4 Development And Evolvability .......................... 54
   3.5 Discussion ............................................. 57
      3.5.1 Theoretical implications .......................... 58
      3.5.2 A hypothetical explanation for the benefit of mortality 60
      3.5.3 Application to Evolutionary Robotics ............. 60
   3.6 Conclusion ............................................. 60
II  Evolving Modular Robots

4  Evolutionary Robotics Platform
   4.1 Neuroevolution .................................................. 64
   4.2 Overview of the Evolutionary Robotics Plugin ......................... 66
      4.2.1 Evolutionary Algorithm ..................................... 66
      4.2.2 Morphology .................................................. 68
      4.2.3 Control ...................................................... 69
   4.3 Current State and Functionalities ..................................... 69
   4.4 Real Modules .................................................... 71
   4.5 Concluding Remarks ............................................... 74

5  Evolving Plant Morphologies with L-Systems 75
   5.1 Introduction ...................................................... 76
      5.1.1 Phytomorphogenesis .......................................... 77
      5.1.2 Simulated models ............................................. 78
   5.2 Approach ........................................................ 79
      5.2.1 Genetic Algorithm ............................................ 79
      5.2.2 L-System .................................................... 80
   5.3 Results .......................................................... 81
   5.4 Discussion ....................................................... 82
   5.5 Conclusion ....................................................... 84

6  Comparing Encodings for Evolving Locomotion in Modular Robots 87
   6.1 Introduction ...................................................... 88
   6.2 Methodology ....................................................... 89
      6.2.1 Common Elements for both Platforms .......................... 89
      6.2.2 Direct Encoding .............................................. 91
      6.2.3 Generative Encoding ......................................... 91
   6.3 Results .......................................................... 92
      6.3.1 Performance analysis .......................................... 93
      6.3.2 Phenotypes .................................................. 93
   6.4 Discussion ....................................................... 96
   6.5 Conclusion ....................................................... 99

7  Toward Energy Autonomy through Evolving Plant-Inspired Modular Robots101
   7.1 Introduction ...................................................... 101
   7.2 Material and Methods ............................................. 103
      7.2.1 Modules ...................................................... 103
      7.2.2 Connectivity ................................................ 106
      7.2.3 Generative Encoding ....................................... 107
      7.2.4 Evolutionary Algorithm ..................................... 108
      Neural Network .................................................. 108
      Fitness function ................................................ 109
      7.2.5 Experimental Setup .......................................... 109
   7.3 Results .......................................................... 110
      7.3.1 Phenotypes Evolved in Different Environments ................. 110
      7.3.2 Movement in Energy Harvesting Modular Robots ............... 111
   7.4 Discussion ....................................................... 112
      7.4.1 Light Tracking .............................................. 112
      7.4.2 Challenges .................................................. 113
      7.4.3 Multi-Robot Systems ......................................... 116
   7.5 Conclusion ....................................................... 118
List of Figures

1.1 Biomorphs .......................................................... 6
1.2 Evolving Virtual Creatures .................................... 6
1.3 Overview of fundamental steps in Evolutionary Robotics .... 7
2.1 Fitness landscape with a peak and a hill .................... 15
2.2 Dentine formation influenced by hedgehog .................. 18
2.3 Configuration of the eye of cephalopods and vertebrates .... 22
2.4 Basic procedure of a generational Evolutionary Algorithm . 24
2.5 Fitness landscape of one-max function ........................ 26
2.6 Fitness landscape of the adjusted H-IFF function .......... 27
2.7 Explanation of the H-IFF function ............................. 28
2.8 The Rastrigin and Schwefel function ......................... 29
2.9 Amount prey and predators when varying the primary energy production .......... 31
2.10 Stable limits in predator prey models ........................ 32
2.11 Biomass over time when simulating plants, rabbits and foxes . 32
3.1 Survivor curves .................................................. 37
3.2 Theories of Aging plotted as an advantage and a disadvantage over time .... 39
3.3 Fitness landscape with one peak and a hill based on the sequence space .. 40
3.4 Relationship between the mutation rate and mortality rate .......... 43
3.5 Evolutionary progress for different mortality rates .......... 43
3.6 Relationship between mutation rate and mortality rate on 128-bit H-IFF .. 44
3.7 Relationship between mutation rate and mortality rate on 32-bit H-IFF .... 44
3.8 Optimal mutation and mortality rate on 32-bit Chuang f1 ........ 45
3.9 Illustration of the spatial model ................................ 46
3.10 Speed of solving H-IFF for the spatial model ............... 48
3.11 Individual runs on the spatial model .......................... 48
3.12 Optimal mutation rate as a function of terminal age .......... 49
3.13 Age Fitness Pareto Optimization .............................. 50
3.14 Single evolutionary runs comparing AFPO and H-IFF ....... 51
3.15 200 evolutionary runs are shown for both AFPO and Mortality Rate .... 51
3.16 Diversity on H-IFF using various genetic algorithms .......... 52
3.17 Evolutionary progression on 64-bit H-IFF of generational Genetic algorithms vs the SSGA implementing mortality ............. 53
3.18 Comparison of evolutionary algorithms on the Schwefel function ... 54
3.19 Correlating developmental stages with finding the global maximum .. 56
3.20 Single evolutionary runs in the developing populations .......... 57
3.21 Venn diagram depicting the relationship between reproductive rate, mortality and mutation rate .......................... 59
4.1 Approaches to controlling a modular robot ................... 66
4.2 Overview of the evolutionary robotics plugin .................. 67
4.3 Illustration of the interface designed for the evolutionary robotics plugin .. 68
4.4 Example of growth in the simulator ............................ 69
4.5 Two modular robots controlled by decentralized neural networks .... 70
4.6 Module connection site ......................................... 71
4.7 Interface Connector ............................................. 72
4.8 Collection of Modules ........................................... 72
4.9 Modular robots connected to V-REP via the plugin .......... 73
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1</td>
<td>Recursion and modularity in plants</td>
<td>76</td>
</tr>
<tr>
<td>5.2</td>
<td>Evolved Phytomorphologies</td>
<td>77</td>
</tr>
<tr>
<td>5.3</td>
<td>Movement of a light source with respect to robot</td>
<td>80</td>
</tr>
<tr>
<td>5.4</td>
<td>Three illustrations of the implemented L-System</td>
<td>81</td>
</tr>
<tr>
<td>5.5</td>
<td>Illustration of an evolutionary progression</td>
<td>82</td>
</tr>
<tr>
<td>5.6</td>
<td>Evolutionary progression of static and actuated phytomorphologies</td>
<td>83</td>
</tr>
<tr>
<td>5.7</td>
<td>The phenotype of the best evolved individual</td>
<td>84</td>
</tr>
<tr>
<td>5.8</td>
<td>Evolution of movement?</td>
<td>84</td>
</tr>
<tr>
<td>5.9</td>
<td>Generated phytomorphologies</td>
<td>85</td>
</tr>
<tr>
<td>6.1</td>
<td>Illustration of simulated and real modules</td>
<td>89</td>
</tr>
<tr>
<td>6.2</td>
<td>Representation of the direct encoding</td>
<td>91</td>
</tr>
<tr>
<td>6.3</td>
<td>Representation of the generative encoding</td>
<td>92</td>
</tr>
<tr>
<td>6.4</td>
<td>Evolutionary progression of individual runs</td>
<td>94</td>
</tr>
<tr>
<td>6.5</td>
<td>Evolutionary progressions of the direct and generative encoding</td>
<td>95</td>
</tr>
<tr>
<td>6.6</td>
<td>Phenotypes obtained through the direct encoding</td>
<td>96</td>
</tr>
<tr>
<td>6.7</td>
<td>Phenotypes acquired through the generative encoding</td>
<td>97</td>
</tr>
<tr>
<td>6.8</td>
<td>Illustration of evolved phenotypes</td>
<td>98</td>
</tr>
<tr>
<td>6.9</td>
<td>Prematurely converged phenotype</td>
<td>98</td>
</tr>
<tr>
<td>7.1</td>
<td>The modules used in the simulated and physical robot</td>
<td>104</td>
</tr>
<tr>
<td>7.2</td>
<td>Schematic diagram of the modular robot containing solar panels</td>
<td>106</td>
</tr>
<tr>
<td>7.3</td>
<td>Block diagram of the modular robotics platform</td>
<td>107</td>
</tr>
<tr>
<td>7.4</td>
<td>Different simulated environments</td>
<td>110</td>
</tr>
<tr>
<td>7.5</td>
<td>Different evolved phenotypes</td>
<td>111</td>
</tr>
<tr>
<td>7.6</td>
<td>Evolved phenotypes in different environments</td>
<td>112</td>
</tr>
<tr>
<td>7.7</td>
<td>Various evolved phenotypes in the environment with the moving light source</td>
<td>113</td>
</tr>
<tr>
<td>7.8</td>
<td>Evolutionary runs done in the different environments</td>
<td>114</td>
</tr>
<tr>
<td>7.9</td>
<td>The effects of an additional energy cost</td>
<td>115</td>
</tr>
<tr>
<td>7.10</td>
<td>Incorporation of the flower module</td>
<td>116</td>
</tr>
<tr>
<td>7.11</td>
<td>Simulated and real modular robot behavior</td>
<td>117</td>
</tr>
<tr>
<td>8.1</td>
<td>Evolutionary robotics approaches to acquire a behavior</td>
<td>122</td>
</tr>
<tr>
<td>8.2</td>
<td>Assembly and disassembly process of modules</td>
<td>124</td>
</tr>
<tr>
<td>8.3</td>
<td>Sequence of automatically assembling modular robots</td>
<td>125</td>
</tr>
<tr>
<td>9.1</td>
<td>The black ghost knifefish</td>
<td>128</td>
</tr>
<tr>
<td>9.2</td>
<td>Experimental setup for evolving the knifefish</td>
<td>130</td>
</tr>
<tr>
<td>9.3</td>
<td>CAD design of the robotic knifefish</td>
<td>131</td>
</tr>
<tr>
<td>9.4</td>
<td>Cross section of the fish design</td>
<td>132</td>
</tr>
<tr>
<td>9.5</td>
<td>Angular deflection of the fin</td>
<td>133</td>
</tr>
<tr>
<td>9.6</td>
<td>Change in angle of fin rays</td>
<td>133</td>
</tr>
<tr>
<td>9.7</td>
<td>Evolutionary progressions of five runs</td>
<td>135</td>
</tr>
<tr>
<td>9.8</td>
<td>The best evolved wave patterns in 5 distinct evolutionary runs</td>
<td>136</td>
</tr>
<tr>
<td>9.9</td>
<td>Performance difference between the best evolved sinusoidal and Fourier series individuals</td>
<td>137</td>
</tr>
<tr>
<td>9.10</td>
<td>Evolved wave patterns</td>
<td>138</td>
</tr>
<tr>
<td>9.11</td>
<td>The evolutionary progression of a single Fourier Series run with a servo malfunction occurring around generation 10</td>
<td>139</td>
</tr>
<tr>
<td>B.1</td>
<td>The UML diagram of the Evolutionary Robotics Plugin</td>
<td>149</td>
</tr>
</tbody>
</table>
# List of Tables

2.1 Example of the score of a 16-bit HIFF genome . . . . . . . . . . . . . . . . 27

3.1 Times the optimal solution is found in the SSGA . . . . . . . . . . . . . 42

3.2 Number of optimal solutions for 32-bit H-IFF on a spatial model . . . . 47

9.1 Specific wavelengths and travel speeds of behaviors . . . . . . . . . . . . 136
Dedicated to my Family
Chapter 1

Introduction

It is interesting to contemplate an entangled bank, clothed with many plants of many kinds with birds singing on the bushes, with various insects flitting about, and with worms crawling through the damp earth, and to reflect that these elaborately constructed forms, so different from each other, and dependent on each other in so complex a manner, have all been produced by laws acting around us. These laws, taken in the largest sense, being Growth and Reproduction; Inheritance which is almost implied by reproduction; Variability from the indirect and direct action of the external conditions of life, and from use and disuse; a Ratio of Increase so high as to lead to a Struggle for Life, and as a consequence to Natural Selection, entailing Divergence of Character and the Extinction of less-improved forms. Thus, from the war of nature, from famine to death, the most exalted object which we are capable of conceiving, namely, the production of the higher animals, directly follows. There is grandeur in this view of life, with its several powers, having been originally breathed into a few forms or into one; and that, whilst this planet has gone cycling on according to the fixed law of gravity, from simple a beginning endless forms most beautiful and most wonderful have been, and are being, evolved.

– Charles Darwin, the Origin of Species

Evolution is the ultimate explanation for life as we know it. The process of descent with modification is not only fundamental to our understanding of nature, its merits are useful in computational models as well. Its utilization is valuable for creating smart algorithms or robots, which enables us to better understand evolution and explain phenomena that we see in nature. Where artificial evolution, defined under the umbrella term evolutionary computation, encompasses the algorithms mimicking biological evolution. It sprouted from the many implementations in computational models that were inspired by evolution. In Evolutionary Robotics, evolutionary computation is implemented as the strategy for the automated design of intelligent robots (Harvey et al., 1992; Nolfi et al., 2000). It has subsequently been shown that research in this paradigm can result in the creation of unique control systems and body plans for simulated and physical robots.

If we could rerun the evolutionary process of life on earth, the emerging entities would likely not resemble humans. Another branch of life could well have dominated and outcompeted the ancestral branch humans sprouted from. If a meteorite coerced preventing dinosaurs from going extinct, an intelligent dinosaur species might have roamed the planet instead of us humans. This aspect is also true for artificial systems, rerunning evolutionary robotics simulations will almost never yield similar phenotypes but rather find unique solutions for an objective; being a creative search process.

Using evolution as a tool can thereby also lead to the emergence of complex robots by simply providing the fundamental building blocks. The evolutionary process in turn generates efficient machines. As will be shown in this thesis, this can be done through evolving simulated robots as well as implementing evolutionary computation on robots directly. In evolutionary robotics we can subject any stage of a robot’s ancestral branch and run evolution from this point onward designating the robot to conform to a variety of objectives, environments, resources, body plans, and control
Chapter 1. Introduction

systems. Hence, an artificial evolutionary process could lead to life, but, most likely, not as we know it (Eiben, 2014).

A classic argument used as proof for the existence of a deity is the analogy of this deity to a watchmaker. The theologian William Paley explained that finding a watch on the floor necessitates the inference that it must have been designed and made by a watchmaker (Paley1802). Organisms seemingly contain design elements that are, just like a watch, complex, as opposed to, say, a stone. Therefore, organisms must have a ‘designer’ as well. This idea was formulated before Darwin’s origin of species. Dawkins (Dawkins, 1986) explained that such complex natural design is simply an emergent property of evolution through descent with modification – where “if it can be said to play the role of watchmaker in nature, it is the blind watchmaker” (Dawkins, 1986). In evolutionary robotics, the blind process of evolution is itself implemented to design robots. Hence us evolutionary roboticists being watchmakers, having come into existence through a blind watchmaker, try to make watches using a blind watchmaker’s approach.

Evolutionary computation is a branch of artificial intelligence. Where contemporary research in artificial intelligence has proven its prominence and applicability. Strategies like deep neural networks and monte-carlo tree search have especially demonstrated to produce smart machines as in the case of the recent defeat of the world best go player (Silver et al., 2017). Though despite the practical value, most research in the field of AI in general is concerned with the creation of arbitrary notions of thought processes and reasoning that are heavily inspired by how humans act and think, or by how machines can be made to act rationally (Russell et al., 2010). It is therefore also highly bio-inspired and usually takes an anthropomorphic perspective. Robotics and AI not only focus on this mimicry, research also entails cognitive and motivational autonomy by investigating memory, imagination, mental life, planning, thinking, dreaming or hallucinating. Although humans can serve as a great model for inspiration in artificial intelligence, it could also be viewed as an over-complicated system that developed its advanced cognitive processes based on maximum parsimony. Maximum parsimony meaning that through natural evolution, the amount of state changes to find a solution is minimized. Humans happen to be the solution thus far in this context.

Many of the physiological traits and structures seen in humans might have served a completely different function in common ancestors. The evolved functionalities we see today might thus not be the most efficient to implement in an artificial system. Our intelligence is surely not modeled with the sole aim of acquiring it, rather, it enables the survival and reproduction of our genes. E.g. there are behavioral and structural commonalities between ourselves and neighboring species that sprouted from a common ancestor. If we dissect the human body, say we split up every compartment of the brain and try to make an artificial module for the specific task allocated to the corresponding part of the brain, we might find more efficient solutions to engender these functional feats without being chained to our ancestral precursive solutions. So, in contrast to the anthropomorphic modeling of an artificial system inspired by hominids, in evolutionary robotics we can reshape the entire collection of functionalities and evolve them without relying on the process of mimicking them. Evolutionary robotics thus allows us to evolve a general bio-inspired artificial system potentially as a precursor to new branches of intelligent systems and artificial life.

1.1 Challenges

Evolutionary computation is often implemented as an objective based problem solver, similar to existing AI techniques. The power of evolutionary computation being manifold, including its ability to optimize discontinuous, non-differentiable functions. The correct parameter set is however highly dependent on the fitness landscape. Simple hills can be climbed through local search, whereas more difficult landscapes necessitate more global search or diversification to find a desired optimum. This is a problem
1.2 Structure and Contributions

in evolutionary computation called the *exploration vs exploitation* trade-off. The capability of traversing the fitness landscapes efficiently through adjusting parameters that favor either exploration or exploitation, are thus of great importance to evolutionary computation. The capacity of traversing a fitness landscape and thereby finding better solutions is what is called *evolvability* – the capacity to evolve. Evolvability in evolutionary algorithms will be discussed mainly in Chapter 3 where it is promoted by implementing an indiscriminate intrinsic mortality factor.

The attributes seen in organism that have led to the emergence of life forms are also of interest in the field of evolutionary robotics. One of these important attributes that changes an organism during its lifetime is called development. The equivalent of development in evolutionary robotics are the processes that change an agent during its lifetime and, of specific interest to this thesis, the mapping from genotype to phenotype. The *genotype to phenotype mapping* being another challenge in evolutionary robotics that will be discussed mainly in Part II, which describes the evolution of modular robots.

One final major challenge in evolutionary robotics is the *reality gap*, or the discrepancy between simulated and real-world robots. In evolutionary robotics, the simulator usually evolves the behavior in a three-dimensional model of the robot that can be transferred to the real-world robot after an adequate solution has been found. However, transferring the evolved robot from the simulation to the real world always presents a performance or reality gap (Jakobi et al., 1995). Contemporary research has not yet created simulators that would allow for morphologically complex evolved robots to be transferred to the real world and expecting similar results. Though certain abstractions can reduce this reality gap as will be discussed briefly in chapters 6 and 7. An alternative approach bypassing the simulator altogether will be discussed in Chapter 9. In this chapter, evolutionary computation is implemented directly on a physical soft robot. This is an efficient approach since a soft robot being difficult to simulate and control in the real world.

1.2 Structure and Contributions

This thesis conveys ideas and experiments about the process of shaping artificial robotic systems without relying on engineered solutions or detailed features derived from natural systems. The thesis is structured in three general parts that address the topics as the title implies: Evolutionary Dynamics (Part I), Evolving Modular Robots (Part II) and Evolution of Physical Robots (Part III). Though the parts are not strict distinctions, the thesis is structured from theory to practice. Every subsequent chapter being more set-in reality than the former.

In the first part, the thesis discusses various concepts of natural systems related to evolution and how these theoretical concepts can be implemented in computational models. In Chapter 2, a general overview is given of high level concepts related to natural and artificial evolution. From here on, experiments on the evolutionary dynamics of genetic algorithms and spatial models are performed in Chapter 3. This chapter investigates specifically the theoretical concept of how senescence (aging) could be an evolutionary advantageous trait in natural and artificial systems. Senescence being simulated by an intrinsic mortality parameter. This chapter describes the first major scientific contribution that is supported by a paper submitted to the artificial life conference of 2018 (“Intrinsic Mortality Governs Evolvability”). The chapter includes many details and additional experiments that were left out in the submitted paper. The hypothesis that is being addressed in this chapter is:

**Hypothesis 1** *Intrinsic mortality benefits the evolvability of a population.*

The results of this chapter indicate that there is a tight correlation between the mutation rate and mortality rate that influences evolvability in a specific manner. Experiments on various landscapes indicate that indiscriminate mortality can enhance the performance of genetic algorithms on benchmark tests and in spatial models. These
experiments show that factors influencing intrinsic mortality are actually an evolutionary advantage for an evolving population.

The second part of the thesis addresses the main body of work done on Evolving Modular Robots. In Chapter 4, an overview is given of the evolutionary robotics simulator that was developed and used for the experiments done in the subsequent chapters. The three chapters that follow are based on three publications on the evolution of modular robots. The first publication (Veenstra et al., 2016) was about evolving plant-inspired virtual creatures for function and aesthetics. It discussed whether evolved artificial plants would likely evolve movement or not in a given environment (Chapter 5). This is formulated by the second hypothesis of the thesis:

**Hypothesis 2** *Actuation in evolving phytomorphologies is beneficial for optimizing light absorption.*

Chapter 5 shows that in fact movement, without any added cost for movement, did not produce better individuals for tracking a light source. It could therefore be that movement is simply not as useful to plants as being able to create many leaves.

Chapter 6 veers the evolutionary robotics approach from optimizing for light absorption to evolving locomotion in modular robots (Veenstra et al., 2017a). The chapter discusses two approaches on generating the modular robots and how this generation affects the evolutionary search process. It therefore addresses the genotype to phenotype mapping. Where one approach consists of a direct encoding and the other of a generative encoding similar to the one implemented in Chapter 5. This forms the third hypothesis of the thesis:

**Hypothesis 3** *A generative encoding increases the efficiency of evolving modular robots compared to a direct encoding.*

The results indicate that a generative encoding is indeed the better encoding strategy for evolving modular robots. Moreover, the video that was produced as supplementary material subsequently won the virtual creatures contest held at GECCO 2017.

The real physical modules that were developed were used to evaluate how to design energy autonomous robots as discussed in Chapter 7 (Veenstra et al., 2017b). Although genuine energy autonomy was too difficult to achieve given the modules that were used, the main contribution of this chapter is its description of a modular robotic system in which energy harvesting modules can be integrated. This approach might be most valuable for multi-robot systems where some robots can be specialized in energy harvesting while higher order robots could utilize the energy gathered by these primary energy producing robots. The main hypothesis being addressed in this chapter is:

**Hypothesis 4** *Energy autonomy in modular robots can emerge from implementing solar panel modules.*

Results show that this implementation of evolution in modular robots provides a stepping stone towards developing legitimate energy autonomy in modular robots. It moreover demonstrates an application where the reality gap is negligible.

The final part of the thesis is about applying evolution in physical systems. It opens with a rough outline on approaches to evolving physical robots and describes a way to automatically assemble modular robots in Chapter 8. Chapter 9 is a standalone experiment where a knifefish-inspired soft swimming robot has been created (Veenstra et al., 2018). Its body is quite complex and would be tedious to simulate. Therefore, a form of evolution bypassing a simulator environment was used. Covariance Matrix Adaptation Evolution Strategy (CMA-ES) was the algorithm implemented to evolve the undulating swimming behavior of the robot. This experiment led to final hypothesis of the thesis:

**Hypothesis 5** *Evolutionary computation finds better solutions for controlling a designed soft swimming robot compared to manually encoded behaviors.*

---

1The video on Evolving Modular Robots can be found here: [https://www.youtube.com/watch?v=HCDft1c1Ada](https://www.youtube.com/watch?v=HCDft1c1Ada)
The results of the evolutionary experiments show that CMA-ES is a viable method for optimizing the behavior of morphologically complex robots outperforming the manually encoded behavior. The thesis concludes with a brief discussion of the implemented work and its prospects in Chapter 10. The general aim of the thesis is to convey how and why evolutionary robotics can lead to the acquisition of functional robots. This aim takes both the biological and artificial processes into consideration. The important common concepts in both natural and artificial evolution – such as selection pressure, the genotype to phenotype mapping, evolvability, and models for evolutionary computation – are therefore described in Chapter 2. This chapter contains essential information for understanding concepts that are described later in the thesis, but might not be a requirement for the reader knowledgeable in these subjects.

### 1.3 Evolutionary Robotics

In evolutionary robotics, we are free to model any potentially useful traits of morphologies and control which can be fed into an evolutionary algorithm as building blocks. This modeling, through using modular robots, forms the core of this thesis. Where robots can adopt certain traits that are useful for an objective and incrementally improve themselves by altering the genetic information that encodes for their phenotype. Any traits and features we provide as building blocks could be viewed as constituents of an artificial primordial soup. It contains all the ingredients that are useful or available to enable the emergence of the robots. Complex robots can thus spontaneously form through the evolutionary process by supplying functional precursors, building rules and control mechanisms. The composition of our artificial primordial soup entails a vast mixture of applicable theories, concepts and algorithms that evolutionary robotics covers. Where the fundamental mechanisms giving rise to our robots being reproduction and development. Reproduction conveying how genetic information is propagated to the next generation whereas development describes the procedure of acquiring a phenotype from this genetic information. Development is thus the mapping of the genotype to the phenotype. In addition, development can convey plasticity and learning in individuals during their life time.

Many of the mentioned challenges have already been considered and a popularized example implementing a simple genotype, an encoder and a user guided evolutionary process (interactive evolution) can be seen in Richard Dawkins’ blind watchmaker program (Figure 1.1; Dawkins 1988). Dawkins created a genotype containing 16 variables that were in turn interpreted by a Lindenmayer-System (L-System; Lindenmayer 1968a) to create biomorphs – objects resembling living organism – based on simple iterative rewriting rules. Where Dawkins emphasized the essential role of developmental genes that encode for recursion expressed in the phenotype of animals. Recursion leading to segmentation in cells, tissues, and overall body plans. The resulting segmentation is also known as a phenomenon called serial homology, or similarity of structures repeated along the body axis.

The genotype to phenotype mapping dictates the encoding that is used to generate an entity from a string of information. In some cases, this string of information is encoded as a generative (or indirect) encoding where bits and pieces of the code are reused in a recursive process to form the eventual phenotype of an individual. In contrast, in a direct encoding every parameter of the phenotype is defined. A fractal or a grammar can lead to incredibly complex patterns in the form of a drawing based on just a few functions or rewriting rules. In this case, the simplicity of the genotype and the complex emergent phenotype is an incredible result of a developmental encoding. This type of encoding is also present in nature and can be functionally utilized in artificial systems.

Karl Sims went a step further than Dawkins and implemented a similar strategy of encoding the genotype to phenotype map for evolving three-dimensional virtual creatures (Sims, 1994b). Subjecting the genomes that encode for the phenotypes to the evolutionary process (Figure 1.2a), many different types of phenotypes emerged with
Chapter 1. Introduction

**Figure 1.1:** Biomorphs. Different biomorph phenotypes evolved through interactive evolution. Taken from Dawkins, 1988

**Figure 1.2:** Evolving Virtual Creatures. The genotype to phenotype mapping (a) and the two evolved phenotypes (b) from Sims’ work (Sims, 1994b). The two evolved behaviors were optimized for swimming (top) and walking (bottom).

The objective of swimming or walking (**Figure 1.2b**). This is the first example of evolutionary computation being used to create simulated, physics-based virtual creatures. Though at the time a computer cluster was necessary to evolve the creatures, it becomes more and more computationally viable to evolve these types of virtual creatures on a personal computer.

The usual approach taken in evolutionary robotics is similar to Dawkins’ and Sims’ approach. Evolutionary computation takes care of automatically adjusting a population of individuals through gathering performance measures from resulting phenotypes. Considering the loop of reproduction and evaluation that are essential for the correct implementation of evolutionary algorithm, a minimal schematic representation of the important aspects in the evolutionary robotics approach is given in **Figure 1.3**. The agent in this case covers the pathway from a genotype to a phenotype through development. The agent in biology being an organism and in the artificial context of this thesis, a robot. Though in an evolutionary setting, the agents themselves aren’t optimized but rather their gene pool. This can mean that many small robots might function just as well as a large complex robot or vice versa.
1.4 Scope and Context

In contrast to conventional approaches in artificial intelligence, understanding intelligence in evolutionary systems may be of lesser importance. Instead of an intelligent thought process being evolved, other characteristics in the morphology of robots could emerge bypassing intelligent control while still adhering to an objective (Pfeifer et al., 2006). More abstract forms of intelligence, or emergent collective intelligence, could as well suffice for artificial systems. Evolutionary robotics is therefore more concerned with obtaining behavior adhering to specific objectives or self-reproduction; not intelligence. This is also true for natural systems. If a morphological change would be more evolutionary advantageous than intelligence, this would be selected for.

In ecosystems there is usually an arms-race within and between populations that drives evolutionary progression. The arms race being an emergent conflict between organisms for gathering finite resources. In artificial systems, the agent’s goal could similarly be set by more ultimate aims, such as energy acquisition. This goal should after all be a requirement for reproduction. For a robotic ecosystem, primary energy producers might simply be smartly arranged solar panels from which higher order robots can gather their energy. In addition, more proximate goals such as the control of movement can be set. Locomotion and energy acquisition being intertwined behaviors in nature that can be isolated as single objectives in artificial evolution which has been done in Part II.

Though the type of optimization algorithm to be used can sometimes be problem specific, evolution can often be implemented as a general strategy. Parameter tuning of the algorithm can in turn be used to adapt the algorithm’s functionality to different problems. This makes an evolutionary algorithm a versatile optimization strategy. Many improvements have been made on evolutionary algorithms to increase their performance. These improvements usually address one or more of the mentioned major challenges in evolutionary robotics: exploration vs exploration, genotype to phenotype mapping and the reality gap. From a problem-solving perspective, these challenges will mainly be addressed in the various experiments of this thesis. In addition, these problems can sometimes be related to equivalent issues in biology. Hence some aspects are taken into consideration from a biological perspective.
The purpose of this thesis is to elucidate various aspects of evolution in artificial systems. It will in part clarify why evolution works, how additional concepts from evolutionary biology might actually increase the efficiency of existing optimization strategies (Chapter 3), and why these concepts are important when designing new robotic systems (Chapters 4, 5, 6 and 7). It is striving for a future where artificial life can emerge through an automated process that transcends the current limitations of the autonomy and technology of artificial systems. Though conceptually the hardware of the eventual system is elusive. Various strategies easing the construction of a potentially variable robotic system have been implemented. Throughout this thesis some main intrinsic problems related to the field of evolutionary robotics are addressed with the predominant experiments focusing on a specific type of robots called modular robots. This is done since modular robots are reconfigurable, making it easy to not only implement different simulated robots, but also adjust the hardware configuration on demand. Using the modular robotics approach presented in this thesis allows any researcher, in theory, to design a few modules, and automatically evolve their composition and behavior for either locomotion or energy acquisition using the presented robotic platform. Simulations are used to elucidate many aspects of robot design and biology with the aid of strategies such as neural networks, evolutionary computation, modular robotics and a range of different objectives which are discussed in the majority of this thesis.
Chapter 2

Evolution

Different sorts of survival machine appear very varied on the outside and in their internal organs. An octopus is nothing like a mouse, and both are quite different from an oak tree. Yet in their fundamental chemistry they are rather uniform, and, in particular, the replicators that they bear, the genes, are basically the same kind of molecule in all of us – from bacteria to elephants. We are all survival machines for the same kind of replicator – molecules called DNA – but there are many different ways of making a living in the world, and the replicators have built a vast range of machines to exploit them. A monkey is a machine that preserves genes up trees, a fish is a machine that preserves genes in the water; there is even a small worm that preserves genes in German beer mats. DNA works in mysterious ways.

– Richard Dawkins, The Selfish Gene

The origin of species – or the "mystery of mysteries, the first appearance of new beings on this Earth" (Darwin, 1845) as Darwin wrote in his diary – has long been apocryphal until the emergence of Darwinian evolutionary theory. Nowadays it has been proven in many different instances that natural selection and descent with modification are the main factors promoting adaptive change in populations of organisms. The self-adapting principles of evolution have in turn been adopted as a metaheuristic optimization strategy in the paradigm of evolutionary computation. The applicability of evolutionary computation as a general problems solver has been demonstrated in engineering for optimizing satellite antennas (Lohn et al., 2003) to bioinformatics for the prediction of RNA structures (Van Batenburg et al., 1995). Evolution is thereby not solely a theory, but a useful applicable optimization strategy.

Natural evolution does not have a predefined goal, it is a blind process where the only criteria are survival and reproduction (Dawkins, 1978). However, we can still view the process itself as being a stochastic problem solver since populations are continuously adapting to their environment. These adaptations in specific traits are selected for if they increase an organism’s survival and reproductive success. With some traits being adaptive during an organism’s life time (acquired), whereas others are inherently encoded in their genome (innate; Bolhuis et al. 2005). E.g. body plans are mostly defined by innate factors while muscle mass is influenced by mechanical stress and thus in part an acquired trait. Genes not only shape the individual organism, but their phenotypic expression also shapes the environment which can assist in their replication, a phenomenon more commonly known as the extended phenotype (Dawkins, 1978). Where the extended phenotype can change the ecological niche across time and thus encompasses cultural evolution as well (Cagnoni et al., 2014). This is the case in ants and bees building hives, or monkeys and crows using tools for capturing insects. The organisms that interact with the environment are only the machinery of the expressed information encoded in the genome. This machinery promotes complex behaviors for locomotion, flying, digging, thinking and even humanity. The resulting machinery would not exist without the genome, and the genome could not replicate without the machinery. All these traits having emerged from information containing reproducing entities subject to evolution by natural selection.
Chapter 2. Evolution

Approach

Tinbergen (1963) found that too often researchers described animal behavior too vaguely and descriptions between researcher were not consistent. He proposed four different approaches to analyze behavior using two proximate questions on how, and two ultimate question on why behavior arises (Bolhuis et al., 2005). The two proximate questions encompass the causation and development of the behavior of the organism. Causation covers how the mechanisms of the underlying structure, or machinery, of the organism function. Development describes the conditions and factors important for the development of specific structures of the organism responsible for behavior. The two ultimate questions are used to explain the evolution and survival value of behavior. Evolution involves the processes that have altered the behavior across generations while the survival value of the organism investigates why the behavior is used by the organism.

In this chapter we consider Tinbergen’s reasoning and start discussing two proximate descriptions of the genome (Section 2.2) and development (Section 2.3) with respect to the fitness landscape (Subsection 2.1.1). The ultimate questions of survival and evolution itself are considered by discussing Evolvability (Section 2.5). This background should have motivated how and why we can implement evolutionary computation (Section 2.6) and how spatial models can relate results from evolutionary computation to biology (Section 2.7). To start off, the next section will boil down the fundamentals of evolution.

2.1 Fundamentals of Evolutionary Theory

Natural selection through descent with modification is the primary driving force allowing for evolution in natural systems. This sole mechanism resulted in the grandeur of immensely diverse and unique creatures living on this planet today. Selection shapes the trajectory towards certain features present in the population, while variation through modification of the gene pool – the set of all genes in a given population – allows for the emergence of novel phenotypic traits. These novel traits are for example driven by mutation, diploidy (chromosomes exist in pairs) and crossover that promote diversity and variations in most organisms.

Even though evolution works, organisms are still riddled with imperfections that are difficult to improve through evolution. A famous example in mammals being the laryngeal nerve connecting the brain to the larynx. This nerve takes a route from the brain, around the aorta and then to the larynx instead of taking a shortcut by directly connecting the brain to the larynx. Though this might not be an issue for most animals, a giraffe’s neck can grow over 2 meters long meaning that signals from the brain to the speech center of the giraffe can take significantly more time than it would when directly connected. Moreover, a direct connection would also reduce the material, or number of cells, needed to make the connection. This denotes the importance of evolution working incrementally where some changes in the design of organisms would simply require leaps that might not be easily achievable from ancestral precursors. Evolutionary adjustments are not sudden, but rather incremental steps towards better adapted populations, species or genes. These incremental adjustments to a population contribute to the noticeable gradual genotypic and phenotypic change that can be seen throughout various phylogenetic trees. The incremental change is both the power and the limitation of evolution and evolutionary computation.

For evolution to occur, there needs to be variation in the population. And for variation to emerge, there must be genetic change. Mutations will allow for offspring to be genetically different from their parent(s). Where mutations can occur in the organism during its lifetime or be passed on to its offspring. Mutations are mainly caused by an inaccurate duplication of a genome during cell division but can also be caused by other factors such as mutagens. And if chance have it, a mutation can lead to a new organism that will be better able to survive and reproduce in its environment. Since selection works in favor of the better adapted individual, there is a higher probability that this individual is selected for reproduction. Evolution being simply an incremental
process whereby a population adapts to the conditions of a dynamic environment over time.

### 2.1.1 Performance Measure

The evolutionary optimization process in evolutionary computation usually needs a measure of performance to acquire adequate functionality. This measure of performance is usually denoted by a numerical value called the *fitness* value. This performance measure can be a number, or multiple numbers in case of multi-objective approaches, directly translating in a fitness value of an agent. In biology, a *personal fitness* (Hamilton, 1964) value is commonly ascribed to the amount of viable offspring an individual is able to make. Another measure, the *inclusive fitness* (Hamilton, 1964) value, of an individual includes altruistic factors that would enable the genes of an individual to propagate to the next generation. An individual’s inclusive fitness is for example increased when it is able to give its sibling, and thereby half of its own genes, more viable offspring at the cost of producing offspring itself. However, as noted by Dawkins (1988), the term inclusive fitness can be a cryptic measure since it is difficult to postulate on how single organisms will affect the population over time. Considering the entire gene pool of a population, the emergent properties of multiple individuals could be measured since the behavior of population is an aggregate, not merely the sum of the actions of agents (Holland, 2012).

The fitness values that are given are an indirect result of an organism’s ability to survive and reproduce in its environment. In evolutionary robotics, fitness values are usually ascribed to the performance of robots on specific tasks. Thereby the fitness value is usually not related to the amount of viable offspring an individual has produced, but rather the other way around. A fitness value is given as a performance measure for an individual in a particular environment, and this performance value will subsequently determine the amount of viable offspring an individual is able to make based on a selection operator. Ultimately, a fitness value is inferred from the genotype and there is thus an indirect mapping from genotype to fitness with a mapping from genotype to phenotype and phenotype to fitness in between. An exception to this approach can be seen in work done trying to embody the evolutionary process itself, an approach called embodied evolution as described by Watson et al. (1999) and Eiben (2014). In this approach, robots do not have a fitness value, but the evolution is an emergent phenomenon from allowing robots to reproduce based on the interactions they have with other robots in an environment.

Considering different possible genomes, naturally and artificially, a *fitness landscape* can usually be formed to map the performance of a gene or genome. This landscape can depict the reproductive rate of specific amino acid sequences of proteins as in Wright (1932) and Nowak (2006). Though the term in evolutionary computation usually conveys the performance of different genotypes. Though being able to produce many low performing offspring compared to a few high performing offspring indicates that the measure of a reproductive rate might not be a good representative for a general performance measure of a system be it biological or artificial. Producing less, but better fit offspring compared to other individuals in the population should be considered. Hence not only reproduction but also survival is important.

In evolutionary computation an individual can be considered an evaluation instance. If an evolutionary algorithm could implement strategies to find a sufficient solution which requires less individuals to be evaluated, it would perform more efficiently. Reproduction is therefore not a logical measure of fitness, but rather a type of cost to evolutionary algorithm. Hence the amount of viable offspring or the reproductive rate is not interesting in evolutionary computation when compared to other performance measures.
2.2 The genome and the Sequence Space

The genes cause the emerging phenotypes in nature and therefore connects evolutionary computation to biologically equivalent genetic representations. Genomes, in biology, are the four-letter sequences of DNA (deoxyribonucleic acid) that forms the lexicon for information storage in life as we know it. DNA and RNA (ribonucleic acid) form the blueprint for all biological life and enable the flow of genetic information through heredity. DNA being the double stranded variant of nucleotides and RNA the single stranded form. These strands are composed of the nucleotides adenine (A), cytosine (C), guanine (G) and thymine (T) in DNA whereas thymine is replaced with uracil (U) in RNA. These chains of DNA and RNA can encode information that is eventually translated in proteins, which form the (death) building blocks of life such as enzymes, hormones, antibodies, etc. The sequence of nucleotides that contains the information of life are the genes, functional sequences that encode for proteins consisting of chains of up to, usually, 20 different types of amino acids. Every amino acid in the protein being derived from a triplet of subsequent nucleotides. Since the genetic code can potentially account for 64 amino acids, the fact that there are only 20 amino acids present in most organisms makes the genetic code is redundant. Included in genes are signals for proteins such as start and stop codons eventually allowing for the transcription and translation of the gene. The combined expression of genes can form complex gene regulatory networks that includes the promotion and inhibition of other specific genes.

Mutations cause the DNA strand in mammals to change. The main cause of mutation being errors during replication of the DNA. Various types of mutations can either be detrimental, neutral and, in very rare occasions, beneficial. It can be brought about by insertion or deletion of specific parts of the genome or even the duplication of an entire genome or chromosome. These random duplication events have been important for speciation. Two rounds of whole genome duplication events in vertebrates where necessary before mammals could emergence (Reece et al., 2010). Although the events of gene duplication are rare, they have been crucial for the generation of new functional proteins as is likely to have happened for red and green color cones that spurred from a duplication event in ancestral species (Dehal et al., 2005).

The Sequence Space

As first described by John Maynard Smith, the sequence space is a way to describe the possible protein chains that can result from a fixed amount of amino acids. Every solution on this sequence space in turn can have a representative performance value that can be related to the reproductive success of the corresponding gene. When a gene, and thereby the resulting protein, is better adapted, it allows for more copies of itself to be passed on to the next generation. A fitness landscape can thereby be created based on the reproduction rate of the genomic sequences (Wright, 1932; Eigen et al., 1977). Considering the potential combinations of amino acid chains that result from the triplets of base pairs, the sequence space is immense. E.g. since there are 20 possible amino acids, the sequence space of a protein is \(20^L\), where \(L\) is the amount of amino acids of a protein. It is therefore impossible to evaluate the entire sequence space of most proteins and the mapping of this entire sequence space on a fitness landscape is therefore difficult. However, based on the functionality of a protein when mutated, the neighborhood of different amino acid sequences can be mapped to create a local fitness landscape.

A collection of similar genomic sequences is usually termed a quasispecies. Not to be confused with biological definitions of a species, the quasispecies refers to a similar set of genomic sequences that are influenced by the mutation-selection process. Where the similarity of sequences can be determined by their Hamming distance, a measure of minimum number of substitutions required to change one string into the other. Therefore, there is an arbitrary distinction between a collection of genomic sequences that can be mapped by genetic distance between two individuals. The distinction is still useful since it can explain how the fitness values of a quasispecies can vary over time.
2.2. The genome and the Sequence Space

This usually results in an adaptation of the quasispecies to local or global peaks in the fitness landscape that vary in their stability depending on the environment.

The mutation rate in a quasispecies determines this stability and the traversability of the population on the fitness landscape. Higher mutation rates allow for more variation and thus higher traversability of the fitness landscape though it also makes certain regions in the landscape unstable. As illustrated in Figure 2.1, a mutation rate \( u \) being above or below certain mutation rate thresholds \( u_1 \) & \( u_2 \) will determine the stability of the population in a part of the fitness landscape (Nowak, 2006). In a robust, stable equilibrium, the resulting quasispecies does not consist of solely the fittest genomes, but rather a distribution of genomes around the stable basin of attraction in the fitness landscape. These principles of stable sequence spaces with relation to the mutation rate can shape evolutionary progressions that will be discussed throughout the thesis.

In evolutionary computation the genes are defined as mutable parameters of a system. As has been introduced in genetic algorithms (Holland, 1975), a binary string can be used to represent the genome and sequence space of an agent in silico. This is the simplest analogy to a DNA string which is also a digital, as opposed to an analogue, information storage mechanism. Since the binary genome is represented by bits, the only difference being that DNA is equivalent to a quaternary numeral system. Similar to biological measures of quasispecies and sequence spaces, a binary genome can be directly mapped to form a fitness landscape. Where the fitness of genomes can be defined by specific functions that will be discussed in Section 2.6.

There are many factors in nature influencing the genetic information present in a species. Mendel first described gene alternatives, known as alleles. Where the genotype contains a set of alleles and the phenotype being the exhibited characteristic of these alleles in the organisms (Reece et al., 2010). In diploid organisms, where individuals have two copies of the same gene, segregation of two alleles from the genome of an individual lead to many potential combinations of alleles in offspring. The random arrangement of gene alleles in the offspring alters their fitness compared to their parents creating changes selection pressures of the offspring. This combination accounts for some of the sustained genetic variation within a population since there can be several

![Fitness landscape with a peak and a hill](image)

**Figure 2.1**: Fitness landscape with a peak and a hill. \( u \) represents a mutation rate, \( u_1 \) a critical value and \( u_2 \) an error threshold. \( u \)'s value determines the region that the quasispecies occupies as represented by the blue area. Taken from Nowak (2006)
combinations of various alleles in an individual. However, considering only Mendelian mechanisms, and considering only one environmental niche, the population would inevitably reside in a zero-evolvability state where no new genetic information enters the gene pool. Especially when considering haploid organisms, the emergence of new genetic information necessitates change in the genetic code. This change is not solely promoted through mutations, it is also promoted by (imperfect) crossover events that allow for the recombination of genomes. Moreover, factors like gene duplication events have been important for the emergent of functional paralogue genes, genes that share a common ancestor.

Genetic diversity through mutation operators that change the genes of artificial agents shapes the resulting potential phenotypes and the robustness of solutions found by evolutionary computation. Although quasispecies can be defined by measuring genetic difference in a population, small population sizes in evolutionary computation usually lead to the convergence of a population to a specific area on the fitness landscape. The relationship between the variation and a population of agents’ ability to traverse the fitness landscape is important to consider in evolutionary computation more so than other natural factors that enable evolution of natural systems.

2.2.1 Selection

Selection acts as the driving factor of what region in the fitness landscape will be occupied by the quasispecies. Variation is necessary for the population to traverse the fitness landscape and enable a quasispecies to climb local hills or explore new regions in the sequence space. Selection pressure can sometimes be negligible for specific areas in the sequence space and this can result in genetic drift. In this case, the genetic information is changing randomly across generational time. This random genetic change can result in neutral evolution whereby change of the genomic sequence does not affect the fitness. Genetic drift can also occur in functional sequences when selection pressure is low, or the mutation rate is high. An advantage of this type of drift is that a gene can transform into a new gene or find a unique sequence in the fitness landscape that could not be reached under conditions of high selection pressure. Though beneficial mutations are rare, a new better suitable gene can potentially be found due to drift. Genetic drift can play an especially important role in the development of paralogue genes when gene duplication events occur.

To see selection work, at least two different types of genotypes need to be present in the gene pool of a specific population. When considering that these two quasispecies reproduce and die in a specific environment and have an exponential growth rate, the quasispecies that has most individuals in an environment will normally take over the entire population. This is true when the fitness values of both quasispecies is the same and lead to a concept called survival of the first (Nowak, 2006). However, when the quasispecies have different fitnesses, there will be a selective pressure towards the survival of the fittest. If both populations have a sub-linear growth rate, one quasispecies will never outcompete the other leading to an equilibrium known as the survival of all. Selection thus shapes the frequency of the genes in the gene pool but does not facilitate evolution on its own. It changes the frequency of genotypes based on the phenotypic fitness of the individuals and the number of individuals in a population. Since the variety of many plants and animals have emerged from the process of breeding through artificial selection, evolutionary computation can be seen as an equivalent where breeding is done through setting objectives. Selection operators usually determine the selection pressure on individuals in a population and, together with variations in the genes, it shapes the evolutionary trajectory of a population.

2.3 Development of the Phenotype

The phenotype entails the observable characteristics of an organism (Wolpert et al., 2007). The genotype is inherited by the parent(s), whereas the phenotype is the expression of the genes that can be influenced by the environment (Nolfi et al., 2000).
The important distinction being that the environment shapes the eventual phenotype of an organism. The process of how the genotype translates into the phenotype is what is called development. The relationship between the genotype and phenotype is a mapping shaped by the environment and the encoding of the genes themselves. In evolutionary computation this relationship between genotype and phenotype is usually called the genotype to phenotype mapping. Where the genes of the genotype determine the generative program for the development of the agent. This genotype to phenotype mapping is important since it can greatly influence the evolutionary progression in an artificial system. The gestalt of all the genes, proteins and cells is what ultimately defines the complexity of an individual. With pleiotropic genes, one gene can influence multiple seemingly unrelated phenotypic traits, that are expressed in a multitude of cells, we can usually not attribute isolated functionalities to specific proteins.

Development in humans is brought about by the intricate network of only around 20,000 functional genes shape the entire phenotype of an individual. Considering the number of cells, and cell types present in a single human, this means that many proteins are reused in different organs/tissues. This conveys that the trillions of cells present in a human exhibit recursion and redundancy. Moreover, organs and specific components of organs tend to isolate part of their functionality, which is usually called modularity in computer science and robotics (Stoy et al., 2010). Where recursion and modularity are important concepts that can greatly affect the evolutionary progression of artificial systems.

In multicellular organisms, an organism develops from a single cell. Usually from a fertilized egg cell which contains a fused set of genes from two haploid parent cells. The egg cell in turn gives rise to a multiplicity of cell types. In this developmental process, many commonalities varying from within species to within kingdoms of animals can be seen. For example, all the animals classified in the subphylum vertebrate, contain a developmental process that generates three germ layers that can already be determined in the blastula stage, an initial developmental stage of an organism containing a spherical layer of cells. Moreover, many substances can act as a morphogen, as first described by Turing (1952), whose graded distribution across cells varies and is involved in pattern formation in organisms (Christian, 2012). A morphogen is thus an important substance encoded in the genes that influences the patterned expression of a variety of traits including germ layers (See Text Box 2.1 for an example). The germ layers sprout from pole cells that are derived from the asymmetric distribution of proteins and granules across the cytoplasm of cells. Any disturbances which change the functionality of these morphogens results in catastrophic changes in the development of the organisms.

Morphogens influence the genotype to phenotype mapping in organisms determining for example the patterned expression of traits across the developing organism. Computationally, having proteins that divide an organism into expressed patterns is relevant for the emergence of complex phenotypic traits in animals and can therefore also be considered in the realm of evolutionary computation as has been done by Dawkins (1988) and Sims (1994a). The process of artificially developing the morphology of artificial systems is largely described by the paradigm of morphogenetic engineering (Doursat et al., 2013) which tries to encompass the self-organization of a complex system for shaping emerging phenotypic traits. The change of the evolutionary progression brought about by implementing development is therefore useful. This approach being derived from evolutionary developmental biology is informally known as evo-devo.

There are moreover a variety of examples of development that allow phenotypic change. Heterochrony, or the developmental timing that influences the size and shape of morphological structures, is another example of a trait that is adjustable factor that only changes the scale of resulting phenotypes (De Beer, 1940). Neural plasticity determines the change of the neurons and synapses during an individual’s life time. Then there are factors such as regeneration and metamorphosis that also drastically change an individual’s layout during its lifetime.

Not all solutions are limited to an organism’s ancestral history and many similar homologue phenotypes have emerged from different species. Flight, for example, is
An example of an important gene that critically influences the development in organisms is called **hedgehog**, which also acts as a morphogen. Hedgehog and its homologues and paralogues across species is another example of an conserved and robust gene that plays a crucial role in the segmentation of parts of the body. In Drosophila it determines, along with other genes, the segmentation process of parasegment boundaries (Wolpert et al., 2007). Implementing a null-mutation results in the loss of this segmentation as can be seen in the expression of denticles in drosophila larva (Figure 2.2).

In other organisms, null-mutations in sonic hedgehog, a paralogue of hedgehog, has a different effect in mammals. Here it has for example been shown to be related to the development of the neural tube and somite and foregut patterning (Varjosalo et al., 2008).

**Figure 2.2: Denticle formation influenced by hedgehog.** Ventral view of denticles of a wild-type (left) and hedgehog knock-out (right) larval cuticle of a Drosophila. In the knockout a 'lawn' of denticles without clear polarity is shown whereas the wildtype depicts the normal eight belts of denticles Adjusted from Desbordes (2003)

not solely evolved in birds, but it has also evolved in mammals (bats). This could be considered as a form of non-bijective mapping. Many different genotypes can be translated into the same phenotype. On the other hand, since many developmental mechanisms are highly conserved, entire lineages of organisms can also be stuck in local sub-optimal solutions. It could therefore be postulated that another developmental strategy would be even more advantageous for developing organisms – by for example having four initial germ layers instead of three. The stepping stones required towards this fourth germ layer might however be too convoluted for mammals that have already evolved efficient behaviors using only three germ layers. If we could do a hard reboot of evolution of life we don’t know if organisms that develop using four germ layers could emerge. More distally expressed genes that might solely shape a single characteristic are likely more prone to change if they don’t affect early developmental mechanisms hence we don’t see the developmental process change much on a phylogenetic time-scale. Moreover, since germ layers and morphogens are usually highly conserved in the genome, it can be seen that developmental stages recapitulate evolutionary stages (Mayr, 1994). Where Darwin already noted that ancestral history can be derived from looking at the developmental stages of an animal (Darwin, 1872).

The genotype to phenotype mapping in addition to the genetic operators are important for the success of an evolutionary system (Floreano et al., 2008). This mapping largely determines how well the specific implementation of an evolutionary algorithm can traverse the fitness landscape. Additionally, acquired behavior can also greatly influence the evolutionary trajectory of a system and this influence is formally known as the Baldwin effect (Baldwin, 1896). A classic example of how learning, a form of development resulting in acquired behavior, can improve the performance in an evolutionary context can be seen in Hinton et al. (1987) where learning has been shown to guide evolution where seemingly no evolutionary paths led. In this simple example, Hinton and Nowlan demonstrated that a learning population (based on binary
representations of individuals) is more efficient since learning grants additional adaptability of individuals during their life time and not only on a phylogenetic time-scale. However, this work has been critiqued since from a computational perspective, the individuals in Hinton et al. (1987) were granted more evaluations and thus also explored more of the search space (Santos et al., 2015). The concept of how learning over a life-time can discover certain aspects that make an individual fitter could still pose a benefit especially in environments that require continuous adaptation. In a static environment, the previously acquired trait could lead to a genetic determination whereby the trait will be incorporated genetically (Smith, 1987). Moreover, a fixed developmental sweep can enable individuals in a population to discover and efficiently express specific phenotypic traits Kriegman et al. (2017). These traits might at first be expressed only momentarily during an individual’s life time but can be selected for which in turn expands the time individuals spend in optimal developmental stages.

Imagine a needle in the haystack problem in a robotics simulator where there is only one efficient gene with a value we denote by the integer 42. Accepting that an individual agent can be simulated for 10 iterations before its fitness value is measured, one can either test one variable during a robot’s life time, or as in the experiments of (Kriegman et al., 2017), decide to evaluate multiple numbers between two points in the life time of the agent. A static genome containing anything but the number 42 would then be equivalent to random search. However, by stating for example that during the 10 iterations the agent will change its phenotypically expressed value by adding one number each iteration, it is able to explore 10 values instead of one. This sweep thus makes it more probable that the number 42 will be found in the evolutionary simulator. However, when the developmental alternative finds 42 it will only exhibit this optimal state 1/10 of the time of the non-developing agent. Being able to evolve the length of expression of the gene in this hypothetical example can thus lead to initial sweeps of the sequence space followed by the encapsulation of the specific efficient timed expression. The sweep thus allowing for an increased search breadth while allowing for subsequent encapsulation of beneficial traits.

We can see the development of agents therefore as making a sweep of multiple solutions, learning behavior and a genotype to phenotype mapping. A developing that exhibits these aspects might therefore be better able to find specific solutions in the search space. Although learning is not discussed in this thesis, the mapping and development as a phenotypic sweep will be important considerations and are discussed in various chapters of the thesis.

### 2.4 Evolutionary Game Theory

The dynamic interactions of populations in their artificial and natural environment are essential for understanding evolutionary dynamics. The struggle for life is immensely influenced by the competition between individuals and between species and can be illustrated by simplifying behaviors or phenotypes in games and contests. Contests in ecosystems can be seen as analogues to rock paper scissors style games where certain traits of individuals can have advantages with trade-offs. Though the actual dynamics are much more complex, experimenting with variable phenotypes can elucidate potential evolutionary trajectories and determine overall stable or unstable phenotypes. Long vs short term adaptive traits benefits leading to equilibria, basins of attractions, oscillations etcetera (Hofbauer et al., 2003).

Perhaps the most influential mathematical example of population dynamics has been given by Vito Volterra and Alfred Lotka. The mathematical model of predators and prey for example displays the interactions between predators and preys over time in elegant equations (Equation 2.1) for prey $x$ and predator $y$ that has been essential in understanding and depicting the stability and permanence of populations in a given environment.

\[
\begin{align*}
\dot{x} &= x(a - by) \\
\dot{y} &= y(-c + dx)
\end{align*}
\]
Simply put, \( x \) represents a population of prey and \( y \) the population of predators. With \( \dot{x} \) representing the change of population \( x \), \( a \) the reproduction rate of prey, \( b \) the death rate of prey \( x \) that is influenced by \( y \). In turn, \( \dot{y} \) encompasses the change of predators over time with \( c \) being the death rate of predators and \( d \) the reproduction rate that is based on the prey population \( x \). These equations can thus depict a change in populations of predator and prey frequencies over time and usually leads to a stable limit cycle where prey and predator populations sequentially exhibit stable oscillations.

The main insights from this model is that predators and prey in a given environment oscillate periodically. It has many types of extensions where the equations can for example be adjusted to describe competition and cooperation between species or for more than two populations. However, the reliance on analyzing population dynamics mathematically can be complex compared to simulating simple populations dynamics in a computer simulation. In computer simulations one can for example test multiple species that can change and evolve over time in dynamic environment. An example of such a computer simulation are agent-based models or finite populations. These simulations can be used to confirm mathematical concepts and test new hypothesis to see how selection and evolvability influence evolutionary models.

To implement evolutionary games in spatial models, stable equilibria of various strategies within populations can be easily simulated. For example, depending on the sole neighborhood analysis of cellular automata, different Prisoner’s Dilemma strategies can form equilibria in populations resulting in various emergent spatial phenomena. If defecting has a large pay-off in a population of cooperating individuals, defecting is a behavior adhering to an environmental niche. This niche is however dynamic since the benefits of defecting can be low if a population is filled with defectors and high in populations only containing cooperators. Defectors and cooperators can be defined with one value for a cell in a spatial model. To expand this type of evolutionary system, each cell could instead contain an entire genome that can elucidate more about the evolutionary dynamics of the evolutionary trajectory. This is done in Chapter 3 and described in Section 2.7 where each cell in a spatial model is given a binary genome that is evaluated by a specific fitness function.

### 2.5 Evolvability

One of the often-used definitions to explain how well a species is able to adapt on a phylogenetic time scale is evolvability – the capacity to evolve. Evolvability can mean the efficiency of finding solutions that make a gene, an organism or a population better adapted to their environment. Evolvability greatly depends on the specific fitness landscape or fitness functions for a given population. A major problem that limit the evolvability of a population is the need for traversing regions of less fit solutions to acquire a better fit solution in the long run. In this case, an adaptive solution for a niche or environment requires a population to traverse a local valley on the fitness landscape to acquire the better fit solution. Accepting a given mutation rate, or variability rate, in the population, this valley is difficult to cross when there is selection pressure in the opposing direction. However, since it is unknown how convoluted the fitness landscape is for higher order organisms like mammals, a common mechanism that could allow for the traversability of these valleys would be beneficial for a species since this prevents it from getting stuck in a local optimum and enables it to find better or unique solutions. Hence here we find overlap between the problem-solving nature of evolutionary algorithms and the evolutionary advantage of species having a greater evolvability than others.

In the scientific literature there doesn’t seem to be a clear consensus on the term evolvability and it differs in meaning from researcher to researcher. Smith (1970) mentions that for a protein to be able to evolve in another protein, it needs to traverse the sequence space of that protein without going through non-functional regions. Smith describes this process by giving an example of changing the word ‘WORD’ to the word
‘GENE’ the word being an analogy to a protein sequence. In the optimal scenario, considering single mutation steps, the change of the word WORD to GENE needs to follow a specific set of mutations that do not let any intermediate steps to be non words. E.g. an optimal evolutionary path for changing WORD to GENE would be: WORD-WORE-GORE-GONE-GENE. In this sequence of mutations, the word has not gone through non-functional regions where the word would not be considered a word. This transition has moreover taken the least possible steps to and therefore obeys maximum parsimony. A problem arises when the transition words are lesser fit variants of the word WORD that necessitates the evolutionary progression to traverse a local valley in the fitness landscape. In this scenario, if we consider the word GENE to be the optimal solution, this change requires the population to traverse regions of the landscape with lesser selective pressure due to the decreased performance. In the long run, finding the word GENE might enable the population to outcompete another population that hasn’t found the word GENE yet illustrating the importance of evolvability.

Although the term evolvability is not mentioned in the original paper by (Smith, 1970), the term has been addressed to his example as explained in Haubold et al. (2006). However, as mentioned, the definition of evolvability is somewhat loosely used in the literature. It is important to clarify our meaning and argue why we didn’t use any other term. One of the early definitions of evolvability in computer science was derived by Altenberg (1994) who equated it to the variation in offspring produced by a parent population. A follow up paper by (Wagner, 1996) indicated that evolvability is "the genome’s ability to produce adaptive variants when acted upon by the genetic system". Similar to Smith’s illustration of the protein space, adaptive variation indicates that the variation has a chance to produce a higher fit organism. In Wagner (2008), the term is still used as the ability of the system to produce evolvable mutations as it has also been used in Floreano et al. (2008). However, despite the more nuanced use of the word system, evolvability is still being measured in Wagner (2008) by "how easily a blind random walk starting from a given phenotype can find a pre-defined but otherwise arbitrary ‘target’ phenotype". Robustness and evolvability where set as properties of one individual genotype (or phenotype in their experiment; Wagner 2008).

A similar way of defining evolvability by Lehman (2012), who derived it from Wagner (1996), defines an individual to be more evolvable when if after mutating the individual it can produce more phenotypically varied offspring compared to others. One of the ways of measuring evolvability as mentioned in Lehman (2012) is that an individual can be said to be more evolvable if, after mutating, the individual is able to produce more varied, and perhaps more fit, offspring. This measurement also takes the solely the individual into account as a measure of evolvability. However, having a varied population and not a high mutation rate can still lead to the population being able to find diverse solutions and fit individuals. Therefore, I would like to stress that it is the entire gene pool as a whole that contributes to evolvability. Basing a measure of evolvability on an individual should thus be similar to how Hamilton (1964) describes inclusive fitness. Taking not just the individual into account, but also the population and the environment. Like inclusive fitness, evolvability being measured from the individual level would be more accurately described as personal evolvability. Using Hamilton’s lexicon, a better term that would allow for a distinction would thus be inclusive evolvability though it can be hard to envision how an individual contributes to the evolvability of the entire gene pool. The selection operator in this case can greatly influence the existing variation in a population. Although Lehman’s and Wagner’s measurement can explain the transition of proteins explained by Smith (1970), the main disadvantage of these interpretations is that time and the rest of the population is not included in the measurement. A population could produce offspring that don’t vary too much. Measuring the change in diversity over 10 generations might indicate that this population has become more fit and more diverse compared to a more evolvable population described by Wagner and Lehman. The key concept is that if a population or gene pool is better able to traverse a fitness landscape to find solutions that are more efficient than the ones that exist, it is more evolvable.

Others, such as Valiant (2009), have used the term evolvability more esoterically by
Text Box 2.2: Locked in Imperfection

The human eye contains photoreceptors on the back of our eyeball that are measure light. There is however one main issue with the organization of the photoreceptors since they are located behind a layers of ganglion cells (Bear et al., 2016). It could be that for some yet unknown reason having the layer of ganglion cells in front of the light receptors might be beneficial in some way, but considering the literature, the consensus is that the layer is in front of the light receptors because due to evolutionary precursors. There are however animals, like cephalopods (Budelmann, 1995), that have evolved their photoreceptors to be in front of this layer of neurons (Figure 2.3). This intuitively seems to be the better option and shows how evolution can be limited by ancestral precursors. Since the eye is a highly conserved structure across species, it must be incredibly difficult to change something in the existing phenotypes to change the photoreceptors position to move in front of the layer of cells. This might even necessitate a few generations of animals that need to cross less functional areas where they might be practically blind. Clearly, mammals that can traverse this landscape to enable the proper organization of cells and photoreceptors might have a distinct benefit improving their eyesight significantly and leading to out competing mammals that have not made this shift. Therefore, considering eyesight as the sole performance measure, being able to find this proper organization quicker would make a population more evolvable.

![Figure 2.3: Configuration of the eye of cephalopods (left) and vertebrates (right). Adjusted from Novella (2008)](image-url)

equating evolvability to learning as defined in Valiant (1984) but with the inclusion of e.g. a polynomial set of hypotheses and the toleration of a decrease in performance. Reisinger et al. (2005) and König et al. (2009) have used the term of evolvability more arbitrarily and define it in terms of the genotype to phenotype mapping being able to create more favorable phenotypes. Reisinger et al. (2005) for example mentions that evolvability is “an adaptive organization of the genotype-phenotype mapping such that the search operators can produce more favorable phenotypes”. However, regardless of the genotype to phenotype mapping, a population with individuals having no mutation or crossover probability will never create a better phenotype and therefore, though the genotype to phenotype mapping can enhance the evolvability of a system by, for example, decreasing the genome length, it is solely a property of evolution. The same genotype to phenotype mapping can produce different variations and improvements depending on the parameters such as the mutation rate that are being set in an evolutionary algorithm.

Natural evolution based on maximum parsimony is still prone to premature convergence, where a population stagnates in a local optimum, locking itself in local regions...
of the search space (Text Box 2.2). Considering benchmark tests for evolutionary algorithms, when a global optimum is known, the best evolvable population would be able to find the global optima quicker than others. We can say that finding the global optima is only possible through change in the genotype, hence evolvability is directly determined by the gene pool of a population. Evolvability, despite its several meanings, important for an evolutionary algorithm to properly be able to traverse the fitness landscape. There are many factors in evolutionary algorithms that will allow a population to traverse the fitness landscape more efficiently. Two of these factors that will be discussed in Chapter 3 are mortality and development. Other factors that influence the evolvability of the system in evolutionary algorithms are the genotype to phenotype mapping, the genome length, mutation rate, and selection operators. Depending on the landscape, these factors should be taken into account for quickly acquiring decent performing algorithms.

The robustness of the system is directly related to the evolvability. In evolutionary computation, features like elitism, low mutation rates and smaller genomes allow for greater robustness of the system since there is a higher probability that the most fit individual is also in the next generation. In the scenario where the best possible solution has already been found in a population, it is therefore beneficial to keep these individuals in the population favoring a robust population over a changing one. Though since evolutionary algorithms can archive all the genomes of all individuals that have been simulated, robustness is of lesser importance to evolutionary computation compared to evolvability.

2.6 Evolutionary Computation

As mentioned, evolutionary computation is the umbrella term unifying all sub fields related to evolutionary algorithms (De Jong, 2006). The first popularized form of evolutionary computation was by Bremermann (1962). From this initial approach, the sub-areas of evolutionary programming (Fogel et al., 1966), genetic algorithms (Holland, 1975) and evolution strategies (Rechenberg, 1973) emerged. Nowadays it is common that the terms aren’t strictly differentiated since there has been a lot of cross-breeding in these paradigms. In this thesis the term evolutionary algorithm specifies the algorithmic implementation of evolution but genetic algorithm will still be used in experiments of Chapter 3 done with binary genomes as it has been used classically. However, an acronym for a steady state genetic algorithm is SSGA and this acronym does not always refer to a binary genomic implementation of evolutionary algorithms. The different techniques have been widely used as optimization tools without a specifically attempting to model biology (Bongard, 2003). Many of the different types of evolutionary algorithms contain subtle differences. There are for example many approaches to encoding an evolutionary algorithm, but a few aspects roughly shape a minimal evolutionary algorithm (Figure 2.4). An evolutionary algorithm is a population-based optimization strategy structured with an iterative procedure. An evolutionary algorithm starts with an initialization step where genomes of a specific population size are usually randomly initialized. After the initialization step the algorithm loops through a few steps until a termination point has been reached. The phenotype of the population of genomes are evaluated to acquire fitness values for every genome. From here on the evolutionary algorithm loops through the following steps:

1. Based on these fitness values, a selection operator is used to select parents that can create offspring for the next generation.
2. In turn, the offspring are produced, and their genome is changed based on mutation / crossover operators.
3. The fitness of the offspring is subsequently determined based on their phenotype.
4. A replacement operator determines how offspring integrated in the existing population.
These four steps are repeated until the algorithm has reached some type of terminal requirement which is usually set to by a maximum number of generations or maximum amount of evaluations.

There are two main variants of evolutionary algorithms called a generational and a steady state evolutionary algorithm. A generational evolutionary algorithm replaces the offspring of a population with the existing population entirely (Vavak et al., 1996a). A steady state evolutionary algorithm only replaces individuals in the existing population when they are outcompeted by offspring (Syswerda, 1991). Thus, the implementations vary only in the replacement operator. In a steady state implementation, instead of discarding the existing population of individuals entirely and replacing them by the offspring, the replacement operator compares the fitness of specific offspring with individuals in the existing population and only replaces an existing individual with an offspring if the fitness value is higher. A steady state algorithm can thus be compared with a population of immortal individuals that can only be replaced when they are outcompeted. There are however various ways in which steady state algorithms can still get rid of certain ‘unwanted’ individuals as in the deletion of the worst individuals of the population (Syswerda, 1991).

Additionally, the selection operator of evolutionary algorithms can be adjusted to give better individuals a higher chance of creating offspring compared to less fit individuals. Selection operators can be totally random but usually tournament selection or some sort of fitness proportionate selection (also known as a roulette wheel sampling) is used (De Jong, 2006). In addition, these techniques can also be implemented in the replacement operator where the offspring and the existing population can be subjected to fitness tournament or proportionate selection in order to reduce the population size back to the population limit. Elitism is another commonly implemented feature in evolutionary algorithms. Elitism usually keeps a certain amount of the best (elite) individuals in the next generation without altering their genetic code. This generally increases the robustness of the evolutionary algorithm since the best genome is not lost. Steady state algorithms do not require an elitism operator since the elite can only be replaced when a better individual replaces it and is thus inherently promoting elites in the population.

Considering the exploration vs exploitation trade-off, certain implementations clearly result in changing evolutionary trajectories. Where elitism or the deletion of the worst individuals usually leads to a more exploitative algorithm and looser selective pressures lead to more exploration. Too much exploitation usually leading to a population stagnating in a local optimum, or premature convergence (Leung et al., 1997). To
cope with premature convergence, many different techniques have been implemented to increase the diversity of individuals in a population. Classic implementation usually considered niching (also described as speciation) methods such as crowding and fitness sharing. In crowding, the replacement of individuals is restricted to other individuals that are similar based on the genotypic comparison of individuals (Mahfoud, 1995). Deterministic crowding being an adaptation to crowding where an initial step groups two parents in a population and offspring are subsequently compared to one of the parents (Mahfoud, 1995). In fitness sharing, the premise was that there is an advantage for individuals to explore different niches when another niche becomes too crowded (Holland, 1992). Where the fitness of population elements is altered by existing fitnesses. Moreover, evolutionary algorithms can usually be tuned (before the evolutionary run) and controlled (during an evolutionary run) that can drastically change the performance in evolutionary algorithms (Eiben et al., 1999). In recent years, many new methods have been introduced to improve diversification of population as can be seen when using an Age-Layered Population Structure (ALPS) (Hornby, 2006; Bongard et al., 2010), novelty search (Lehman, 2012) or Age Fitness Pareto Optimization (Schmidt et al., 2011) or developmental approaches (Kriegman et al., 2017) to name a few.

However, considering the existing approaches to diversification, one should also consider that this usually helps steady-state approaches to evolutionary algorithms whereas generational algorithms tend to diversify more on their own. Although only some of the mentioned diversity promoting methods are implemented in the experiments of this thesis, it is useful to take these methods into account when talking about the specific implementation of an evolutionary algorithm and its relationship to the evolvability of a system. Since evolutionary algorithms can be adjusted in many ways, sometimes simple adjustments such as a lower selection pressure or a higher mutation rate can already make a population more diverse and potentially more evolvable which sometimes makes it difficult to compare novel implementations to genetic algorithms that are tuned to solve a specific problem.

Computer simulations can serve as a tool for analyzing systems that are much more complex than those analyzable mathematically (De Jong, 2006). Accepting this premise, we can empirically investigate various models of natural evolution that have been classically defined mathematically as described in Hofbauer et al. (2003) and Nowak (2006). Moreover, implementing various strategies on different benchmark problems removes the necessity of mathematical models when implementing new strategies. To investigate the performance of variants of evolutionary algorithms, fitness landscapes can be manually defined for a specific sequence space. It is therefore commonplace to use benchmarks which might consist of functions that translate the genome of an agent to a specific performance measure. These functions can be defined in many ways though the important functions implemented as testbeds in this thesis are either continuous single objective functions or binary functions. Through this approach, a basic implementation of the evolutionary algorithm would just consist of the problem function and the evolutionary algorithm itself. Although the eventual aim of using benchmarks could be to find efficient algorithms to implement in a robotics simulator, testing new algorithms directly on a robotics simulator is of limited value since the fitness landscape is largely unknown due to the large potential search space.

### 2.6.1 Binary Approach

In genetic algorithms as classically defined by Holland (1975), the genome of an individual is binary and represented by a binary genome or a bit string. To test how well an evolutionary algorithm performs, a function can give the bit string a fitness value based on the sequence of bits in its genome. For example, the one max function Figure 2.5 provides each genome with a fitness value based on the number of bits with a value of one present in the genome. Since every possible gene can directly improve itself based on the addition of a one in their genome or the replacement of a zero to a one, the one-max function/problem is one of the most basic functions on which exploitative algorithms work best. Due to the random probability of bits in these bit strings being mutated, a normal GA applied to this function quickly converges to a
Chapter 2. Evolution

population of genomes that have a genome that consists of solely ones. The one-max function is hardly interesting for solving with evolutionary algorithms since it is not at all representative of robotic systems or natural systems due to its linearity. It is not a challenge for any optimization method.

There are a few binary fitness landscapes that are more challenging to solve and have been designed to be convoluted and deceptive such as the Chuang f1 (Chuang et al., 2010), Royal Road (Mitchell et al., 1991) or the Hierarchical If-And-Only-If (H-IFF) function (Watson et al., 1998). H-IFF is the landscape mostly used in this thesis. The H-IFF function creates a fractal deceptive fitness landscape and can be used to evaluate the performance of implementations of evolutionary algorithms. In H-IFF, a binary genome is evaluated based on self-similarity. One can check for self-similarity in the genome in multiple layers by initially checking the similarity of a pair of bits across the genome, continuing in the next layer by checking a pair of two bits, followed by checking a pair of 4 bits etc. In each layer a fitness value can be ascribe to the self-similarity score of the genome. This score is usually derived from the amount of self-similar parts in the genome and the layer depth that is being checked.

As an example, Table 2.1 shows how one can derive a fitness value from a 16-bit genome that results in 4 layers on which to check for self-similarity. Note that in the original implementation a null bit was possible in the genome which resulted in an additional fifth layer. This null possibility has been omitted in this thesis to increase the computational efficiency and ease the visualization of genetic change over time. The omission of this null possibility makes the evolutionary progression easier to visualize by plotting the fitness value over the number of ones in the genome Figure 2.6 similar to the representation of the one-max function depicted in Figure 2.5. The gray area in Figure 2.6 illustrates the possible fitness values an individual can achieve when having a certain number of zeros in its genome. The landscape is, however, unchanged, with as many local optima as the original implementation. The maximum fitness of an individual with either only zeros or only ones is 192 and this is the value of the global maxima on H-IFF. However, when half of the genome is composed of zeros and the other half of ones, the fitness value of that particular individual ranges somewhere between 4 and 160 depending on the specific order of the bits.

For H-IFF there are two potential global maximum no matter the length of the genome. One global maximum contains a bit string of only ones while the other contains only zeros. In between these extremes there are many local optimum and one can generally state that there is a local optimum between each two high optima. This makes the landscape inherently fractal and deceptive. To understand how different sets of genomes correspond to fitness values based on the H-IFF landscape, Figure 2.7 illustrates how four genomes are located on the fitness landscape of Figure 2.6 using the explanation in Table 2.1. A score for self-similarity in this illustration is simply denoted by a red color. The area of the fields of the table being colored red directly translates in

![Figure 2.5: Fitness landscape of one-max function.](image-url)
2.6. Evolutionary Computation

Table 2.1: Example of the score of a 16-bit HIFF genome. The table shows the fitness derived from the bit string 0001-1011-1111-1111. The genome has a fitness value of 14 (out of a maximum of 32). The explanation is based on Watson et al. (1998).

<table>
<thead>
<tr>
<th>row</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>tot</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Figure 2.6: Fitness landscape of the adjusted H-IFF function. The global optima in this landscape are at the edges of the distribution and in there are local optima fractally dispersed between every two other local optima.

the fitness value. In this case, the best fit genome is a bit string of only zero’s.

Considering a binary genome that consists of 64 bits, there are a total of $2^{64}$ total possible configurations of a genome. This large search space in turn makes it difficult for algorithms to solve binary functions. Taking a brute force approach to solving the H-IFF function would take an incredibly long time hence genetic algorithms have been employed to ease the search process. Since unknown fitness landscapes of robotics simulators might contain many local optima that might not be interesting to explore, an evolutionary algorithm that is able to traverse the landscape efficiently is essential. Though many factors influence the breadth of exploration vs the exploitation behavior of an evolutionary algorithm, it would be ideal to have an algorithm that can be adjusted over time or even be self-adjusting to deal with premature convergence. Moreover, implementing strategies we can derive from population dynamics might feedback into why we see certain features in nature arise. A large chunk of the next chapter will discuss this issue in detail by showing how mortality as a bio-inspired implementation influences the efficiency of traversing the fitness landscape on H-IFF while comparing it to existing methods.

2.6.2 continuous single objective functions

Considering domains differing from binary functions, we can look at continuous single objective functions. In this case, a variable of the genome that can be represented by one or more floating-point numbers. In these implementations a mutation operator does not change a bit but changes the value of the numbers slightly. Usually with a mutation operator based on a Gaussian distribution (De Jong, 2006). An objective function can in turn determine the fitness value of the specific gene containing a floating-point number. Based on the specific parameters implemented in the evolutionary algorithm,
Chapter 2. Evolution

Figure 2.7: Explanation of the H-IFF function. 4 genomes of length 64 are given (a,b,c,d) and with their corresponding fitness value. Left shows the scoring tables where red indicates a reward for self-similarity as shown in Table 2.1. The red area directly translates into a fitness value of the individuals.

the efficiency at which the algorithm can achieve the maximum fitness changes. Similar to the one-max binary function, there are many functions available that have a similar effect on continuous objective functions. Some specific evolutionary algorithms are especially effective to be implemented on these types of landscapes such as Covariance Matrix Adaptation Evolutionary Strategy (CMA-ES; Hansen et al. 2003). In CMA-ES, the algorithm is able to predict mutations toward a specific basin of attraction. CMA-ES is usually better than other evolutionary algorithms on these landscapes since mutations are not directed in normal evolutionary algorithms. It is also able to efficiently traverse some types of convoluted landscapes such as the Rastrigin function (Figure 2.8a). However, the implementation of CMA-ES becomes problematic on more deceptive landscapes that are similar to H-IFF. One specific example of a difficult to solve deceptive continuous objective in this case is the Schwefel function (Figure 2.8b). CMA-ES has more difficulty finding the global optima since there is no gradient towards the optimum unless it contains a large enough population to already sample the local area of the global peak (or valley in this case since it is a minimization function). To emphasize, the control and creation of robots is probably also deceptive in certain regions of the search space. Hence the ability of an algorithm to be able to solve deceptive landscapes is important to evolutionary robotics where the fitness landscape is largely unknown.

2.7 Spatial Models

To further investigate the performance of parameters of evolutionary algorithms and to test biological hypotheses, spatial models can be used to see how a population of agents performs in a minimal environment. Predator-Prey like models can therefore be implemented to see how one the relationship between predator and prey change over time and shape the evolutionary trajectories. Since H-IFF is a difficult function to optimize for a population of agents, it has therefore been implemented in the spatial model to evaluate how well a population of simulated agents is able to traverse the search space.

The spatial models implemented in this thesis was originally based on a Cellular
Automata program architecture \(^1\) \(^2\). This is a C++ based application that has been modified to generate a predator prey like spatial model. In the spatial model it is possible to create cells that are primary energy producers (plants), primary energy consumers (rabbits) and secondary energy Producers (foxes). It has been modeled as an abstraction of energy flow in a natural ecosystem. Energy in nature is acquired by autotrophic creatures such as plants that, through the utilization of solar energy, create organic compounds. Considering a food chain, heterotrophs gather energy by consuming autotrophs, they cannot produce their own energy. Heterotrophs are in turn called primary energy consumers, secondary energy consumer, tertiary energy consumers etc. depending on their hierarchical level in the food web that is determined by the trophic structure. Rabbits are autotrophs that consume plants and turn the energy from plants into their own energy. Foxes being secondary energy consumers in turn gather this energy from rabbits.

The different cell types are defined as a property of each cell. In the spatial model, each cell contains its own genome, its type, its movement efficiency and subsequent fitness values. Type determining characteristics of movement and biomass. A type 0 cell is for example considered to be equivalent to a plant cell. However, most of the experiments done will focus on primary energy producers and primary energy consumers. The plant cells steadily acquire energy or biomass at a fixed rate (simulating light absorption) and each cell that is not occupied by either a rabbit or a fox will automatically increase the biomass of that specific cell. This biomass can in turn be consumed by the rabbit cells that transform the plants biomass into usable rabbit biomass. If the rabbit has accumulated enough biomass it has a chance to reproduce depending on a reproduction rate. A simple pseudocode of this spatial model can be seen in algorithm 1

Though the spatial model is similar to others, a few more factors are introduced in the model for studying complex evolutionary dynamics. First, to make the cells in the spatial model evolvable, each cell contains a genome and a corresponding fitness value. The fitness value represents an efficiency measure with which the rabbit cells are able to consume plant cells. The genome of the rabbit cells consists of a bit string either of size 8, 16 or 32. From this genome, the fitness value is computed by evaluating the genome with the H-IFF function at the start of a new individual rabbit cell. The model is now evolvable, and we can study how mutation rate, population size and mass acquisition influence the evolvability of the rabbit population.

\(^1\)Original code used as a template can be found at: https://github.com/Muzkaw/Cellular-War/
\(^2\)A minimal source code of the adjusted version used in this thesis can be found here: https://github.com/FrankVeenstra/ALife2018
Algorithm 1: Spatial Model Simple Pseudocode

initialize population $P = \{X_1, \ldots, X_N\}$;

\[
\textbf{while } g := 1 \text{ to } G_{\text{max}} \textbf{ do } \text{ New Cycle } \\
\textbf{for } n := 1 \text{ to } N_{\text{max}} \textbf{ do } \text{ Update Cell } \\
\quad \text{if } \text{Cells}[n] \neq X \text{ then } \\
\quad \quad \text{Mass} := \text{Biomass Production Rate}; \\
\quad \text{else } \\
\quad \quad \text{Mass} := -\text{Mass Loss Rate}; \\
\quad \quad \text{Move; } \\
\quad \quad \text{Eat; } \\
\quad \quad \text{Reproduce; } \\
\text{end } \\
\text{end } \\
\text{end }
\]

A model for development and mortality are implemented in the spatial model and experimented with in Chapter 3. In the case of the mortality experiments, the cells can have a maximum age or terminal age, a probability of death and the condition of being either mortal or immortal. The terminal age has been changed in Chapter 3 to see how this affects the evolvability. For investigating development, the algorithm includes multiple genomes that are expressed in different stages of their lifetime. This implementation is also further described in Chapter 3. The full algorithm of the spatial model can be seen in Appendix A.

The spatial model consists of a matrix that has the same width and height specified by the user. Smaller worlds require less computational power but also make the simulation more prone to extinction events. Considering computational requirements, the spatial world has been limited to a grid of cells that is 250 by 250 cells in width and height. In order to equate relationships in evolutionary algorithms to biology, spatial models are used as abstractions of ecosystems. Since spatial models can contain inherent elements, such as local competition, this would better represent changes in population dynamics than standard evolutionary algorithms. With this spatial model system, implementing and evaluating existing biological concepts that are based on e.g. Lotka-Volterra equations are feasible. Text Box 2.3 describes the implementation of the spatial model to recreate predator prey equations of the Lotka-Volterra model as an example of how the spatial model can be used.
Text Box 2.3: Simulating Predator Prey Models

Population dynamics can contain counterintuitive elements that can be easily evaluated by the aforementioned spatial model without the knowledge of the mathematical formulation. Consider the Lotka-Volterra Predator Prey equations. These equations state that in the absence of a predator, the prey reaches a population size close to the carrying capacity. Adding predators to the model influences the steady equilibrium, or stable or unstable limit cycles, of the population model. In this spatial model one can discover equilibria to which the population size of predators and prey are converging to. By looking at the rabbit and fox population, the same dynamics as explained by the equations arise. The population size of predators on the other is greatly influenced by the reproduction resources available to the prey. Considering a spatial with an initial starting condition containing a low number of predators and prey, one can see that there are various equilibria that depend on the amount of resources available to the prey.

By varying the resource availability for the prey, one can see that when increased resources for prey does not influence the population size much Figure 2.9. A slight drop in population size can even be seen when more resources are available which seems counter-intuitive. When increasing the initial biomass acquisition of the primary energy producers, allowing for more food for the prey, the eventual population size of the prey is steady around 750 individuals while the amount of available resources was increased tenfold. The number of predators however do vary greatly as there are about 500 predators when the biomass production is low while there are around 1250 predators when the biomass production is high. Varying the energy acquisition for the primary energy producer yields various equilibria and limit cycles as represented in Figure 2.10. Some of these parameters leading to instability as can be seen in the case of an energy accumulation rate of 0.064.

The spatial model in this case uses three hierarchical steps in the food chain: plants, rabbits and foxes. Another surprising effect can be seen in the biomass of plants, with the absence and presence of predators Figure 2.11. In the absence of predators, the average plant and rabbit biomass reaches a quite low equilibrium. The average plant biomass is increased again with the introduction of the predators since the foxes protect the plants by eating the rabbits.

![Figure 2.9: Amount prey and predators when varying the primary energy production. Over time, the predators and prey reach a somewhat stable population size while varying the energy production of 0.002 (A), 0.004 (B) and 0.032 (C)](image)
Figure 2.10: **Stable limits in predator prey models.** The stable limit cycle gathered from experiments with the spatial models using different biomass production rates ranging from 0.002 to 0.064. Every arrow on a line represents an interval of 20 iterations.

Figure 2.11: **Biomass over time when simulating plants, rabbits and foxes (primary energy producers, prey and predators)**

The addition of another predator complexifies the spatial model even more and one can now see that the prey population in turn grows larger with the addition of the extra predator. From an ecological stand-point this means that one cannot directly determine the health of an environment by looking at the plants or rabbits, rather, one should look at the state of the apex predators. A large apex predator population can determine how well the initial layer of prey, primary energy producers, is able to generate acquire energy and produce biomass that can be distributed to lower layers of the strata. Hence, conservationists should never look at the population size of prey for screening the healthiness of an ecosystem, they should rather look at the health of the apex predators.
2.8 Concluding Remarks

This chapter gave an overview of the important factors influencing evolution in natural and artificial systems. These concepts are in turn taken into consideration and form the foundation on which the remainder of the thesis builds. These concepts should be taken into account when implementing them in evolutionary robotics since they can shape the evolvability of the simulated or real evolving entities significantly. The importance of these concepts can be summarized as follows:

- The sequence space and corresponding fitness landscapes are important to consider shape evolutionary trajectories
- Mutation and selection operators determine both the robustness and evolvability of a population
- A measure of evolvability is valuable to describe the efficiency of an evolutionary algorithm
- Development can change the breadth of the search space by abstracting the genotype to phenotype mapping and allows for changes in behavior during a life time of an individual
- Simple evolutionary algorithms using binary or continuous objectives can serve as test beds to elucidate the effectiveness of them
- Spatial models are required for equating observable evolutionary dynamics to biology

In Chapter 3, the evolvability of populations is evaluated on benchmark applications and the influence of mortality on the evolvability of a simulated population is of key interest. It concerns an in silico implementation not relying on any mathematical formulations. In Chapter 3, both evolutionary algorithms and spatial models are used as testbeds. Development, as discussed, will also form a small part of the next chapter displaying some preliminary results of a developmental approach to evolvability. Principles of development are implemented in the form of generative encodings in chapters 5, 6 and 7. Ultimately, the evolvability of a population and the genotype to phenotype mapping of the individuals largely determine the emergence of specific phenotypes in an evolving population and are thus important for the effective design and implementation of evolutionary computation to robotics.
Chapter 3

Evolution and Longevity

The secrets of evolution are death and time – the deaths of enormous numbers of lifeforms that were imperfectly adapted to the environment; and time for a long succession of small mutations.

– Carl Sagan, Cosmos

Death is a seemingly frivolous property of life that is counter intuitive to the evolutionary advantage old age has when considering classical Darwinism. The older an individual can become, the more offspring it potentially is able to produce. Hence, the personal fitness of an individual is directly increased by living longer. Nature is however riddled with mechanisms that seem to be silly restrictions to longevity. For instance, senescence, the deterioration of function with age, is prevalent in a multitude of species. Though the individual benefit is usually related to fitness, Darwin (1872) already noted that longevity is likely a product of the complex interactions between a species and its environment. Could longevity itself somehow be a determinant for the evolutionary trajectories we see in nature?

Considering difficult fitness landscapes that can be convoluted and deceptive, finding a good genotypic solution with evolutionary computation often requires the implementation of additional exploration operators. These types of operators are valuable for evolutionary algorithms since they promote the evolvability of a population by preventing the population to get stuck in local optimum. This chapter discusses the potential influence that mortality has on evolvability and why it could be a useful concept to implement in evolutionary computation. It furthermore demonstrates a potential evolutionary advantage of senescence in nature. In this chapter, some curiosities and theories of senescence (Section 3.1), the impact of mortality on genetic algorithms and spatial models (Section 3.2 & Section 3.3), considering the potential evolutionary advantage of mortality in general (Section 3.5). Moreover, some preliminary experiments were done to see how developmental mechanisms influence the evolvability in a similar manner. These preliminary results on the influence of development on evolvability and age in spatial are therefore also discussed in Section 3.4.

Approach

Although many examples of evolution can be seen as a gradual change over generational time, some might require evolutionary steps that would make individuals worse than their ancestors; descending the fitness landscape. They would be required to cross valleys in the fitness landscape. Crossing these valleys would enable progeny to find a solution in the search space that is more distant, and perhaps more efficient than the ancestor’s solutions. As a testbed for this potential leap, we used a deceptive fitness function—an adjusted version of the H-IFF function (Watson et al., 1998) that was described in Subsection 2.6.1. This deceptive fitness function is implemented on both a SSGA and a spatial grid model to simulate an evolvable population of individuals. The main hypothesis that is being addressed in this chapter therefore speculates on the nature of intrinsic mortality and its benefit to evolving populations (Hypothesis 1).

Hypothesis 1 Intrinsic mortality benefits the evolvability of a population.
3.1 Theory on the Evolution of Senescence

Most octopuses are semelparous, reproducing only once in their life time. Most types additionally only live around a year. An observation already mentioned in Aristotle’s History of Animals (Aristotle, 1910). They “live young and die fast” (O’Dor et al., 1986). The process of senescence in the *Enteroctopus dofleini* for example, is regulated by secretions from an endocrine gland that normally causes death by starvation after reproduction (Anderson et al., 2002). After reproduction, the octopus suddenly stops foraging and instead takes care of eggs and hatchlings, followed by death. However, by simply removing the endocrine gland, octopuses seemingly live significantly longer than usual, even being able to reproduce more than just once (Wodinsky, 1977). Octopuses in specific are quite elusive, and the true advantage of this type of senescence might be caused by various phenotypic traits and selection pressures. Does a short life have an evolutionary advantage? Or is the decreased life span a byproduct from the mechanism inhibiting foraging behavior, which enables the protection of offspring with the dire side effect of mortality?

Other animals such as specific salmon undergo a similar process of senescence: dying after having laid eggs. Some spiders are cannibals and kill their male counterpart after sex (a feature also exhibited by octopuses). Elephants run out of teeth, a form of *mechanical senescence*. While some turtles express *negligible senescence*. Naked mole rats grow significantly older than other rodents. Artificial selection of *drosophila* can allow them to live 50% longer after a few generations (Nusbaum et al., 1994) and the proteins DAF-2 and DAF-16 directly regulate life span in *C. elegans* (Lin et al., 2001). And as a final example, it has been shown that long-lived yeast mutants are outcompeted by short-lived wildtypes (Kryyakov et al., 2016). Longevity and aging thus seem to emerge differently across species. But what is the evolutionary value of senescence if there is any? Is aging somehow beneficial to a species? Or is it mere chance that individuals undergo senescence? To this day, this is a debated topic in as can be seen by the recent publications by Kowald et al. (2016), and Goldsmith (2016).

3.1.1 Summary of Theories on Senescence

Mortality is a fundamental component of natural systems that is either caused by intrinsic factors (senescence) or extrinsic factors such as predation, disease and accidents.

It initially seemed that aging is an evolutionary disadvantage for individuals since their personal fitness is lowered when an individual dies from internal mechanisms. There are however several theories explaining the cause and function of this biological phenomenon as an alternative to being a direct disadvantage. Darwin already mentioned in the sixth edition of the origin of species (not in older editions) that longevity is related to the scale of organization, expenditure and general activity of organisms which has probably been determined by natural selection Darwin (1872). Although having an evolutionary disadvantage for the individual, it may have several advantages for the maintenance of a species. Weismann. (1889) claimed that aging is determined by the “needs of the species” which is subject to the same mechanical process of regulation as to other structures and functions of organisms.

An alternative theory by Medawar (1952) proposes that aging could just be a phenomenon that arises due to the simple neglect of selection pressure on older organisms. Where older organisms are by chance more likely to have been prone to mortality inducing factors that limit their lifespan. Considering a steady probability of death for each individual in a specific population, a survivor curve can be created displaying how many individuals in a specific age group are alive (Figure 3.1). Consider a population of 100 individuals wherein every month there is a specific chance for individuals to die; a fixed probability of death, or mortality rate. This probability shapes the age range that evolutionary selection can act on. In the case of a 20% mortality chance every month, the range of selection is quite low, whereas this range is quadrupled when the chance is only 5% (Figure 3.1). Depending on this rate, the number of older individuals in a population differs. Medawar proposed that the random accumulation
could lead to this type of survivor curve. An accumulation of mutations leading to senescence could therefore be the result of a decline in selection pressure on older individuals in the case that individuals die due to mere chance, or wear and tear processes. Genes beneficial in early life would therefore have a higher selective advantage and thereby higher chances to propagate themselves into the next generation. Mere accumulation as an explanation of senescence is however difficult to hold for most species since single genes that cause aging have been conserved throughout different species over evolutionary time (Guarente et al., 2000).

Genes could also have evolved to be beneficial early in life while being potential deleterious later in life. This is a theory known as antagonistic pleiotropy (Williams, 1957). In this theory, a gene can have a pleiotropic effect by promoting reproductive success and survival early in life while being detrimental later in life. Through this process evolutionary biologists have argued that this inherent trade-off makes it difficult for natural selection to evolve old age in the first place. Although the scientific literature contains a ubiquity of examples of pleiotropic genes, it is more difficult to see how mutations in genes can improve the personal fitness in early life while having a deleterious effect later in life. These genes are sometimes defined as “putative” disease alleles (Carter et al., 2011) since acquiring evidence that these types of alleles really have a benefit early in life is still to be determined.

Another alternative theory to the accumulation of mutations and antagonistic pleiotropy theories is the disposable soma hypothesis proposed by Kirkwood (1977), which is one of the dominating theories nowadays (Shefferson et al., 2017). In the disposable soma theory, the body of an individual organism can allocate limited resources to various cellular processes and needs to make compromises between its metabolism, reproduction, repair and maintenance functions. E.g. a population only focusing on repair can be outcompeted by a population that instead spends more energy on growth. Combined with Medawar’s survival curve, which can also be caused by extrinsic factor such as predation, this would suggest that maintenance and repair are also of lesser importance later in life since the probability of an individual reaching old age by chance is already low. Not allocating any resources to the repair of an organism with increasing age would thus in turn lead to an organism’s deterioration with age as a side effect.

Other more recent theories consider the potential altruistic effect of senescence where aging can be beneficial for coping with a changing environment (Yang, 2013; Mitteldorf et al., 2014; Herrera et al., 2016). In this case it has been artificially shown

![Figure 3.1: Survivor curves. Considering a fixed number of individuals entering the population every iteration, and accepting that there is a 5%, 10% or 20% chance for an individual to die by chance, it can be seen that after a certain number of iterations there will simply not be any survivors left in a specific age category. Based on Medawar (1952).](image)
that a terminal age is beneficial for a population in rapidly changing environments that necessitates adaptive changes in the genome. Similarly, a resulting benefit from senescence, or intrinsic mortality, is the reduction of over-consumption of environmental resources which gives a selective incentive for intrinsic mortality (Werfel et al., 2017). In addition, Lehman et al. (2015) showed that extinction events can lead to a better evolvable evolutionary algorithm, though in this case the extinction events were discriminative and kept certain elites in the population. From an optimization perspective, it seems that senescence, or simply mortality, can be beneficial for a population in terms of evolvability and altruistic aging.

The evolvability theory of senescence claims that senescence increases the evolvability of a population (Goldsmith 2014; See Section 2.5 for the discussion on evolvability). As I see it, there are two ways in which mortality might aid evolvability:

1. When individuals die, a higher turnover rate of new individuals arises in the population
2. Mortality reduces selective pressure on the best individuals in the population and thereby decreases convergence; promoting diversification

For the first reason, a greater number of individuals that can live in a specific period leads to a larger proportion of acquired genetic adaptations by a population. Or: more individuals that have been ‘evaluated’ in the environment yield more individual fitness results from potentially differing phenotypes. Mortal populations thus contain a higher turnover rate of individuals in the population compared to immortal populations. For the second reason, if older fit individuals are prevented from outcompeting younger, slightly less fit, individuals, the population can get stuck in a local optimum, or in a state that is less evolvable. The second reason is of great interest since this not only has biological relevance, but can also change the efficiency of evolutionary computation.

The actual mechanisms of senescence would most likely be a combination of the theories of senescence that have been described. However, by considering the gene pool as a whole instead of thinking about the benefits of the individual, there is no reason for individually detrimental phenotypic traits to not pose an evolutionary advantage. Say mutations are the main factor driving senescence, mutations also drive evolution due to the introduction of new variations of genes in the population. A non-mutating population with an immortal life would reside in a zero-evolvability state (Goldsmith, 2008). The evolution of complex organisms can thus be a compromise between evolvability of the species and personal benefit to the individual (Goldsmith, 2008).

The antagonistic pleiotropy theory might surely be an explanation for senescence though if senescence turns out to be beneficial for a population, this antagonistic effect would actually be an altruistic effect; or altruistic antagonistic pleiotropy. If the personal fitness of an individual could moreover be prolonged by adjusting the self-repair energy expenditure as mentioned by the disposable soma theory, there are evolutionary pressures towards better self-repair mechanisms. However, the lack in self-repair might also simply lead to more mutations, making a population more evolvable. Or the lack of self-repair mechanisms could lead to senescence, which could be advantageous as well. If there is a selection pressure towards senescence, we can speak of programmed death.

3.1.2 Programmed vs Non-programmed

The theories on senescence can be further categorized in non-programmed and programmed theories. Considering recent publications by Kowald et al. (2016) and Goldsmith (2016), this is a debated topic. In Kowald et al. (2016), non-programmed senescence supports theories such as mutation accumulation, (Medawar, 1952); antagonistic pleiotropy, (Williams, 1957); and disposable-soma theory, (Kirkwood, 1977). In contrast, Goldsmith (2016) supports programmed aging theories on evolvability (Weismann, 1889; Goldsmith, 2016; Mitteldorf et al., 2014) and altruistic aging (Yang, 2013; Werfel et al., 2017; Herrera et al., 2016) and claims to be corroborated by biological
3.1. Theory on the Evolution of Senescence

**Figure 3.2:** Theories of Aging plotted as an advantage and a disadvantage over time. Where line 1 denotes the advantage of longevity of an individual if longevity where age would not lead to a decrease in personal or inclusive fitness. Line 2 (solid red line) would represent Medawar (1952) as the advantage of longevity would decrease with age but would not have a negative impact. Antagonistic pleiotropy and disposable-soma theories would be represented by line 3 (dotted). In this case an increased lifespan does have an advantage to the inclusive fitness of a population but decreases with age. Weismann and Goldsmith support concept 4 where there exists an optimum life-span. Individuals that grow older than this lifespan will have a negative impact on the population. The Figure is taken from Goldsmith (2014).

Examples. It seems that supporters of the non-programmed theories generally exclude programmed theories (Kowald et al., 2016; Shefferson et al., 2017) whereas the programmed senescence supporters do not specifically exclude non-programmed theories. The impact of the theories considering lifespan as an advantage or disadvantage are summarized in Figure 3.2. Where the increased age is only considered to be a disadvantage in line 4 (supporting programmed senescence), while not in lines 1 (classical no negative effect), 2 (mutation accumulation) and 3 (antagonistic pleiotropy and disposable soma).

In Kowald et al. (2016), Figure 3.2 has been critiqued due to line 3 not representing the antagonistic pleiotropy and disposable soma theory well. It is critiqued since the parameters of these theories lead to a specific average lifespan that is optimal, hence mortality is an emergent factor. However, if there was a gene that, at no cost, would improve the lifespan of the organism, it would have an advantage for both the antagonistic pleiotropy and disposable soma theories but not for the programmed aging theories. Age itself is not the disadvantage, but the disadvantage is the product of the pleotropic gene, or the trade-off between soma, respectively. Hence the distinction is still valid but should be taken lightly. The main distinction being the evolutionary advantage of the process of senescence – or the evolutionary disadvantage of long life – that line 4 displays.

3.1.3 Why Mortality Promotes Evolvability

To illustrate why mortality could potentially work in favor of evolvability, we should consider Nowak’s model on the quasispecies in the sequence space again. As explained by Nowak (2006), when considering a sequence space of a specific gene, there can be several optimal regions (Schuster et al., 1988). Recall from Section 2.2 that the average mutation rate $\mu$ shapes the eventual region the population occupies based on
Chapter 3. Evolution and Longevity

Figure 3.3: Fitness landscape with one peak and a hill based on the sequence space. Where u represents a mutation rate as in Figure 2.1. \(\delta\) represents the mortality rate in a given population. The two threshold values for u and \(\delta\) are believed to have a similar result on the stable region the population occupies on the fitness landscape.

The threshold mutation rate values \(u_1\) and \(u_2\) (Figure 3.3). However, if genes in a population of individuals already reside in the broader lower fitness state, how can they traverse the sequence space to end up in the narrow peak that is the better fit solution? Traversing this fitness landscape would either require an individual to drastically mutate into that region, or a population could gradually move to the region through genetic drift. Nowak’s mutation rate threshold values are moreover only valid for a population of mortals. If immortality could occur, the immortal individual residing in the narrow peak will always stay there (since it cannot be outcompeted) and eventually, its offspring have a chance to also occupy the narrow region no matter how high the mutation rate is. Mortality in Nowak’s model is thus a requirement. Therefore, if we consider \(\delta\) to be the mortality rate, I claim that there exists a mortality rate threshold \(\delta_1\) and \(\delta_2\) similar to the mutation rate thresholds (Figure 3.3). This was the initial hypothesis that formed the premise of why mortality promotes evolvability.

A common issue with genetic algorithms is that the parameters for their optimal performance highly depend on the domain. Generational genetic algorithms inherently implement a mortality mechanism since the entire population is replaced by a new population of offspring every generation when no elitism is implemented. Moreover, deletion in steady state algorithms has also been investigated for e.g. dynamic environments and shown to perform similar to generational genetic algorithms (Vavak et al., 1996b). The application of a mortality rate in genetic algorithms can therefore elucidate if one should implement it in existing genetic algorithms to better traverse the fitness landscape. If mortality influences evolvability it is thus not only of value to evolutionary biologists, but also useful for optimization methods.

Accepting this premise that the mortality rate and mutation rate both affect the evolvability, the next section will try to experimentally verify it. Using H-IFF (Subsection 2.6.1) as the difficult to solve deceptive fitness landscape on both an SSGA and a spatial model, can help us understand how this relationship influences the evolvability of a population. The SSGA is used as an abstract model to see the general effects of mortality on the evolutionary progression on this deceptive fitness landscape (Subsection 3.2.1). Whereas, as described by Werfel et al. (2017), spatial models can elucidate aspects of mortality that equate to natural systems which will be discussed in Subsection 3.2.2. The spatial model – that contains an inherent extrinsic mortality rate emerging from local competition – is used to isolate the influence of intrinsic mortality.
to see whether it affects the evolvability in natural systems.

3.2 Mortality and Mutation Rate shape Evolvability

Though evolvability has been suggested to be a determining factor shaping longevity in nature, there haven’t been any artificial experiments directly looking at longevity and evolvability in this respect apart from extinction events (Lehman, 2015; Lehman et al., 2015). Looking at the evolution of a species as an optimization problem, we can define fitness landscapes in simulation environments where simulated individuals composed of binary genomes can be evolved to cohort to the maximum fitness. The experiments that will back up the simulation results are divided in a benchmark optimization implementation and a spatial agent-based grid model. In both simulations, the fitness results from the H-IFF function (Subsection 2.6.1). The selection/deletion operators in the spatial model are inherent properties emerging from the interactions of the individuals and their environment while they are defined in the SSGA. Including an extrinsic mortality mechanic in the spatial model will demonstrate whether the mutation rate can alter the stable region in the sequence space of the genomes as explained by Nowak (2006) (Figure 3.3) as well as see if an additional intrinsic mortality rate influences the evolvability of the population. The aim of the experiments in this section is to show how mortality alters the evolutionary progression of a population and to check whether this enables the population to traverse the fitness landscape more efficiently. The code is kept minimal, no crossover operators and only an asexual reproduction operator is used.

3.2.1 Mortality in the Steady State Genetic Algorithm

Methodology

The SSGA with a mortality rate implements a population with individuals containing binary genomes of length 64. The initial genomes were composed of a bit string of ones and zeros that were randomly initialized. The bits in the genome are mutated with a probability given by the mutation rate. Note that mutating a gene will randomly assign a bit of 1 or 0, so the gene swaps a bit with a half the contingency in mutation events. A mutation rate of 0.1 means that a gene is mutated with 10% probability, and thereby changes with only a 5% probability. This has been implemented to make sure a mutation rate of 1.0 would not produce offspring with the complementary bit string of their parent’s genome, but rather an entirely random set of bits.

After initializing the population, each iteration of the SSGA is as follows:

1. Choose a random individual
2. Copy the genome and subsequently mutate and evaluate it
3. Compare the new genome to a random individual in the existing population and replaces it when its fitness is higher

For a population size \( n \), a generation consists of \( n \) iterations of this process. After each generation, individuals are independently checked for deletion with a probability given by the mortality rate. Deleted individuals were marked with a fitness value of -1, but kept in the population to maintain the population size, though they were unable to reproduce. The population was logged after each generation. No crossover was implemented to isolate the effect of solely the mutation rate.

To see how mortality influences this SSGA, 20 evolutionary simulations ran for 100,000 generations and a population size of 50 individuals with different values for the mortality rate and mutation rate. A mutation rate sweep from 0.0 to 1.0 was done changing the mutation rate approximately exponentially. A similar sweep was done for the mortality rate, although the 0.64 and 1.0 mortality rates have been excluded.

---

\(^1\)The source code for both the SSGA and spatial model implementing mortality can be found at: https://github.com/FrankVeenstra/ALife2018
TABLE 3.1: Times the optimal solution is found in the SSGA. Varying the mutation rate ($u$) and terminal age ($\delta$). Results are taken from 20 runs of each set of parameters on 64-bit H-IFF. The subscript values represent the average number of generations (thousands) that had to be simulated before finding the global optimum. Mutation rates above 0.32 and below 0.01 have been omitted since the global optima is never found in these scenarios.

<table>
<thead>
<tr>
<th>$u \times \delta$</th>
<th>0.0</th>
<th>0.005</th>
<th>0.01</th>
<th>0.02</th>
<th>0.03</th>
<th>0.04</th>
<th>0.06</th>
<th>0.08</th>
<th>0.12</th>
<th>0.16</th>
<th>0.24</th>
<th>0.32</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.00</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0.005</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0.01</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0.02</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>0.03</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>0.04</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>20</td>
</tr>
<tr>
<td>0.06</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>14</td>
</tr>
<tr>
<td>0.08</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>0.12</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>18</td>
<td>19</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0.16</td>
<td>2</td>
<td>11</td>
<td>17</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0.24</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0.32</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0.64</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1.0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

since these values led to early extinction of the population and did not convey any important results.

Results

The number of times the global maximum has been found in each of the 20 evolutionary runs is presented in Table 3.1. In addition, the subscript values in the table represent the average number of generations ($\cdot 10^3$) it took for the runs to find the global maximum on average. As can be seen in the table, the relationship between the mortality rate and the mutation rate in the SSGA is very specific for finding the global maximum on 64-bit H-IFF within the given simulation time. Moreover, the mutation rate and mortality rate explain 89% of the variation seen in the ability to traverse to the global optimum in H-IFF (Figure 3.4). The results of twenty runs using the SSGA is shown in Figure 3.5 (top). To see how single runs are able to traverse the fitness landscape, Figure 3.5 (middle and bottom) depicts the fitness and diversity on the H-IFF landscape over generational time. Shown as blue dots on the graphs are the individuals of a population at specific intervals.

Mortality rates or mutation rates that are too high lead to too much variation and as a consequence to less fit individuals. In contrast, if the mortality rate or mutation rate is too low, the population quickly stagnates in a local optimum. The proper ratio of mutation rate and mortality rate leads to a population residing in an unstable local optimum, but still fit enough to traverse the top of the fitness landscape and explore multiple peaks. Using the optimal mutation rate to mortality rate ratio, the ability of a population to produce adaptive diversity over generational time can be seen as creating diversity while still hugging the top of the landscape. I have called this phenomenon informally as hill-hugging since the genetic algorithm crosses valleys but does not steep low in the search space compared to higher mutation/mortality rates.

Moreover, the experiments were done using 64-bit genomes on H-IFF. However, though the optimal ratio between the mutation rate and the mortality rate changes when varying the size of the genome, this optimal ratio still exists. On 128-bit H-IFF, which contains an immense number of possible configurations almost 20 orders of magnitude larger than 64-bit H-IFF, it is still able to find the global maximum within 100,000 generations when a specific mutation rate to mortality rate ratio is used (Figure 3.6). On 128-bit H-IFF, the maximum achievable fitness value is 448 instead of
3.2  Mortality and Mutation Rate shape Evolvability

![Graph showing relationship between mutation rate and mortality rate.](image)

**Figure 3.4:** Relationship between the mutation rate and mortality rate. Mutation rate is shown in logarithmic scale. Symbols represent number of optimal solutions found for 64-bit H-IFF. Darker colors represent more solutions for those parameters (up to 100% success). Exponential fit for the data: $y = 0.1538 \times e^{-7.28x}$, with $R^2 = 0.89$

![Graph showing evolutionary progress for different mortality rates.](image)

**Figure 3.5:** Evolutionary progress for different mortality rates. (top) The average fitness and percentiles (25-75 dark grey; 0-100 light grey) of 20 runs using a mutation rate of 0.08 and a mortality rate of 0.04 (A), 0.08 (B) and 0.16 (C). Distribution of the population across the H-IFF landscape of a single run in comparing the distribution and fitness of individuals across the landscape (middle) and plotting the distribution and fitness over generational time (bottom).

Out of 20 evolutionary runs, the global maximum on H-IFF, though highly unstable, was found 3 times in different runs when using a mutation rate of 0.03 and a mortality rate of 0.12; it was also found 3 times using a mutation rate of 0.06 and a mortality rate of 0.02; and, it was found only once when using a mutation rate of 0.02 and a mortality rate 0.16. The other combinations of mutation and mortality rates that didn’t find the global maxima were similar values as used for the sweep in Table 3.1.
Chapter 3. Evolution and Longevity

Figure 3.6: Relationship between mutation rate and mortality rate on 128-bit H-IFF. Showing the evolutionary progression of 20 runs with a mutation rate and mortality rate of 0.02 and 0.16 (A), 0.03 and 0.12 (B), and 0.06 and 0.02 (C) respectively. A relationship with mutation rate or mortality rate that was too high leads to the evolutionary progression depicted in D. While a too low mutation and mortality rate led to a progression depicted in E.

Figure 3.7: Relationship between mutation rate and mortality rate on 32-bit H-IFF. Showing the evolutionary progression of 20 runs with a mutation rate and mortality rate of 0.1 and 0.128 (A), and 0.1 and 0.192 (B). Using a mutation rate of 0.1 and removing 90% of the individuals each generations as extinction events results in a similar performance (C).

When performing 20 runs on only 32 H-IFF bit, the optimal mutation rate to mortality rate ratio is much broader as can be seen when implementing a mutation rate of 0.1 and varying the mortality rate to be 0.128 and 0.196 (Figure 3.7). Interestingly, implementing mortality in the form of an optimal extinction events ratio, similar to the implementation of Lehman et al. (2015), results in a similar performing evolutionary run. In the case of the optimal extinction events ratio, 90% of the individuals were removed every 10 generations as displayed in Figure 3.7c.

In addition, the relationship between the evolvability and the mutation and mortality rate has been tested on the Chuang f1 (Chuang et al., 2010) function as it has been implemented in Fortin et al. (2012). When changing the fitness function to the Chuang f1 function for 64-bit genomes, the optimal mutation ratio is comparable to 64-bit H-IFF as depicted in Figure 3.8. In this case, a similar mutation rate and mortality rate was required to find the global maximum in the evolutionary runs. Decreasing or increasing these rates had a similar effect as they had on the H-IFF function.
3.2. Mortality and Mutation Rate shape Evolvability

Figure 3.8: Optimal mutation and mortality rate on 32-bit Chuang f1. Showing the evolutionary progression of 20 runs (A) and the diversity of all the individual in the 20 runs (B). The mortality rate was set to 0.04 with a mutation rate of 0.1.

When not implementing any type of mortality rate, the SSGA does not produce genetic variety and converges quickly. As can be seen in Figure 3.5a, the lower mortality rate clearly lessens the diversity of solutions found during the evolutionary progression. When only doing a mutation rate sweep on a SSGA, one might falsely conclude that the problem is similar to a needle in a haystack scenario. In the case of a needle in a haystack scenario, random search would be the best possible approach and one might therefore falsely set the mutation rate of the evolutionary algorithm to 100%. However, as demonstrated by the mortality rate, this is not a fair conclusion and hence the use of steady state approaches in general may be of limited value.

Though the relationship between a mortality rate and a mutation rate is interesting as it significantly changes the performance of an SSGA, it is of limited interest to biologists since it doesn’t represent a plausible spatial ecosystem at all. Therefore, the implementation of mortality and H-IFF on a spatial model is the main experiment conducted in the next section.

3.2.2 Spatial Model

Methodology

The spatial model is an agent-based grid model as discussed in Section 2.7. Like the SSGA, the spatial model implements the H-IFF fitness function producing the deceptive landscape. However, the genome size was limited to 32 bits. The genomes were only composed of 32 bits to reduce computational requirements which were considerably higher in comparison to the SSGA. Another difference to the steady state implementation was that the population was initialized with genomes in the middle local optimum of the H-IFF fitness landscape (i.e. 0000-0000-0000-0000-1111-1111-1111-1111, with corresponding fitness values of 64). From this starting genome, it is particularly challenging to find the global optima since no individuals of the population are close to any of the global optima, which could be the case when randomly initializing the genomes. Moreover, when randomizing genomes, the fitness of random individuals can by chance be so low that the population is never able to survive without manually increasing their fitness. Hence, the middle local optimum was chosen as the initial genome of all individuals in the population.

The spatial model is similar to a predator prey model and various features were included serve as an analogy to natural systems. The experiments were performed on a 250 x 250 grid (Same as in Kowald et al., 2016) where cells are either type 0 (prey) or 1 (predator). One can imagine the prey and predators to be plants and rabbits respectively. Where predators where subject to evolution and each rabbit cell contained a binary genome. The fitness value derived from a rabbit’s genome translates into food consumption efficiency, or a metabolic efficiency. The ability to acquire food from
Chapter 3. Evolution and Longevity

Figure 3.9: Illustration of the spatial model. Green represents plant biomass, blue rabbit biomass. Snapshot taken after the first few cycles of the spatial model.

the environment efficiently enables rabbits to grow faster, thereby producing more offspring. The spatial model is visualized in Figure 3.9 where green cells denote plants cells (the intensity of the color displaying how much biomass has been accumulated) and blue cells represent rabbit cells.

In the experiments, some parameters were set that shaped the environmental dynamics of the system. These values are however not strict and can be altered to produce similar results. The specific values described here are set for computational efficiency. E.g. the environment is not too large or too small, not to few or to many rabbit cells roam around under certain conditions, and food is not too scarce or to plentiful. Changing these values ± half of the value do not change the presented results significantly.

At each iteration, all plant and rabbit cells are updated. A given amount of biomass is added to plant cells according to a biomass production rate which was an absolute value of 0.0016. A plant cell from here on could grow to contain a maximum biomass value of 1.0. Rabbit cells attempt to move to a neighboring cell with 8/9 chance at each iteration. If the target position is occupied by another rabbit, the rabbit will stay in position. For computational efficiency, the grid was sequentially updated from left to right and top to bottom, and it was ensured that rabbits move only once per iteration. Furthermore, rabbit cells reproduce with a 1/10 chance if their biomass was above twice the reproduction cost (reproduction cost was 0.4). The reproduction cost was subtracted from the biomass of the parent rabbit. Offspring start with a biomass equal to the reproduction cost multiplied by 0.8 (0.32). This multiplication stood for an additional reproduction cost where a value of 0.08 biomass was lost during a reproduction event.

When a rabbit cell moves on a plant cell it consumed the plant’s biomass with an efficiency rate of \( \frac{fitness}{maxfit} \). The rabbits could not increase their biomass over the 1.0 limit; any unused plant biomass is left in the plant cell and thus stayed available for consumption. At every iteration, rabbit cells lose 0.02 biomass as a maintenance cost. Rabbits with a biomass below 0.01 were removed from the population expressing extrinsic mortality through starvation (or local competition for food).

In contrast to the SSGA, the spatial model implements both intrinsic and extrinsic mortality. For intrinsic mortality, a terminal age has been implemented and extrinsic mortality was thus the result from local competition. In order to explore the relationship between intrinsic mortality and mutation rate, different mutation rates and terminal ages were compared. The main results indicate how often, and how quickly, the global maximum was found on 32-bit H-IFF.
3.2 Mortality and Mutation Rate shape Evolvability

Table 3.2: Number of optimal solutions for 32-bit H-IFF on a spatial model. Results are taken from 20 runs for each combination of mutation rate ($u$) and terminal age (TA). $\epsilon$ marks combinations where the population went extinct in all runs. Data from terminal age 1000 and 2000 not shown.

<table>
<thead>
<tr>
<th>$u \setminus$ TA</th>
<th>40</th>
<th>50</th>
<th>60</th>
<th>80</th>
<th>120</th>
<th>160</th>
<th>500</th>
<th>1000</th>
<th>2000</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.01</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0.015</td>
<td>19</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0.02</td>
<td>$\epsilon$</td>
<td>20</td>
<td>15</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0.03</td>
<td>$\epsilon$</td>
<td>10</td>
<td>20</td>
<td>20</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0.04</td>
<td>$\epsilon$</td>
<td>1</td>
<td>11</td>
<td>20</td>
<td>12</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0.06</td>
<td>$\epsilon$</td>
<td>2</td>
<td>6</td>
<td>20</td>
<td>20</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0.08</td>
<td>$\epsilon$</td>
<td>1</td>
<td>1</td>
<td>20</td>
<td>20</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0.12</td>
<td>$\epsilon$</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>9</td>
<td>20</td>
<td>3</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>0.16</td>
<td>$\epsilon$</td>
<td>$\epsilon$</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>20</td>
<td>20</td>
<td>20</td>
<td>17</td>
</tr>
<tr>
<td>0.24</td>
<td>$\epsilon$</td>
<td>$\epsilon$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>13</td>
<td>15</td>
<td>14</td>
</tr>
<tr>
<td>0.32</td>
<td>$\epsilon$</td>
<td>$\epsilon$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>0.48</td>
<td>$\epsilon$</td>
<td>$\epsilon$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Results

Similar to the steady state implementation, the spatial model showed a tight correlation between the mutation rate and mortality rate as can be seen in Table 3.2 for finding the maximum fitness on 32-bit H-IFF. Apart from these optimal ratios, the speed of finding the global maximum could also be determined (Figure 3.10). Though not tested for significance, a terminal age of 120 and a mutation rate of 0.06, and a terminal age of 160 with a mutation rate of 0.08, were optimal for finding the maxima the quickest.

Additionally, the optimal mortality rate to mutation rate ratio resulted in a population that was less prone to lose the global optimum once it was found (Figure 3.11). This might be because a better fit population is able to sustain more individuals than a lower fit population (Figure 3.11). There is again a specific ratio of mutation rate to terminal age that is optimal for the population to traverse the fitness landscape (Figure 3.12). The population of the spatial model, in contrast to the SSGA, was initialized with individuals residing in the middle of the fitness landscape. The results illustrate that despite being in a local optimum furthest away from the global optima, the solution can still be quickly found under the right parameters. The parameters being tuned to exhibit the optimal mutation rate and mortality rate.

Looking at the individual runs when using a terminal age of 60 (Figure 3.11), we can see a similar phenomenon as to the SSGA where a lower mutation rate leads to premature convergence more quickly while a high mutation rate creates an unstable population. Moreover, the speed of finding the global maximum in the spatial model also differs when using different mutation rate and mortality rate variables (Figure 3.10). In this case, the speed was derived from the number of cycles the spatial model ran before finding the global maximum. A different measure of speed would be to count the number of individuals that have been simulated before the maximum has been found. E.g. the only real difference to Figure 3.10 when comparing a mutation rate of 0.02 and a terminal age of 50 with mutation rate 0.16 and terminal age 500, is that the latter needs to simulate significantly less individuals before the maximum is found (two-sided Mann Whitney-u test p value 0.008). The difference in speed of the number of cycles was not significant in this comparison (two sided Mann Whitney-u test p value 0.2). I.e. median number of individuals simulated before finding the global maximum was 971,792 in the low terminal age scenario and 188,148 in the high terminal age scenario. Thus, a higher terminal age in this comparison needed to simulate less individuals. Apart from this anomaly, the speed plot of using individuals as a measure looks almost identical to Figure 3.10. For evolutionary algorithms in general,
Chapter 3. Evolution and Longevity

**Figure 3.10:** Speed of solving H-IFF for the spatial model. Times the global maximum found divided by the average number of iterations · 1000 the spatial model ran varying the mutation rate (x) and the maximum age.

**Figure 3.11:** Individual runs on the spatial model. Individual runs (illustrated with different colors) displaying maximum fitness values across generational time when implementing a terminal age of 60 and a mutation rate of 0.02 (A), 0.03 (B), 0.04 (C). Each cycle represents 100 iterations of the spatial model.

the number of individuals simulated should be minimized though simulation time in terms of cycles is what matters in real populations.

Another interesting result when using no terminal age is that although the average age of the individuals in the population stays relatively the same across different mutation rates, the maximum age is significantly higher in high mutation rate scenarios (two sided Mann whitney-u p value of $7 \cdot 10^{-8}$). These results were interpreted as the best fit individual being unable to produce many functional offspring due to the high mutation rates. This meant that the older fitter individuals had in turn a higher chance to outcompete the other, likely less fit, individuals in the populations. The elite thus becomes a lot older in scenarios where there is a high mutation rate. I.e. when looking at no terminal age, the maximum age of the population under high mutation rates can grow as high as 20,000 cycles. As shown in Table 3.2, as high mutation rates also lead to more lesser fit individuals in the population, the mutation rate is necessarily low, otherwise the population will go extinct as denoted by $\epsilon$ in Table 3.2. Both a mortality rate and a mutation rate could thus lead to an error catastrophe, where a species goes extinct due to excessive mutations. In conclusion, the mortality rate and mutation rate shape the evolvability of the population simulated in the spatial model.
3.2 Mortality and Mutation Rate shape Evolvability

Figure 3.12: Optimal mutation rate as a function of terminal age. Note the logarithmic scale for terminal age. The continuous line shows an exponential fit: $0.1903 - 0.1907 \cdot e^{0.0028}$, with $r^2 = 0.9936$ (values closer to one indicate a better fit).

3.2.3 Comparison to Other Algorithms

Although in this section are somewhat preliminary, the mortality rate implementation on the SSGA with H-IFF has been compared to some other conventional implementations of genetic algorithms. In particular, it has been compared with Age-Fitness Pareto Optimization (Schmidt et al., 2011), generational implementations, extinction events and an SSGA containing a speciation operator as implemented in Stanley et al. (2002b). The results from these experiments do not display a miraculous efficiency of the mortality rate implementation, but they can provide evidence for why mortality works for increasing evolvability, bolstering Hypothesis 1. This section therefore displays that evolutionary algorithms may not be effective and tuning existing algorithms might lead to better results.

Age Fitness Pareto Optimization

Since Age Fitness Pareto Optimization (AFPO) (Figure 3.13) has been used to create more diversity in evolutionary algorithms, the main initial aim of the mortality rate implementation was to beat this algorithm on a highly deceptive fitness landscape. This was initially done to see if implementing mortality was indeed useful. AFPO is a single population based evolutionary algorithm that utilizes the deletion and selection of individuals based on pareto dominance. Where pareto dominance is based on two factors, a fitness value and the age of an individual. A lower age being advantageous. The different ages and fitness values in the population creates a pareto front – region yielding optimal solutions for both objectives as displayed in Figure 3.13.

AFPO as implemented for the experiments\textsuperscript{2} starts by initializing a population of random individuals. Each individual has been assigned an age of 0 and these individuals and their asexually produced offspring increment their age value by one after each generation. After evaluating all the individuals in this initial population, all fittest individuals in all possible age categories are kept while the rest of the individuals are discarded. Afterwards, all ages of all individuals are incremented, and a random individual is inserted in the population. This random individual has an age of 0, similar to the initial population. The new population, including the random individual, is used to generate an offspring population that is the same size as the maximum population size. Afterwards, tournament selection (with tournament size 2) determines which

\textsuperscript{2}Researchers at the University of Vermont would call it ’AFPO-101’ since no additional parameters were tuned
individuals from the offspring population and the existing population will form the population in the next generation. This tournament is however also steady state since offspring are set up to compete with an individual in the original parent population sequentially. An individual can only outcompete and replace another individual if its fitness is higher and its age is lower thus dominating on both fronts.

To compare how efficient AFPO is on H-IFF, an initial experiment was done evaluating it on 32-bit H-IFF. Compared to the normal steady state implementation with and without mortality, it seemed to perform similar to the mortality SSGA. Increasing the size of the genomes to 64 bits quickly changed the outcome of the performance of AFPO compared to the SSGA implementing mortality. A mutation rate sweep was done to figure out the optimal mutation rate for AFPO. This optimal mutation rate was around 0.1 and was subsequently used in the experiments. The optimal mortality rate was tuned using the 0.1 optimal mutation rate of AFPO and was set to 0.05. The population size was again, kept at 50 individuals similar to the previous SSGA implementations.

As can be seen in a single evolutionary run of 100,000 generations Figure 3.14, the mortality rate implementation seems to exhibit more diversity while staying at the top of the landscape. In contrast, many individuals in AFPO reside in less fit (lower) areas of the fitness landscape space likely due to too frequent insertions of random individuals at each generation. The evolutionary progression of the SSGA implementing mortality should convey the informal principle of hill-hugging that we introduced in the previous sections since, especially compared to AFPO, the steady state implementation does not seem to occupy low fit regions of the fitness landscape. 200 of these evolutionary runs were done to compare the efficiency of both algorithms, the measure of efficiency was correlated to the number of times a global maximum has been found after running 100,000 generations Figure 3.15. Out of the 200 runs, only one run of AFPO found the global maximum whereas this number was 190 out of 200 runs in the SSGA with a mortality rate. AFPO performs better than steady state algorithms when no mortality is implemented, but for this landscape, the mortality rate implementation was superior.

**Other Conventional Implementations**

There are a plethora of different algorithms that have been developed over the years that could be compared with the mortality rate. Simply looking at the difference between five strategies, a SSGA, a generational genetic algorithm implementing 5% elitism, AFPO, speciation and extinction events clearly shows how different strategies
3.2 Mortality and Mutation Rate shape Evolvability

Figure 3.14: Single evolutionary runs comparing AFPO and H-IFF. The graphs display the maximum fitness (top), distribution of the population in 2d (middle) and the distribution of the population in 3d (bottom) of a single run.

Figure 3.15: 200 evolutionary runs are shown for both AFPO and Mortality Rate. The solid black line represents the average value of the maximum fitness of each evolutionary run. Though it can be seen that the average maximum fitness in AFPO is ever increasing, the average maximum fitness using the mortality rate stays around the same value. The graph’s color is reddish due to the order of the curves of the evolutionary runs being plotted in the graph.
3.3 Changing the Domain

Single objective continuous functions can be used as an alternative to the binary functions of H-IFF. The efficiency of evolutionary algorithms can then be evaluated to how well an implementation is able to evolve the correct parameter sets. There are many different types of objective functions that are used as benchmarks for evolutionary algorithms but of specific interest to this section are deceptive fitness landscapes where there is no apparent gradient towards the global optima. The convoluted landscapes with gradients (such as Rastrigin or Griewank functions) are comparatively easy to solve, even with regular SSGAs and thus not discussed. Since a continuous function is used, more advanced strategies like Covariance Matrix Adaptation Evolutionary Strategy (CMA-ES; Hansen et al. 2001) can be implemented and compared with our implementations and can convey how the different algorithms work.

The alterable parameters of the continuous functions implemented in evolutionary algorithms are the mutation rate and the mutation spread. This spread is usually controlled by a Gaussian mutation operator as in the case of the following experiments. The standard deviation of the Gaussian mutation operator is usually denoted by a $\sigma$ value to indicate the spread. The change of the value of a gene becomes larger when
3.3. Changing the Domain

Figure 3.17: Evolutionary progression on 64-bit H-IFF of generational Genetic algorithms vs the SSGA implementing mortality. The graphs show the evolutionary progression of 20 evolutionary runs for 100,000 generations on 64-bit H-IFF (top) and the genetic diversity of all the individuals in all runs (bottom). Results shown for the SSGA implementing mortality (A), the generational genetic algorithm implementing tournament selection (B) and the generational genetic algorithm using roulette sampling (C). The optimal mutation and mortality rates were used for each implementation.

σ is increased. In the SSGA with mortality, the σ value is thus additionally correlated with both the mutation rate and the mortality rate.

The Schwefel function (Figure 2.8b) was used to create the deceptive fitness landscape whose genomic values could range between -500 and 500. A σ value of 200 was therefore chosen as the mutation spread where a gene having a value of -500 would thus not be likely mutated into a value of 500. Therefore, the landscape cannot be directly crossed after one mutation step. The σ value of 200 means that a value that is being mutated has a 68.2% chance to be altered through the addition of a value between -200 and 200 (one standard deviation) and a 95.4% chance to be between -400 and 400 (two standard deviations). In CMA-ES, the mutations are directed and changed based on the spread of the population over generational time (Hansen et al., 2001). Hence, in the comparison to CMA-ES, no specific mutation rate or sigma value is implemented.

For the experiment, a genome of size 8, containing 8 floating point numbers, was used. A mutation rate sweep was done for AFPO to see what the best mutation rate to implement was. Approximately a mutation rate of 20% yielded the best results. Using this mutation rate, the optimal mortality rate was determined for the SSGA. The optimal mortality rate was coincidentally also around 20%. These rates are however determined for quite a short run and could be subjected to further tuning. The rates do however show nicely how the genomes in the population change over time. The optimal mutation rates for AFPO and the optimal mortality rate for the SSGA were based on 4 evolutionary runs using different rates and simulating a population size of 50 for 2000 generations. Afterwards, 12 evolutionary runs were done with these optimal mutation rates, again for 2000 generations. CMA-ES implemented a population size of 1000 for 100 generations. This is done because larger populations are more efficient in CMA-ES since smaller populations converge incredibly quickly to a local optimum. However, the number of evaluations needed (100,000) stayed the same in all approaches.

Figure 3.18 depicts the difference in the evolutionary progressions of the different runs (top) and shows the change in the values of individual genes over time for a single
run that was representative (bottom). As can be noted, within the 2000 generations, the CMA-ES strategy is the only one to find the global optimum in 2 out of the 12 runs. AFPO and the SSGA do not find the global optima. Longer simulations and larger populations sizes would most likely led to the global optima to be found eventually. What is interesting to see in these plots is the genetic change of each implementation over time. The SSGA implementing mortality can be seen to vary the values of its genes over time, were a gene sometimes changes its value from one side of the landscape (close to -300) to the other side (close to 400). The SSGA implementing mortality thus keeps genetic variation. AFPO keeps a lot more genetic variation on average in the population due to the insertion of random individuals. The graph therefore shows that average values of each of the genes in the population drastically changes with time. CMA-ES starts off with a lot of genetic variation due to the large population size and converges to specific values quickly after which the genes do not change anymore. Where two runs of CMA-ES find the global maximum after approximately 60 generations. These effects on diversity are typical for each optimization type shaping the evolutionary progression of each type of genetic algorithm.

### 3.4 Development And Evolvability

Though mortality and mutation rate change the overall evolvability of an evolutionary algorithm, these are most likely not the only mechanism that increase the evolvability. As mentioned in Section 2.3, organisms contain inherent developmental programs that enable the patterned expression and recursion of cells, tissues and organs. This relates to how the genotype is mapped onto the phenotype. This section therefore focuses on the impact of development on the evolvability of the spatial model. In addition, an interesting emerging dynamic on the age of the individuals in the system was noted and included in the section. However, it should be taken into account that these results are also preliminary.

Hinton et al. (1987) demonstrated how developmental plasticity could smooth the search space evolution operates in; but they considered only an abstract control system and a one-to-one genotype to phenotype mapping. In needle in a haystack problems, random search has been proven to be the best search strategy (Hinton et al., 1987). Random search could be seen as having a 100% mutation rate. It is therefore not viable in nature since it would lead to an error catastrophe or to spending valuable resources on sub-optimal individuals. A more gradual stepping stone mechanism...
which includes intermediate sub-optimal fitness values, would allow for individuals to more effectively traverse the search space by keeping the population close to the top of the search space as has been shown by implementing mortality. As an alternative to increase this traversability, or evolvability, of a population, an individual could exhibit a small genotypic sweep by evaluating multiple genes expressed at different time intervals. This is what I consider to be a minimal developmental representation in this section. The question posed here is: *can this minimal development model lead to an increase in evolvability?*

**Methodology**

A fixed patterned change of the genome, through the developmental processes of an individual, could increase the search of various phenotypic expressions in an individual’s life time. To investigate how during a life time of an individual can explore multiple phenotypic traits, a spatial model is implemented again using H-IFF. The experimental setup consists of a predator prey like spatial model where a binary 16 bit genotype is translated into the phenotype of a cell as a metabolic efficiency as has been implemented in Subsection 3.2.2. Note that in previous experiments a 32-bit genotype was used. The only way for individuals to be removed in this experimental setup is through starvation.

Two experiments using the developmental stages were performed. The first experiment evaluated the efficiency of finding the global maximum similar to the spatial experiment with the mortality rate (Subsection 3.2.2). Instead of varying the mortality rate, the number of developmental phases were altered. Where a timed developmental phase changed the expressed genome of a cell to another. The number of developmental phases as well as the timing of these phases were fixed and the only mutable parameter in this instance was the mutation rate. The timing of moving from one developmental stage to another was set to 100 iterations. 100 iterations were chosen since a cell on average lived around 500 iterations. An age of 500 iterations is thus enough for an individual to reside in all up to five developmental phases during its life time.

To investigate how the population of cells in the spatial model is able to traverse the H-IFF landscape, 50 simulations ran for 200,000 iterations (2000 cycles) with differing parameters of mutation rate and number of developmental phases.

**Results**

Since the genomes with different developmental phases can evaluate multiple genomes in one lifetime, it was expected that the developmental stages to be directly correlated to the evolvability of the population. More genomes mean more exploration, however, this might also mean a shorter lifespan since genes that are deleterious later in life will limit the lifespan. An elite that goes from an efficient developmental stage to a suboptimal one, would be quickly outcompeted by better fit cells and thus the relationship between the mutation rate and the number of developmental stages will likely be skewed. In this setup, evaluating more than one set of genomes could be considered to be equal to evaluating multiple individuals in conventional evolutionary algorithms though the speed of acquisition of the global maximum in the spatial model might translate to evolutionary algorithms as well. The aim is to check how these developmental phases can aid, or if they are detrimental, to evolving spatial models.

Conventionally, in evolutionary robotics, the computational time for evaluating an individual mostly depends on the physics simulator since this requires the most computational power. The evolutionary algorithm itself usually doesn’t require much computational power and measuring the efficiency solely based on the number of individuals that have been simulated is therefore a good measure when considering the efficiency of the evolutionary algorithm. To evaluate if development is computationally more efficient, it is thus considered how many individuals have been simulated until the global maximum was found. As depicted in Figure 3.19, a clear relationship
between the mutation rate, the number of developmental stages and the times the global maximum has been found can be seen. Though this relationship exists, it is not known how useful this is for evolutionary computation.

In a second preliminary experiment, the timing of the developmental stages itself was subject to evolution. Where each developmental stage could span between 10 and 1000 iterations. In this case, the global optima can be found in different stages of development of the populations and this has an effect of the evolutionary progressions of the developing cells. Namely, if a population of cells finds the global optimum in the initial stage, it can be seen that the time the individuals spend in this stage is maximized (Figure 3.20; right). In contrast, there is no selection pressure to maximize this phase when the global maximum has not been found.

When comparing the different developmental times and mutation rates, a clear relationship between the mutation rate and the number of developmental phases can be seen in Figure 3.19. High mutation rates are efficient when the cells do not have developmental phases. However, with lower mutation rates, no developmental phase is very inefficient for finding the global optima compared to developing populations. When using mutation rates lower than 0.12, none of the non-developing populations found the global maximum while the developing populations were still able to do so. Other than that, a 100% mutation rate seems to work well in the spatial model. Producing many random offspring is therefore still efficient on 16 bit H-IFF. This would however not have a high probability of working using larger genomes as can be derived from Table 3.2 where mutation rates larger than 0.32 do not yield optimal solutions.

The developing populations of these experiments can be altered by a timing mechanism as developmental change; single bits change in timed expression in the genomes of the individual cells. This development increases the breadth of the search and in turn enhances the evolvability of a population. In Figure 3.20 (right), the period during which the maximum possible fitness is found alters the average age of the population. This limit in age can be seen as being the result of different bits changing that in turn cause a deleterious effect. This change in bits causing a detrimental effect could
be an antagonistic pleiotropic effect since it lowers the age of individuals; supporting the antagonistic pleiotropy theory. It could, however, also pose a dual benefit, where development both increases the breadth of the search and limits the lifespan of individuals further increasing the evolvability of a system. Though no quantifiable data is shown, it is an idea that might be worth investigating in the future. The negative effect of pleotropic genes on the personal fitness might thus actually increase the inclusive fitness.

### 3.5 Discussion

As shown, the evolvability is greatly influenced by the mutation and mortality rate ratio in both evolutionary algorithms and a spatial model. In particular, the H-IFF function, despite its deceptiveness, can be traversed by a SSGA through simply including an indiscriminate mortality rate. Since a fitness landscape in nature is likely highly convoluted and possibly deceptive, we speculate that programmed aging could be, as Goldsmith (2014) mentions, beneficial for the evolvability of a population. The better a species can traverse the fitness landscape without going through low fitness
regions, the more plausible it is that the population finds more adaptive traits. This in turn makes it better and potentially increases a population’s ability to cope with changing environments.

In the presented experiments, the shortest path to the global maximum of H-IFF from the center of H-IFF was to mutate individuals in the appropriate lesser fit local maxima. The results show that there is an optimal mortality rate for a given mutation rate, which could be considered to improve existing evolutionary algorithms, especially the steady state variants. SSGAs with no chance of removing elite individuals are prone to premature convergence shown by the few times the global maximum has been found when no mortality rate was implemented. But as shown, the addition of indiscriminate mortality enhances such algorithms and allow them to efficiently traverse the state space landscape. It is further shown that this effect is likely transferable to other domains and landscapes with preliminary results.

AFPO relies on a new random individual to be inserted in a specific region close the maximum fitness of the landscape by accident. From this starting point, the new pareto efficient individuals in this age category need to quickly find a good solution before they are outcompeted by a younger strain climbing the local hill. A potential improvement to AFPO is thus also to insert a new random individual at intervals, not every generation, so the individuals in the new age category have time to climb a local hill preventing them to be outcompeted by chance. Despite the occasional loss of the best individual in a population, the entire population of individuals remain close to the top of the fitness landscape.

The correlation between the mutation rate and mortality rate indisputably works on SSGAs and spatial models for solving deceptive fitness landscapes. Even the most advanced optimization strategies have difficulties with solving these types of landscapes. Other than that, there is something to be said for the ease of implementing a simple mortality rate to a genetic algorithm. Simply remove individuals from the population at random to increase the performance.

### 3.5.1 Theoretical implications

The mutation rate and mortality rate have an optimal ratio that depends on multiple factors. However, factors such as reproduction speed, development, population size, selection pressure, and crossover likely influence the optimal parameter set as well. Speculatively, a minimal relationship between these factors that always shape evolvability is the mortality rate, reproductive rate and mutation rate that shapes the overall evolvability of a system. This relationship is clarified in the Venn diagram shown in Figure 3.21. Where non-reproducing or non-mutating populations reside in zero evolvability states. In this case, individuals are evaluated too long and genetic variation is low. I speculate that through changing the parameters of for example, mutation and mortality rate, one can map the deceptiveness of the landscape as shown in the plots that depict diversity and fitness over generational time (Figure 3.5). Potentially, changing just one parameter without taking into consideration the other.

Species in natural environments suffer from both intrinsic (aging) and extrinsic mortality (predation, accidents, parasitism). Extrinsic mortality is known to fluctuate, both in predictable ways (seasons) and depending on external factors (diseases, variable predator pressure). As there is a clear correlation between mortality and mutation rate for optimal evolvability, this means that such fluctuations in mortality rates could have a negative or positive impact on the evolvability of populations. Evolving an intrinsic mortality factor may alleviate this problem: when external pressure is high, aging is not a dominant factor. If external mortality is decreased, then intrinsic mortality prevents the death rate to mutation rate equilibrium to be out of balance, preserving evolvability. Therefore, this would be an advantage of an intrinsic mortality rate.

Though mortality has been shown to be beneficial for the evolvability of a population, it is still difficult to see how mortality in itself would evolve. Mortality as seen in a prisoner’s dilemma scheme could define individuals that exhibit mortality to be
3.5. Discussion

Reproduction Rate/Efficiency
Mutation Rate and other Variability mechanism
Intrinsic and Extrinsic Mortality

Reproduction Rate/Efficiency

Mortality increases the turnover rate and determines the stability of the population

Zero-Evolvability State
no new individuals

Premature Convergence
unless the algorithms promote diversification

Zero-Evolvability State
no variation

FIGURE 3.21: Venn diagram depicting the relationship between reproductive rate, mortality and mutation rate. Reproduction, mortality and mutation rate shape evolvability. For a given sequence space, there exists an optimal relationship between these factors which is an overlap of the four elements. No mortality leads to premature convergence, no reproduction and no mutation leads to a zero-evolvability state since no new individuals or variation is being introduced in the population.

cooperators while immortals to be defectors. Since an immortal can produce more offspring than a mortal during its lifetime, its personal fitness would be higher. However, if all individuals in the population are immortal (defectors), the overall benefit to the population is less than when all individuals are mortals (cooperators). Hence, the evolution of mortality could have an attractor state for immortal individuals that would promote older individuals. If a population with mortals would be more efficient it would be more likely to outcompete a immortal population, eventually.

There is however another interesting detail that can explain the evolvability of mortality. Consider a population residing in stable limit cycle. The number of individuals in this population fluctuates depending on how much prey is available to them, and how much predators decrease their population. A periodic fluctuation would mean that the extrinsic mortality rate also fluctuates. In a period when many individuals of the population get eaten, the population can exhibit more genetic drift. Once less individuals of the population get eaten again, this genetic drift will also decrease when we accept that the mutation rate does not change. This change in evolvability of a population could however be counteracted by continuously changing the mutation rate. Lower mutations rates when many prey are eaten would keep the evolvability the same. Similarly lower predations rates would require higher mutation rates. This process of altering the mutation rate is likely a cumbersome to evolutionary adaption. However, as mortality has been shown to change the evolvability as well, this mortality rate could instead have been evolved to provide a steady evolvability rate in the presence and absence of high predation rates. This would keep the evolvability of a population the same and therefore the mutation rate would not need to be altered. A possible benefit for senescence could be to keep the evolvability of a population steady under conditions of fluctuating extrinsic mortality rates. As discussed by Herrera et al. (2016), intrinsic mortality seems to be beneficial in changing environments and therefore a steady optimal evolvability rate might be required. Intrinsic mortality could therefore be a mechanism that ensures a baseline evolvability rate in a dynamic, changing environment.

There is a major argument to be made why mortality is likely programmed by combining a changing environment as in Mitteldorf et al. (2014) and Herrera et al. (2016) and the experiments of H-IFF shown here. Consider for example a H-IFF type
function, and a changing environment where the global maximum switches from one side of the H-IFF landscape to the other side of the H-IFF landscape. How could one possibly hope to make a population traverse this landscape efficiently? Either the mutation can be increased to its maximum, or mortality could enable a drift across the top of the landscape. I claim that mortality is a more plausible explanation in natural systems since they gradually change the genetic information in the population instead of producing drastic changes in the genome. A high mutation rate would produce many individuals that are non-functional, these individuals might need to be brought into the world that can have a massive reproductive cost. Moreover, can such a potentially high mutation rate really be achieved locally in germ cells? Or would high mutation rate mechanisms mean that the individual cells are also subject to high mutation rates? This could bring about many detrimental effects in the individual and could lead to error error catastrophes in species.

3.5.2 A hypothetical explanation for the benefit of mortality

The issue of mortality also becomes more interesting when considering multiple niches. Say we have a hypothetical single population of finches on an isolated island with different beaks. The population hasn’t resulted in any speciation yet. Additionally let’s assume there are three types of food sources we simply call food source a, b and c. Let’s say that these food sources require appropriate beak shapes to optimize the consumption which we can call beak A, B and C respectively. Clearly, depending on the steady food supply, the beak frequency of beak types would roughly correspond to the availability of a food type. However, when food source a and b are low, beak C will begin to dominate. Let’s say that this domination is so vast that all beaks A and B have been out-competed after 100 generations. A phenomenon in artificial life that can be termed Catastrophic forgetting. The long-term effects of beak C dominating led to a decrease in food source c and a subsequent increase in food source a and b. What is now the quickest pathway to rediscover beaks A and B? Well, if the path to beak A and B is slightly deceptive, and the mutation rate is unalterable, a population of mortals will simply allow for better adaptive radiation of the population to rediscover either beak A or beak B than a population of immortals. This hypothetical dynamic shift of niches therefore promotes a population of mortals and is, to me, a conceivable logical explanation supporting the evolutionary advantage of intrinsic mortality.

3.5.3 Application to Evolutionary Robotics

Undefined domains with potential deceptive landscapes such as robotics simulators may also have an optimal ratio between mutation rate and mortality rate. It is likely that this optimal ratio changes in different parts of the landscape. An additional feature of the mortality to mutation rate ratio is that it will determine the broadness of the hills in the landscape that can be robustly occupied across generational time. This is derived from accepting the relationships of mortality and mutation rate to stable sequence spaces as presented in Figure 2.1. A higher mutation rate or mortality rate would then change the stable region the simulated population is occupying, and a broader stable region might result in more abstract general phenotypes that could be better transferable to robots. The field of evolutionary robotics is thus a promising field for future applications. It would be especially interesting to see if an optimal ratio of mortality and mutation rate can surpass existing algorithms like Age Fitness Pareto Optimization (Schmidt et al., 2011), Age Layered Population Structures (Hornby, 2006) and novelty search (Lehman, 2012) that have been frequently applied to such simulators.

3.6 Conclusion

An explicit relationship between the mutation rate and mortality rate for optimal evolvability on a deceptive fitness landscape in both spatial and non-spatial evolutionary
models has been presented. As an alternative to proposed theories on aging showing how intrinsic mortality is advantageous for altruistic aging, we claim that intrinsic mortality governs evolvability and that it is thereby a potentially evolvable trait, ultimately supporting the theories on programmed death. Moreover, in scenarios of fluctuating extrinsic mortality rates, an intrinsic mortality rate would keep the evolvability the same which might further support why intrinsic mortality has an evolutionary benefit. Also, considering potentially unrealistically high mutation rates to otherwise grant the same level of evolvability, these high mutation rates in nature would likely lead to an error catastrophe, leading to the extinction of the species altogether. Senescence might therefore be a better explanation for evolvability. The results not only increase our understanding of senescence but hold potential benefit in applications to evolutionary algorithms and robotics. As it has been shown that the mortality to mutation rate ratio influences evolvability on deceptive landscapes, one question that can now be posed to bolster the significance of this in chapter is:

Do natural systems contain deceptive dimensions that are traversable through mortality induced evolvability?
Part II

Evolving Modular Robots
Chapter 4

Evolutionary Robotics Platform

Everyone knows that a machine tool is more complicated than the elements which can be made with it, and that, generally speaking, an automaton A, which can make an automaton B, must contain a complete description of B, and also rules on how to behave while effecting the synthesis. So, one gets a very strong impression that complication, or productive potentiality in an organization, is degenerative, that an organization which synthesizes something is necessarily more complicated, of a higher order, than the organization it synthesizes. This conclusion, arrived at by considering artificial automaton, is clearly opposite to our early conclusion, arrived at by considering living organisms.

– John von Neumann, Theory of Self Reproducing Automata

In Part I the importance of evolvability and the genotype to phenotype map was discussed with respect to evolutionary dynamics. The part conferred how evolutionary computation can be implemented to evaluate theoretical hypothesis of biology and how its performance can be altered. In contrast to the theoretical undertone of the previous part, this part of the thesis illustrates scientific questions that arise from an engineering perspective. I.e. it will express the application of the types of evolutionary computation on the creation and control of modular robots.

Three of the challenges of evolutionary robotics – exploration vs exploitation, the genotype to phenotype mapping and the reality gap – are being addressed in this part of the thesis. In particular, modular robots are being used since the reconfiguration of robotic modules allows for a change in both the morphology and control systems of a robot. Non-modular approaches can also allow for morphological change though the use of modular robot parts additionally enable the evolved robots to be easily transferred to the real world, which expands the potential real world solutions that can be evolved. Another advantage of using a modular approach is that incremental improvements or the exchanges of modules can alter the functionality of the modular robot through simply replacing or adding modules to the system.

To gain more insight in how to evolve the behavior and control of a modular robot, a large portion of this chapter is devoted to explaining the evolutionary robotics plugin developed for the robotics platform: Virtual Robot Experimentation Platform (V-REP; Rohmer et al. 2013). This plugin covers many of the evolutionary algorithm and its parameters, the inclusion of an encoder able to construct modular robots from a genome, and the implementation of various controllers. The modules, environment and objective used can be changed in the plugin allowing for the emergence of a variety of modular robots. The plugin itself is not necessarily unique since others have developed similar software platforms to evolve modular robots as can be seen in Faîta et al. (2013) and Auerbach et al. (2014). The plugin was developed due to its ease of use, potential integration with other projects and having previous experience with it (Veenstra et al., 2015). Though the plugin in its current form is unable to evolve real-world modular robots directly due to the lack of a standardized performance feedback from a physical robot, the simulator can directly control a robot in the real world and in the simulation simultaneously. The robots that were transferred to reality in Chapter 7 were simply evolved in the simulation environment. Part III of the thesis considers online evolution directly applied to physical robots without using the plugin.
Chapter 4. Evolutionary Robotics Platform

Modular robots are important since they can eventually give us insights in how to construct various (reconfigurable) robotic morphologies. This both eases the production of robots and allows for a feedback loop to evaluate various developmental models and their relevance to robotics. Chapter 5 and Chapter 6 implement a generative encoding and discuss it with regards to developing a robot morphology. Chapter 6 specifically highlights the importance of the genotype to phenotype mapping. Chapter 6 describes the evolution of modular robots evolved for locomotion while Chapter 7 utilizes the same modules with the addition of solar panel modules, to evolve modular robots with energy autonomy.

The control of the modular robots in Chapter 6 and Chapter 7 is based on neuroevolution. Each module could contain a neural network that was evolvable. However, the next three chapters didn’t implement neural networks to their full potential due to the necessity of keeping the control as simple as possible. But since it has been implemented and shown to be useful, this chapter will therefore also give a brief overview of contemporary neuroevolution strategies (Section 4.1) and the manner neural networks were encoded for the evolutionary robotics plugin (Section 4.1). Hence, a brief overview of the control strategies that were implemented in the simulator (Subsection 4.2.3) is presented. Afterwards, it describes a high level overview of the various features of the plugin in Section 4.2. In Section 4.4, the real modules that were developed are presented.

4.1 Neuroevolution

Neuroevolution, or the implementation of evolving neural networks (ENNs), is a bio-inspired optimization strategy that usually implemented to investigate the acquisition of behavior in artificial creatures (Sims, 1994b) and robots (Nolfi et al., 1994). A key question in neuroevolution is which type of neural network to use as the basis for evolution. This section therefore gives a brief overview of some prominent neuroevolution strategies.

ENNs can be optimized without the strict forms of learning algorithms such as backpropagation that are usually implemented in neural networks. Many different types of neural networks with various learning and evolutionary strategies have been developed over the past few decades with auspicious results. In Stanley et al. (2002b), the technique Neuroevolution of Augmenting Topologies (NEAT) was introduced and subsequently implemented in many instances as an effective strategy to generate neural networks to control robots. NEAT is an efficient method since the neural networks of the initial population starts with basic connections of the input neurons to the output neurons subsequently augments the networks. The mutation operators facilitate the addition or deletion of neurons (nodes) and the connections (edges) between the neurons. The mutation operators in addition alter the activation function of the nodes and adjust the weights of the edges. One more feature that has been implemented to protect innovation is speciation, also known as niching (Stanley et al., 2002b). Speciation is made possible in the population of neural networks through implementing historical markings that track the historical origin of the evolved genes.

A prominent extension of NEAT is Compositional Pattern Producing Networks (CPPNs; Stanley 2007). In this case, a collection of various types of functions are employed to produce patterned outputs. This approach is not only used for control, but also for outputting a spatial pattern for the morphology of robots (Auerbach et al., 2011; Cheney et al., 2013). It has also been used to create static 2D images (Secretan et al., 2011) and 3D objects (Clune et al., 2011). Another extension is the use of CPPNs to create the neural network itself, an approach called HyperNEAT (Stanley et al., 2009). This approach has been further extended by Risi et al. (2010) to evolve the substrates of the neural network. I.e. specific regions of the network could change the placement and densities of the neurons in that area.

The utilization of CPPNs as an abstraction of development is valuable as a generative encoding. Multiple neurons can be created from a range of input values. Other classical approaches of neural networks are usually produced by direct encodings
as in the original NEAT version and other approaches (Angeline et al., 1994; Yao et al., 1997). Moreover, within the evolutionary loop, other machine learning strategies, like backpropagation, can be implemented in the neural network combining evolutionary algorithms with other learning strategies. For example, (Belew et al., 1992) found that evolving the initial weights enables backpropagation to find better solutions compared to without evolving the weights. An algorithm implementing both backpropagation and neuroevolution would thus be similar the approach of Hinton et al. (1987) where development during an individual’s life time could potentially increase the overall fitness. It should therefore be taken into consideration that a learning strategy could always be implemented in addition after a network has been evolved.

One more feature that is frequently implemented in neural networks are Central Pattern Generators (CPGs). CPGs mimic neural circuitry by generating continuous activation loops within neurons. In most animals, rhythmic activation of motor neurons enabling locomotion is regulated by some sort of CPG (Bear et al., 2016). The activity patterns generated by CPGs are important for locomotive gaits since they provide rhythmic actuation of muscles (Still et al., 2006). As CPGs do not require any sensory information they may be of particular importance in evolving robots. Therefore, patterned inputs to the simple neural networks used in chapters 6 and 7, implement a patterned input to the neural networks of the modules by simple sinusoidal functions.

Neural networks are usually implemented as centralized control strategies in robots; one controller processes all the inputs and generates all the outputs across the entire robot. A neural network creates outputs based on the network topology, internal activation and the sensory input. In the case of CPPNs, the network has also been used to create the robots themselves, where control can be localized producing a type of decentralized control. Auerbach et al. (2011) and Cheney et al. (2013) showed how local body parts created by the CPPN also encoded for the control system in the robot. In modular robots, this type of decentralized approach is already prevalent where individual modules can be assigned “roles” and act accordingly (Stoy et al., 2002). The ability of the change in these roles furthermore increases the adaptiveness of the robotic system (Pfeifer et al., 2006).

As will be described in Chapters 5, 6 and 7, a Lindenmayer System (L-System; Lindenmayer 1968b) has been implemented as the generative encoding to create robot morphologies. An approach similar to Hornby et al. (2003). The L-System was initially chosen as a plant-inspired approach to creating robots. Additionally, each of the different modules was controlled by a neural network where inputs and outputs could be distributed from one module to the next. Each module exhibiting its own role through its neural network. Modularity again being effective for isolating the functionality in specific parts of the robot (Stoy et al., 2010). The basic approach to controlling the modular robot is thus guided by a generative encoding and results in a decentralized control as illustrated by the dark blue arrow in Figure 4.1. The other arrows in Figure 4.1 depict a few other potential pathways one can take to create a decentralized or centralized approach to controlling robots when implementing neural networks.

Though natural systems might seem to exhibit control in a centralized manner, many organs and tissues also contain local control elements. One of the most prominent examples is the nervous system of octopuses whose neurons are largely distributed throughout its entire body. Each of its arms containing a staggering 3 million motor neurons (Levy et al., 2017). This can have several advantages for octopuses. For example, reflexes and reaction speed of arm movements are quicker when their control is localized. The octopus has therefore also been an interesting model itself as discussed by Laschi et al. (2012). The reflexes are similar to how they are controlled in for example the lower body of humans through a feedback loop from the spinal cord, not the brain. In contrast to the usual centralized controllers, a modular approach can lead to decentralized control scheme where parts of the body are controlled locally; a decentralized embodiment of robotic control.
Chapter 4. Evolutionary Robotics Platform

4.2 Overview of the Evolutionary Robotics Plugin

The robotics simulator used in this thesis was Virtual Robot Experimentation Platform (V-REP; Rohmer et al. 2013). This simulator was chosen since it is open-source and has many capabilities through which one can interface with it using various coding languages. To potentially maximize the performance, a C++ based plugin was developed and integrated with V-REP as a Dynamically Linked Library (DLL) plugin. The DLL plugin communicated with V-REP continuously; at start up, and most importantly, every simulation time-step. The DLL plugin was developed through using a C++ based template provided by V-REP that communicated with V-REP through functions that were called during the start, stop and iteration of a simulation.

For the evolutionary algorithm, a loop was integrated automatically starting and stopping the simulator based on a settings file that was modifiable. The main function of the DLL was to instantiate an evolutionary robotics class which was linked to the evolutionary algorithm. This class encapsulated the environment, morphology and control with three abstract factory patterns. The abstract morphology and control classes could moreover contain class instances of modules and neurons respectively. The modules and neurons were also created using a factory pattern and were instantiated whenever the settings file contained information indicating that they should be used.

The in total five factory patterns created robots, with neural networks and modules, in specific environments, which depended on the initial settings. This approach enabled the quick replacement of a type of module with another type of module by simply specifying in the settings file which modules the factory pattern should instantiate. The aim of this approach was to keep the simulator as flexible and accessible as possible. In theory, another researcher could create and integrate another module to directly incorporate it and evolve this new module with the existing modules by either supplying a ".ttm" file (model file in V-REP) or a blueprint C++ class describing how the module should be constructed and controlled. However, though potentially a useful tool for other researchers, the main purpose of the simulator was to conduct scientific experiments and therefore the functionality and the flexibility of the simulator in this state are obviously limited and contain messy code. Instead of a detailed description that can take up numerous pages, some important highlights will be given to further explain its functionality. The basic functionality can be found in Figure 4.2 while a full UML description is included in Appendix B.

4.2.1 Evolutionary Algorithm

The evolutionary algorithm was implemented in two ways: (1) in the main loop of the DLL plugin and (2) as a server-client implementation. In both cases it was initially
written as an SSGA. It has been extended to encompass other types of evolutionary algorithms though these were not implemented in the experiments in the remainder of the thesis. In the first approach, where the evolutionary algorithm ran in the main loop of V-REP, it simply initializes a random population of individuals by instantiating a fixed number of genomes for a given population size. The simulation subsequently starts by sequentially evaluating the genomes in the population through the following steps:

1. Construct the Robot and its control from the genome
2. Initialize its position
3. Start the simulation and update its control at every simulation time step
4. Receive a measure of fitness based on e.g. distance traveled or energy acquired after a specified period

From here on, it follows the same steady state evolutionary loop as explained in Section 2.6. After a specified number of generations, the genetic algorithm elicits a stop function that shuts down the simulator.

The later server-client implementation was used to evaluate multiple individuals in parallel speeding up the evolutionary progressions and allowing parallelized single runs on computer clusters. This required one processor of a node on a cluster to be responsible for the evolutionary algorithm while a number of client applications (depending on the amount of processors available on a node) of V-REP applications booted and waited for commands from the server. The server application ran the evolutionary algorithm, created genomes and stored the genomes in a shared folder. Once a generation of genomes was created, the server sequentially assigned a client instance to evaluate a specific genome. After the client was done evaluating a genome, it sent a fitness value back to the server application which was in turn stored in the client application. Since genomes could result in different phenotypes and thereby different computational requirements per client, as soon as a client was done evaluating an individual, it received a new genome from the server to evaluate; a process repeated in every client until the entire generation was evaluated. Once every individual in a generation was evaluated, the server incremented the evolutionary algorithm by going
through the selection, reproduction and replacement phases producing a new generation of individuals. Based on the fitness values, all elites of the genomes were stored in a specific shared folder. The size of the robots that were simulated by the client applications could vary in size which impacted the computational requirements in each different run. The runs that evolved smaller robots where therefore also significantly faster to evaluate since the computational requirement was significantly reduced.

For testing the simulator, the plugin was used to directly evolve a population on a single terminal with a Windows operating system. The server-client implementation was mainly used on computer cluster though could also be used on in a Windows operating system directly. This server-client implementation is thereby still useful to use on a personal computer when using multi-core processors.

### 4.2.2 Morphology

Each genome stores a base class reference to a morphology. The morphology type, instantiated by the morphology factory, is acquired from the settings file. This morphology type indicated that the genome should load either a fixed, or a modular morphology. This fixed morphology was used in Veenstra et al. (2015) and the modular morphology is used in this thesis. This modular morphology, construed by a generative encoding, was randomly initialized for the initial generation. In the simulation environment, each variable of the implemented L-System represents a specific module state which encompasses all the parameters of the morphology, control and attachment rules inherent to a module state. The genome of the generative encoding is thus composed of a fixed amount of module states predefined before an individual is generated and evaluated in the simulation environment. The mutable parameters of the module state are used to create new types of attachment rules and connections of the modules. The manner through which these rules produce the modular robots will be discussed in the next few chapters and are therefore not illustrated here. They are illustrated in detail in Figures 5.4 and 6.3.
4.3. Current State and Functionalities

All modules have a male connector site and zero, or multiple, female connector sites. When assembling a modular robot, the robot is constructed from an initial module (axiom). This initial axiom of the L-System is always the first module type that is defined in the settings file. Male connector sites of other modules can potentially connect to the female connector sites of the initial module. Through this process, the modular morphology can be represented by a tree and hence the use of an L-System is further justified. Each module type that has been elicited for use by the evolutionary robotics plugin, stores information about the child modules and their orientations that specify how the modules are connected when incrementing the L-System. However, when this incrementation step causes a generated module to collide with an existing structure, it will not be created. The outcome of the structure is thus also influenced by the environment as in a context sensitive L-System (Prusinkiewicz et al., 1990). Each module is created directly by an instantiation of a virtual module class. This module class contains all the instruction to construct, and subsequently control the module if it contains any actuators.

4.2.3 Control

The control of the modules is regulated by neural networks that were generated for each module. In Chapters 6 and 7 the neural networks used for each module type were however simple and open-loop, the networks didn’t utilize any sensory input other than predefined actuation functions. The neural networks consisted of three layers, a feedforward input layer, a potentially recurrent hidden layer and an output layer. The input neurons could be connected to sensory input such as light absorption or servo position. The output neurons could be connected to available actuators in the modules. The input neurons could give a time dependent patterned output that was defined by a sinusoidal function when specified. The amplitude, frequency and phase offset in the sinusoidal input neuron were mutable parameters.

The neural network was initially developed to allow for complex communication patterns to arise in the modular robot between modules. This was done through allowing additional input and output neurons across different modules to influence one another. E.g. an output neuron could connect to the input neuron of an adjacent module directly. These implementations are however reserved for future experiments using the evolutionary robotics plugin to evolve modular robots. This will hopefully shed light on how to improve our designs in robots and whether it is efficient to control a robot in either a centralized or decentralized manner.

4.3 Current State and Functionalities

The current state of the simulator permits the adjustment and control of many parameters of the evolutionary robotics plugin. A small user interface was developed to
quickly see the effects of a specific evolutionary approach (Figure 4.3). In this user interface, the parameters of the evolutionary algorithm, the encoding, the environment, the control type, and the fitness function can be specified.

Some noteworthy additional features were implemented to potentially improve the effectiveness of the evolved modular robots. One of these features was online growth of the robot (Figure 4.4). This function was implemented since growth was one of the objectives of the plant-inspired approach to building modular robots. Where the input of a light sensor input could potentially guide the addition of new modules towards a light source, similar as in Zahadat et al. (2017). Growth is however not used since the current setup would require a human-in-the-loop to adjust the phenotype of the modular robot. This would make the acquisition of modular robots more complex since a researcher would not only have to evaluate an evolved robot in the real world by assembling and controlling it, but also through adjusting it during its "lifetime". A truly autonomous approach would be to facilitate the modular robot to self-reconfigure. One approach to accomplish this is through using an industrial robot arm to construct the modular robots; an approach that will be briefly discussed in Chapter 8.

Another feature not utilized but worth mentioning is the connectivity of separate neural networks in each module whose activation could be visualized as seen in Figure 4.5. The figure shows that neural networks with a single output ranging from -1 to 1 can control the desired angle the servo motors. Where an output of 0 kept the servo motor in the center (grey), an output of 1 moves the servo motor +90 degrees (white), and a value of -1 moves the servo -90 degrees (black). The movements were controlled by PID controllers that were similar to the implemented Robotis AX-12a and AX-18a servo motors used in the real modules. Cheney et al. (2013) similarly visualizes the voxels of a simulated soft robot depending on their activation pattern useful.
4.4 Real Modules

In conjunction with the modules designed for the simulator, some of the modules were also created in reality. The modules that were used in Chapter 6 were based on modules that were collaboratively developed. The EMERGE module, which is a variation of the module used in this thesis, is discussed in Moreno et al. (2017)\(^1\). In the EMERGE module PCBs have been developed with additional infrared sensors that have not been implemented here. Instead, I have designed PCBs with six instead of 3/4 electrical channels. The six channels allow electricity to flow both to and from a power source allowing for the potential automated charging of a power source. However, more channels meant less space, hence no infrared sensor was integrated.

The connection mechanisms and the servo motors used where the same in the different module types. The specific mechanism that connects the two modules, and the PCBs attached to these connection sites, is depicted in Figure 4.6. The real modules are connected to one another via magnet-based connection sites (Figure 4.6a). The connection sites contained PCBs with pads and spring pins that enabled electricity to be routed between modules. As mentioned, the connection sites could either be male or female. The male connector sites contained spring pins that were soldered on the pads of the PCB (Figure 4.6b). The female connection site contains 3mm pads to ensure a connection with the spring pins on the male connection site. The PCBs included six separate channels through which electricity could be routed. 3D printed hulls housed to directly see how the phenotype adjusts itself to input signals changing over time.

The implementations of communication between modules sprouted from an investigation on the effect of the directionality of communication between neural modules. In a decentralized control, communication from the axiom module to the distant modules, \textit{vice versa}, and communication in both directions was possible. Here, the sensory input and motor output can be transmitted across the modular robot. This approach might shed light on whether a decentralized embodied control of modular robots is advantageous. Which implementation is truly useful, and whether any of these implementations could explain signal propagation in nature, remains to be evaluated in future experiments.

4.4 Real Modules

Figure 4.6: Module connection site. (A) The connection mechanism of the modules is composed of a female connector (left and middle) and male connector (right). Each module contains six channels through which modules can distribute power (PWR), ground (GND) and data (D1,D2,D3,D4) from one module to another. Both the male connector and the female connector have copper pads. Spring pins are attached to the pads on the male PCB. These spring pins (right) allow for a current to flow between modules (B). The connection sites contain magnets through which male sites can be connected to female sites. The blue (left) PCBs have places to connect the spring pins and is used for the male connector sites while the black (right) PCBs are used for the female connector sites.

\(^1\)An overview of the different module types and the parts needed to construct them can be found here: https://sites.google.com/view/emergemodular/home
Chapter 4. Evolutionary Robotics Platform

Figure 4.7: Interface Connector. The connection site, USB cable and power hub that powers and controls the modular robot. This connection site could be connected anywhere on a female connection site of the modular robot.

Figure 4.8: Collection of Modules. 5 solar panel modules, 7 servo modules and 3 structural modules are depicted.

Four cylindrical NdFeB (neodymium, iron and boron) magnets (12mm diameter, 3mm depth). Two magnets can hold one another together with a force of roughly 13.5N. The male connector hull has protrusions that ensure the fit and connection of the male connector to the female connector.

Two solar panel modules have been designed for the modular robot and are implemented in the experiments discussed in Chapter 7. One ‘flower module’ has been collaboratively designed with a MSc. intern, Chloé Metayer. This flower contained an additional vibration sensor and a servo motor that could open and close the module base on the sensory input. In addition to the experiments in the next chapters, preliminary results of the automated assembly of modular robots is also presented in Section 8.2 and spurred from a collaboration with Rodrigo Moreno and Andrés Faíña where we wanted to create a platform that could automatically create modular robots and evaluate them in the real world. Since no evolutionary algorithm or three-dimensional simulation was implemented in this approach, it is briefly discussed in Part III of the thesis.

The modular robots created were connected to an external power source and a laptop via an additional connection site (Figure 4.7). This connection site could be connected to any open female connection site that would in turn automatically power and control all modules. A personal computer is connected to a central 3-pin power
hub via a USB2AX interface. As can be seen in Figure 4.8, the collection of modules was composed of servo modules, solar panel modules and structural modules. The simulator could in turn limit the number of modules used based on the modules that were available in reality. Hence it would always evolve robots that were feasible to create in reality.

Figure 4.9 depicts how a modular robot containing five servo modules, and a modular robot containing two servo modules and a solar panel, could be controlled in simulation and reality. Through moving the servo modules in reality, the real modular robot also adjusted its servo motors accordingly. The rays that come out of the solar panel module represent how the solar panels absorb light. The light absorption rate becoming lower when the solar panel is not perpendicular to the light source. Ultimately, the connection of the simulator to the modular robot can lead to a feedback loop of automated evolution where multiple individuals can be evaluated in the simulator while some elites are evaluated in reality.
4.5 Concluding Remarks

The evolutionary robotics plugin discussed in this chapter conveys a wide variety of potential applications in the study on the evolution of modular robots. The plugin implements an evolutionary algorithm, a morphology constructed through a generative encoding and a decentralized method of controlling the modular robot with neural networks. It has been implemented for the evolution of plant-inspired virtual creatures (Chapter 5), comparing encoding strategies (Chapter 6) and evolving energy autonomy in plant inspired modular robots (Chapter 7). There are many preliminary implementations that can shed more light on how to evolve and design robots in the future, such as growth and decentralized control. The use of modules being essential for the rapid prototyping of various morphologies with corresponding control strategies that can aid in our understanding how to best create robots.
Chapter 5

Evolving Plant Morphologies with L-Systems

The chemical differences among various species and genera of animals and plants are certainly as significant for the history of their origins as the differences in form. If we could define clearly the differences in molecular constitution and functions of different kinds of organisms, there would be possible a more illuminating and deeper understanding of question of the evolutionary reactions of organisms than could ever be expected from morphological considerations.

– Edwin Ray Lankester

Plants are primary energy producers. Without them, we likely wouldn’t see the higher order terrestrial organisms we see today. All organism being tangled in a trophic structure, where organisms higher in the food chain are solely dependent on the lowest. Where these food chains are usually short due to inefficiencies of energy transfer across the chain (Reece et al., 2010). For natural evolving systems, energy acquisition is therefore most important. The design of an artificial entity can therefore start with energy acquisition. And what better suitable model for this than plants?

In this chapter we therefore verge in realm of phytomorphogenesis, the acquisition of artificial plants optimized for the acquisition of light. Since the evolved creatures are plant-inspired and have light absorption as their fitness function, the morphologies are referred to as phytomorphologies (plant morphologies). Although the initial aim of the presented research was not directly related to modular robots, there are many features of plants that make it relevant for modular robots and robots in general. This chapter describes an approach, related to modular robots, to evolve the morphology and a minimalistic control of plant-inspired robots. In order to achieve this, a Lindenmayer-System (L-System; Lindenmayer 1968a; Lindenmayer et al. 1992) is implemented to see what type of morphologies emerge from the platform. The use of L-systems for the creation of 3D robots is thereby also used in other experiments using the robotics simulator as a generative encoding.

Considering macro structures in different organisms, plants exhibit many recursive patterns as self-similarity and modularity considering its leaves and branches (Figure 5.1). As described in Section 2.3, many of these characteristics are potentially beneficial for the evolution of any type of agent. As plants are relatively simple organisms, investigating how to evolve phytomorphologies might thus eventually lead to the creation of complex robots that could display characteristics of higher order organisms while also being ‘primary energy producers’. A curiosity that will be investigated in this chapter is whether phytomorphologies that have been evolved will exhibit movement to track a light source or not. Hereby formulating our second hypothesis:

Hypothesis 2 Actuation in Evolving phytomorphologies is Beneficial for Optimizing Light Absorption.
5.1 Introduction

The development of phytomorphological elements of plants ultimately arose from a dynamic interaction between genetic, ontogenetic and environmental forces. Phyto-morphological traits have emerged through the evolution and selection of plants, favoring those that were adequately adapted to their environment. Different environments stimulate the development and evolution of specific qualities in plants and contribute to the adaptation of plants to specific environmental niches. Light-absorption being one of the most essential characteristic prevalent in almost all plants. The resulting role of plants as primary consumers conveys their fundamental impact on any terrestrial ecosystem.

Urban environments have replaced a large share of plant-rich environments meaning that the potential energy up-take in these environments is exposed and primed for solar exploitation. For an efficient, but still aesthetically pleasing, deployment of solar cells, the developmental processes manifested by years of plant evolution is investigated. Hence, gaining insights into how plant development works and how this can be mimicked in intelligent robotic and autonomous systems is the main interest of this chapter. For investigating how to properly embody such systems, an evolutionary developmental model was used for investigating various factors that have contributed to the emergence of phytomorphologies.

In plants, various signaling mechanisms have evolved to communicate environmental factors to remote cells and tissues. Moreover, the cell walls of plant cells contribute to the relative immobility as well as the rigidity of plants, limiting cell migration and thereby actuation. Lacking a nervous system, plants are forced to utilize relatively slow signaling molecules for communication. These molecules atone for the lack in efficient communication mechanisms through various diffusion and transduction pathways. The signaling molecules can be transported through an apoplastic (through the cell wall) or symplastic (via the cytoplasm; through plasmodesmata) pathway. Various molecules can also be transported over long distances through the vasculature of the plant.

Although plants acquired efficient dynamic behavior that directly influences morphogenesis, a complete modelling of plant signalling, as done in Zahadat et al. (2016), was not done since this might over complicate the design process and the potential increased computational requirement. Where Zahadat et al. (2016) implement a strategy that is similar to the concept of morphogens (Wolpert, 1969), in this case simulated hormones. Since actual robotic implementations of evolved phytomorphologies are likely not able to grow or move once created, grammars seem to be a suitable method to implement for generating potential phytomorphologies. Convention approaches of these grammars can be simulated by simple grammars have been popularized with the use of Lindenmayer systems (L-Systems; Lindenmayer et al. 1992). Furthermore,
since L-Systems work with variables, they can easily be extended to contain signal propagation algorithms and even morphogens themselves if one chooses to. One could potentially also model morphologies with direct encodings. A comparison of evolving robots with these two methods is discussed in the next chapter (Chapter 6). In this chapter, an evolving context sensitive L-System is implemented to engender phytomorphogenesis of artificial plants.

5.1.1 Phytomorphogenesis

Variation in plant features is influenced by many factors including ecophysiological, phenological, morphological and ontological traits. Other important factors driving plant-evolution include resource allocation, biochemistry, metabolism, and leaf morphology and function (Ackerly et al., 2000). All the genes are in turn subject to evolution and specific genetic components are selected for across generations. Where the absolute fitness value of a plant is, in biology, determined by either the amount of viable seeds it produces during its lifetime. Or as a more useful measurement, its fitness can be represented by the amount of genetic information it is able to propagate into the next generation. In both cases, a difficult value to measure value.

However, it has been shown that the photosynthetic rate of leaves in plants has a direct influence on the absolute fitness (the proportional change in the abundance of a genotype over one generation) of Arabidopsis thaliana. One specific gene (Alt1g61800) causes leaves to produce more chloroplasts when plants were placed in a different environment where they were subjected to higher light-intensity (Athanasiou et al.,
78 Chapter 5. Evolving Plant Morphologies with L-Systems

2010). This is an example of the importance of dynamic feedback to plants. Solar cells don’t have to worry about receiving too much light but a solar cell’s capacity of acquiring energy can be modeled as a trade-off between average available light and amount the peak light intensity a solar panel is able to capture. Since dynamic behavior in plants is evolutionary speaking a result of various compromises a plant takes to optimize its survival and reproduction, this behavior might not be necessary for plant evolution under controlled conditions. The experiments presented in this chapter therefore focus more on investigating intrinsic properties of plants that contribute to the generation of phytomorphologies.

*Phyllotaxis* is the main factor driving phytomorphogenesis (Cells, 1997). The most common patterns formed in plants through phyllotaxis include distichous, spiral, decussate and whorled patterns (Kuhlemeier, 2007). Notably, the divergence angles of *primordia*, tissues containing cells capable of phyllotaxis, differ usually by 180°, 90°, 137.5° (Fibonacci; Newell et al. 2005) and some other more uncommon angles (Kuhlemeier, 2007). These, mostly unimodal, angles influence how well the leaves sprouting from the primordia can absorb light and overshadow other leaves (Falster et al., 2003). Leases can also be positioned at a certain level of steepness which is advantageous for either preventing self-shading or capturing light from the morning and evening sun (Falster et al., 2003). Steeper angles of lamina (blade of the leaf) are also more beneficial for plants that receive an amount of light higher than the maximum photosynthetic potential of a plant. When the leaves are steeply oriented, other leaves, that would otherwise be overshadowed, can receive more light and thus the overall photosynthetic activity of the plant is increased. Other evolutionary trade-offs that emerge in the leaves of plants include e.g. mass-to-area ratio, sap flow versus heat processing, CO$_2$ uptake to water loss ratio, and the leaf size-to-number ratio (Nicotra et al., 2011). Moreover, hormones, such as auxin, play an important role in embryonic development, cellular elongation and phyllotaxis (Prasad et al., 2013). The driving factors that bring about phyllotaxis in plants can make the search space of an evolutionary model quite convoluted which artificial plants do not necessarily have to take into account. As will be discussed next, an L-System can result in patterned formation that is similar to phytomorphologies.

### 5.1.2 Simulated models

Computer models of plants have generally been implemented in computer graphics (Habel et al., 2009) for accurate modelling of plant dynamics (Courmède et al., 2008; Prusinkiewicz et al., 2012; Merks et al., 2013; Runions et al., 2014) and for assessing the role of evolution on the emergence of plant traits (Valladares et al., 2000). Moreover, evolutionary computations and generative encodings have been implemented to efficiently simulate plant models (Zamuda et al., 2012; Zamuda et al., 2014) with some biological accuracy. In previous work on generating patterned morphologies, and for keeping the morphological encoding simple, generative encoding strategies, such as the parametric encoding used in the work of Sims (Sims, 1994a), are usually implemented since they can recursively generate body segments. Different types of generative encoding strategies have been developed over the past two decades to abstract developmental strategies towards generating both morphology and control of virtual creatures (Eggenberger-Hotz, 1997; Yeom et al., 2010). One strategy for generating artificial structures linked to neural networks is known as artificial ontogeny (Bongard et al., 2001; Bongard, 2003). In this method, an agent’s simulated spherical elements can grow by increasing in size and splitting in two. As a result, repeated divisions can transit a single unit in a fully developed agent. Each separately created unit contained up to six joints and diffusion sites. These diffusion sites could in turn contain zero or more sensory, motor and interneurons. Despite a promising application of artificial ontogeny to produce plant-like structures, the implementation of neural networks can result in a great increase of the search-space making it a less attractive system to implement. Nonetheless, it is an interesting methodology to implement in the future.
5.2 Approach

A Lindenmayer system (L-system; Lindenmayer 1968a) is another grammatical generative encoding approach, originally used to mimic plant development by iteratively rewriting variables and constants through a set of rules (Lindenmayer et al., 1992). L-Systems can be seen as a developmental representation of a virtual plant. Like some other generative encoding strategies, the similarity of L-Systems to biology includes its reuse of rules and variables comparable to how organisms reuse genes. Further relevance L-systems have to biology can be derived from the fact that cells, or parts of plants, can change their state, or cell fate. This determines the behavior and ultimately the phytomorphogenesis of plant form and structure. L-Systems are thus an attractive method to implement for creating artificial plants both as they somewhat mimic biological development as well as being extremely simple and efficiently encoded. L-systems have furthermore been used to create the morphological structure of virtual creatures with reactive controllers (Hornby et al., 2001). This approach can similarly be effective for the generation of virtual plants.

5.2 Approach

Virtual Robot Experimentation Platform (V-REP; Rohmer et al. 2013) is used as the simulator to create and evaluate plant-like robotic morphologies. The simulated components are controlled via a C++ based DLL plugin created with visual studio 2013 as described in Chapter 4. The plugin is divided into three parts: a genetic algorithm, a morphology generator and a control part. The genome of the morphology is encoded as the rules and parameters of the L-Systems. Two experiments were done to simulate 16 evolutionary runs for evolving static phytomorphologies as well as 16 runs for evolving phytomorphologies which contained joints that could rotate.

5.2.1 Genetic Algorithm

The implemented genetic algorithm is a steady state genetic algorithm (SSGA; Wu et al. 1995). And in our case, a random offspring is generated asexually, without crossover, and evaluated against a random individual in the population. The random selection and a population size of 100 individuals was used to keep the population somewhat diverse and to slower the convergence of the evolving L-System to a local optimum. The genomes of the initial population were furthermore randomly initialized. The individuals were evaluated based on their ability to absorb light in an environment that only contained a flat surface, a light source and the individual itself. When comparing an evaluated offspring with a random individual of the population, the offspring would only replace the selected individual if its fitness value was higher. Based on preliminary experiments, the mutation rate was set to 5% meaning that each variable of the genome had a 5% chance of being changed in an offspring. When mutating the variables, either a completely random new variable could be assigned, or a local mutation could cause the variable to change slightly. Were local spread as mutations was implemented to explore the local search-space whereas the random variable was intended to potentially promote diversity. To recall a morphology without needing an L-System to construct it, the evaluated morphologies were also stored.

Ten evaluation steps contributed to the eventual fitness value of a virtual plant. At each time-step, the amount of light absorbed by the simulated leaves of an individual was calculated. The orientation and surface area of the leaves have a direct influence on the amount of light absorbed by the leaves. The amount of light absorbed is calculated by the multiplication of one light-sensitive surface area of the artificial leaf with the z-directional vector of the leaf relative to the directional vector that is oriented from the leaf’s origin to the origin of the light source. Furthermore, if there is anything between the artificial leaf and the light source, the leaf does not contribute to the overall fitness value of the individual. The light source that directly influences the fitness of the virtual plants is moved at each time-step. Starting at the Cartesian coordinate (2.0, -4.0, 10.0) and ending at the coordinate (2.0, 5.0, 10.0). This light source thus moves in the direction of y with a directional vector of (0.0, 1.0, 0.0) as illustrated in Figure 5.3.
The fitness function for each individual is given in Equation 5.1.

\[ F = \sum_{i=1}^{n} \left( \sum_{j=1}^{o} \alpha \delta - \sum_{k=1}^{p} \beta \right) \]  

(5.1)

The fitness \( F \) is the sum of the acquired fitness values at ten time-steps as represented. \( n \) representing the upper bound of the number of time-steps. The total number of leaves is given by \( o \) and \( p \) represents the total amount of objects formed by the individual. \( \alpha \) represents the surface area of the artificial leaves which is multiplied by the \( z \) directional vector \( \delta \). \( \beta \) represents the volume of the objects.

### 5.2.2 L-System

The implemented L-System was a context sensitive L-system (Prusinkiewicz et al., 1990). The context refers mainly to the simulated environment. For example, in order to prevent objects from overlapping, a feedback loop to the L-System ensures that the created morphology does not contain any overlapping/colliding objects. The L-system contains a total of 10 variables which are referred to as the specific states of the objects that are created. Each state of the object contains corresponding rule sets that define what child objects are created. An example of how the states, rules and constants of the L-System influence morphogenesis is displayed in Figure 5.4.

The L-System generates morphologies by iterating seven times through the state parameters of the morphology. Seven iterations were subjectively chosen as they seemed to display a good diversity of morphologies without requiring too much computational power. The axiom of the L-System is a state 0 object. Before the first iteration of the L-System, an object in state 0 is therefore created at the center of the environment on top of the floor. Afterwards, the first iteration of the L-system will generate objects that the rules in state 0 produce. Having only seven iterations, an object chain from the initial object to the outer most child consists of a maximum of 8 objects. Some loopholes in the L-System can quickly result in a very high computational demand and thus specific constraints are implemented. Every object in a given state can potentially...
5.3. Results

Figure 5.4: Three illustrations of the implemented L-System. The Genotypic representation shows how the production rules result in the generation of the morphology. The symbolic representation shows the developmental instructions and the relationships between states as similarly represented by the work of Sims (Sims, 1994a). The phenotype generated by the example is shown on the right. Note that the + constant represents a three dimensional orientation to which a new object is rotated relative to its parent.

create up to six new child objects. The maximum amount of objects that can be created is therefore limited to 50. Likewise, the amount of loops the L-System can make for generating these objects is limited to 200 to further limit the calculation time that would otherwise arise. To enable individuals to absorb light from the environment, two object states of the L-System genome represent artificial leaves that are expressed as rectangular cuboids. These leaves are colored orange. All other states represent spherical objects that shape the overall morphology. Spherical objects where chosen in order to effortlessly calculate the position of new objects without having to worry about collisions and overlapping objects. The objects in four other object states are colored red, blue, green and yellow while the remaining objects are colored black by default. Note that the first object created is always in state zero which is always colored red. An illustration of how the L-System generates the phenotype from a specific genome is depicted in figure 5.4.

Additional components are included in the L-System to enable movement of the joints in simulation time. Whether a joint moves is represented by one boolean value. The angular rotation a joint can make per time-step is limited to 36 degrees meaning that a joint can rotate a maximum of 360 degrees in a positive or negative direction during one evaluation.

5.3 Results

As can be seen in Figure 5.6, the average acquired fitness values of the population with static plants is similar to the fitness of the population of plants that could potentially actuate their joints. Since the evolutionary runs were not normally distributed (confirmed by a Shapiro Wilk test) a Mann-Whitney U Test was performed to see whether the results were significantly different. As can be inferred by looking at the graph (Figure 5.6), The Mann-Whitney U test confirms that the data is insufficient to reject its null hypothesis. For the amount of generations that we have simulated the populations, no statistical difference between the efficiency of static versus actuated phenotypes could be seen. Considering that the runs shown in Figure 5.6 did not plateau, a difference might emerge when simulating far more generations. Although a few phenotypes did utilize moving parts (such as Figure 5.8), the majority of the phenotypes that evolved did not move. Out of the 16 evolutionary runs of rotating individuals,
Chapter 5. Evolving Plant Morphologies with L-Systems

**Figure 5.5:** **Illustration of an evolutionary progression** A top view of individuals of one evolutionary run are depicted to illustrate how evolution shapes new more efficient individuals. The figures depict individuals from generation (gen) 20, 120, 220 and 320.

The best individuals of the final generations seldom utilized any actuation in joints that would change the shape of the artificial plants significantly.

The fitness values depicted in the graph of Figure 5.6 are quite arbitrary at first sight. However, with some extra knowledge the fitness value can make sense. For example, the fitness value of the best evolved individual (Figure 5.7) was 23.841. Without the negative contribution of the volume of the individual, it’s fitness would have been 31.9399. The division of this value by the amount of time steps results in the average surface area of the artificial leaves that was exposed to the light source. This is area is corrected by the relative angle the leaves had in respect to the light source. 3.19399 m² is thus the two-dimensional projection of the average light absorbing surface area of the artificial leaves. The total volume of an individual could also be extracted by checking the negative contribution of the volume. For the example the total negative fitness contribution of the volume of the individual discussed in this paragraph was 8.0989. The total volume of the simulated individual was thus 0.80989 m³. Hence, the phenotype seen in Figure 5.7 thus represents a structure with an average light-absorption area of 3.19399 m² and a volume of 0.80989 m³.

The phenotypes of the evolved phytomorphologies are quite diverse and different spiral patterned morphologies can be seen (Figure 5.2). In Figure 5.5, the best evolutionary run is mapped across different generations. Looking at the top view of this figure, one can see that the total amount of surface area exposed by the artificial leaves (orange rectangles) gradually becomes larger.

### 5.4 Discussion

An evolutionary developmental algorithm was employed to engender various phytomorphologies optimized to absorb light. As can be seen in Figure 5.2, a wide variety of phytomorphologies evolved. Functionally, these evolved morphologies don’t look
5.4. Discussion

Figure 5.6: Evolutionary progression of static and actuated phytomorphologies. The average fitness values of the populations across generations. The runs are not significantly different from one another (p-value was 0.782) when using the Mann-Whitney U test.

particularly optimal for light absorption as one would expect all the orange surfaces to point somewhat upwards instead of in the various directions shown in the resulting morphologies. Making longer evolutionary runs could shed more light on whether the evolutionary L-System can generate more efficient models. Moreover, actuating the morhologies did not change the population fitness values significantly when compared to the statically simulated populations. Blind tracking of a moving light source may have caused the search space to become more convoluted making the algorithm inept for finding solutions where actuation was more beneficial than not actuating anything. Another explanation is that growing many leaves is simply more useful than considering movement and actuation. Considering that plants do not exhibit complex light tracking behavior in most cases, the conditions set in the experiment were not enough to confirm Hypothesis 2.

The evolved virtual plants were quite voluminous considering that the volume has a negative effect on the fitness value. However, making large objects and dispersing the morphology over a large area, while making leaves with a thin volume but large surface area, is an intuitive result given the simulation environment. It is expected that different phytomorphologies arise when artificial plants have an additional restriction to grow horizontally. In biological environments, factors such as the overshadowing of neighboring plants, form an additional pressure that stimulate specific types of plants to grow tall quickly. Co-evolving the same L-System can therefore yield results that are more diverse than the ones shown in this chapter.

Considering the results, various future improvements of the evolutionary algorithm may increase the efficiency of a population to traverse the search-space. Since no crossover function was implemented, this might definitely increase the efficiency of the evolving L-System considering that specific states and rules of the L-System can be recombined with between individuals within the population. As mentioned earlier, the implementation of neural networks in addition to artificial development can be interesting for developing more dynamic morphologies (as in Bongard et al. 2003). Moreover, morphogens (Wolpert, 1969) are also an attractive strategy to implement.
Figure 5.7: The phenotype of the best evolved individual. Note that object chains are surrounded by artificial leaves.

Figure 5.8: Evolution of movement? An individual that rotated some of its joints during the simulation. T0, T2, T4 and T6 represent the respective time steps 0, 2, 4 and 6.

to mimic long range communication in plants. An algorithm that checks for diversity besides quality, as has been implemented in novelty search (Lehman et al., 2008), might also be useful to speed up the search process. Moreover, novelty search can lead to the evolution of very distinct morphologies making it more useful for people that possibly want to generate phytomorphological structures for aesthetic purposes. Additional implementations that have not been published but might be interesting for the reader is shown in Text Box 5.4.

5.5 Conclusion

L-systems can be utilized to create a wide variety of 3-dimensional phytomorphologies that can be optimized for light absorption. These phytomorphologies were generated with the aim of implementing them in urban environments for both functional and aesthetic purposes. Interestingly, evolution did not exploit possibly beneficial joint actuation but instead converged on various types of static phytomorphologies instead. The plots did seem to somewhat plateau and noticing that movement has not evolved makes us reject Hypothesis 2. In our system it is more likely to simply grow a lot of leaves than to additional actuate them. This evolving L-System can be extended by implementing additional algorithms to increase the effectiveness of traversing the fitness landscape for acquiring both more efficient and more unique phytomorphologies. it can moreover also include a potential dynamic feedback controller than might lead to movement in leaves that do aid the individual.
Prior to the publication of the work shown, an approach that was more related to Sims (1994a) was taken to create the actual morphologies. The difference between the approach described in this chapter was that objects where able to collide and simple rectangular shapes were used. Rectangular shapes eventually being less computationally demanding. This approach was however altered with the thought of crossing the reality gap in the future. The spherical approach was therefore easier since collisions were easy to avoid. Aesthetically however, the rectangular phytomorphologies look more interesting though this also depends on personal taste. Figure 5.9.
Chapter 6
Comparing Encodings for Evolving Locomotion in Modular Robots

Today’s organisms are phylogenetically descended from others which were vastly simpler than they are, so much simpler, in fact, that it’s inconceivable, how any kind of description of the latter, complex organism could have existed in the earlier one. It’s not easy to imagine in what sense a gene, which is probably a low order affair, can contain a description of the human being which will come from it. But in this case you can say that since the gene has its effect only within another human organism, it probably need not contain a complete description of what is to happen, but only a few cues for a few alternatives. However, this is not so in phylogenetic evolution. That starts from simple entities, surrounded by an unliving amorphous milieu, and produce, something more complicated. Evidently, these organisms have the ability to produce something more complicated than themselves

– John von Neumann, Theory of Self Reproducing Automata

The use of physical modules in robots enables both the morphologies and the controllers of robots to be subjected to evolution. The modular robotics approach additionally allows us to potentially quickly create the modular robot in the real world. It can then serve as a fast prototyping mechanism to evaluate different phenotypes in the real world. However, one major constraint is the modeling of the modular robot. To subject a modular robot to evolutionary algorithms requires an encoding from the genotype to the phenotype. This encoding greatly influences the emerging phenotypes of the robotics platform and should be considered in order to either explore or exploit specific parts of the search space.

Andres Faina had already developed a system capable of evolving modular robots originally developed for ROS (Robot Operating System). Since we thought V-REP was a promising new robotics simulator, Andres implemented the direct encoding he wrote for evolving modular robots (Faiña et al., 2011; Faíña et al., 2013) to V-REP. This enabled us to compare both the algorithmic implementation of the direct encoding and the generative encoding of the L-System as developed in Chapter 5. Though his API was written in Java, and mine in C++, the results of the evolutionary experiments were done in two separate implementations. The modules, stored as .ttt files, were simply loaded by the algorithms and after some optimization of parameters, the results of the simulators when fixing certain settings. Since a generative encoding creates a mapping from genotype to phenotype that is more recursive than a direct encoding, the expectation was that the generative encoding would be more efficient than the direct encoding for evolving modular robots. Hence the third hypothesis of this thesis:

**Hypothesis 3** A generative encoding increases the efficiency of evolving modular robots compared to a direct encoding.
6.1 Introduction

Evolutionary Robotics has covered a vast amount of research on the automated design of Robotic entities via artificial evolution (Lipson et al., 2000; Hornby et al., 2003; Eiben et al., 2013). To rapidly explore different robotic morphologies and control systems that can be physically assembled in the real world, robotic modules are useful as evolutionary building blocks. A robotic module being an independent unit that encapsulates part of its functionality (Stoy et al., 2010) This encapsulation is important for the (re)configuration of modular robot compositions. In contrast to static robotic entities, modular robots can be reconfigured enabling researchers to quickly explore different morphologies.

However, it is difficult to design a representation of the genotype to phenotype mapping of a modular robot and we can either evolve all parameters of every simulated module or reuse parts of the genome to construct and control a modular robot. The latter approach – a generative encoding – would require a smaller genome and could in turn evolve decent morphologies and control more quickly. In contrast, evolving all parameters of every robotic module enables us to fine tune behavioral parameters but also increases the search space. So, is it more efficient to fine-tune or to have recursive morphologies and control? Both encodings determine greatly influence the functionality of the evolutionary approach since being able to fine-tune is more likely an exploitative evolutionary approach while the abstractions taken in a generative encoding is likely more explorative.

In nature, most multicellular organisms develop from a zygote (Reece et al., 2010). The zygote and its genome comprise the developmental representation of the organism (Floreano et al., 2008). The resulting developmental process allows for the reuse of genes which can give rise to recursive structures in the phenotype. Computational models representing an artificial organism’s phenotype either use a direct or generative encoding (also indirect encoding). A direct encoding constitutes a one-to-one mapping of genotypic components into the phenotype meaning that the genes encode for every simulated module. In contrast, generative encodings – similar to the development of an organism from a zygote – reuse elements of the genome for constructing the phenotype. Generative encodings have a smaller genotypic space due to this reuse of genes.

Since the morphological search space in modular robots is limited to the number of connection sites available on each module, encodings that directly map the assembling process of modular building blocks have been implemented for the generation of robot morphologies (Marbach et al., 2005; Faïña et al., 2013; Guettas et al., 2014). Usually, these direct encodings implement an additional symmetry operator that increases the effectiveness of artificial evolution. A multitude of simple generative encodings have however been implemented to evolve robot morphologies and control (Sims, 1994b; Hornby et al., 2001; Auerbach et al., 2011; Cheney et al., 2013) have been shown to quickly lead to useful robot morphologies. It is, however, unclear whether designing platforms that evolve modular robot morphologies should rather use a direct or generative representation and if the generative encoding is still useful when just a few modules are being used. A generative encoding should no longer have an advantage if the genetic sequence space in both encodings is of similar size since the number of mutable parameters are equal.

Whether a generative encoding or a direct encoding is more useful for evolving modular robots for locomotion is the main concern of this chapter. Moreover, the experiments that will be described use modular robots of different sizes, only allowing up to a number of modules to be simulated in an individual. Both encodings make use of evolutionary algorithms to optimize the simulated robots. The direct encoding utilizes the ‘Evolutionary designer of heterogeneous modular robots’ (Edhmor; see Subsection 6.2.2) system (Faïña et al., 2013). Edhmor contains a specific evolutionary algorithm that evaluates robots through multiple mutation phases as explained in Subsection 6.2.2. The generative approach utilizes a steady state evolutionary algorithm (Syswerda, 1991). The generative encoding is based on a parallel rewriting system called a Lindenmayer System (L-System; Lindenmayer 1968a) as discussed in Chapter 5.
6.2 Methodology

Many modular robotic systems make use of central pattern generators for controlling the modules (Kamimura et al., 2005; Sproewitz et al., 2008; Bonardi et al., 2014). These central pattern generators are derived from their natural equivalents seen in biology (Still et al., 2006; Ijspeert, 2008). The implementation of modifiable central pattern generators seems a logical step toward evolving modular robots. However, this convolutes the search space of the evolutionary system unnecessarily for the aim of this chapter. To still achieve a patterned output in the modular system, sinusoidal functions control each module individually in a decentralized manner. By fixing the morphological parameters of the simulated modules and limiting the control parameters of the modules to sinusoidal functions, we were able to analyze how the different encodings can be implemented for evolving robotic structures. For evolving simulated robot morphologies, two evolutionary platforms were used to evaluate the direct and the generative encoding. Both platforms employ the robotics simulator ‘Virtual Robot Experimentation Platform’ (V-REP; version 3.32; Rohmer et al. 2013). The next sections will discuss the common elements as well as the differences for each platform.

6.2.1 Common Elements for both Platforms

Both encodings simulate the exact same modules modeled in V-REP. A cube module and a servo module were designed for the platforms. The modules are based on earlier designs of physical modules (Figure 6.1a). In turn, the simulated modules (Figure 6.1b) are modeled according to the physical properties of these modular units. The real modules can be attached to one another via magnets and the breaking force and torque parameters resulting from these connections is modeled in the simulated modules.

The modules contain male and female connection sites that enable the modules to connect. The connections are modeled with a force sensor in V-REP. If the force on a connection site exceeds 1.7 N m of torque or 80.0 N of force, the force sensor between the modules breaks leading to the fragmentation of the morphology. 10 consecutive connections do not fail in a row. The modules are simulated in the same way on both platforms and the same time step is used for all simulations. This is because the two platforms are of the same type and share the same robot models. The only differences are the specific implementation and the pragmatics of the evolutionary optimization algorithms.

The main difference between both implementations is the size of the search space. The search space of the direct encoding grows exponentially with the number of modules that determine the number of mutatable parameters. The search space of the generative encoding stays roughly the same size. Although more variants in phenotype are available when allowing more modules to be simulated, the genome of the individual stays the same size. The ability of the direct encoding to mutate parameters of individual modules enables more local, fine-tuning, improvements. In contrast, since small mutations in the generative encoding can lead to drastic phenotypic changes, the generative encoding might be more prone to stagnate in local optima. Though the scope of this chapter does not encompass transferability, the implemented encodings serve as a stepping stone towards evolving feasible modular robotic entities in reality.
threshold violations for the force sensor had to be registered before a connection could break.

The cube module (dimensions x,y,z is 55mm,55mm,55mm; weight is 100g) is used as an initial building block for the modular robot to which other modules are attached. This cube has five female connection sites (top, right, left, front, back). The servo module (dimensions x,y,z is 55mm,55mm,80mm; weight is 160g) has three female connection sites (top, right, left) and one male attachment site (bottom). The bottom male connection site of the servo module is thus able to connect to any of the female connection sites of other cube or servo modules.

The joint of the servo module implements a PI controller (P is 0.1 and I is 0.01) and could exert a maximum torque of \(1.5 \text{ N m}\). A sinusoidal wave function controls the position of the joint in the servo module. The maximum amplitude of the sinusoidal wave function ranged from -90° and +90° degrees from its original position. The offset, phase and amplitude of the sinusoidal function are mutable parameters. When a new module is added to the simulation, only the male connection site of the new module can be connected to any female connection site of the robot. The new module has four different orientations in which it can attach to a new connection site (note that the number of orientations is a bit different in the direct encoding: Subsection 6.2.2). The servo modules implemented the default simulation material while the cube module used the "rest_stack_grasp_material" as material types simulated by V-REP.

The goal of the simulated robots was to move as far away from its initial position in a horizontal direction as possible within 20 seconds of simulation time. This distance is measured by the horizontal distance that the initial cube module has traveled. Before starting a simulation in V-REP, modules are joined together to form a robot morphology. The entire robot is then shifted upwards so that its lowest point is 0.1 millimeter above the simulated ground. To take into account the movement due to the robot simply falling over, the distance traveled in the first 2.5 seconds of the simulation is discarded. An additional cost function was added to compensate for modules that were disconnected due to the breaking of a connection site. The fitness value of each individual is directly correlated to the horizontal distance traveled multiplied by the number of connections broken between the modules to the power 0.8 and can be derived from Equation 6.1.

\[
F = \sqrt{(p_e x - p_1 x)^2 + (p_e y - p_1 y)^2} \ast \eta^{0.8}
\]

(6.1)

Where \(F\) represents the fitness value obtained by calculating the eventual position \((p_e)\) minus the position after 2.5 seconds \((p_1)\) traveled in both x and y directions. \(\eta\) represents the number of broken module connections of the morphology after 20 seconds of simulation time.

A simulation environment consisted of a default floor and was simulated using the bullet dynamics engine (version 2.78). The dynamics settings were set to accurate (default) with a time-step of 50ms. Six experiments were done comparing the different encodings. Three of the experiments ran twelve evolutionary runs whereby a maximum of 5, 10 or 20 servo modules and one cube module were allowed. These three experiments were done to see how the direct encoding performed. The other three experiments analyzed the efficiency of the generative encoding and was also composed of twelve evolutionary runs simulating a maximum of 5, 10 or 20 modules. The runs are limited to a fixed number of evaluations. In the simulations that could simulate a maximum of five modules, 12,500 evaluations were done. The other runs were limited to 25,000 evaluations; more evaluations were performed in these runs since the search space is larger when increasing the number of simulated modules. 25,000 evaluations were chosen as a trade-off between performance and computational requirements. Since a high-end physics simulator is used, the computational requirements are considerable. The next sections will cover the direct and indirect encodings in more detail.
6.2. Methodology

6.2.2 Direct Encoding

The ‘Evolutionary designer of heterogeneous modular robots’ (Edhmor; Faiña et al. 2011; Faiña et al. 2013) system is used as the direct encoding strategy to assemble and evaluate robot morphologies. The Edhmor system is organized as a tree representation, where nodes represent control parameters of a module and its type and edges represent how a module is attached to a parent module. The direct encoding is used together with a constructive algorithm. This algorithm starts building a random population of robots with just a few modules. Afterwards, different mutation phases are applied cyclically. The mutation phases of the algorithm are:

- **Add Module**: Add a module into a morphology.
- **Mutate morphology**: Change the orientations or the place where some modules are connected
- **Mutate control**: Change the control parameters of some modules
- **Prune robot**: Test all the morphologies generated by removing a module and its children.

In every phase, a mutation operator is applied several times to produce different random mutations of the same individual which are tested in the simulator. For example, when adding a new module to a robot, five different robots are generated and each of them have a new module placed in different positions and orientations. These phases revert to the previous robot if the mutation does not increase the fitness of the robot, except in the add module phase. This phase forces morphological evolution to take place which has been shown to be advantageous when evolving virtual creatures (Cheney et al., 2016).

The evolutionary algorithm of Edhmor is furthermore generational, the 10% worst performing robots are removed from the population every cycle. Half of them are replaced by random robots with a low number of modules, the other half is generated by applying symmetry operators to the best robots. The symbolic representation and its phenotype are depicted in Figure 6.2. A more detailed overview of the system can be found in Faiña et al. (2013).

6.2.3 Generative Encoding

The implemented generative encoding is based on a context sensitive Lindenmayer-System (L-System; Lindenmayer 1968a; Lindenmayer et al. 1992) – a parallel rewriting system. The variables used in the L-System represent the modules employed to construct a robot (Figure 6.3), similar to the L-System used Chapter 5 and as described in
Chapter 6. Comparing Encodings for Evolving Locomotion in Modular Robots

**Chapter 4.** The generative encoding was limited to using five different module states. The first state (the axiom) represents the cube module and the four other states represent the servo module. The four states that represent a servo module encode for the same module but can differ in their mutable parameters responsible for the sinusoidal function that controls the servo module. The attachment rules of the cube module included the information of which module is connected to what connection site and in which orientation. The same attachment rules are possible in the servo modules but the servo modules only contain three attachment sites. The implemented attachment rules are in essence similar to the rewriting rules of a normal context sensitive L-System (Lindenmayer et al., 1992). It is context sensitive since a module cannot be placed at an attachment site if another module is already occupying it. Furthermore, modules cannot be created if this causes a collision with other created modules.

The internal sinusoidal function that controlled the PI controller of the modules could be mutated in the genome of the module states. This means that the robot can actually not have more than four distinct sinusoidal controllers. For illustrating the different object states, they are colored in the phenotype. The modules could either be red, yellow, blue or pink depending on their state. Four iterations of the L-System were done to create the robot phenotypes starting with the cube module as the axiom.

All parameters of the module states were subject to evolution. There was a 15% chance of a morphological parameter to be mutated and a 5% chance of a control parameter to be mutated. A symmetry mutation operator enabled an object state to arise at the opposite site of a module where it originally was expressed. Though symmetry is an inherent trait to an L-System, the symmetry operator enhanced the probability of creating symmetrical phenotypes. Since the genome of an individual is represented by different module states, a crossover operator enabled different states to be exchanged between individuals. The crossover function had a 20% chance that a module state of an individual came from a different individual than its original parent.

### 6.3 Results

The results of the different evolutionary runs were divided in a performance analysis and a phenotype analysis. The performance analysis was done to get a clear insight in the efficiency of the encodings. Knowing a bit of what type of phenotypes resulted

---

**Figure 6.3:** Representation of the generative encoding. (left) The L-System parameters form the genotype of the morphology whereby the variables of L-Systems are replaced by module states. The '+' constant represents the placing of the next module at the specific attachment sites of a module. The symbolic representation of the genotype (middle) serves as a visualization on how the genotype constructs the phenotype (right).
from the evolutionary runs gives us more insight in what prominent evolved characteristics where and how we can ultimately improve the simulator for the design of actual modular robots.

### 6.3.1 Performance analysis

As can be seen in Figure 6.5, the average fitness values – as well as the averages of the maximum fitness values – of the evolutionary runs is quite different per encoding. The generative encoding seems to be able to quickly find decent behaviors that are rewarded with a high fitness value. A Mann-Whitney U test has been performed at specified intervals to check whether the encodings performed significantly different. The performance difference was measured using the average fitness values of the maximum fitness of each individual evolutionary run at a specified time. The test resulted in a significant difference between evolved populations after 6,250 evaluations (p-value: 0.000612) and 12,500 evaluations (p-value: 0.003674) when simulating a maximum of 5 modules. There was also a significant difference between the two encodings at 6,250 evaluations (p-value: 0.00328), not at 12,500 evaluations (p-value: 0.012410) but again at 25,000 evaluations (p-value: 0.001617) when evolving a maximum of 10 modules. The runs of the simulation evolving a maximum of 20 modules was also statistically different at evaluation 6,250 (p-value: 0.00332) but not at evaluation 12,500 (p-value: 0.0124106) and also not at evaluation 25,000 (p-value: 0.209462). The maximum and average fitness values of the individual runs can be seen in Figure 6.4.

### 6.3.2 Phenotypes

Different distinct phenotypic behaviors emerged after a certain amount of evolutionary time. The direct encoding evolved various kinds of strategies though the generative encoding had evolved more simple, distinct types of locomotion due to the similarity in behavior seen in several modules. Caterpillar like behavior could be seen evolved conglomerates that were composed of a single chain of modules (Figure 6.7b). A single chain of modules could also result in a different type of rolling locomotion as seen in Figure 6.6a and Figure 6.6b. For some evolved robots there was no apparent logic to how they moved. Two robots tossed their weight around which resulted in complex rolling (Figure 6.6c, 6.7c and 6.7d) and one robot performed a crawling (Figure 6.7e) behavior. The types of behavior should become more evident when consulting the supplied video.

The constructive strategy of the direct encoding tends to to add modules to the robot. This results in the best individual of all the different evolutionary runs to be composed of 5 modules when simulating a maximum of 5 modules. In the case that the maximum number of modules is 10, 8 out of 12 runs have reached the maximum number of modules and the average is 9 modules. When 20 modules are allowed, the average is 12.33 with a standard deviation of 2.87. In this experiment, the number of modules are limited by the fact that the excess of torque breaks the connections between the modules, which are heavily penalized by the fitness function.

All the robots with a maximum of five modules developed similar morphologies, linear structures, with a rolling behavior. One of them is shown in Figure 6.6a. With a limit of 10 modules, branches in the structure of the robots appear. Despite that the rolling behavior is still predominant, a crawling behavior can be found in some individuals (Figure 6.6b). When increasing the maximum number of modules to 20, some unspecified conglomerates of modules are found but most of the behaviors roll or crawl as in Figure 6.6c.

In the generative encoding there was a recurrence of simple friction-based phenotype (Figure 6.7a) when simulating a maximum of 5 modules. This friction-based phenotype seemed to exploit friction parameters of the simulator. Exactly half of the evolutionary runs that allowed for a maximum of 5 servo modules stagnated in a local optimum with this kind of phenotype. This phenotype was moreover also found...
Figure 6.4: Evolutionary progression of individual runs. The graphs represent the individual runs done for each experiment. The six graphs represent the direct encoding simulating a maximum of 5 servo modules (a); generative encoding simulating a maximum of 5 modules (b); direct encoding simulating a maximum of 10 servo modules (c); generative encoding simulating a maximum of 10 modules (d); direct encoding simulating a maximum of 20 servo modules (e), generative encoding simulating a maximum of 20 modules (f). The bold black line represents the average maximum fitness values for all runs while the black dotted line represents the average of the average fitness values of all runs. The colored lines represent individual runs, where the solid line represents the maximum fitness value of the population and the dotted line the average fitness of the population.
6.3. Results

**Figure 6.5:** Evolutionary progressions of the direct and generative encoding. The graphs display the average maximum fitness values of the different evolutionary runs when simulating a maximum of 5 modules (a), 10 modules (b) and 20 modules (c). The solid blue line marked with circles represents the average maximum fitness value of all the runs of the generative encoding. The red solid line marked with triangles represents the average maximum fitness values of the direct encoding. The dotted lines represent the median of the two types of encodings. The thick error bars depict the 25-75 percentiles and the thin error bars depict the 0-100 percentiles.

When simulating a maximum of 10 or 20 servo modules (Figure 6.5), the fitness values of these individuals were quite low while other simple, more effective morphologies, such as the phenotype shown in Figure 6.7b, were possible to evolve. Considering the number of modules of the resulting phenotypes, the number of modules present in all robots was considerably smaller in the generative encoding compared to the direct encoding. The average number of modules in the best evolved individuals of all runs was 3.5, 6.83 and 8.615 for the runs allowing 5, 10 and 20 modules max respectively. Five out of 12 runs, when simulating a maximum of 20 modules, led to the evolution of morphologies composed of more than 10 modules; 10 out of twelve led to the use of more than 5 modules. Seven out of 12 runs simulating a maximum of 10 modules led to the evolution of using more than 5 modules.

In the generative encoding, not all genes of module states are represented in the evolved phenotypes. On the contrary, it seems that the evolutionary algorithm actively selects against the use of more modules. Out of all the evolutionary runs, the runs with a maximum of 5 modules only evolved phenotypes with on average 1.33 expressed servo module states. The runs simulating a maximum of 10 modules had an average of 2.66 expressed servo module states and the runs of simulating a maximum of 20 modules had on average 2.23 expressed servo module states. The phenotypes seen in
Figure 6.6: **Phenotypes obtained through the direct encoding.** Resulting phenotypes simulated with a maximum of 5 modules (a), 10 modules (b) and 20 modules (c). For a more detailed visualization of the phenotypes, see the video at [https://www.youtube.com/watch?v=HCDftic1AdA](https://www.youtube.com/watch?v=HCDftic1AdA)

Figure 6.7d and Figure 6.7e are examples of large phenotypes using only two types of servo module states.

### 6.4 Discussion

As can be derived from the graphs (Figure 6.4 and Figure 6.5) there is a difference in performance between the generative and direct encoding. The most striking difference in performance can be seen in the initial phase of the generative encoding where it outperforms the direct encoding. Over time, the direct encoding was able to catch up with the generative encoding and the performance differences were no longer statistically significant. The generative encoding still had an advantage in the long run when only a maximum of 5 modules could be simulated. This result was counter intuitive since it was expected that the generative encoding would perform better in the long run when more modules could be simulated.

A smaller portion of the genome of the generative encoding can lead to modular robots containing more modules than the direct encoding. The number of servo module states used in the generative encoding was 2.23 on average in the final evaluations of the evolutionary runs of the generative encoding. Since 4 servo module states could be stored in the genome it is noteworthy to see that not all genetic information is expressed in the phenotype of the generative encoding. This result illustrates the usefulness of reusing the genome for creating modular robot morphologies. Being able to evolve robots with just a few genotypic parameters is furthermore an advantage and might lead to discovering abstract recursive mechanisms that are useful for the specified objective.

All evolutionary runs of the direct encodings led to phenotypes that utilized more modules compared to the generative encoding. This is due to a strong pressure in the direct encoding for adding new modules to the existing morphology. The mutations in the generative encoding can lead to destructive genotypes more quickly potentially posing a limiting factor to the number of modules simulated for the individuals. Although the generative encoding outperformed the direct encoding, the generative encoding was still prone to premature convergence. This premature convergence was
Figure 6.7: Phenotypes acquired through the generative encoding. Resulting phenotypes simulated with a maximum of five modules (A;B), ten modules (C) and 20 modules (D;E). For a more detailed visualization of the phenotypes, see the video at https://www.youtube.com/watch?v=HCDfticiAdA.

Not seen in the direct encoding due to other evolutionary parameters that were used in the encoding. An improved version of the evolutionary algorithm could implement methods to increase diversity and evolvability as done in speciation (Cook, 1906) – implemented in Neuroevolution of Augmenting Topologies (NEAT; Stanley et al., 2002a) –, novelty search (Lehman et al., 2008) or Age Layered Population Structure (ALPS; Hornby, 2006; Hornby, 2009). Regarding the L-System, an alternative generative encoding, such as a Compositional Pattern Producing Network (Stanley, 2007), can be a relevant alternative generative encoding for evolving modular robots (as applied in Auerbach et al., 2015). Though as a subjective interpretation, the L-System did evolve robots that seem to exhibit more generic behaviors that might well aid in their transfer to the real world (Figure 6.5).

Albeit out of the scope of this chapter, the presented data is of limited use for robotic applications since it is not known how well the evolved behaviors transfer to reality. However, I expect that a hybrid approach of the two encodings would be a useful strategy to cope with the reality gap. The generative encoding can be used to evolve the global morphology and control of the robot while the direct encoding would tweak
Text Box 6.5: Generic Phenotypes

When considering Figure 6.6 and Figure 6.7, the generative encoding, to its more generic behavior are sometimes easier to see how they work. Figure 6.8 depicts two more phenotypes from which one might know how the first moves by simply looking at the images. This is, although I am biased, a lot more difficult to imagine for the bottom robot created by the direct encoding. Though the behaviors are best displayed in the video.

Curiously, the phenotype frequently evolved as depicted in Figure 6.7a, considerably altered the performance of the generative encoding when being able to simulate up to 20 modules as seen in Figure 6.9. Being unable to evolve a new morphology was detrimental to the overall performance difference between the encodings that might have well changed the statistical significance if it did not evolve this phenotype. This is a major constraint of premature convergence using the L-System that should be addressed in future implementations.

Figure 6.8: Illustration of evolved phenotypes. Evolved morphology using the generative encoding (top) and the direct encoding (bottom)

Figure 6.9: Prematurely converged phenotype The figure illustrates the low fitness solution at the end of the evolutionary run and its corresponding phenotype.
6.5 Conclusion

morphic and control parameters online or in a feedback loop with the simulator. This would be beneficial since the generative encoding cannot locally change parameters specific to individual modules. Nonetheless, it might also be better to evolve phenotypes using the generative encoding and have an online learning system – such as a form of local decentralized learning (Christensen et al., 2013) – adjust the control of the modules accordingly.

The presented semi-homogeneous modular robot system presents a promising step in the direction of evolving feasible modular robots. Increasing the heterogeneity in the system would give us additional insight in how the modules can be better modeled in the future to produce even better robots. One could think of applying additional structural modules that have a variable stiffness. Since many organisms exploit various biomechanical attributes – be it elasticity, friction, strength – adding this type of module can enable evolution to come up with morphological solutions (Pfeifer et al., 2005) and reduces the need for every part of the robot morphology to be actuated. Additionally, sensory modules can be implemented to extend the functionality of the system giving the robot inputs to its control system. The products of evolution of these potentially evolved heterogeneous modular robots can become experimental platforms that can be consulted before designing and building a non-modular equivalent.

6.5 Conclusion

Much work in evolutionary robotics is devoted to brain-body optimization strategies though few studies consider the transferability of the evolved morphologies and control systems. This approach tries to decrease this gap and enable researchers to have a fast way of evolving and evaluating robots in simulation and reality. The presented robotic platform that evolves conglomerates of modules showed that a generative encoding, despite, having less optimization freedom, is more effective for evolving locomotion in simulated robots. The reuse of genes in the generative encoding seems to work well for the evolution of robot morphologies and control. This is a great advantage when constructing a robot out of many modules since many of them can be assigned with the same control parameters. To facilitate both exploration and exploitation, I conceive that the generative encoding is able to evolve more abstract and simple robots and suspect that a hybrid system would be ideal for experimenting with the reality gap of the evolved robots. This hybrid system can initially use a generative encoding in simulation followed up by a direct encoding that locally optimizes parameters in a real robot.
Chapter 7

Toward Energy Autonomy through Evolving Plant-Inspired Modular Robots

I, on the other hand, am a finished product. I absorb electrical energy directly and utilize it with an almost one hundred percent efficiency. I am composed of strong metal, am continuously conscious, and can stand extremes of environment easily.

– Isaac Asimov, I, Robot

As stated in Chapter 5, plants as primary energy consumers are the initial source of energy that accumulates in higher order terrestrial organism. This chapter explores how to design a robotic system with a similar objective as plants in a potential artificial ecosystem to act as primary energy producers. In this case, the robotics simulator as implemented in Chapter 6 is used and the solar panel modules as described in Section 4.4 were implemented. The experiments convey how modular robots can be optimized for energy autonomy and describes the potential applications of solar panel modules in robotics. The current modular robots make use of energy intensive servo motors, but these can potentially be replaced with other less energy demanding structural modules. This chapter will have a more abstract hypothesis since there isn’t much comparable preliminary work that used solar panels to evolve robots for optimal energy consumption. Hence, the title of this chapter begets the use of the word ‘Toward’. The hypothesis that was formulated to guide this chapter was therefore:

**Hypothesis 4** Energy autonomy in modular robots can emerge from implementing solar panel modules.

### 7.1 Introduction

Energy autonomy in artificial systems is beneficial for long-term autonomous behavior in single or multi-robot applications required for e.g. monitoring and exploration. However, it can be complicated to design energy autonomous systems since this depends on the energy demand and energy acquisition of the robot. In evolutionary robotics, locomotion and object manipulation are among the most prominent objectives for robots (Vargas et al., 2014). Though the same principles in evolutionary robotics can be implemented for energy autonomy in robotic systems. Energy autonomy is usually implemented on different robotic systems where the robot can to utilize energy from light (Noth et al., 2006; Afarulrazi et al., 2011), or microbial fuel cells (Ieropoulos et al., 2003; Philamore et al., 2015). Being able to automatically design robotic systems that are geared towards energy autonomy could give us unintuitive solutions that might be more effective than traditional solutions. Since plants have mastered extracting energy from light in terrestrial environments, they are taken as a source of inspiration. Although plants have many unique features, we do not consider the implementation of many of them since its either impractical or infeasible to implement in robotic systems.
Chapter 7. Toward Energy Autonomy through Evolving Plant-Inspired Modular Robots

However, an abstraction of plant development is implemented in the form of a generative encoding (Subsection 7.2.3). Having a modular robotic system that conforms to energy optimization could give rise to self-reconfigurable robots that maximize energy acquisition.

The ultimate mechanisms that drive evolution in organisms is enacted through the external energy influx into an open system that innately works against its thermodynamic equilibrium. Considering the earth as an open system, this energy influx is mainly acquired from the sun in the form of light and to a lesser degree in the form of heat and chemicals from earth. Being able to acquire this energy from the sun to generate complex organisms is thus a vital for evolution. The initial multicellular organisms that roamed the planet acquired energy from their environment in the form of chemicals and light (Reece et al., 2010). These carbon-based life forms were the precursors to plants which became the expert terrestrial organisms for acquiring energy from light. Although plants appear to be slow, they are highly optimized for gathering energy from their environment. Many plants contain track and actuation mechanisms that optimizes their productivity by absorbing light more efficiently (Ehleringer et al., 1980). This tracking behavior emerges in the form of circumnation through heliotropism (Graham et al., 2013). In phototropism, a form of heliotropism, the plant actively grows towards a light source as in the case of the sunflower (Atamian et al., 2016; Kutschera et al., 2016). Some other species of plants can adjust their leaves or flowers to bend towards the light with a structure called the pulvinus. The pulvinus is a structure below the leaves and flowers that twists and pivots them through adjusting turgor pressure (Song et al., 2014). In general, leaves adjust their angle and move their surface perpendicular to the sun when conditions are optimal in a process called diahlotropism. These are some general adaptive mechanisms through which a plant can adapt their morphology based on specific stimuli. Other morphological traits are hard-coded in the genome such as phyllotaxis (Cells, 1997), the arrangement of leaves on a plant. Since the number of solar modules used in the robotic system is limited, as will become apparent in the next sections, no fair comparison can be made with the complex forms of phyllotaxis seen in plants. However, phyllotaxis is mainly driven by innate factors requiring limited feedback from the environment which is similar to the open-loop control implemented in the generative encoding and evolutionary algorithm.

A mechanism exactly like phototropism is a feat that we are unable to implement in the robot since this would require some form of growth. This could however be accomplished with soft robots (Sinibaldi et al., 2014; Sadeghi et al., 2017; Heinrich et al., 2016; Vergara et al., 2017). Although allowing for continual growth show promising bio-inspired applications in robotic systems (Sadeghi et al., 2014), this also brings forth difficulties regarding the reconfigurability and reuse of robotic parts. The modular robotics approach therefore doesn’t allow continual growth but enables reuse and reconfigurability. The bio-inspiration of the implemented robotic platform is motivated by the rotational movements of the pulvinus since this structure allows for movement of the leaves without major morphological change. With an open-loop control system, simple control mechanisms that allow for the optimal energy absorption. While artificial plant systems have been implemented in cellular automata (Hogeweg, 1988; Balzter et al., 1998) as well as virtual creatures (Zamuda et al., 2014; Zahadat et al., 2016; Veenstra et al., 2016; Corucci et al., 2016), they have rarely been investigated in a three-dimensional embodied approach other than light-tracking solar panels (Prinsloo et al., 2015). Many evolutionary robotics experiments have focused on acquiring behavior typical of consumers (Sims, 1994b; Peifer et al., 2006; Vargas et al., 2014), whereas this chapter looks at how primary energy producers can evolve in artificial systems. Some robotic platforms have been designed to cope with energy autonomy (Greenman et al., 2003; Ieropoulos et al., 2003; Philamore et al., 2015) although research in this area is still limited. The evolutionary system, instead of modeling nature, aims at implementing feasible evolved designs into real modular robots. The modules are therefore based on a heterogeneous modular design. A robotic module being an independent unit that encapsulates part of its functionality (Stoy et al., 2010). The modular robotics approach eases the construction process of different morphologies as well as the capability of changing the morphologies on-the-spot. By simply implementing the same
connection mechanism on each module of the modular robot, different modules can be joined together to form unique and feasible robot phenotypes. The platform thus allows for light-absorption in a robotic system that implements actuator with similar degrees of freedom as the pulvinus with the aid of a plant-inspired developmental algorithm.

7.2 Material and Methods

A simulation environment is used to evolve modular robots and the evolved phenotypes are transferred to the real world. The simulated robots were optimized for harvesting energy from light using simulated solar panels. The physical robot simply follows the parameters that have been evolved by the simulator. Different environments were simulated that in turn shaped the search space of the robot. Moreover, five different types of modules were designed that could be simulated. These modules could all be connected using the same connection mechanisms. Enabling morphological change and open-loop evolution of the control system allows the platform to evolve unique robot phenotypes that can be transferred to reality. The simulated robots could never use more modules of a type than was maximally allowed. Every evolved robot could thus be created in reality. Varying the number of solar panels that we can implement in the robotic system gave us an idea on the number of modules we require to evolve sufficient robotic phenotypes. A modular robot was constructed by attaching one module to another module manually. Since the connection mechanism is based on magnets, the construction process is a simple snap-on procedure.

Virtual Robot Experimentation Platform (V-REP; version 3.4.0) (Rohmer et al., 2013) is used as the robotics simulation platform and the evolutionary algorithm was implemented as a C++ based DLL plugin. The simulation environment consisted of a default floor and was simulated using the bullet dynamics engine (version 2.78). The dynamics settings were set to accurate (default) with a time-step of 50ms. The modules used for the simulator are based on the physical properties of the real modules (Figure 7.1) approximating similarity in size and weight. The modules consisted of a base module, a servo module, a cube module and two types of solar modules as will be discussed in the next section. All modules contained connection sites that could either be male or female. The pairing of these two connection sites established a connection between two modules allowing for the composition of a modular robot (Subsection 7.2.2). A list of the materials used to construct these methods can be found on our website.

7.2.1 Modules

Five different types of modules were used: a cube module (Figure 7.1a), a base module (Figure 7.1b), a servo module (Figure 7.1c) and two types of solar modules. One of the solar modules simply contained two solar panels that were joined together (Figure 7.1d). The other, more elaborative, solar panel module (flower module) was designed with the aim of allowing for more plant like adaptive behavior in the system (Figure 7.1e). However, the increased complexity of this flower module also brings about increased complexity in the simulator. Therefore, this flower module is not used in the experiments described in this chapter but rather serves the role of informing the reader about the potential future implementations of solar modules. Throughout different modules, all custom parts were 3D printed using polylactic acid (PLA).

The base module (Figure 7.1b) is composed of a simple custom structure with three female connection sites. Three female connection sites were used to reduce the number of possible connection sites limiting the search space of the robot. However, more connection sites could make more sense in a future implementation of the base module though this makes the state space landscape more convoluted due to more possible configurations of the modules. This base module was simulated statically meaning that its physical properties were not simulated. Instead, the module was fixed in place

---

1List of Materials can be found on our website at https://sites.google.com/view/emergemodular/projects/energy-autonomy
ensuring that the structure stays in the same position conforming to the sessile nature of plants. The three female connection sites were 55 by 55 mm and were placed next to each other at a 45° angle. The cube module (55 mm by 55 mm by 55 mm; weight is 300 g) is used as a structural building block for the modular robot to which other modules are attached. This cube has five female connection sites (top, right, left, front, back) and one male connection site (bottom).

The servo module (80 mm by 55 mm by 55 mm, weight is 160 g) is based on a module from the EMERGE modular robotic platform (Moreno et al., 2017). This module contains three female connection sites, attached to the shaft, and one male connection site on the bottom attached to the chassis of the servo motor. It houses a Dynamixel AX-18a servo motor. In the experiments the range of the servo motors is limited to +90° and -90°. The difference of the servo module used in this chapter is the attribute of having 4 data channels compared to the 2 data channels implemented in the EMERGE module. Also, infrared sensors that were present in the EMERGE module are not implemented in the servo modules used. Infrared sensors were not implemented since they take up additional space on the PCBs and consume additional energy. At the location of the infrared sensor in the EMERGE module there is simply a 3 mm wide hole that can be used to adjust a screw to tighten the connection between the servo bracket and the servo motor. This tightening can in turn lead to a stiffer joint that is harder to actuate but also harder to move passively. This is beneficial to adjust on the spot since we don’t

**Figure 7.1:** The modules used in the simulated and physical robot. The cube module (A), base module (B), servo module (C), solar panel module (D) and flower module (E).
want an inactive servo joint to buckle under the weight of the robot. Servo motors closer to the base of the robot can therefore be tightened more while those at the end can be looser. The dynamixel AX-18a servo motor is controlled via a PID controller that is directly connected to the robotics simulator.

The first type of solar panel module (referred to as solar module) implemented two simple solar panels (141mm by 124mm by 6mm, weight is 66g) on a 3D printed bracket (Figure 7.1d). Two 0.8W, 5.5V monocrystalline silicon solar panels (118mm by 63mm) were mounted on the case of the solar module. These two solar panels were connected in series and could output a voltage of 11V (Figure 7.2a). An additional diode was attached to the solar module to prevent current flowing in the opposite direction. The current acquired from the solar panels was converted to 12V using a step-up voltage regulator (not shown in the figures) to ensure a 12V output. This output voltage was connected to two data channels connected to the base module. The solar module was simulated with a basic morphology with similar dimensions and contained five proximity sensors to determine light absorption. These proximity sensors were positioned at each corner and in the middle of the solar module. Each proximity sensor served as a ray tracer simply determining whether there was an object in between the starting position of the ray and a light source. Additionally, the difference between the z directional vector of the ray compared to the orientation of the solar panel gives us the angle at which light impacts the solar panel. This angle of attack linearly influenced the contribution of light absorption to the fitness of an individual.

The flower module (Figure 7.1e) is composed of five SP3-37 flexible solar panels. Each panel is mounted on a 3D printed petal. These artificial petals could not bend making the use of the flexibility in the solar panels redundant. The flexible solar panels thus effectively function the same as a regular solar panel. The five petals are connected together through a system of cranks and a small circular platform. A rack and pinion system combined with a small servomotor pushes the platform and actuates the petals. Two hinge joints connect the circular platform to the petals, one at the inner edge of a petal (center of the flower) and closer to the middle of the petal. A hinge joint on the edge of the petals is attached to the platform containing the rack while the hinge joint closer to the middle of the petal is connected to the piston. When the piston moves the rack up and down, the petals actuate and open or close the flower respectively. The petals can open to a certain degree and this position can be optimized depending on the solar intake. The energy harvested is stored in a LiPo battery. This battery was used to power the servomotor actuating the petals making the flower module (not the modular robot) energy autonomous. Though the flower module is not used in the evolutionary runs, it can also be connected to the same power grid of the modular robot to charge the 12V battery. Additionally, the flower module contained a MMA842Q accelerometer (3-axis), which enables the flower module to detect movement. This sense of movement can inform the flower module whether it should close or open its petals under harsh or favorable conditions. For example, at night, during heavy rain or in windy scenarios, it would be better for the flower module to close and protect its solar panels. Especially if the solar panels can unfold in an origami like manner (the original intend of implementing the flexible solar panels), the surface area would be relatively weak highlighting the need for the flower and solar panels to close. The flower module contained five solar panels and five shapes were used to simulate them. Similar to the solar module, each solar panel simulated five proximity sensors that were used to measure the light absorption. This led to the increased computational requirements since 25 proximity sensors and six shapes were used in the flower module. Therefore, the flower module was excluded in the robot simulator.

One base module, one cube module, eight servo modules, five solar modules and two flower modules were created in reality and the simulator was thus restricted to use this number of modules. This limits the potential resulting phenotypes of the evolutionary algorithm and constrains the search space but enables evolution of feasible modular robots. A direct feedback from simulated modular robots and actual modular robots is thus present. All the morphological parameters of the individual modules
Figure 7.2: Schematic diagram of the modular robot containing solar panels. The solar panel module consists of two solar panels that are connected to one another in series. A diode is attached to prevent current from flowing in the reverse direction. As can be seen in 7.2b, the solar panel modules pass power down to the base module where the battery is located.

were fixed to represent their physical counterparts. The eventual physical modular robot was constructed based on the phenotypes that evolved in the simulation environment. An implementation of the dynamixel protocol (protocol used to communicate with the servo motors) in the plugin allowed for the direct control of the actual robot in the simulation environment. The dynamixel protocol utilizes serial communication to communicate with the dynamixel AX-18a servo motors. Through sending instruction packets individual motors can be controlled using a single communication bus. The values of the dynamixel servos needed to be transformed into hexadecimal values that indicate parameters such as the desired speed and position of the servo motors. The solar modules implemented in the evolutionary algorithm did not have any type of actuation. Connecting a few modules together to form a robotic entity enabled current to flow from a power source towards the servo modules while also allowing for the flow of current from the solar panels to the power source (Figure 7.2b). The solar modules could directly charge a 12V lithium ion battery that was used though the efficiency of charging the batteries greatly depended on the light saturation of the solar panels.

7.2.2 Connectivity

The real modules are connected to one another via magnet-based connection sites as was presented in Figure 4.6. Two of the data channels implemented in the PCBs were used to enable a current to flow from solar panels back towards the initial power source. 3D printed hulls housed four cylindrical NdFeB (neodymium, iron and boron) magnets (12mm diameter, 3mm depth). Two magnets can hold one another together with a force of roughly 13.5N. The male connector hull has protrusions that ensure the fit and connection of the male connector to the female connector. The connectivity of the modules in the robotic system is depicted in a block-diagram (Figure 7.3). Two of the channels were dedicated to power and ground while the four other channels routed data channels. Two of these data channels were used to enable a current to flow from solar panels back towards the initial power source.

In the simulated modules, a force sensor was put in between two connection sites to detect the torque and force between two modules. Since the modules have never disconnected in the physical robot yet, the modules could only break from one another if 100 consecutive threshold violations occurred where the threshold was set to $10\,000\,\text{N m}$ of torque and $80\,000.0\,\text{N}$ of force. The chosen values are incredibly high to ensure that the modules stay connected to one another except in the case of a faulty
7.2 Material and Methods

Figure 7.3: Block diagram of the modular robotics platform. A terminal is connected to a central 3-pin power hub via a USB2AX interface. A 12V power supply is also connected to the same 3-pin power hub. The power hub is in turn connected to a male or female connector face of the modular robot. This connection distributes power to all connected modules. The block diagram only shows the Servo and the Solar module. The base and cube module simply transfer all electrical current from their male connection sites to their male connection face and vice versa. Two of the channels are connected separately to the solar panel modules which transfer power from the solar panels to the power source. The red and black connections indicate power and ground while the green connection represents the communication wire. The dotted lines represent the power and ground connections to the solar panels.

collision. When the simulated robot simulated modules that inaccurately collided, disconnection between modules could occur. These inaccuracies did not occur in the eventual simulator. As a failsafe, the fitness value of individuals that contained broken force sensors was set to zero. The construction of the modular robot occurred before the simulation started. A generative encoding (Subsection 7.2.3) translated the genome and created the robot phenotype. This genome consisted of simple morphological parameters and an additional neural network (Section 7.2.4) for the servo modules.

7.2.3 Generative Encoding

The generative encoding was based on a context sensitive Lindenmayer-System (L-System; Lindenmayer 1968a; Lindenmayer et al. 1992) – a parallel rewriting system – as implemented in (Chapter 6; Veenstra et al. 2017a). The variables of the L-System represented the different modules of the robot. A variable describes a state that is decoded as the morphology, control and attachment rules of the modules. In the simulation environment, these modules represent the cube, base, servo and solar modules. One state was dedicated to the base module and was the axiom of the L-System. Another state represented the cube module. Four states represented the servo module and two
states contained the parameters of the solar module. Every state can assign unique attachment rules and control parameters. The generative encoding does not necessarily lead to all genotypic parameters to be expressed in the phenotype. Instead, some genotypic parameters could be dormant which allowed for genetic drift to occur across generations. E.g. eight servo modules could be generated in the phenotype if one servo state is expressed eight times in the phenotype, but this phenotype could also be the result of one state being expressed six times and one state being expressed two times in the phenotype. The phenotypic representation of the modules is determined by the attachment rules of the L-System that were stored in the genome. The attachment rules of the modules included the information of which module is connected to what connection face in which orientation. The L-System was moreover context sensitive since a module cannot be placed at an attachment site if another module already occupied it. Modules cannot be created if this caused a collision with other created modules. The different modules were colored in the phenotype based on their states so recursive expression could easily be detected. Five iterations of the L-System were done to create the robot phenotypes starting with the base module as the axiom.

7.2.4 Evolutionary Algorithm

The implemented evolutionary algorithm was based on a steady-state genetic algorithm (Syswerda, 1991). The simulations were limited to 46,000 evaluations per evolutionary run. 46,000 evaluations were chosen as a trade-off between computational time and performance. A population size of 92 individuals was used and was simulated for 500 generations. The population size is a multiple of the 23 simulation instances that ran in parallel on a cluster node containing 24 computing cores. One core was dedicated to running the evolutionary algorithm itself while the remaining cores evaluated individuals in V-REP. For each experiment, 12 evolutionary runs were performed with different initial seeds. The evolutionary algorithm was generational and randomly selected one parent from the population to produce an offspring. The initial population consisted of individuals created from randomized genomes. Though the offspring were haploid, a crossover function allowed certain states of another parent to be transferred to the offspring with a 20 percent chance. Up to eight states were simulated. After crossover occurred, the offspring were mutated with a morphological mutation rate of 0.15 and a control mutation rate of 0.1. The morphological mutation accounted for any aspect of the generative encoding replaced by a random value with a 15% chance. Only the four states of the servo modules contained a neural network and the neural network had several mutation operators that could be activated with a 10% chance. There were four mutation operators working on the neural networks: change connectivity that altered the edges between the neural network; add neuron; remove neuron; and change neuron swapping an interneuron for a new interneuron of another type (Section 7.2.4). The maximum amplitude of the servo arm ranged from -90nd +90degrees.

Neural Network

In a previous experiment simple sinusoidal wave functions were implemented to control simulated servo modules (Veenstra et al., 2017a). The same sinusoidal patterns are implemented here though they are implemented in a network of neurons. This artificial neural network, implemented in the servo modules, consists of one input neuron, up to six interneurons and one output neuron. The input neuron is always activated. A recurrent interneuron layer consisted of neurons that simulated a fixed sinusoidal output pattern or a neuron with a binary step function. The phase, amplitude and frequency of the sinusoidal neuron could be altered, and the mutable parameters of the binary step neuron were the threshold value (between -1.0 to 1.0) and the output weight (-1.0 to 1.0). The output neuron simply outputs a value between -1.0 and 1.0 based on its inputs. This value is in turn transformed into a value that represents the absolute position of the servo motor of the module. In this neural network, the connections were not weighted, instead, once a neuron gets activated, all the neurons
7.2. Material and Methods

connected to the activated neuron receive the same output. This was implemented due
to the small size of the neural network (one input and one output) and to limit the se-
arch space. A neural network could be implemented in each state of the servo module.
This means that multiple servo modules could express the same neural network if the
generative encoding created these neural networks from the same gene.

Fitness function

The goal of the simulated robots was to absorb light within five seconds of simulation
time. The simulation time was limited to five seconds since the modules were able to
appropriately adjust the positions of their joints to the light source within the given
simulation time. To approximate an actual day cycle in the physical world depends
on the transformation of the simulation time which can be stretched according to a
given environment. The fitness of each individual was determined by the amount of
light that was absorbed by each ray of all the solar panels present on the robot and can
be derived from Equation 7.1. The gathered light was calculated by subtracting the
z directional vector of the z axis of each ray on the solar panel from the z directional
vector of x axis of the solar panel. An additional cost function was added to represent
an arbitrary energy expenditure of the robot.

\[
F := \rho_i \sum_{i=1}^m - \epsilon_j \sum_{j=1}^n
\]  

(7.1)

Where \( F \) represents the fitness value obtained by calculating the total amount of light
absorption that occurred within the 5 seconds of simulation time. The ":=" operator
represents an update of the fitness value with right hand side at each time step. \( \rho \) re-
presents the amount of energy gathered by each ray \( i \) at every time step. For each servo
module \( j \), the amount of energy \( \epsilon \) used by all servo modules is subtracted. An additio-
nal cost function was added to compensate for modules that were disconnected due to
the breaking of a connection site (not shown in equation). Breaking of modules never
occurred in experiments though the initial fitness cost was implemented to ensure we
didn’t reward malfunctioning robots.

7.2.5 Experimental Setup

The evolutionary runs were performed in four different environments (Figure 7.4). The
first environment contained a light source located directly above the modular robot.
The second environment consisted of four walls surrounding the modular robot con-
straining direct outward growth of the phenotype of the modular robot. The third en-
vironment contained an object that blocked the phenotype from receiving direct light
absorption. This was done in to motivate outward growth of the modular phenotype.
The last environment consisted of a moving light source that mimicked the trajectory
of the sun in winter in the northern hemisphere of earth. It was expected that the last
environment would promote the evolution of blind control systems that enabled solar
panels to tilt towards the sun. The position of the light source was calculated by two
sinusoidal functions that were transformed into cartesian coordinates (Equation 7.2).
The ":=" operator represents an update of the left hand side variable with the term on
the right hand side as it is performed at each time step.

\[
P_x := P_{x,\text{start}} \alpha_x + \tau \alpha_x \sin(\tau) \alpha_y \quad P_y := \sin(\tau) \alpha_y \quad P_z := \sin(\tau) \alpha_z
\]  

(7.2)

Where \( \alpha \) represents a specific scaling factor for transforming the position in carte-
sian coordinates. The x y and z positions of the light source are updated at each time
step denoted with \( \tau \).

In the environment with the moving light source, the energy cost was a custom
value. This cost value was either 0.0, 0.1, 0.5, 2.0 or 8.0. For every energy cost, 12 evo-
lutionary runs were done. These values were chosen as they changed the evolutionary
trajectory of the different evolutionary runs whereas above a cost of 8.0, no change
Chapter 7. Toward Energy Autonomy through Evolving Plant-Inspired Modular Robots

Figure 7.4: Different simulated environments. Environment with stationary light source (top left); an environment where the initial base module is constrained (top middle); an environment where direct light is blocked by an object (top right); and an environment where the light source is moving in an arc (bottom). The line represents an approximation of the path the light takes.

In the evolutionary runs could be seen. Additionally, for each simulated environment twelve evolutionary runs were performed when simulating a maximum of one, two or five solar modules. This is done to see if movement evolves in environments where a different number of solar panels can be simulated. It is expected that when simulating a maximum of one solar panel, on average more angular movement can be detected in the solar panels compared to simulating more solar modules. This hypothesis is tested by measuring the difference in angular movement of each solar panel at each time step. This arbitrary measure of angular movement is stored together with the fitness values of the individuals.

7.3 Results

The results shown in this section are separated in the resulting phenotypes of the eventual population of robots in different environments and the results on the impact of movement and energy cost on evolving the modular robots. The phenotypes give a clear overview of the types of robots that could be evolved while the energy costs explain emergent behaviors seen in the evolved robots.

7.3.1 Phenotypes Evolved in Different Environments

The four types of environments led to major differences in evolved phenotypes (Figure 7.5). The individuals in the environment with the stationary light source evolved simple morphologies where all solar panels are pointing upwards avoiding collision with one another. The individuals in the constrained environment were more difficult to evolve as can be seen in the graph depicting the evolutionary progressions (Figure 7.8). In particular, only three individuals in the last generation of the evolutionary run displayed phenotypes that were different from the top right and bottom right...
Figure 7.5: Different evolved phenotypes. Four individuals of each simulation environment are shown. (A) The top left individuals were evolved in the environment with a stationary light source and no objects in the environment. (B) The top right individuals were evolved in an environment where the surroundings of the initial module were blocked by four adjacent wall structures. (C) The bottom left depicts individuals evolved in the environment where the light source was blocked by an additional structure. (D) The bottom right individuals were evolved in an environment where the light source moved.

individuals seen in Figure 7.5. It can therefore be said that the search space of this environment is a lot more rugged than the search space in the other environments making it harder for the evolutionary algorithm to get out of a local optimum. As expected, the environment where the light source was blocked directly from above, the solar panel modules were located on the outside of the modular robots. Some of these evolved phenotypes were easily implemented in the real world since they didn’t actuate any servo modules (Figure 7.11). As can be seen in Figure 7.7, the eventual population of the modular robots that evolved in the environment with the moving light source were also easy to transfer to the real robot. Eventually, although it is a conceptual model, the modular robot that incorporates the flower modules is depicted in Figure 7.10.

7.3.2 Movement in Energy Harvesting Modular Robots

As could be seen in the simulation environment with the moving light source, movement can increase the fitness value of evolved robots compared to the evolutionary runs if the cost of moving is low enough (Figure 7.9). A statistically significant difference can be seen when comparing the average movement angle of all solar panels when comparing maximum and minimum energy cost. The difference in fitness for a high energy cost versus a low energy cost for movement is highest when simulating one module and lowest when simulating five modules. Based on a Mann-Whitney U test we can see that there is a significant difference in the angular movement of the leaves when a large energy cost is applied when simulating a maximum of 1 (p-value: 0.0000779), 2 (p-value: 0.000300) and 5 (p-value: 0.0050966) solar panel modules. The difference in angular movement between simulating a maximum of 1 and 5 solar modules without a energy cost was also significant (p-value: 0.0030495). Similarly, the difference of angular movement when simulating a maximum of 1 and 5 solar modules was significant with the highest energy cost (p-value: 0.0070689). When comparing
Chapter 7. Toward Energy Autonomy through Evolving Plant-Inspired Modular Robots

7.4 Discussion

The aim of this chapter was to provide a deeper understanding in how evolution of modular robots could shape robotic entities towards energy autonomy in different environments. The resulting phenotypes of each environment indicate that the simulated individuals adjusted differently to optimize for light absorption. Some of the simulations led to solutions in local optima indicating a rugged search space for the evolutionary algorithm, as was the case of the constrained environment. The other environments typically evolved more varied and more complex phenotypes. This demonstrates that our simulator is versatile in creating novel robots using a fixed number of modules in different simulated environments though improvements in the evolutionary algorithm as well as the encoding could be made.

7.4.1 Light Tracking

When movement was costlier and when more solar panels were available to be implemented in the modular robot, it was expected to see a relative decrease in angular movement of the leaves when compared to runs without implementing cost of movement and simulating less solar panels. Our results indicate that when less solar modules were allowed in the evolutionary runs, more movement could on average be seen in the phenotypes that evolved. The reason that this difference occurs is due to the increased search space of simulating more than one solar module as well as the constraining available positions for solar modules when simulating a limited number of modules. Our results indicate that having more than one solar panel does lead to robots tilting their solar panels towards the light source. However, when only one
7.4. Discussion

When simulating five solar modules (A), a maximum of one solar module (B) and a four solar module phenotype that has been transferred to the real world (C). The camera stayed in the same location after each subsequent picture.

Solar panel is being used, the robot evolved more movement and movement has a clear evolutionary advantage when it tilts the solar module towards the light. Similar to Veenstra et al. (2016), evolution does not necessarily pick up movement of solar panels even when there is no cost for movement attributed to it. A significant difference could however be seen when a large cost was implemented for movement in the modular robots. Having more solar modules may be more beneficial than requiring tracking a light source. When robots are constructed in the real world, the actual movement costs and solar uptake of the entire robot can be modeled and used as a feedback mechanism to the simulation environment. In this scenario we could determine if it is better to make small modular robots at different locations only containing a few solar modules, or if it is better to create a single modular robot containing many solar panels.

7.4.2 Challenges

One major challenge of the presented robotic modular platform is the size and weight of the modules. Since the modules are only connected via magnetized connector sites, the structure will become quite heavy which is especially detrimental at the extremities of the modular robot. Modules connected to the base module would have to cope with more force than the modules at the extremities. Similar to plants, the main stem or tree trunk is usually the strongest and heaviest while its branches are lighter and its leaves even lighter than the branches. Unable to distribute weight accordingly is however, a common issue in modular robotics and one solution is to remove redundant connection sites. Another solution is to vary the shape, size and weight of the modules. Such a system might require modules that isolate their mechanical connections and
Chapter 7. Toward Energy Autonomy through Evolving Plant-Inspired Modular Robots

Figure 7.8: Evolutionary runs done in the different environments. The three graphs depict the individual runs where bold solid line represent the average maximum fitness value all evolutionary runs. The colored surrounding area represents the 25-75 percentiles while the less intense colored surrounding area represents the 0-100 percentiles. The evolutionary runs are shown of the environment with a stationary light source (left), the constrained environment (middle) and the indirect light environment (right).

communication. Since every connection face of our modules contains the same number of magnets, each connection between modules has the same strength. A future implementation could allow for modules at the near the base to be connected more forcefully and modules at the extremities be connected with less force.

Only a limited number of generations were simulated in the evolutionary runs due to conversion of the evolutionary algorithm. Improvements to the evolutionary algorithm itself would be valuable to cope with this. These improvements could consist of implementing speciation (Stanley et al., 2002b), novelty search (Lehman et al., 2011) and/or pareto optimization techniques (Schmidt et al., 2011; Brodbeck et al., 2015). However, doing more evolutionary runs might lead to the discovery of better performing individuals by having an initial population that can be exposed to incremental improvements. The demonstrated robotic platform has potential for integrating interactive evolution (Sims, 1992; Graf et al., 1996). A human in the loop could for example design the modular robot by simply connecting the modules to form a robot morphology. The simulator could in turn evolve behaviors for that specific robot’s configuration in the simulation environment similar to Wagy et al. (2015) with potential continuous self-modeling (Bongard et al., 2006). Although evolutionary algorithms were used to generate the control and morphology of the modular robots, online adaptation could be a more beneficial strategy, especially for real world robots. A hybrid approach of initially evolving arbitrary modular robots followed by an online learning strategy could therefore lead to more feasible robots. This would be especially valuable if the adaptive control can be localized and its genetic information could be reused similar to the generative encoding.

The generative encoding abstracts the complexity of the genome and has been shown to work well to quickly acquire decent performance. However, as is common in evolutionary algorithms, some runs stagnate in a local optimum making it harder for the evolutionary algorithm to discover novel strategies. However, the variety of modular robots that evolved in each environment varied a great deal. It is unknown whether doing more evolutionary runs will increase this variety of decent modular robots. The generative encoding implemented might also be improved through using more implementations that have been used as abstractions of development such as compositional pattern producing networks (Stanley, 2007).

The evolutionary system limited the number of usable modules to the number of robotic counterparts in the real-world. Simulating more modules could lead to more effective phenotypes when making. However, simulating less modules could potentially lead to good solutions as well when considering that many modules that have been implemented in the evolved robot didn’t seem to be of any particular advantage.
7.4. Discussion

Figure 7.9: The effects of an additional energy cost. Where a maximum of 1 solar panel (a), 2 solar panels (b) and 5 solar panels (c) were simulated. The black line and depicts the average of the maximum fitness values of 12 individual runs. The lighter gray lines depict the average of the maximum fitness of 12 individual runs where energy cost was applied to the simulation. The lighter gray lines represent higher energy cost. The box plot (d) depicts the average movement of the solar panels in each simulation with different energy costs. Though a lot of disparity could be seen between runs, a trend can be seen that when energy cost is higher, the resulting phenotypes on average move their solar panels less. This distinction can most clearly be seen when maximum 1 solar panel is being simulated (blue). The different colors of the box plot represent the different number of maximum solar panel modules that could be generated.

The implemented modules are also not ideal when optimizing for light harvesting in modular robots. E.g., the servo modules contained a hinge like joint although it might be better to implement a twisting joint in the environment with the moving light source. Moreover, other modules with less energy demanding actuation, or elastic and rigid structural properties, could increase the performance of the evolutionary algorithm to find more suitable solutions for absorbing light from the environment. One possible new module type could be a soft-module similar to (Vergara et al., 2017). If a soft module could be used to inflate multiple chambers, directed movement of solar panel modules towards a light source could be implemented with ease. Such a soft module would be similar to a pulvinus structure seen in many plants. Even the energy gathering system could potentially be combined with modules that gather energy from different sources such as microbial fuel cells. The robotics platform does allow for the effortless integration of new modules in the system – both for the hardware and the software – although explorations of these systems is left for future work.

Four types of environments were used in the evolutionary runs to optimize for light absorption while the fitness function stayed the same across environments. An additional approach could be to evolve a population of individuals initially in one environment and afterwards in more advanced environments which can either be done incrementally (Bongard, 2008) or through encapsulation of behavior in specific evolved environments (Lessin et al., 2013). The light source could also be used as a stimulus...
Chapter 7. Toward Energy Autonomy through Evolving Plant-Inspired Modular Robots

Figure 7.10: Incorporation of the flower module. (left) different design stages of the flower module. (right) An example of an assembled modular robot with flower modules. Note that the servo modules are an older version with one data channel instead of four. Photograph by Phil Ayres.

In mobile modular robots. In this case the evolutionary algorithm could optimize the robot to locomote towards light as a main stimuli that could potentially give rise to modular Braitenberg vehicles (Braitenberg, 1986). Evolving locomotion would add an additional reality gap that was not present in our sessile modular robots. Additional experiments could also include lattices with many solar panels which greatly increase the uptake of light. As seen in the evolutionary runs, modular robotic structures with many solar panels would not necessarily give any freedom for complex behavior to evolve. Moreover, the robotics platform can be used to evolve any arbitrary task that evaluates compositions of robotic modules. If the same connection sites are used for the modules, any type of module could be implemented. For locomotion it would be useful to have additional modules that can act as feet or tendons. The robots evolved for different tasks could then be further evolved towards energy autonomy by allowing solar panels to be implemented in the evolved individuals.

7.4.3 Multi-Robot Systems

Considering food chains in natural ecosystems, organisms higher up the food chain extract the energy that has been harvested by primary energy producers (Reece et al., 2010) simply through consuming and reusing the molecules they produced. Hence most plants and other photosynthetic organisms are known as primary energy producers. Primary, secondary, tertiary, etc., energy consumers subsequently depend on the acquisition of chemical energy that has been created by their prey. The modular approach allows for a potential implementation of a multi-robot system where a robot specialized in absorbing light could share its energy with other robots, potentially making robot energy autonomy viable in environments that are off the electrical grid. Moreover, humanity’s ecological footprint is already unsustainable and in overshoot (Toth et al., 2016). This calls for an approach towards energy autonomy in robots that will not increase our already existing ecological footprint. Instead of relying on renewable energy sources to be implemented in our energy grid, we can imagine a robotic system where robot entities can specialize in gathering energy from the environment while higher order robots can tap into the power grid of these robot entities to recharge
7.4. Discussion

Text Box 7.6: Transferring locomotion from simulation to reality

The modular robot has also been used for other objectives such as locomotion. There is however a severe reality gap when testing the evolved phenotypes in the simulator with reality. Although as Figure 7.11 depicts, the behavior of the real and physical modules is almost identical. As illustrated in this chapter, the modules are both electrically and mechanically connected by the connection sites. However, when the robot performs a strong movement that causes it to forcefully slam on the ground, it can happen that the connections of the spring pins are temporarily or permanently lost. This loss in connection due to mechanical stress could therefore be isolated by isolating the mechanical and electrical connections between the modules.

![Figure 7.11: Simulated and real modular robot behavior. The figures A to F show the behavior of an individual instance of a modular robot optimized for locomotion with the simulator environment in the top right corner of each figure.](image)

Since the tasks of our robotic platform is largely undetermined it is unsure what the eventual robotic entity will look like, how it moves, manipulates objects and gathers information from the environment. Instead of having robotic entities created for specific functions, a range of robotic modules can be implemented that can be optimized for a variety of tasks. In an approach where multiple small modular robots are used to gather energy, an additional modular robot could be used to extract energy from the energy harvesting modular robots simply by connecting a docking module to the solar harvesting modular robot. This docking module could simply be a male connector face that attaches to a female connector face of the energy harvesting robot. This can in turn give rise to an artificial ecosystem whereby some robots are specialized in energy uptake while others can be specialized for different tasks. Such a system could enable symbiotic relationships between modular robots that are optimized to harvest energy and other robots that are able to extract energy from the energy harvesting modular robot. The modular connection mechanisms would allow for the energy to flow directly from one modular robot to another without too much of a hassle. The advantage of energy autonomy in an ecosystem of robots instead of a single robot is that some robots can specialize for very specific tasks not having to worry about spending much energy. This division of labor in multi-robot systems conforms to symbiotic energy
Chapter 7. Toward Energy Autonomy through Evolving Plant-Inspired Modular Robots

exchange that can contribute to the survival of the entire robotic system.

7.5 Conclusion

It has been shown how an evolutionary algorithm can evolve light harvesting robots that can easily be transferred to the real world using a modular robotics approach. Furthermore, there is an advantage for evolutionary algorithms to exploit movement when only one solar panel module is simulated but not when more are simulated given the constrains of the simulator. It has been demonstrated that the robotic platform is able to evolve a variety of modular robots based on the various objectives in the environment. This might prove especially advantageous when evolving a multi-robot system which can form a stepping stone towards artificial ecosystem. This artificial ecosystem can consist of multiple modular robotic entities that interact with one another based on energy. The described platform was limited to the available types of modules and the flower module wasn’t used in the evolutionary algorithms. However, a robotic module containing more complex electronics could simplify evolutionary search when as smart morphology and online learning is applied. All the described experiments and proposed methods can thus lead to an automated design of homeostasis in robots that can potentially scale up to artificial ecosystems.
Part III
Evolving Physical Robots
Chapter 8

Evolutionary Approaches in Physical Robots

Embodiment of robots conveys robots experiencing the world directly, as opposed to simulated robots. Where "the actions of robots have an immediate feedback on the robot’s sensations" (Brooks, 1991). An embodied robot is a real physical robot whose behavior can be observed in the environment (Pfeifer et al., 2006). Though already here, some sort of embodiment can also take place in the simulator since the 'environment' itself is also simulated. Pfeifer et al. (2006) especially advocates that embodiment is a necessary prerequisite for the emergence of intelligence. However, a major problem with the application of evolutionary algorithms in physical systems directly is that the evolutionary process can take a long time to complete (Nolfi et al., 2000) compared to a robotics simulator. Hardware failures are common in the real-world which adds an additional requirement of a robust robotic system when implementing evolution on a robotic system. There are therefore many ways to acquire evolved behaviors in robotic systems. One pathway was described in Chapter 7 by directly building a modular robot based on the evolved simulated phenotype. In this case, due to the sessile nature of the plant-inspired robot, the reality gap was negligible. If the evolved robots would have looked more like the work of Sims (1994b), or the phenotypes displayed in Figure 6.8, the transference of the evolved phenotypes to the real-world would be problematic.

An alternative strategy to circumvent the problem of the reality gap is to directly evolve the robot in the real physical world. This bypass of the simulation environment is especially beneficial for robots that are hard to simulate, e.g. robots that contain many parts, springs or soft structures. This chapter will therefore give a brief overview of evolutionary approaches in physical robots (Section 8.1) and discuss an implementation of the automated assembly of a modular robot (Section 8.2). The latter implementation would allow an autonomous robot platform to automatically evolve and evaluate simulated and real-world robots that differ in their morphology. In Chapter 9 an experiment was done to directly evolve a physical knifefish-inspired soft robot using the strategy.

8.1 Evolving Physical Robots

Different evolutionary robotics approaches can be taken to evolve the morphology or behavior of robots as summarized in Figure 8.1. One of the most direct approaches to evolving robot behavior is through having a terminal run the evolutionary algorithm and either uploading the genomes, or directly control the robot itself. These types of approaches are not very common but are discussed Floreano et al. (1994) and Zykov et al. (2004). Alternatively, an evolutionary algorithm can first evolve simulated robots and afterwards transfer and further evolve the robots further in the physical world (Nolfi et al., 1994; Pollack et al., 2000). In this case, the reality gap is present but reduced by evolving the evolved behaviors for a few generations in reality afterwards. However, this additional evolutionary step might not be needed with the increasing accuracy of simulation environments. Jakobi et al. (1995) for example saw that including noise in a simulation environment makes the evolved simulated robots more
transferrable to reality. Furthermore, this approach relied on the additional simplification of most behaviors to minimize the reality gap (Jakobi, 1998). The reality gap can also be decreased through implementing adaptive traits in the simulated models that can be adjusted in the real robot through for example plasticity in neural networks (Floreano et al., 2001). A feedback loop from the robot and the simulator has even been used to reconstruct the parameters and the dynamics of a physical robot themselves through continuous self-modeling (Bongard et al., 2006). In this case, an evolutionary algorithm can self-adjust its simulation environment based on the model it creates of itself. Even the simulator itself can be evolved based on the acquisition of real data from the environment. Another strategy that has been implemented is called Map-Elites (Mouret et al., 2015). In this approach, elites with different characteristics are transferred to the real-world and a heat map of behavioral traits that are efficiently transferred can be constructed that in turn illuminates the search space of the robot. This map can in turn inform the robot which behaviors are likely to be transferrable compared to others as has been demonstrated in Cully et al. (2015).

Another type of implementation bypassing both the reality gap and the evolutionary algorithm running on a terminal is known as embodied (artificial) evolution. The term was originally proposed by Watson et al. (1999) and embodies the evolutionary process itself in a population of physical robots. As proposed by Eiben et al. (2012), Embodied Artificial Evolution is a specific form of embodiment where (1) physical units are involved as opposed to virtual ones (2) real birth and death is implemented (3) the environment shapes the fitness of an individual (4) individuals decide when and how to mate. This embodied evolutionary approach is perhaps the most cumbersome to achieve, but could in a future with autonomous robots be an emergent property of artificial life; since robots that make robots would also be bound by evolutionary principles in a struggle for artificial life.

As depicted in Figure 8.1, there are many approaches that can be taken in evolutionary robotics to acquire a physical phenotype. The simplest form, without requiring a robotics simulator, is to directly evolve the behavior of a robot in a physical system as implemented in Floreano et al. (1994). This is the type of evolutionary strategy that is implemented on a knifefish inspired soft robot as described in Chapter 9. Before discussing this approach to evolving the soft robot, the next section presents a method for the automated reconfiguration of modular robots directly in the physical world which can allow for the automated evaluation of simulated and real robots simultaneously. This is an approach toward autonomous evaluation and self-reconfiguration of modular robots.
8.2 Automated Assembly of Modular Robots

In Brodbeck et al. (2015), an approach is described for the automatic generation of modular robots. The robots, composed of active and passive modules, were picked up by a gripper attached to a UR5 robot arm and connected to other modules with hot glue adhesives. This process allowed Brodbeck to automatically assemble and evaluate a variety of modular robotic structures that were evolved. The main disadvantage of this approach being the difficulty of removing the glue once a robot has been constructed and reconfiguring the morphology afterwards. The implementation of a similar assembly strategy using the magnet based modular robotics system described in Chapter 4 and Chapter 7, would allow for a versatile automated assembly, and disassembly of robotic modules without the necessity of hot glue or other adhesives. This makes the platform potentially resilient and enables robots to be sequentially reconfigured and evaluated. This section describes a methodology for automatically recognizing modules and how to subsequently pick and place the modules to create a robot morphology.

8.2.1 Methods and Environment

For the assembling process of a modular robot, a robot arm and a webcam were used in an environment where modules could be automatically detected, connected and disassembled. The environment consisted of a rectangular plate with four fiducial markers (markers used as a point of reference) on the corners. These four fiducial markers were used to extrapolate the position of the modules in the environment. An affine transform was used to translate the pixel coordinates in the positions in the reference system of the robot. The modules themselves also contained fiducial markers. These markers were oriented so that the male connector site was always facing in the same direction. The positions and orientations of the fiducial markers on the modules were used to plan how the robot arm would connect the modules to one another similar to Faíña et al. (2017).

The robotic platform used a magnetic gripper attached to a Universal Robotics UR5 robot manipulator. The gripper utilized permanent magnets to detach and attach magnetic modules for the creation of the robot as can be seen in Figure 8.2. A module is attached to another by attaching the male connector site to another specified female connector site of another module. The gripper did not contain any moving parts and thus relied on the specific movement of the arm for connecting and disconnecting itself to the modules. The end effector moved in an arc perpendicular to the male connector site to disconnect itself from the modules. When detaching a module from the modular robot containing several modules, the direction of the movement was changed to produce an arc around the two connected module sites.

The basic assembly steps are:

1. Move and align end effector above the marker
2. Move the end effector down until a force of 30N is applied; avoiding jamming and ensuring that the module is well connected to the gripper
3. Lift the module up to a safe distance above the floor
4. Move the end effector to align with a desired site to connect to
5. Move the end effector down hovering above the floor
6. Move the end effector towards the site to attach
7. Perform the movements required to release the module that is attached to the gripper.

With this system, various types of morphologies can easily be constructed. However, the morphologies in this case could only be assembled in 2D since the fiducial markers were only connected to one connector site of the modules which was identified by a camera from a top view. Hence only planar configurations (one layer of modules) were considered. The process of assembling of modular morphologies is
displayed in Figure 8.3. The assembly of the morphology was achieved without difficulties due to the self-alignment properties of the connectors. It was more challenging to disassemble the morphology where the robot arm would sometimes jam. The main problem being that the complex movements of the end effector were sometimes near singularity points leading to a safety stop causing the jam. Additionally, the end effector sometimes caused the attached module to be pressed against the arena floor while rotating, also causing a jam. These challenges should be taken into consideration in future implementations.

### 8.2.2 Prospects on the Automated Assembly of Modular Robots

The process of creating modular robots discussed in this section allows for the automated generation of modular robots which would, in its current state, be especially useful for chain type modular robots (Stoy et al., 2010) where some self-reconfigurations are not possible due to kinematic restrictions. It is moreover useful as a testbed for rapid prototyping. When combined with a robotics simulator that can automatically optimize potential modular robot configurations, this platform would allow us to automatically generate robots with a unique morphology and control. A fruitful prospect is to combine the approach with the evolving modular robots as discussed in Chapters 4, 5, 6, and 7.

Some challenges remain to be addressed in order to improve the system to a state where continuous experiments can be done with reconfiguring robot morphologies. Visual fiducial markers can currently only be attached to the modules at specific parts...
occupying and obstructing a potential connection site. A redesigned fiducial marker or another way of recognizing the robot modules should be used in future implementations. Another limitation of the current setup is that the robot is unable to be controlled and acquire power without the help of an operator attaching a control cable to one of the modules. This problem could simply be bypassed by adding a docking station that the modular robot can be connected to. To have a fully automatic assembly process, a base module that couples to a docking station can be implemented. The base module/docking station assembly can then be used as the initial module from which robot morphologies are assembled. Battery modules with wireless capabilities can also be added to the system when the assembled robot must move away from the docking station. 3D configurations still represent a challenge as they would require more complex reconfiguration movements and adjustments to the visual tracking system.

Automatic reconfiguration can be especially beneficial in fields that optimize the morphology and control of robots. It can be specifically well suited for evolutionary
robotics experiments that are usually time consuming due to the number of morphologies that need to be tested. Although some approaches already show promising results by implementing evolved robots in simulation environments and afterwards transferring them to the real-world (Auerbach et al., 2015), they are still time consuming considering that all parts have to be glued or screwed together. This is also addressed in Brodbeck et al. (2015) by using an evolutionary algorithm to generate the robot morphologies, however, the approach presented here allows for different robot morphologies to be more quickly tested in an environment and a combined approach of using simulations to optimize the robots with the subsequent transference of the evolved simulated robots to reality is therefore promising. This can, furthermore, give insights in reality gap related issues that would allow us to improve simulators and evolutionary algorithms to minimize the gap.

8.3 Concluding Remarks

In the first part of this chapter, an overview has been given on approaches to evolve physical systems. This can be done in various manners either including a simulation environment or not. Though in order to rapidly evaluate various robotic morphologies that can be evolved, as discussed in Part II, an automated assembly process of these modular robots is beneficial. Section 8.2 has shown how modular robots can be automatically assembled and disassembled using a robot manipulator. Without the need for power consuming autonomous modules, the merit of the approach would potentially alter the way we evolve robots. Otherwise human approached prototyping can take long hours whereas the assembly and positioning of the robots with the robot arm, spans only a few minutes. The automatic reconfigurability thus enables fast prototyping of different robotic morphologies and control systems, insightful to constructing an efficient robotic end-product for a given task. The visual feedback system could be improved by being able to identify modules in three-dimensional and the robot arm can be improved by having an active gripper and a better positioning algorithm. In addition, the potential integration of a docking system could allow for the automated repositioning and recharging of modular robot conglomerates. Many improvements can be made to engender the automated design and creation of robot behavior and morphologies that would truly allow us to evolve the simulated and physical robot in an automated manner.
Chapter 9

Evolution in a Knifefish-inspired Soft Robot

The physicality and embodiment of robots accompanies the morphological complexity that can be hard to simulate. One way to deal with the limitations of not being able to simulate a robot is to directly implement the optimization strategy on the robot. Bypassing the simulator ensures that the strategies found conform to reality not violating the laws of physics. In this chapter, evolution is implemented directly on a bio-inspired soft robot that was based on the black ghost knifefish. This knifefish species is interesting since it has an undulating fin underneath its body responsible for its movement Figure 9.1. The robot fish was made from soft materials used to cast the entire body and included solid fin rays connected to servo motors for actuating the fin. This chapter discusses the design, the algorithmic implementation, and the resulting behaviors compared to natural knifefish. The comparison of the robotic knifefish to the actual knifefish through evolving its controllers combines the paradigms of biorobotics and evolutionary robotics also known as evolutionary biorobotics (Long, 2012). Through evolving the robotic knifefish we can learn how to better optimize soft robots and start to understand the natural functionality of undulation in knifefish. It is a step toward automatically generating the control for robotic soft robots. Hence, this chapter brings us to the fifth and final hypothesis of the thesis discussing the evolution in a physical robot:

Hypothesis 5 Evolutionary computation finds better solutions for controlling the designed soft swimming robot compared to manually encoded behaviors. ¹

9.1 Introduction

Despite recent advances in evolutionary robotics, the reality gap (Jakobi et al., 1995) is still a prevalent issue. Especially in the emerging field of soft robotics it becomes more difficult to simulate the physical properties of soft materials accurately (Rus et al., 2015). In cases where this was accomplished successfully, it required high computational power and complex algorithms (Coevoet et al., 2017). For aquatic robots, the integration of flexible materials can lead to increased performance by the principle of morphological computation, i.e. by exploiting that dynamic interactions with the environment can be useful for achieving a desired behavior efficiently. The complex mechanics of silicone and its hydrodynamic interactions are, however, computationally heavy to simulate, especially when the morphology is driven by multiple actuators. For these reasons, an evolutionary approach was implemented to directly evolving physical systems (Rieffel et al., 2017) as a feasible alternative method to evolve efficient behavior of a bio-inspired soft robot.

Soft robots have been proposed for several applications that include exploration and search and rescue operations. For such tasks high maneuverability is usually necessary. Since the family of ghost knifefish (Apteronotidae) contain examples of dexterous aquatic animals capable of high multidirectional maneuverability at low

¹The limitation of this hypothesis being the ‘manually encoded behavior’ that was designed. If it was manually designed by the leading expert on knifefish, it might have well resulted in a different behavior.
Chapter 9. Evolution in a Knifefish-inspired Soft Robot

Figure 9.1: The black ghost knifefish. Vectorized image taken from Youngerman et al. (2014). Images show the aperonotus albifrons from a lateral (A) and ventral (B) view; orange line depicting digitized points across the tip of anal fin of the knifefish. Six digitized points are shown by orange circles along the individual fin rays laterally (C) and ventrally (D).

speeds (MacIver et al., 2004), this fish was chosen as the model whose control was subjected to evolution. Knifefish are able to produce thrust in many directions by undulating a single anal fin located underneath the body. By generating propagating waves across their fin they can easily move backwards and forwards depending on the directionality of the wave (Curet et al., 2011a). Vertical thrust is accomplished through sending counter-propagating waves towards and away from the center of the fin canceling out longitudinal forces. In undulatory swimming, the thrust is produced through a reaction force on the fluid adjacent to the body or fin surface. Bending of the body part, the fin, enables wave propagation. The combination of the lateral forces produced on both side of the fin should cancel out each other to produce a net forward thrust (Biewener, 2003).

9.1.1 Evolution of Soft Robots

The evolutionary robotics approach to soft robotics has thus far only been implemented in simulation environments such as VoxCad (Cheney et al., 2013; Cheney et al., 2016; Kriegman et al., 2017) or off-the-shelf physics engines where morphologies are represented by tetrahedral meshes and the controls and morphology have been evolved (Rieffel et al., 2013). Computational power is, however, a major constraint when using simulations. Computational requirements usually scaling proportionally to the amount of tetrahedra or voxels simulated, usually exponentially. Morphologies found through the VoxCad approach have only been replicated physically by means of soft volumetrically expanding materials that require changes in the pressure of the surroundings for actuation (Hiller et al., 2012), though these results are again take from evolving the soft robot in a simulator and transferring its phenotype to the real world afterwards.

Controllers for simulations of existing partially soft morphologies have also been evolved and in some cases transferred to hardware. A genetic algorithm with a "lumped" dynamic model simulation has been used to evolve the gait of a soft caterpillar-inspired robot and has resulted in an increase in performance of a physical prototype (Saunders et al., 2011). In another instance, both an objective-based and a novelty-driven (novelty search; Lehman et al. 2011) approach have been utilized to optimize the design of a crawling octopus by discovering self-stabilizing dynamic gaits (Corucci et al., 2015). A differential evolution algorithm was used to optimize a model-free adaptive controller (MFAC) in a simulation of a robotic fish with a flexible caudal fin.
For the same morphology an evolutionary multiobjective optimization technique (NSGA-II algorithm) found morphological and control parameters in simulation that maximize the swimming speed and minimize the power usage with subsequent validation in hardware (Clark et al., 2015b). However, in this approach it was found that the "best speed" parameters of the evolved simulated model were considerably faster than seen in the experiments due to hardware limitations. This illustrates that although reasonable performance can be transferred from the simulation to reality, discrepancies are still persistent. In the above examples the evolution of soft robot morphologies and controllers was made possible by confining the search space to highly abstracted morphologies (fish where only a simple tail is flexible, caterpillar-like shapes) or by decomposing the morphologies into a finite number of voxels. While such approaches have yielded interesting results, they are still lacking in relation to realizing the full potential of soft robotics technology as they limit the design space to very simple or highly abstracted shapes. By evolving the controller in the physical hardware instead, an effective strategy can be attained through (1) a bio-inspired design that mimics a natural model closely and (2) the automated discovery of its most optimal behavior.

9.1.2 Knifefish-inspired Swimming Robots

Due to their unique morphology, knifefish have served as inspiration for a number of research robots. Building on the work of Low et al. (Low et al., 2006; Low, 2009), Siahmansouri et al. constructed an untethered robot with 6 fin rays capable of regulating the direction and depth of swimming by moving the fin relative to a buoyancy tank (Siahmansouri et al., 2011). Curet et al. built a knifefish-inspired robot with 32 individually actuated fin rays and were able to show that its optimal actuation parameters were similar to the ones of the black ghost knifefish (Curet et al., 2011b). They were also able to generate upward forces on the robot with counter-propagating undulation waves (Curet et al., 2011a). Sfakiotakis et al. (2015) devised a linear slide equipped with a fin composed of 8 individual fin rays and implemented open-loop velocity control and closed-loop position control.

A common denominator of the previous work on knifefish-inspired robots is the use of sinusoidal functions as an undulation pattern for the fin. This occurs even though a sine function is only an approximation of the actual undulation pattern of the species, which could be reproduced more accurately (Youngerman et al., 2014). The design of our robot also departs from the earlier work as it is an integrated silicone morphology constructed with contemporary soft robotics fabrication techniques. This approach simplifies the fabrication of the fin and fin rays significantly. Moreover, elasticity is added to the fin, which has been hypothesized to be a means of increasing energy efficiency (Low, 2009).

9.2 Methodology

A soft swimming robot with a single undulating fin inspired by the anatomy of the black ghost knifefish was designed. To be able to evaluate its swimming speed with different motion patterns, the experimental setup shown in Figure 9.2 was constructed. As only the forward swimming speed was evolved, the robot is fixed on a linear slide. It is not submersible and kept at a level of neutral buoyancy. The robot (E) is placed in the water surface of a 100 cm × 40 cm × 40 cm aquarium. It is tethered with power and signal cables for its 6 servo motors. It is attached to a cart (F) with four ball-bearing wheels that is mounted on a T-slot beam linear slide (C) atop the aquarium. A plastic attachment piece (D) connects the cart to the linear slide and prevents the robot from turning. The slide is equipped with two IR sensors to measure when the beginning and end of the slide has been reached. For the evaluation of an undulation pattern, the robot starts on the left side of the track at the first IR sensor. During evaluation a swimming pattern is played on the robot and an ultrasonic distance sensor (A) measures the

2A video of the robot can be found here: https://www.youtube.com/watch?v=3Xjg2bs0t2g
distance to a plastic plate (B) on the cart. The cumulative sum of the distance readings are used directly as the fitness value for the undulation pattern that was evaluated.

9.2.1 Mechanical Design of the Robot

The main parts of the robot are its hull, frame, and fin rays (Figure 9.3). The hull and fin of the robot were constructed from Ecoflex 00-30 silicone (Young’s modulus approx. 0.1 MPa, Shore A hardness 00-30; Mosadegh et al. 2014). The uncured material was degassed after mixing and poured into a three part 3D printed mold (two sides and one inner part). The inner mold part holds the fin rays in place during casting and blocks out a compartment for the rigid inner frame, which was mounted after casting. The inner frame was constructed from laser cut acrylic parts that were glued together. The servo motors are held in place with bolts and nuts.

Six bamboo sticks (approx. diam. 3mm) serve as fin rays. With 6 fin rays it is theoretically possible for the robot to hover and to move forward, backward, up, and down by generating traveling and counter-propagating waves (Curet et al., 2011a). Each fin ray is attached to a servo motor via a servo bracket. The servo motors used were initially six H-KING HK 15148 mini servo motors. Due to malfunctions three of them were replaced with a two tower Pro SG90 micro servos and one EMAX ES08AII servo motor. The servo motors are connected to the fin rays with a crank-like mechanism (Figure 9.4). The angle of a fin ray as a function of the servo angle is given by:

$$\alpha = \tan^{-1}\left(\frac{\sin(\theta) \cdot 21}{30 - \cos(\theta) \cdot 30}\right)$$

(9.1)

where the constant 21 is the distance (in mm) from the center of rotation of the servo to the piston that connects to the fin ray and the constant 30 the distance from the center of rotation of the servo to the approximate center of rotation of the fin ray (see Figure 9.4). The change in the bamboo angle and based on the angle of the servo is displayed in Figure 9.6.

This equation, however, does not take into account the additional angular deflection caused by slack between the pistons and the fin ray, the elasticity of the soft body resisting rotation (see Figure 9.5), and the tilt of the soft body when actuating the

---

3The CAD files can be accessed at: https://cad.onshape.com/documents/51d2c0394f6e3aa7b3fc06b3
Figure 9.3: CAD design of the robotic knifefish. The white parts represent the laser cut acrylic parts, the blue part is the silicone part (top), the black parts depict the 6 servo motors that were used for each fin ray. The bamboo sticks that serve as the fin rays are displayed in green. The robot’s full dimensions are 272x60x136mm and the fin is 70mm high and 210mm long. The fin rays are each spaced 40mm apart.

9.2. Methodology

9.2.2 Evolutionary Experiments

In the pre-experiments a generational evolutionary algorithm without crossover was implemented to create the genome for our robot controller. Due to the long evaluation time of the generational evolutionary algorithm, and servos being prone to overheating, it was decided to implement Covariance Matrix Adaptation Evolutionary Strategy (CMA-ES; Hansen et al. 1997; Hansen et al. 2003) instead, to quickly find the basin of attraction and thereby speed up the evolutionary process.  

Encoding

The genome created for an individual is composed of a string of 15 bytes. Each triple of three bytes translates into a sinusoidal function with a specific frequency, phase, and amplitude.
amplitude. The total of five sine functions are summed to yield the first five terms of a standard Fourier series. With this function an arbitrary continuous periodic function can approximated and used as a fin undulation pattern on the robot to be evaluated. The mutable parameters were the amplitude, phase, and frequency of each sinusoidal function. These parameters are converted into servo angles $\alpha_n$ for the 6 servo motors with the following function:

$$\alpha_n(t) = \left( \frac{g_1}{255} \cdot \theta_{\text{max}} \right) \cdot \sin((g_3 \cdot t) + (g_2 \cdot s_n))$$

(9.2)

where $g_1$, $g_2$, and $g_3$ represent the mutable parameters of a genome triple as bytes. $\theta_{\text{max}}$ is the maximum angle that the servo motors can move. $s_n$ stands for adjacent servo motor numbers (values from 0 to 5) and $t$ represents the time steps.

In the second experiment the wave patterns that have evolved using the Fourier series were evaluated. In this case, the mutable parameters are transformed into outputs through summing up 5 sinusoidal functions of the type in Equation 9.2.

**Evolutionary algorithm**

The evolutionary approach was divided into a control system and an evolutionary algorithm. The evolutionary algorithm made use of functions from the Distributed Evolutionary Algorithms in Python (DEAP) library which included an implementation of CMA-ES (Fortin et al., 2012). The CMA-ES implementation implemented a population size of 10 and ran for 20 generations. CMA-ES was able to find similar solutions in 20 generations as running a normal generational evolutionary algorithm for 100 generations which was advantageous for limiting the duration of the experiments. Our CMA-ES implementation included an initial standard deviation value of 50 and a centroid value of 125 for every gene (half the max value of the bytes in the genome).

**Controller system**

An Arduino Mega 2560 controlled the robot by actuating the servo motors and received the sensor readings of the ultrasonic distance and infrared sensors. Through serial communication, a genome is uploaded from a PC running the evolutionary algorithm
9.2. Methodology

Figure 9.5: Angular deflection of the fin. Front view of the robot showing the angular deflection of the fin. The actual maximum angle of the fin can be seen to be less than the calculated angle (red dashed lines).

Figure 9.6: Change in angle of fin rays. The calculated angle of the fin ray based on the angle of the servo motors is shown on the left. The right side depicts the angle of the fin ray based on a simple sinusoidal input. The wave is not a perfect sine wave but approximates one.

to the Arduino Mega. The Arduino Mega evaluates an individual using the genome it received. This evaluation consists of:

1. Move robot to the starting position (by using a manually coded swimming behavior)
2. Move the servos to a central position and wait for six seconds (this delay was implemented to prevent overheating of servos and reduce waves in the tank)
3. Evaluate genome for 10 seconds
4. Send back a fitness value based on the distance the robot has traveled within the 10 seconds

All steps take roughly between 20-30 seconds for one individual depending on how far the robot was able to swim. When the same genome was evaluated multiple times the error difference in fitness was negligible (standard deviation of samples of size 4 was less than 1% for each run). Each individual is therefore only evaluated once.

There was a 20 ms delay inserted between each time step for updating the servo angles. 500 time steps were done for each individual. The fitness value of each individual is calculated as a summation of the ultrasound distance measurements at every
consecutive update of the servo positions. At each time step the ultrasonic distance sensor initiates a sound pulse and measures the time difference between the pulse and echo. This time interval becomes higher the further the robot moves away from its initial position. The fitness value for a controller that is not moving robot lies around $100 \cdot 10^4$. At the start of the evaluation of a genome, the entire wave pattern for each servo was calculated for each time step. This required six arrays to store 500 byte values derived from the genome. Although this takes up a lot of memory on the Arduino Mega board, it circumvents doing calculations on the spot that might have caused an additional delay between every time step. Such a delay was, however, caused by the ultrasound sensor which required an 8 microsecond delay for measuring the distance.

Experiments

Since earlier examples of robotic knifefish have been able to swim with only a single sinusoidal wave function as a control signal for the fin, experiments were conducted where the genome is reduced to three bytes that translate into the frequency, phase and amplitude of a single sine function. We tested if evolution is able to efficiently optimize these three parameters for increased swimming speed. Our second set of evolutionary experiments evaluate functions that are generated from all 15 mutable parameters, and yield the first five terms of a Fourier series. This is done to see whether an arbitrary periodic function can increase the performance compared to a single sine wave. For both sets of experiments, whether evolution will find swimming behaviors similar to the ones of actual knifefish, and if the performance of the evolved controllers can rival a manually programmed controller, was also tested. For both the sinusoidal and the Fourier series approach, 5 evolutionary runs were done with the exact same hardware setup. Since the slightest change in hardware and the environment can influence the evolutionary runs drastically, all the 10 runs were done consecutively. A manually coded swimming behavior is used as a baseline to compare with the evolved controllers. This behavior was the fastest swimming behavior we were able to find by manually adjusting the genome parameters during a two-hour trial session with the platform. Its control function is:

$$\alpha_n = 40 \cdot \sin((64 \cdot t) + (100 \cdot s_n))$$

(9.3)

These control parameters correspond to a genome with the following three bytes: 255 for the amplitude, 64 for the phase, and 100 for the frequency.

9.2.3 Comparing behaviors of the robot with actual knifefish

Bale et al. (2015) found that a diverse group of aquatic animals that use median/paired fin swimming, including knifefish, have evolved a similar optimal swimming strategy. More specifically, the result of dividing the length of an undulation on the fin by the mean amplitude of undulations along the fin, during steady swimming, consistently yields around 20. This wavelength, which maximizes the force generated by the body and the swimming speed, is referred to as the optimal specific wavelength (OSW). The specific wavelength (SW) of the evolved undulation patterns was therefore calculated to compare them with the swimming behaviors of the knifefish. The SW was calculated by dividing the wavelength of undulation $\lambda$ by the average amplitude of oscillation $\bar{a}$. In general, this average amplitude $\bar{a}$ is given by

$$\bar{a} = h_{mean} \sin(\frac{\theta_{max}}{2})$$

(9.4)

Where $\theta_{max}$ is the mean maximum angle of excursion of the fin rays and $h_{mean}$ is the mean height of the fin.
9.3 Results

9.3.1 Performance Analysis

After running CMA-ES for 20 generations using the sinusoidal and the Fourier series approaches, different wave patterns were acquired. Both evolutionary progressions of the 5 runs of each approach (Figure 9.7) evolved decent swimming behaviors though the Fourier series evolutionary progressions seem to have more variation in performance and did not plateau as clearly as the sinusoidal evolutionary progression. This corresponds to a larger, perhaps more convoluted, search space when evolving Fourier series.

The periodic control signals that have evolved in the sinusoidal approach are similar to each other while the best individuals of the Fourier series exhibit more erratic wave patterns (Figure 9.8). Looking at the individual wave patterns and their corresponding fitness values, the best individual evolved in the Fourier series has a significantly higher fitness value than the others.

In Table 9.1 the evolved swimming behaviors of the best candidates were compared to see if the OSW ratio also applies here. The approximate wavelengths of the traveling waves have been obtained from ventral view video recordings of the robot with the best candidates and the manually coded behavior controlling its swimming. The average amplitude of oscillation was calculated from Equation 9.4 using a maximum angular excursion of 28 degrees (derived from video recordings) and that the fin height is 7 cm. The average travel speeds were also measured from video recordings (of the manual behavior and the best evolved individuals being replayed on the robot; Bale et al. 2015. The black ghost knifefish that was the inspiration of our robot has an SW of around 18 (Bale et al., 2015). From Table 9.1 it can be seen that the best evolved sinusoidal controller has a specific wavelength of 16, i.e. it approximates, but is lower then, the optimal specific wavelength found by Bale et al. (2015). Although our manually programmed controller has a SW of 17 and comes closest to the actual knifefish, in reality it performed considerably worse than most of the evolved controllers (see Table 9.1).
Chapter 9. Evolution in a Knifefish-inspired Soft Robot

**Figure 9.8:** The best evolved wave patterns in 5 distinct evolutionary runs using the sinusoidal approach (a) and the Fourier series approach (b). The graphs show two seconds of a resulting wave from each genome. The blue line represents the trajectory of the first servo motor while the green dotted and red dashed lines depict the positions of servos two and three respectively. The trajectories of servo four, five and six are not depicted. The difference in the wave of different servos visible in some of the Fourier series is due to including potentially high frequencies and querying the function every 20ms.

**Table 9.1:** Specific wavelengths and travel speeds of behaviors. The evolved behaviors resulted in wave patterns with varied wavelengths and speeds. (Wavelength of Four. (Run 4) has been omitted as the wave function was too erratic for it to be measured)

<table>
<thead>
<tr>
<th>Genome</th>
<th>Wavelength</th>
<th>SW</th>
<th>Speed (cm/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Manual</td>
<td>28 cm</td>
<td>17</td>
<td>3</td>
</tr>
<tr>
<td>Sine (Run 1)</td>
<td>26 cm</td>
<td>16</td>
<td>8</td>
</tr>
<tr>
<td>Sine (Run 2)</td>
<td>23 cm</td>
<td>14</td>
<td>6</td>
</tr>
<tr>
<td>Sine (Run 3)</td>
<td>26 cm</td>
<td>16</td>
<td>6</td>
</tr>
<tr>
<td>Sine (Run 4)</td>
<td>23 cm</td>
<td>14</td>
<td>6</td>
</tr>
<tr>
<td>Sine (Run 5)</td>
<td>24 cm</td>
<td>15</td>
<td>8</td>
</tr>
<tr>
<td>Four. (Run 1)</td>
<td>26 cm</td>
<td>16</td>
<td>4</td>
</tr>
<tr>
<td>Four. (Run 2)</td>
<td>26 cm</td>
<td>16</td>
<td>2</td>
</tr>
<tr>
<td>Four. (Run 3)</td>
<td>24 cm</td>
<td>15</td>
<td>5</td>
</tr>
<tr>
<td>Four. (Run 4)</td>
<td>-</td>
<td>-</td>
<td>5</td>
</tr>
<tr>
<td>Four. (Run 5)</td>
<td>22 cm</td>
<td>13</td>
<td>1</td>
</tr>
</tbody>
</table>

Being able to evolve wave patterns to control the swimming behavior of the robot is of limited use if their phenotype cannot be reproduced. Since the robot was slightly worn down after a lot of different experiments and after having replaced several malfunctioning servo motors, we evaluated the performance of the evolved wave patterns again. When comparing the evolved Fourier series wave patterns with the evolved sinusoidal wave patterns it can be seen that the sinusoidal wave patterns also outperform the manually encoded wave pattern significantly in terms of fitness value (Figure 9.9). Though this could have been caused by many factors, it seems that
9.4. Discussion

CMA-ES proved an efficient method for automatically evolving the swimming behavior of our soft swimming robot inspired by the ghost knifefish. Although the search space was quite small, failing hardware is usually a problem that makes evolving physical robots arduous. Predefining the controller by only utilizing periodic wave functions and only running CMA-ES for a brief period was enough to generate efficient swimming behavior. One of the main challenges when evolving physical robots is about how to deal with malfunctioning hardware. Considering a death toll of 17 servo motors during these experiments, using CMA-ES seemed a lot more viable compared to initial experiments with a generational evolutionary algorithm that took almost five times longer to get to results compared to the CMA-ES approach. CMA-ES was however not completely immune to the death’s of the servo motors as discussed in Text Box 9.7.

The robotic platform presented in this chapter is constrained by predefined functions and the limited movement sets acquired in the evolutionary runs. However, the presented robot fish could evolve many different behaviors that the knifefish is also capable of. This could make it a viable option for autonomous underwater vehicles. A next submersible iteration of the fish could evolve vertical thrust through sending
counter-propagating waves towards and away from the center of the fin canceling out longitudinal forces as discussed by (Curet et al., 2011a). A selection of these behaviors can be evolved and encapsulated in a fixed environment, removing manual programming of the behavioral repertoire.

Zoological studies of knifefish kinematics have shown that the wavelength of the propagating wave varies across the fin during steady swimming (Youngerman et al., 2014). Given that the swimming behavior of the knifefish has been optimized through natural evolution, implementing this feature in the encoding of the controller could lead to better performance. Additionally, this could be accomplished by using a compositional pattern-producing network (CPPN; Stanley 2007) with servo number and time as inputs. A similar approach has previously been used successfully to generate the oscillatory controller for a quadruped robot (Morse et al., 2013). To discover a greater variety of controllers that perform well, novelty search (Lehman et al., 2011) or other diversity enhancing methods can also be applied instead of a goal directed approach which is often prone to premature convergence or over-fitting. Another aspect worthy of further inquiry is the materials used for the fin. It is possible that a material with another elastic modulus might better exploit the interactions with the water to facilitate the emergence of dynamics that aid the swimming.

With this robotic platform, we were able to automatically evolve the behavior of an
9.4. Discussion

Text Box 9.7: Servo malfunctions

One of the main issues of the experiments demonstrated here was the malfunctioning of the servo motors as described by the ‘death toll’ of 17 servo motors. It should be considered for any future implementations that the servo motors are robust enough to survive the evolutionary progression. Meaning that we should have bought more resilient servo motors for the experiments. However, considering this limitation, CMA-ES was an essential implementation since other implementations took longer to achieve similar fit individuals. The implementation of CMA-ES relieved us from having to tweak the parameters of the evolutionary algorithm to get the correct results. In order to have comparable results of multiple evolutionary runs, it was a requirement that the same set of servo motors survived for all the ten evolutionary runs of the sinusoidal approach and the fourier approach. However, in many cases a servo failure actually occurred. When plotting the percentiles and the average fitness values of the population in this run the resulting evolutionary progression contains a sudden change in performance as depicted in Figure 9.11.

![Figure 9.11: The evolutionary progression of a Fourier Series with a servo malfunction occurring around generation 10. The grey area represents the 25-75 percentiles of the population, the upper and lower solid lines the maximum and minimum fitness values in the population, the blue line the average fitness value of the population and the white line is the median fitness value of the population.](image)

intuitively functional soft robot using CMA-ES. Considering the increasing advances of automated manufacturing methods and readily available materials to create detailed robots with various features, we think this evolutionary approach on physical soft robots can become viable as a tool for directly optimizing the behavior of the physical systems.
9.5 Conclusion

This chapter demonstrated that evolving the controller for a knifefish-inspired soft robot is feasible directly on the physical robot. The majority of the evolved behaviors outperformed a hand-designed controller in terms of speed. Additionally, evolution was able to exploit the dynamical properties of the flexible material to produce feasible swimming strategies for the robot that have similar phenotypes but different genomes. Evolutionary experiments on physical robots, which have so far only been applied to traditional non-soft robots, are especially relevant for soft robots that are difficult to simulate computationally. In the future, the presented approach could be combined with more explorative search methods such as novelty search and different fish models, to solve tasks for which even a simple hand-designed controller is an infeasible option.
Chapter 10

Discussion

Complex now [the mechanical reproductive system], but how much simpler and more intelligibly organized may it not become in another hundred thousand years? or in twenty thousand? For man at present believes that his interest lies in that direction; he spends an incalculable amount of labour and time and thought in making machines breed always better and better; he has already succeeded in effecting much that at one time appeared impossible, and there seem no limits to the results of accumulated improvements if they are allowed to descend with modification from generation to generation. It must always be remembered that man’s body is what it is through having been moulded into its present shape by the chances and changes of many millions of years, but that his organisation never advanced with anything like the rapidity with which that of the machines is advancing. This is the most alarming feature of the case, and I must be pardoned for insisting on it so frequently.

– Samuel Butler, Erewhon

Artificial life – the paradigm concerned with the recreation of biological phenomena seen in life – is slowly progressing as a field enabling the emergence of it. From the perspective of artificial life, this thesis gave an overview on theory of evolutionary dynamics and displayed both simulated (Chapters 3, 5, 6 and 7) and physical (Chapters 7, 8 and 9) experiments that tried to engender the recreation of phenomena of life using an evolutionary approach. The experiments not only teach us about life, but promotes the development of future technologies. Researchers of evolutionary robotics simply try to nudge the evolutionary search process to give rise to a potential wide spectrum of robots – making us the watchmakers. We allocate building blocks and divide laborious tasks across isolated functionalities of robots. Some functionalities having straightforward solutions, like rotating a wheel through a DC motor, while others require crossing convoluted fitness landscapes. Machines have already been designed to, for example, exhibit optimal arithmetic, and these attributes could simply be implemented as a module, being one more building block in our artificial system. The composition of the various theories, methods and materials together can build a cascade of incremental steps engendering the emergence of a robotic conglomerate that can be optimized toward any objective. Toward this objective, this thesis discussed the role of death on the evolvability of a population; the evolution of modular robots; and using evolutionary computation directly evolve behavior on a physical robot.

10.1 Evolutionary Dynamics of Intrinsic Mortality

To improve our understandings of life and its emergent phenomena, it is important that we understand as much as we can about its evolutionary dynamics. For evolutionary robotics this means especially manners in which we can traverse fitness landscapes. For robotics applications there may be many landscapes that are interconnected and could potentially be isolated. And some landscapes might be dependent on others, where a seemingly unrelated stepping stone needs to be found before the population can be further evolved (Stanley et al., 2015). E.g., a robot might need to see before it is able to grasp objects. Although sight might give a direct evolutionary advantage
Chapter 10. Discussion

when discovered, other potential strategies can emerge from precursors that had different functionalities when they initially appeared; as in the feathers that gave rise to flight in birds. Evolution has therefore a seemingly random explorative nature and in turn the capacity to exploit any type of property that deems sufficient. The competence of a population to traverse the fitness landscapes empowers evolvability but is likely produced due to happenstance.

Chapter 3 has specifically shown that one factor, mortality, drastically changes the evolvability of a population when implemented on deceptive fitness landscapes. I contemplate that deceptive landscapes in particular are the most important analogy to natural systems. In nature, these deceptive landscapes are dynamically shaped by the environment. This dynamic aspect can come from us changing, removing or adding an objective to the artificially simulated environment or physical robot. The mortality implementation shows how a population can traverse the top of the fitness landscape. This hill-hugging phenomenon – emergent from a simple decrease in selective pressure on the fittest individuals – might well be a powerful way through which artificial systems can find solutions by means of apparent drift. Most algorithmic implementations instead rely on diversification methods through the insertion of random individuals (Schmidt et al., 2011), or the active search for novelty (Lehman, 2012). Though diversification is promoted when using mortality, we could postulate that this implementation is functional novelty through death, or mortality induced adaptive radiation.

As discussed in Kowald et al. (2016), senescence might well be emergent through a trade-off between advantageous cellular machinery (disposable soma) or a coincidental effect of genes that are early life fitness promoters and detrimental later in life (antagonistic pleiotropy). Proving that senescence increases the algorithmic performance does solicit the advocates of non-programmed aging to take into account that a mortal population might simply be more effective on a time-scale of hundreds of generations. The difficult question that remains is still: when is senescence an evolvable feature? Not merely a feature that increases the evolvability. This means that future experiments should make the parameters of mortality themselves evolvable. Depending on the fitness landscapes that are being used, intrinsic mortality is likely not advantageous on landscapes with an obvious gradient towards a maximum. Even when there is a landscape that makes mortality advantageous, individuals exhibiting no mortality can well take over the population in the long run. When considering aging individuals to be cooperators, on immortals defectors, these defectors likely have a direct advantage over cooperators creating a prisoner’s dilemma scenario. A prematurely converged population stuck in a local optima has a chance to outcompete mortal populations that haven’t found a higher fit solutions in the landscape yet, though are likely in trouble when the environmental niche they occupy changes. Nevertheless, intrinsic mortality affects evolutionary search and is thereby likely an evolvable trait.

10.2 Evolving Modular Robots

The importance of using modularity in robots is the isolation of the functionality in specific parts of a robot. Additionally implementing generative encodings can engender a recursive succession of modules. This recursion of simple parts could in turn give rise to complex phenotypic traits. This process of recursion, or genotype to phenotype mapping, was implemented in Chapter 5 for creating artificial plant morphologies. It’s specific importance has subsequently been discussed in Chapter 6 for the acquisition of locomotion in modular robots. Although the L-Systems that were implemented might not be the most efficient type of generative encoding, they are a simple method to quickly generate a patterned phenotype that can be represented as a hierarchical tree.

To ensure that the robots that are evolved in a simulator are feasible, the type and amount of modular parts available can be specified to determine the morphological bounds of the evolutionary search process. This evolutionary search process can furthermore be altered by simply adding modules with their own unique attributes to the
same modular system with a different objective. The addition of solar panel modules is an example of changing the objective and building blocks of the evolutionary search process (Chapter 7). The application of solar panel modules could potentially be extended in artificial settings through implementing various solar panels that are optimal for different light intensities, potentially maximizing light absorption while minimizing resource requirements.

In a future simulation environment implementing different types of modules, specific modules of the robots could be subject to evolution while others remain fixed. For example, if a bipedal robot has been constructed, the leg modules could be subjected to evolution potentially changing the size and control mechanisms of various parts. In turn, when a bigger brain or additional arms are incorporated in this robot, the evolutionary algorithm could specifically re-evolve the structure of the legs to compensate for the added mass and shifted center of mass. Hence, a feedback loop of quickly tweaking local parts of a robot can be achieved at various scales.

The methods that were presented were however limited by the mechanical properties and requirements of the modules. In the case of the presented modules, the mechanical and electrical connections were distributed through the same connector site. However, from a engineering perspective this approach is flawed since a disconnection – be it temporarily – of two modules would result in the inactivation of the modules that in turn briefly lose power and communication. The future design of modules for a modular robot could take this into account by separating the electrical wiring and the mechanical connections. This can be done through additionally having magnet based wires distribute power and communication to other modules. Though this implementation would also make the robot more difficult to construct, especially automatically. In addition, mechanical connections close to the center of the robot would require more power and mechanical strength to actuate parts of the body compared to more distant parts. Recursive modules that change in scale and strength are therefore likely contributors to more efficient modular robots. Although despite the limitations of the presented work, the modular approach to evolving robots is an efficient way in which morphological designs can quickly be altered and explored in robotics.

The use of generative encodings can lead to phenotypes that are better transferable to the real world. As generative encodings allow the same behavior to occur in multiple modules in a system, this recurrence can mean that the solutions found are transferable because the result of the patterned behavior is likely more generic. It is not fine-tuned to its environment as would be more likely the case in a direct encoding. Fine-tuning in the simulator might well lead to the evolution of a behavior that isn’t transferable. In addition, a generative encoding is able to make a wider sweep across the search space while a direct encoding increments more locally. This wider sweep might also contribute to the generative encoding finding more generic strategies. Combining the approaches might however be an even better method for designing robots; initially making a somewhat large sweep with a generative encoding to find generic solutions and afterwards fine-tuning the phenotypes using a direct encoding. Where a generative encoding can evolve the robots in a simulator while the direct encoding can be implemented to tune the real robot.

The presented approach to evolving modular robots has demonstrated that modular robots can automatically emerge by giving the simulator a specific objective, be it energy acquisition or locomotion. With the addition of being able to reconfigure the modular robot physically, the approach considers a genotype to phenotype mapping emphasizing the morphology and decentralized control while also working toward a minimizing the reality gap. Being able to correctly abstract the morphology and control of robots, as well as being able to efficiently traverse the search space, has potential to yield a plethora of robot types optimized for any objective. This could potentially be transferred to real world scenarios where a user specifies the environment and the objective and the evolutionary plugin will automatically evolve the morphology and control based on the available modules.
10.3 Evolution of Physical Systems

Chapter 8 has given a brief example of a methodology to automatically assemble modular robots using fiducial markers and a UR5 robot arm. The results depicted in this approach were preliminary, but are aimed towards the automated production and evaluation of modular robots. Since the modules presented in this chapter were not active, there are some additional hurdles that need to be overcome before a complete autonomous modular robot assembler can be created. Moreover, though easy to use by researchers, the reconfiguration of modular robots can be difficult since disconnecting modules requires some force due to the strong magnetic connections between modules. There are many improvements that can be made e.g: using active grippers; having mechanical connectors; adding a power source module perhaps with wireless communication capabilities; and potentially adding a docking to automatically charge the robot and reset its position.

The evolution of the soft robot inspired by the black ghost knifefish was presented in Chapter 8. In this case, due to hardware limitations, it became apparent why the Covariance Matrix Adaptation Evolution Strategy was so efficient. Implementing sinusoidal wave functions that were mutable quickly led to the discovery of various undulation controls. Evolving a fourier series and a regular single sine wave function showed a similar evolutionary progression – the fourier series being a bit slower. After replacing all the servos with new ones, a drastic change in performance was noted. On average, solutions found by the fourier series performed worse than the solutions found by the sinusoidal controller. The sinusoidal approach being simpler probably means that a more generic solution is more robust and is not likely to exploit specific behavioral characteristics that the fourier series might well have done. However, the resulting behavior in the fourier series might also have emerged due to the exploitative nature of CMA-ES. Implementing another strategy, like the mortality rate, might be less exploitative and instead promote diversity.

There is an argument to be made for bypassing the simulator and implementing evolutionary approaches on robots directly. Taking the embodied robotics approach. For bioroboticists, an organism can be physically reconstructed and its behavior can subsequently be evolved. Elucidating not only principles of biological (perhaps extinct) organisms, but also aiding in the constructing of effective robots (Long, 2012). Moreover, the embodied approach – that is either implementing evolutionary computation directly on robots or the embodiment of the evolutionary algorithm itself in embodied evolution – does not have a reality gap. Simulators frequently exploit inaccurate dynamics/physics which are a major problem when dealing with the reality gap. Having efficient evolutionary methods that can adequately explore the fitness landscapes is thus a prominent technique for acquiring behavior in robots directly.

10.4 Concluding Remarks

This thesis discussed death, modularity and physicality for the ultimate acquisition of robot morphologies and control through evolutionary computation. Programmed death induces evolvability on deceptive fitness landscapes and might thereby be an explanation for senescence. Beyond being a mere explanation, it can prevent populations in evolutionary computation to prematurely converge enabling more solutions to be found. Generative encodings in modular robots increase the performance of evolving modular robots and illustrate the importance of the genotype to phenotype mapping and how development is beneficial. While the behavior in robots can be transferred from the simulator when the phenotypes are simple, directly evolving behavior is also feasible in the absence of simulation models, bypassing the simulator. The exploration vs exploitation trade-off, genotype to phenotype mapping and physicality are interlinked concepts that in conjunction can be implemented toward the generation of ever more adapted robots.

The issues presented in this thesis are moreover relevant to the general paradigm of AI. Conventional methods in artificial intelligence implement learning strategies
that update control parameters usually in a directed manner, following gradients and promoting novelty. Since evolutionary computation mainly works on the phylogenetic time scale, the incorporation of concepts in artificial intelligence on the ontogenetic time-scale can provide us with a toolset for improving existing machines and algorithms in general. Where the implementation of evolutionary computation can be beneficial for a wide-range of applications, but especially for the generation of complex robots from mere building blocks.

All biological machinery has evolved through the blind process of evolution. The elegant evolutionary dynamics have in turn formed the backbone of this thesis were it was utilized with the aim to create robots by defining the environment, the task, the building blocks and the encodings. The presented methodology is valuable for both science and engineering, where self-adaptation through evolution is key. Prospective improvements on the presented methodologies might well alleviate the design burdens of watchmakers, where instead of being designed, robots are evolved.
Appendix A

Full Spatial Model Pseudocode

The spatial model updates the biomass of all the cells at every iteration. Where $g$ represents the cycles and $n$ all the cells on the grid. If the cell that is being checked is not part of the evolvable population (!= X) then the biomass of the cell is updated depending on the biomass production rate set by the initial parameters. The cells that are part of the evolvable population can have a developmental trigger and a mortality trigger. If the cell should develop, its genome is swapped with another one that is stored in the subsequent developmental phase. If the cell is mortal, and the terminal age is reached or a random death factor function returns true, the cell is deleted and part of its biomass is left as plant biomass. After the developmental and mortality triggers, the cell can move, eat and reproduce. The data of the entire grid is saved every 100 cycles.

**Algorithm 2: Spatial Model Extended Pseudocode**

```
Initialize and Evaluate population $P = \{X_1, \ldots, X_N\}$;

while $g := 1$ to $G_{\text{max}}$ do New Cycle
  for $n := 1$ to $N_{\text{max}}$ do Update Cell
    if Cells[$n$] != X then
      Mass := Biomass Production Rate;
    else
      Mass := -Mass Loss Rate;
    end
    if Developmental Trigger then
      Develop Cell;
    end
    if Cells[$n$] = immortal then
      if Cells[$n$].age > Cells[$n$].maxAge || Cells[$n$] = Selected by Probability then
        Remove Cells[$n$];
      end
    end
    Move;
    Eat;
    if Reproduction then
      Reproduce;
      Mutate Offspring;
      Evaluate Offspring;
    end
  end
  if Cycle % 100 == 0 then
    Log State
  end
end
```

---

1The full implementation can be found here: https://github.com/FrankVeenstra/ALife2018
Appendix B

UML Evolutionary Robotics Plugin

The UML of the robotics plugin is presented in Figure B.1. The UML shows the relationships between the genetic algorithm, genome, morphology, modules, control, neurons, environment and V-REP. Five factory patterns, from which two were dependent on V-REP, were implemented.

**Figure B.1:** The UML diagram of the Evolutionary Robotics Plugin. The dark blue boxes with white letters depict the main functions. The white boxes are the features of the genetic algorithms which were the Evolutionary Algorithm and the Genome. To the left of the genome, in yellow, the classes relevant to the morphology are depicted. On the other side, blue depicts the control architecture of the neural network that was implemented in the modules/robots. Finally, in green the environment and corresponding fitness functions are defined. Dark grey boxes additionally present the factories enabling factory patterned instantiation of classes.


Bibliography


Medawar, P. B. (1952). An unsolved problem of biology. DOI: 10.1016/S0140-6736(00)99799-X.


Index

Adaptive radiation, 60
AFPO (Age-Fitness Pareto Optimization), 49
Allele, 15
Altruistic Aging, 38
Antagonistic pleotropy, 37
Baldwin effect, 18
Biomorphs, 5
Blind Watchmaker, 2, 5
catastrophic forgetting, 60
Chuang f1, 44
CMA-ES (Covariance Matrix Adaptation Evolution Strategy), 28, 52, 132
CPG (Central Pattern Generator), 65, 89
CPPN (Compositional Pattern Producing Network), 64
Development, 17
Diploidy, 12
Direct encoding, 88
Disposable soma, 37
DNA, 14
Edhmor (Evolutionary designer of heterogeneous modular robots), 88
Elitism, 24
Embodied evolution, 122
Embodiment, 121
Energy autonomy, 101
Error catastrophe, 48, 61
Evo-devo, 17
Evolutionary algorithm
  Extinction events, 38, 52
  Generational, 24
  Genetic algorithm, 23
  Speciation, 52
  Steady state, 23, 24, 67, 79, 88, 108
Evolutionary computation, 1, 11
Evolutionary games, 19
Evolutionary robotics, 1, 3, 5
Evolvability, 3, 20
Evolvability of mortality, 59
Exploration vs exploitation, 3, 24, 87, 88, 99, 142
Extended phenotype, 11
Fiducial marker, 123
Fitness, 13, 77
  Absolute, 77
  Inclusive, 13, 21, 57
  Landscape, 13
  Personal, 13, 35, 36
Gene pool, 12
Generative encoding, 68, 78, 87, 88, 102, 107
Genes
  Homologue, 18
  Paralogue, 16
Genetic drift, 16
Genotype, 17
Genotype to phenotype mapping, 5, 13, 17, 87, 88
H-IFF (Hierarchical If-and-Only-If problem), 26, 35, 40
Hedgehog, 18
Heterochrony, 17
Knifefish, 127
L-System, 65, 68, 75, 76, 79, 89, 107
  Context sensitive, 69, 80
Light tracking, 112
Longevity, 35
Lotka-Volterra, 19
Maximum parsimony, 2, 21, 23
Modular robot, 101, 102
  Automated assembly, 123
  Simulated, 88
Modular robots, 63
Modularity, 65, 75
Modules
  Cube, 90, 103
  Flower, 103
  Servo, 89, 103
  Solar, 103
Morphogen, 17
Morphogens, 76
Morphological computation, 127
Mortality, 35
  Extrinsic, 36, 58
  Intrinsic, 35, 36, 58
Mutation accumulation, 37
Mutation accumulation theory, 37
Mutation rate threshold, 15
Mystery of mysteries, 11
NEAT (Neuroevolution of Augmenting Topologies), 64
Neuroevolution, 64, 108
ENN (Evolving Neural Network), 64
Octopus, 11, 36, 65
Open-loop, 69, 102
Phenotype, 16
Photoreceptors, 22
Phototropism, 102
Phyllotaxis, 78, 102
Phytomorphogenesis, 75
Phytomorphology, 75
Pleiotropy, 17, 37
Premature convergence, 23, 24
Protein sequence, 21
Quasispecies, 14, 39
Reality gap, 3, 122, 126, 127
Schwefel, 53
Selection, 16
Self-similarity, 75
Senescence, 35, 36
Programmed vs non-programmed, 38
Sequence space, 14
Sessile, 76
Soft robotics, 127
Spatial model, 28, 41, 45
Speciation, 14
Survival of the fittest, 16
Traversability, 15
Trophic structure, 29, 75
Undulation, 128
V-REP (Virtual Robot Experimentation Platform), 63, 66, 87, 103
Watchmaker, ix, 2, 145
Zero-evolvability state, 16, 38, 58