Charles University in Prague

Faculty of Mathematics and Physics

DOCTORAL THESIS

Peter Krčah

Evolution and Learning of Virtual Robots

Department of Software and Computer Science Education

Supervisor of the doctoral thesis:	RNDr. František Mráz, CSc.
Study programme:	Computer Science
Specialization:	Artificial Intelligence

Prague 2016

First of all, I would like to express my great gratitude to my supervisor RNDr. František Mráz, CSc. for his patient guidance, countless reviews of the text and the friendly working atmosphere during the years of my doctoral study.

I would also like to thank my life partner Petra for her continuous support, encouragement and patience during the period of my studies without which this thesis would not have been possible. Finally, I would like to thank my parents and my brother for supporting me in this work and in all other aspects of my life.

Název práce: Evoluce a učení virtuálních robotů

Autor: Peter Krčah

Katedra: Katedra softwaru a výuky informatiky

Vedoucí disertační práce: RNDr. František Mráz, CSc.

Abstrakt: Evoluční robotika využívá evoluční algoritmy k automatickému návrhu stavby těla i chování robotů. Tato práce přináší dvě metody do oblasti automatického návrhu virtuálních robotických organizmů. První metoda ukazuje přírodou inspirovaný algoritmus, který umožňuje virtuálním robotům měnit jejich morfologii učením v průběhu života. Vysvětlujeme, jak tato morfologická plasticita umožňuje vyvíjet organizmy, které mohou dynamicky měnit svoji stavbu těla podle prostředí, do kterého jsou umístěny. Dále ukazujeme, že učení redukuje výpočetní čas potřebný k evoluci robota s předem určenou hodnotou fitness. V druhé části popisujeme pro vybrané problémy evoluční robotiky, jak změna zaměření evoluce z původního cíle na hledání libovolných nových forem chování může předejít předčasné konvergenci do lokálního minima (tato technika je známá jako Novelty search).

Klíčová slova: Evoluce virtuálních organizmů, koevoluce stavby těla a chování robota, morfologická plasticita, neuronové sítě, učení

Title: Evolution and Learning of Virtual Robots

Author: Peter Krčah

Department: Department of Software and Computer Science Education

Supervisor: RNDr. František Mráz, CSc.

Abstract:

Evolutionary robotics uses evolutionary algorithms to automatically design both body and controller of a robot. We describe two contributions to automated design of virtual robotic creatures. First, we introduce a nature-inspired method that allows virtual robots to modify their morphology through lifetime learning. We show that such morphological plasticity makes it possible to evolve robots that can dynamically adjust their morphology to the environment they are placed into. We also show that by reshaping the fitness landscape, learning reduces computation cost required to evolve a robot with a given target fitness even in a single environment. In the second contribution, we show that for certain problems in evolutionary robotics, premature convergence to local optima can be avoided by ignoring the original objective and searching for any novel behaviors instead (a technique known as Novelty Search).

Keywords: Evolution of Virtual Creatures, Body-brain Coevolution, Morphological Plasticity, Neural Networks, Learning

Contents

Chapter 1

Introduction

As robots are becoming more widely used in today's society, techniques used for their design are gaining importance. Traditional methods of design where an engineer designs both physical structure and controller of the robot are being increasingly supplemented by automated techniques made possible by recent advances in machine learning and automated manufacturing methods such as 3D printing. Such advances are bringing the field of robotics closer to a point when full end-to-end automation of robot design and construction will be possible. Once such level of automation is accomplished, the task for the human designer will be to provide a high-level description of the tasks which the robot should perform and a set of constraints on robot's body and the target environment. Both physical structure and control system of the robot will be designed automatically and the resulting robot will be manufactured. While this level of automation is is still years away, many of the building blocks have already been achieved.

One of the most promising approaches to automatic robot design are methods inspired by natural evolution. The astounding diversity and level of adaptation of organisms found in nature is a testimony to the power and flexibility of autonomous evolutionary processes. The field of evolutionary robotics is motivated by the hope that this power can be harnessed and used for automated design of robots as well. The fact that evolutionary algorithms can be used for designing both body structure and behavior of simulated robots (also called virtual creatures) has first been demonstrated in a landmark paper published by Sims [84] in 1994. Evolutionary robotics has been an active area of research since then (an overview of results achieved so far is provided in Section 2.1, a more comprehensive version also published in Krčah [44]).

This thesis builds on results presented in my master thesis [40] and several subsequent publications [41, 42, 43, 46] which introduced a new algorithm for evolution of virtual creatures called HierarchicalNEAT (see Section 2.4 for a description of the algorithm). In 2014, virtual creatures evolved using the new algorithm have placed second in a competition of similar projects [48] and software developed as part of the thesis (ERO framework—Framework for Evolution of Robotic Organisms) has since been used by other researchers internationally for further research on virtual creatures [58, 39, 63].

This thesis contributes two new methods to the field of evolutionary robotics. First, we study the impact of morphological plasticity on evolution of virtual robotic creatures. Nature provides many examples where animals benefit from being able to adjust their morphology to the environment they are placed into. Such morphological plasticity often allows animals to increase their chances for survival and reproduction compared to animals not capable of such adaptation. However, the effects of such morphological plasticity have not yet been studied in the context of evolutionary robotics. The first contribution of this thesis is that we show (motivated by examples from nature) that allowing robots to make even minor alterations to their morphology during their lifetime through learning can significantly reduce computational cost required to evolve a robot capable of successfully solving given task. Moreover, we show that such plasticity makes it possible to evolve creatures capable of adapting their morphology to different environments to increase their fitness [49].

In the second contribution we address a common problem in evolutionary robotics and evolutionary computation in general—premature convergence to a local optimum in the search space. We show that when the problem is inherently deceptive, suitable body structure and behavior of a robot can sometimes be discovered significantly faster by ignoring the original objective completely and instead searching for virtual creatures exhibiting any previously unseen behavior [45, 46, 47, 50, 51]. These results are based on a method called Novelty Search recently introduced by Lehman and Stanley [57] initially in the context of evolution of neural network controllers.

1.1 Publications

Partial results have been presented in the following publications and a competition (at the time of writing, results have been cited by 44 publications according to Google Scholar, 14 according to Scopus and 4 according to Web of Science, excluding self-citations):

Competitions

Peter Krčah. Virtual creatures evolved using speciation and historical markings, 2014. Runner-up in Virtual Creatures Contest, GECCO 2014: conference on Genetic and evolutionary computation `http://www.cs.utexas.edu/~joel/virtual_creatures_contest/`.

Book Chapters

[1] Peter Krčah. Solving deceptive tasks in robot body-brain co-evolution by searching for behavioral novelty. In Tauseef Gulrez and Aboul Ella Hassanien, editors, *Advances in robotics and virtual reality*, volume 26 of *Intelligent Systems Reference Library*, pages 167–186. Springer, 2012.

[2] Peter Krčah. Evolučný návrh robotických organizmov. In *Umelá inteligencia a kognitívna veda I*, pages 195–229. STU Press, Bratislava, Slovak Republic, 2009. Editors: V. Kvasnička, J. Pospíchal, Š. Kozák, P. Návrat and P. Paroulek.

Conference Proceedings

[3] Peter Krčah. Adaptation of virtual creatures to different environments through morphological plasticity. To appear in *SAB 2016: The 14th International Conference on the Simulation of Adaptive Behavior*, Lecture Notes in Computer Science (Subseries: Lecture Notes in Artificial Intelligence). Springer, 2016.

[4] Peter Krčah. Effects of speciation on evolution of neural networks in highly dynamic environments. In Youssef Hamadi and Marc Schoenauer, editors, *Learning and Intelligent Optimization*, volume 7219 of *Lecture Notes in Computer Science*, pages 425–430. Springer, 2012.

[5] Peter Krčah and Daniel Toropila. Riešenie zavádzajúcich úloh v koevolúcii ovládania a morfológie robotov pomocou hľadania noviniek v správaní. In Jiří Jelínek and Radim Jiroušek, editors, *Znalosti 2011.* VŠB-Technická univerzita Ostrava, Fakulta elektrotechniky a informatiky, 2011.

[6] Peter Krčah. Solving deceptive tasks in robot body-brain co-evolution by searching for behavioral novelty. In *10th International Conference on Intelligent Systems Design and Applications (ISDA)*, pages 284–289. IEEE, 2010.

[7] Peter Krčah and Daniel Toropila. Combination of novelty search and fitness-based search applied to robot body-brain co-evolution. In *Proceedings of the 13th Czech-Japan Seminar on Data Analysis and Decision Making in Service Science, Otaru, Japan*, pages 1–6, 2010.

[8] Peter Krčah. Towards efficient evolutionary design of autonomous robots. In Gregory S. Hornby, Lukas Sekanina, and Pauline C. Haddow, editors, *ICES '08: Proceedings of the 8th international conference on Evolvable Systems: From Biology to Hardware*, volume 5216 of *Lecture Notes in Computer Science*, pages 153–164, Berlin, Heidelberg, 2008. Springer-Verlag.

[9] Peter Krčah. Towards efficient evolution of morphology and control. In *GECCO '08: Proceedings of the 10th annual conference on Genetic and evolutionary computation*, pages 287–288, New York, NY, USA, 2008. ACM.

[10] Peter Krčah. Evolving virtual creatures revisited. In *GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, page 341, New York, NY, USA, 2007. ACM.

Research presented in this thesis was supported by Charles University Grant Agency under contract no. 9710/2011 and under Grant-No.358/2006/AINF/MFF.

1.2 Structure

The thesis is organized as follows. Chapter 2 provides an overview of evolutionary robotics including different methods of designing genetic representation of robots and methods how robots evolved in simulation can be constructed in reality. The chapter continues by introducing software used for all experiments in this thesis (a framework called Evolution of Robotic Organisms, or ERO) and the representation of virtual creatures used in this thesis. Chapter concludes with the description of HierarchicalNEAT algorithm—an efficient algorithm for co-evolution of body and control system of virtual creatures. Chapter 3 introduces a new method of accelerating evolution using morphological plasticity achieved through lifetime learning. Chapter 4 then describes results showing how novelty search can be used to evolve virtual creatures with highly deceptive problems. Conclusions are provided in Chapter 4.6, followed by two appendices. First appendix (Appendix A) contains algorithms and configuration parameters for generating and mutating virtual creatures. The second appendix (Appendix B) provides an overview of supplementary materials provided on a CD attached to the thesis.

Chapter 2

Background

This chapter provides an introduction to the area of evolutionary robotics and to some of the underlying methods used in the rest of the thesis. The chapter starts with an overview of recent advances in evolutionary robotics, including different methods for generating robots from their genetic description (Section 2.1.1) and the problem of transferring evolved behaviors to real robots (Section 2.1.2). More comprehensive review of evolutionary robotics can be found in Krcah [44]. In the next section we describe in more detail the structure of virtual creatures used in the rest of the thesis (Section 2.2), the ERO framework (software used for all experiments presented in this thesis) and the algorithm used to evolve both body and behavior of virtual creatures efficiently (Section 2.4).

2.1 Evolutionary Robotics

Evolutionary robotics is a technique for designing robots using methods inspired by evolution of natural organisms. While evolutionary computation is a relatively young field (started just four decades ago [29, 78]), it has been used already to solve difficult problems in a range of domains such as automated design of electrical circuits [38], invention of new algorithms for quantum computers [87] or solving open problems in mathematics [88]. In many cases evolutionary algorithms have been shown to be able to find better solutions to a problem than a human designer [38, 3]. While evolutionary computation has been applied to many domains, the similarity with nature is perhaps most apparent in the field of evolutionary robotics. Robots have a lot in common with their living counterparts: both must solve tasks in a world governed by the same physical laws, both have a body and a control system which are both optimized by evolutionary processes. These similarities make robotics a very tempting area for application of evolutionary algorithms. An interesting consequence of these similarities is that robots designed using evolutionary algorithms often resemble animals evolved in nature in both structure of their bodies and their behavior (examples can be found in virtual creatures evolved by Sims [84] two decades ago, up to recent works with damaged robots by Cully et al. [14] or soft robots in [11]).

The term *evolutionary robotics* was first used in 1993 by Cliff at al. [12]. The study of evolutionary robotics started from two different directions. In the first direction, the goal was to evolve control system of a real robot with fixed body structure (using robots such as Khepera or e-puck) and to solve the problem of crossing the *reality gap*, i.e. ensuring that the behavior of a robot evolved in simulation can be maintained after transferring it to a real robot [73]. The second direction was to evolve robots purely in simulation without attempting to construct them in reality. This line of research approached evolutionary robotics from the point of view of artificial life. Since projects in this category did not attempt to construct physical robots, they were free to design robots with much richer set of features and to evolve not just the behavior of the robot but also the structure of its body. Properties of the robot and of the fitness function used to measure its performance were in this case limited only by the creativity of the researcher and constraints imposed by the physics simulation. The first landmark paper in this area by Sims in 1994 [84] introduced evolved virtual creatures performing simple tasks such as jumping, swimming, land locomotion or following an object. The striking similarity of evolved creature to organisms in nature continues to inspire researches to this day. In recent years, these two research directions are starting to converge in projects which attempt to evolve both body and behavior in simulation and then build the resulting robot in reality [31, 62, 101, 27].

2.1.1 Representation of Robots in Evolutionary Algorithms

The ultimate objective of evolutionary robotics (and evolutionary algorithms as such) is to automate the design of robots and bring the complexity of their behavior and morphology closer to the complexity found in living organisms. Evolution in nature is capable of discovering extremely complex organisms made up of trillions of cells. How can an evolution efficiently navigate search space of such enormous size? It would be impossible for evolution to optimize each attribute of each cell directly (and independently). The answer to this question lies in the fact that in evolution of natural organisms, the genetic representation (or *genotype*) is very different the physical form of an individual (its *phenotype*). Genotype is the primary carrier of information, it is inheritable and it is acted on by mutation and recombination. Phenotype, on the other hand, is the body of a living individual which has developed from a single cell through ontogenesis. The relationship between the genotype and the phenotype of an individual forms a crucial component of evolution in nature and increasingly in simulated robots as well. A non-trivial mapping between genotype and phenotype of an animal or a robot can greatly simplify and restructure the space of all solutions, making it much easier for evolution to navigate. For instance, genotype could be constructed in a way where a single mutation can duplicate a limb of a robot (including all of its properties), making it possible to reuse previously discovered element in a different part of the robot. Evolution could then reuse elements discovered through a previous long search process without having to rediscover them from scratch again. In nature, perhaps the most striking example of a compact description of a complex object is human brain, whose hundred billion neurons are described using DNA molecules containing just tens of thousands of genes.

In order for the genotype to phenotype transcription to increase the ability of organisms to evolve it must have a set of attributes [73]. The first is *expressive power*, i.e. ability to represent a wide range of different phenotypes. In evolutionary robotics this includes the structure and physical properties of a robot (its *morphology*), the control system of a robot, or even properties of the transcription process itself. Expressive power increases diversity of robots accessible to evolutionary search, making it easier to find a robot better adapted to a given task. For example, if both morphology and control system of a robot are part of the genotype, evolution can optimize both together, allowing morphology and control system to cooperate when solving a task.

The second attribute is *compactness*. The relationship between complexity of the genotype and the complexity of the phenotype should be weak, so that even simple genotypes should be able to represent complex phenotypes. This attribute is important because evolution searches the space of all possible genotypes, not the space of all possible phenotypes. Smaller genotypes results in search space with smaller number of dimensions, which means it can be searched more effectively. There are several approaches for achieving compactness of the genotype, but most approaches use the idea of modularity and symmetry to make it possible to use a single gene to represent many parts of a robot.

Another important attribute that genotype to phenotype encoding should address is *evolvability*, the ability of the encoding to not just generate diversity of creatures through mutations, but to generate diversity that is adaptive. Success or failure of an offspring robot is influenced by (1) the shape of the fitness landscape in the vicinity of the parent robots (2) on genetic operators (which alter the offspring in some way) and (3) on the method used to transcribe genotype to phenotype, which influences how changes in genotype affect the phenotype.

The most straightforward method of representing robots in evolution is using *direct encoding*. In direct encoding, structure of genotype is identical to the structure of the phenotype and transcription from genotype to phenotype is therefore reduced to a copy operation. There are many cases when direct encoding is sufficient to represent a robot (e.g. when the problem is simple and solution requires only a small number of elements that need to be optimized). The main advantage of this approach is ease of implementation, which is one of the reasons why some projects in evolutionary robotics use it [62, 7].

A more powerful set of methods for representing robot phenotypes are inspired by formal grammar systems. First use of grammars for modelling of biological systems can be found in the works of Lindenmayer [61], who used rewriting systems to model the growth of plants. Resulting models are also known as *L-systems*. While the original L-systems have been used to model biological systems, they can be used to describe a wide variety of different structures including robots. An example of a very simple grammar could be two rules "A→B" and "B→AB" and a starting symbol "A". Applying rule "A→B" results in each occurrence of symbol "A" to be replaced by symbol "B". Repeated application of these two rules (starting for a selected starting symbol) results in sequence A → B → AB → BAB → ABBAB → BABABBAB → . . . (in each step, a rule is applied to each symbol in the string). Even two very simple rules and a starting symbol can create long strings of symbols with complex internal structure. Recursive nature of L-systems leads to self-similarity of generated structures which often result in complex fractal shapes. If strings of symbols produced by such grammar are interpreted as commands for building a robot, resulting robots can

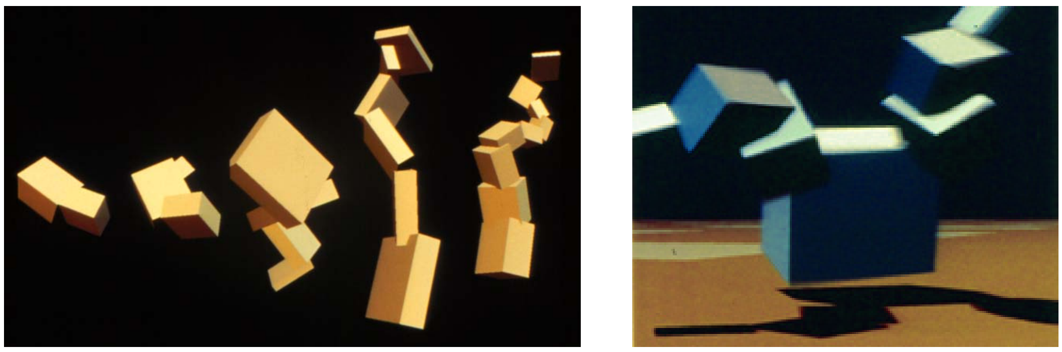

(a) (b)

Figure 2.1: **Virtual creatures evolved by Sims [84, 83].** Creatures were evolved for swimming (left) and land locomotion (right). Figure on the left shows how the swimming creature was incrementally improved in successive generations.

have very complex morphologies represented by a very compact L-system.

The first system where such recursive construction of phenotype of a robot was used can be found in work on Sims [84, 83]. The subject of evolution in Sims' work were virtual creatures living in a simulated three-dimensional world with simulated laws of rigid body dynamics. Phenotype of a creature is in this case represented by a rooted tree of nodes, each of which represents body part of a creature (a single block) and connections between nodes represent joints connecting the body parts. Genotype of a robot is the same as phenotype with one crucial difference: cycles are permitted in the graph representing the genotype. Transcription of genotype to phenotype starts in a designated root node and proceeds in traversing the genotype graph in depth-first order, adding copies of all encountered nodes to the phenotype tree. To prevent infinite recursion during this traversal, each node defines a recursive limit which determines the maximum number of times transcription is permitted to visit the given node during the depth-first traversal. Moreover, each genotype connection can have a *reflection flag* enabled which causes a mirrored copy of the subtree to be added to the phenotype graph. Recursive transcription and reflection flags allow compact representation of symmetric structures (see symmetric arms of creature in Figure 2.1b) and of appendages composed of repeated segments (see tail of the snake-like creature in Figure 2.1a). For instance, mutation can increase the number of segments in creature's tail by increasing value of a single parameter (recursive limit of the node forming the tail). Due to the flexibility and compactness of this encoding, it has become a common choice in evolution robotics research [77, 43, 58, 37, 52, 59, 67, 66]. Notably, Lessin [59] has recently extended Sims's model to allow virtual creatures to learn new behaviors while retaining previously learned behaviors (see Figure 2.2). Since experiments in this thesis use encoding based on encoding used by Sims, a more detailed description is provided in Section 2.2. For a step-by-step demonstration of how a virtual creature is constructed see Figure 2.11.

Transcription from genotype to phenotype used by Sims in many ways resembles L-systems. While genotype is not stored explicitly as a series of rewrite rules of a formal grammar, nodes and connections of the genotype chart can be viewed as symbols used in the grammar and the root node can be viewed as a starting

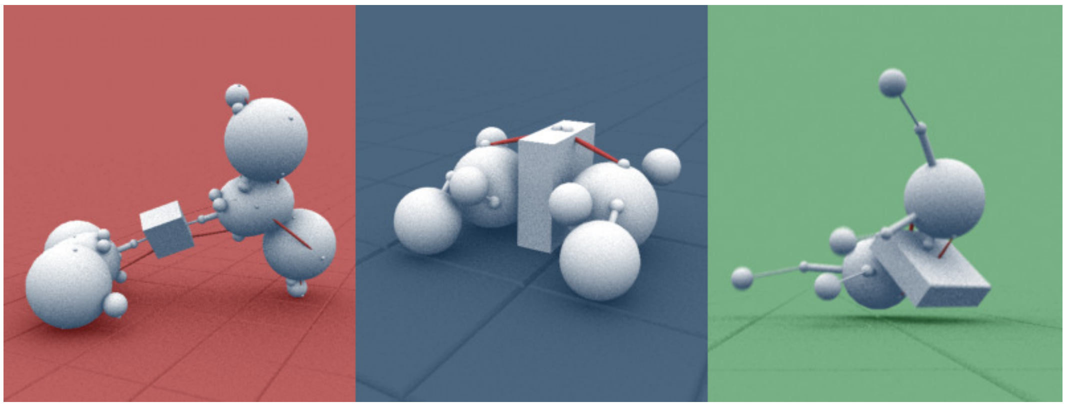


Figure 2.2: **Virtual creatures evolved to perform multiple tasks by Lessin [59]**. Actuation is provided by muscles (rendered in red color in the figure above) attached to two different body parts of a creature as opposed to the force being exerted directly at joints connecting two body parts.

symbol. A set of rewrite rules could then be defined to perform transcription of genotype to phenotype in a similar way to the model used by Sims.

There are examples of models where the robot is constructed directly from a set of symbols generated by a formal grammar. One such example is a work by Hornby et al. [33, 31]. In this work the string of symbols created by the grammar is used as a sequence of instructions for controlling a LOGO-like turtle. Examples of such instructions are: step forward, step back, turn, etc. Structure of robot's body is determined by the path of the turtle. Joints are created using a special *joint* command. Joints have only one degree of freedom and the angle of the joint is controller by an oscillator. Turtle can also use a stack for saving and restoring its state (symbols "[" and "]") and commands can be performed a fixed number of times (symbols "{" and "}"). The power of the grammar is increased by allowing parameterized and conditional rewrite rules. Example of rules defining genotype of a robot:

$$\boldsymbol{A}(n) \rightarrow \{\boldsymbol{A}(n-1)\}(n) \qquad \text{for } n > 2 \tag{2.1}$$
$$\rightarrow \text{ joint}(1)\boldsymbol{B}(2n) \text{ clockwise}(2) \qquad \text{for } n \le 2 \tag{2.2}$$
$$\boldsymbol{B}(n) \rightarrow [\boldsymbol{B}(n/4)] \qquad \text{for } n > 2 \tag{2.3}$$
$$\rightarrow \text{ joint}(1) \text{ forward}(1) \qquad \text{for } n \le 2 \tag{2.4}$$

Using the starting symbol of $\boldsymbol{A}(3)$, the following sequence of strings will be generated by applying the genotype above:

$$\begin{array}{ll}
\boldsymbol{A}(3) & \text{(using rule 2.1)} \\
\{\boldsymbol{A}(2)\}(3) & \text{(using rule 2.2)} \\
\{\text{joint}(1)\boldsymbol{B}(4) \text{ clockwise}(2)\}(3) & \text{(using rule 2.3)} \\
\{\text{joint}(1)[\boldsymbol{B}(1)] \text{ clockwise}(2)\}(3) & \text{(using rule 2.4)} \\
\{\text{joint}(1)[\text{joint}(1) \text{ forward}(1)] \text{ clockwise}(2)\}(3) &
\end{array}$$

Resulting robot constructed using the final generated string is shown in Figure 2.3a. Robots are evolved for locomotion in two- and three-dimensional environments. Some of the evolved robots were constructed in reality as well (see

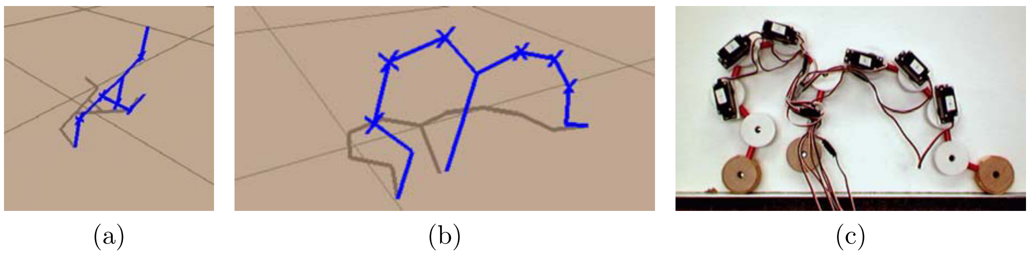
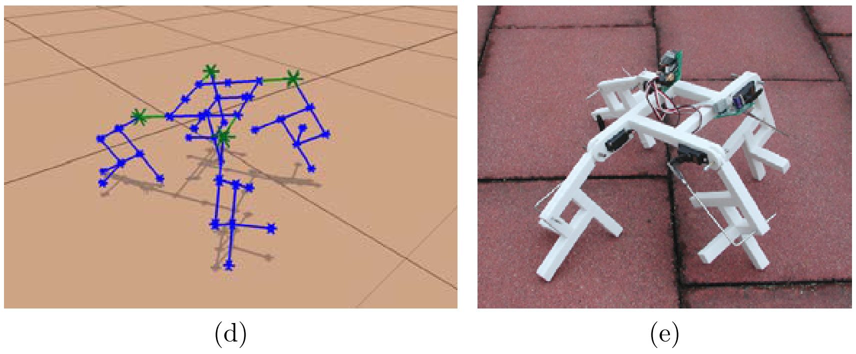
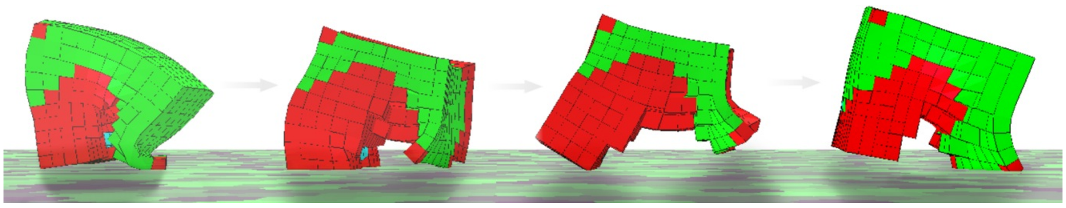
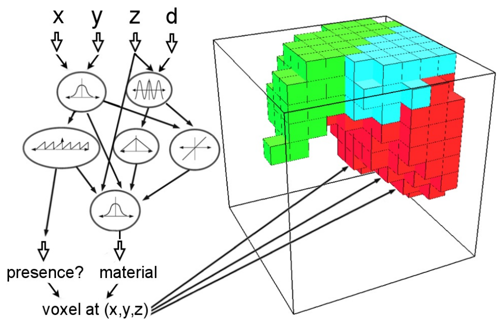



Figure 2.3: **Examples of robots evolved using L-system encoding in Hornby [33, 31]**. Robots were evolved in simulation (a, b, d) and subsequently constructed in reality (c, e).

Figure 2.3). Authors claim that encoding using L-systems achieved better results than direct encoding, by leveraging the ability to express repeated structures and symmetry.

All genotype to phenotype mappings described so far are limited in the number of independently operating body parts that the genotype describes (most robots shown so far consist of less than a hundred parts and often less than ten parts). While L-systems and Sims-like encodings can theoretically describe robots with large number of body parts, such robots have not yet been demonstrated using these encodings. One method that has achieved morphological complexity of 10^3 body parts recently (Cheney et al. [11, 10]) is based on the idea of describing robot morphology using a three-dimensional grid of voxels where each voxel can have different physical properties (see Figure 2.4). Some voxels provide actuation (red and green) while others are passive with different degree of stiffness. Genotype of a robot needs to specify the type of each voxel and whether the voxel is present or not. While this information can be encoded in the genotype directly for each voxel (a direct encoding), a much superior encoding is provided by a Compositional Pattern-Producing Network (or CPPN) evolved using a method called NEAT (see Section 2.4 for more details about NEAT). CPPN networks are a variation of artificial neural networks originally created to compactly represent two-dimensional images—network receives coordinates of a pixel on its input and produces an output representing color of the pixel using a set of internal neurons (neurons in this case use a wider set of activation functions, such as sine and Gaussian). When used to describe robots, CPPNs "paint" the morphology of a robot instead of painting an image. CPPN receives three-dimensional coordinates of a voxel and outputs the type of material and a flag indicating whether the voxel should be present or not. Transcription of genotype to phenotype thus consists


(a) Robot evolved for fast locomotion.



(b) Presence of each voxel and its type is determined by evaluating a CPPN for each voxel independently.

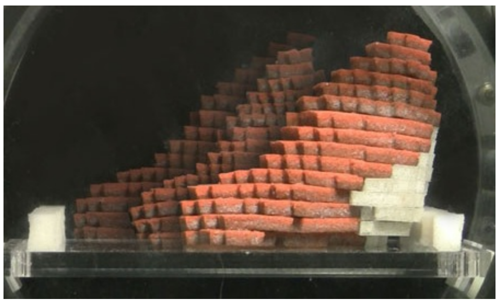

(c) Prototype of a physically constructed voxel-based robot actuated by changing air pressure [27].

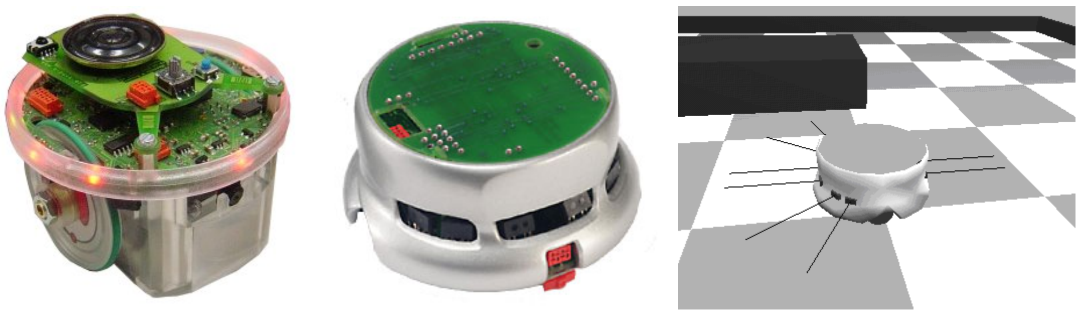
Figure 2.4: **Voxel-based robots with morphology defined by a CPPN network.** Green voxels periodically change their size with a fixed frequency, red voxels also periodically change their size but with reversed phase, light blue voxels are passive but deformable [11].

of evaluating CPPN for each voxel separately. Besides making it possible to describe complex morphologies compactly, the advantage of this approach is that CPPN is resolution-independent so robots at different resolutions can be generated from the same evolved CPPN. The CPPN-based model of robot morphology has recently been extended with more flexible actuation where oscillations of different voxels do not occur at the same time but spread through the body using simulated electrical signals [9].

The problem of how to best represent robots during evolution is an active area of research. Some of the other nature-inspired approaches include a method by Gruau based on simulation of cell growth [26, 25], methods based on gene regulatory networks [4, 18, 79] or methods based on simulating soft muscle tissue [60]. More detailed survey of methods is provided in Stanley et al. [93], Nolfi et al. [73] and Krcah [44].

2.1.2 From Simulation to Real Robots

The traditional way of designing physical robots using evolutionary algorithms is to use a robot with fixed structure and optimize only its behavior. A popular type of platform for such experiments is a small circular robot such as e-puck or Khepera (see Figure 2.5). These robots have circular footprint with diameter of 5-6cm and height of 3cm. They move using two wheels and they support a range of sensors. Several simulators are available for the robots as well (e.g. Webots). Such simple two-wheeled robot were used in a large number of early studies of evolutionary robotics (see [73] for examples).


(a) E-puck robot.

(b) Khepera II robot.

(c) Simulation of Khepera in Webots simulator.

Figure 2.5: **Examples of robots with a fixed structure.**

A more complex platform for evolutionary experiments is robotic dog called AIBO [20]. AIBO is a four-legged robot with 18 degrees of freedom, a set of sensors and a processor powerful enough to run evolutionary computation on the robot directly. This made it possible to use the robot for online evolutionary experiments in Hornby et al. [30]. Evolutionary search in this case was a fully automatic process requiring no human intervention. Fitness function was set up to evolve fast walking gaits. Authors claim that the evolved style of walking was faster than any walking style designed by a human designer.

Robots described so far all use a fixed construction. The first robotic project that attempted to break away from that constraint was Funes et al. [21]. The goal of the project was to evolve static LEGO structures capable of performing a task, such as being able to function as a crane holding a large weight. Simulator was used for actual evolution and resulting structures were then constructed and tested. While the resulting structures were static, this work paved way for future projects which attempted to build mobile robots.

One of the first examples of projects constructing evolved robots capable of locomotion was Hornby et al. [31]. While the main purpose of the work was to study genetic representation of a robot, some of the two- and three-dimensional evolved robots were successfully constructed in reality (see Figure 2.3).

Another early example of constructing robot based on evolved blueprint was project GOLEM (Genetically Organized Lifelike ElectroMachines) by Lipson et al. [62]. Robots were evolved offline in a simulator designed with an emphasis on making it possible to build resulting robots in reality and retain their performance. In project GOLEM, the robot is represented by a series of segments connected using ball joints. Robot is controlled by a neural network. Each neuron can be attached to one of the segments, in which case the length of the segment starts to be controlled by the current value of the neuron (in real robot this is realized by a linear actuator). Robots were manufactured from an evolved blueprint using a 3D printer, with motors and controllers manually inserted into the printed plastic skeleton afterwards (see Figure 2.6 for an example of resulting robots). Similar approach has been used recently Megaro et al. [65] in a method which allows casual users to design a robot using an intuitive editor—controller and blueprints for 3D-printing the robot are then created automatically by evolution in a simulated environment.

The ultimate goal of evolutionary robotics is to be able to fully automate

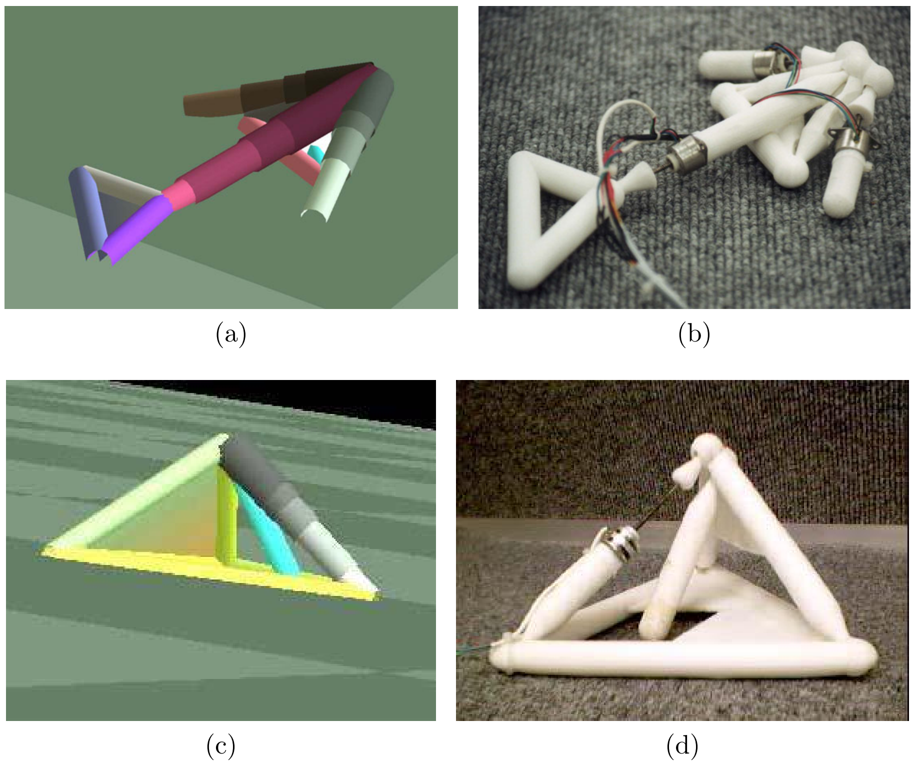
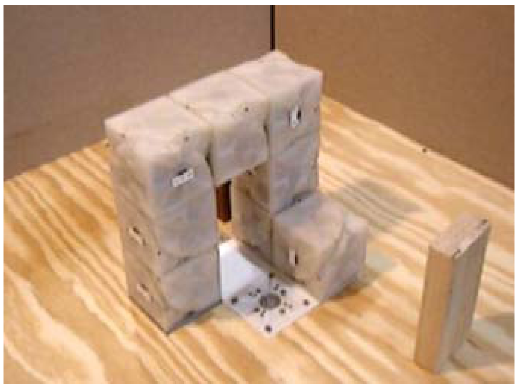

(a) (b) (c) (d)

Figure 2.6: **Examples of robots evolved for locomotion in GOLEM project [62].** Robots were evolved in simulation (left) and then constructed using a 3D printer (right).

design and manufacturing of a robot. A recent new approach to this problem uses 3D printed robots consisting of a grid of voxels of different types (e.g. static rigid voxels or flexible voxels). An early prototype of such printed robot has been demonstrated which can achieve locomotion using changes in ambient air pressure [27] (see Figure 2.4c).

An alternative approach to using 3D printers for building robots is to use a set of pre-manufactured building blocks that can be attached to each other to form the body of the robot. Several researcher teams have proposed methods of building such reconfigurable robot platforms (e.g. M-TRAN [70], Molecubes [101], Sambot [98] or SuperBot [82]). Zykov et al. [100] have demonstrated that it is possible to build a system where individual modules can rearrange themselves without human intervention. Each module contains a servomotor and electromagnets embedded in its walls which it can use to attach itself to other such modules. Robots constructed this way (see Figure 2.7a) have been shown to be able to self-replicate (recreate a copy of themselves). Studer et al.[94] has shown (in simulation) that an ecosystem consisting of large number of such cubical modules shows emergence of self-replicators, i.e. different types of chains of modules that copy themselves at the expense of other modules.

The need for including morphology of a robot in evolutionary optimization may be required even for robots with fixed structure. For example if a robot becomes damaged during normal operation (e.g. one of the servomotors stops functioning or part of its construction is damaged or broken off), it needs to adapt its behavior to the damaged morphology. In Bongard et al. [7], the subject of


(a)

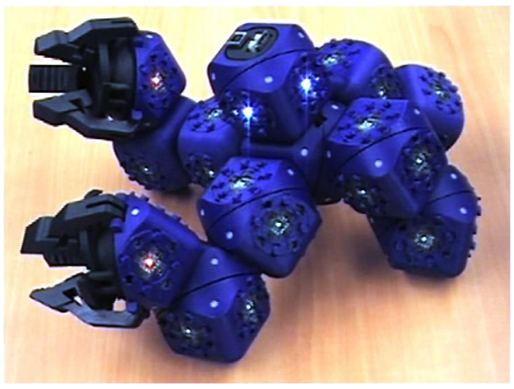
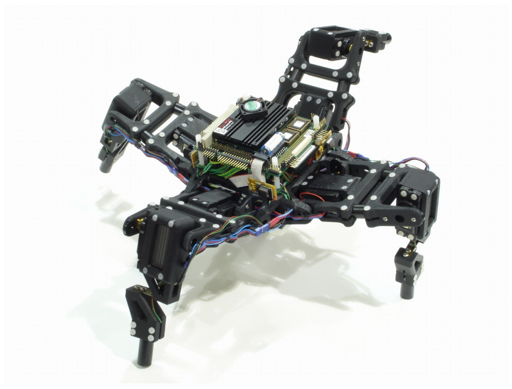

(b)

Figure 2.7: **Reconfigurable modular robots.** Self-replicating reconfigurable robots [100] (left) and open-source Molecubes platform for experimenting with reconfigurable robots [101] (right).


(a) Resilient robots by Bongard that use evolutionary algorithm to find the best model of their own morphology after damage occurs. [7].

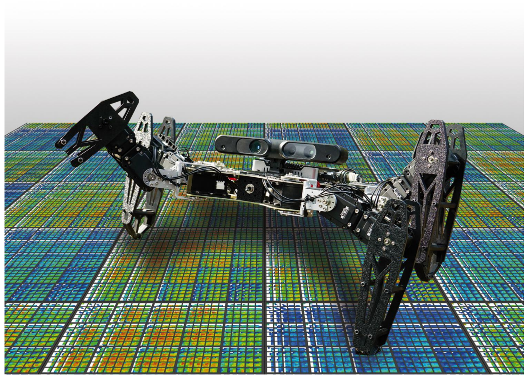

(b) Robot that can adapt to damaged morphology rapidly (in less than 2 minutes) by performing a small number of experiments.[14]

Figure 2.8: **Examples of robots with fixed structure that can adapt to different types of damage to their body.**

optimization was a four-legged robot actuated by 8 servomotors (see Figure 2.8a). Robot has 8 sensors, each reading the current angle of one of the servomotors, and two additional tilt sensors. The goal of the project was to make it possible for a robot to recover from damage by discovering the extent of damage automatically and adapting its gait to it. The robot infers changes in its morphology using only the sensors described above (no additional information about the damage is provided). The process of adaptation works incrementally: at the start of the experiment, robot performs a set of random movements and remembers sensor values received during those actions. In the first adaptation phase robot uses evolutionary optimization (in simulation) to find 15 models of its own morphology which best describe values measured by sensors. The second phase of adaptation then looks for the single best action that could be performed by a robot to find out which of the 15 models best represents the damaged robot. This action is then performed by the real robot. When this process is repeated 16 times, the robot finds enough information to correctly guess its new morphology. The best model is then used to find (again through evolution in simulated environment) the set of actions that move the robot forward most efficiently. The result of this process is a robot that learns how to move forward efficiently even after one or more parts of its body are damaged.

While the method tries to use the smallest possible number of fitness evaluations, its main disadvantage is that the amount of time required to perform evolutionary optimizations between tests with a real robot can be quite long. Cully et al. [14] recently proposed a method called Multi-dimensional Archive of Phenotypic Elites (MAP-Elites) that works around this problem by using evolutionary search to construct a *behavior-performance* map. For each behavior, the map stores the robot with highest performance discovered during the preliminary evolutionary search. In the case of hexapod robot (shown in Figure 2.8b), the behavior vector consists of 6 values (one per leg), each value representing the proportion of time a given leg spends touching the ground. After the damage occurs, the behavior-performance map is used to find the most promising controller for the robot and the controller is tested on a robot. After the test, the actual behavior and performance of the robot is used to upgrade the behavior-performance map using Bayesian optimization and new controller is found. While the method requires significant amount of computation time for the initial construction of the behavior-performance map (authors report two weeks of computation time on a single multi-core CPU), the map then allows the robot to rapidly (within two minutes) adapt to previously unseen damage.

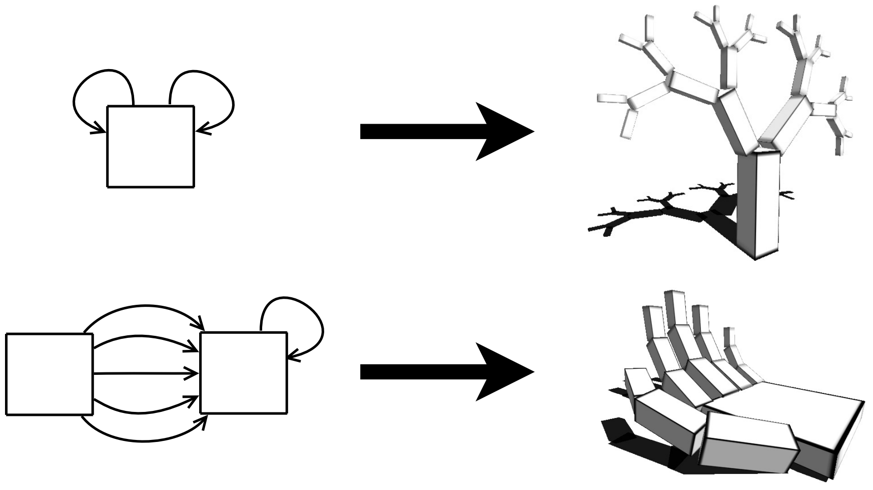


Figure 2.9: **Examples of manually designed creatures and their genotypes.**

2.2 Virtual Creatures

This section introduces the encoding of virtual creatures used in the rest of the thesis (detailed description is also available in [40]). Creature representation is inspired by the work of Sims [84, 83]. In the following text, the term *genotype* refers to the genome of a virtual creature (analogous to DNA in animals) and the term *phenotype* refers to the body of the creature constructed from the genotype.

Section 2.2.1 describes creature morphology and a procedure of constructing physical body of a creature from the genotype. Section 2.2.2 presents a distributed control system of creatures. Section 2.2.3 describes mutation and mating of creatures. Section 2.2.4 describes fitness evaluation. Finally, section 2.2.5 introduces a testing mechanism for early removal of faulty creatures.

2.2.1 Creature Morphology

Body of the creature is represented by a rooted tree of morphological nodes. Each node corresponds to a single body part (e.g. a box) and each connection between two nodes corresponds to a physical joint between two body parts (see Figure 2.9 for examples of manually designed creatures and their genotypes).

Creature phenotype is created from a corresponding genetic template (i.e. a *genotype*). Genotype is a directed graph (not necessarily a tree; cycles are permitted). In each genotype graph, one node is marked as the *root node*. Phenotype is created from genotype by first copying the *root node* and then recursively traversing connections in depth-first order and adding encountered nodes and connections to the phenotype graph. Since the genotype graph may contain cycles, recursive traversal could run indefinitely. To prevent this, each genotype node has a *recursive limit*, which limits the number of passes through the given genotype node. Each genotype node can thus be copied multiple (but finite) times to a phenotype graph. Each genotype connection also has a *terminal flag*. Terminal flag can be used to represent structures appearing at the end of chains or repeating units.

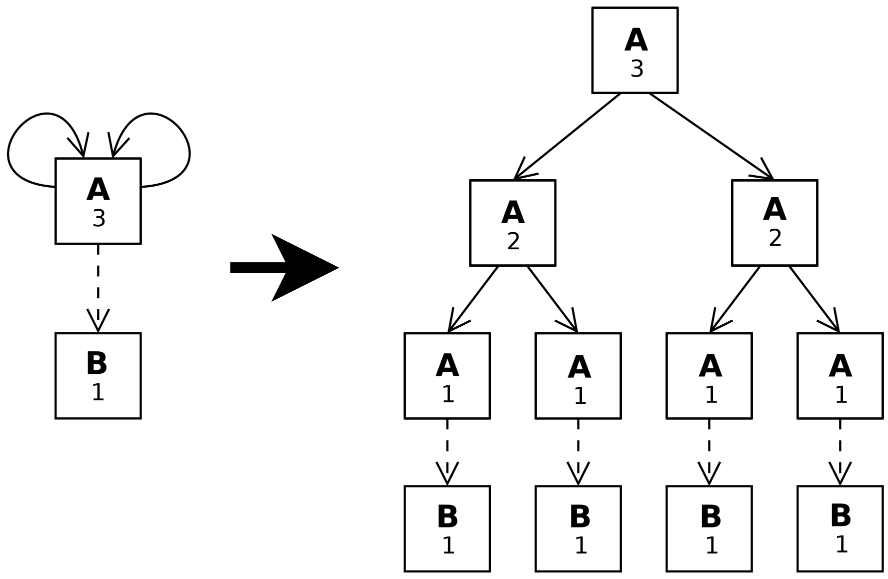


Figure 2.10: **Creature genotype (left) and phenotype (right).** Dashed line represents a terminal connection. Recursive limit of each node is shown under the node mark (A, B).

The transition from a genotype graph with a terminal connection to the corresponding phenotype graph is illustrated in Figure 2.10.

Both genotype node and genotype connection have a set of properties used for building their phenotype counterparts. Each genotype connection contains information about the position of the child node relative to its parent node. The position is represented by child and parent *anchor points*, relative *rotation*, *scaling factor* and a set of three *reflection flags* (one for each major axis).

Each of the two anchor points lies on the surface of the child or parent node. Position of an anchor point on the surface of the node is specified by two angles $\alpha \in [0, 2\pi]$ and $\beta \in [-\pi/2, \pi/2]$. These two angles specify (in polar coordinates) the direction of the vector originating at the center of the body part. Position of the anchor point is determined by the intersection of the vector with the surface of the body part (see Figure 2.12a). Local coordinate system is defined at each anchor point using the surface normal vector and two vectors tangent to the surface (see Figure 2.12b).

When building a creature from its genotype, child and parent nodes are first aligned, so that child and parent anchor points become identical and surface normals at anchor points become opposite. Afterwards, the child node is scaled by the scaling factor and rotated by the rotation parameter. Each enabled reflection flag causes a mirrored copy of the child node to be added to the phenotype graph (along with the original non-mirrored child node). All enabled reflection flags are always applied (if one, two or three reflection flags are enabled, two, four or eight mirrored copies of a child node are created in the phenotype graph, respectively).

All geometric transformations (such as scaling, rotation and reflection) are cumulative, i.e. they are applied to an entire subtree of the phenotype graph during its construction. For example, when the scaling factor of a connection doubles, the size of each node in the subtree started by this connection in the phenotype graph is doubled. The effect of individual morphological parameters

Joint type	**n**	**u**	**v**	**DOFs**
fixed	no	no	no	0
hinge	no	yes	no	1
twist	yes	no	no	1
hinge-twist	yes(2)	yes(1)	no	2
twist-hinge	yes(1)	yes(2)	no	2
universal	no	yes	yes	2
spherical	yes(1)	yes(2)	yes(3)	3

Table 2.1: **Joint types used for connecting parts of creature's body.** Values in columns **n**, **u** and **v** specify whether two body parts connected by the given joint are permitted to rotate freely around the given vector of the anchor point local coordinate system (as shown in Figure 2.12b). Numbers in parentheses show the order of non-constrained rotations. Last column shows the number of degrees of freedom for a given joint type.

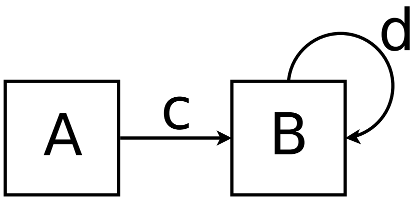
is further illustrated in Figure 2.11.

Each genotype node contains information about the *shape* and *size* of the resulting morphological node (in the case of a box, its dimensions are specified) and a *joint-type*. A joint-type defines properties of a joint connecting the current node and its parent in the phenotype graph. The following joint types are used: *fixed*, *hinge*, *twist*, *hinge-twist*, *twist-hinge*, *universal* and *spherical*. Each joint type is defined by a set of rotational constraints imposed on two connected body parts. Constraints are expressed in terms of vectors **n**, **u** and **v** in anchor point local coordinate system of the child node (as shown in Figure 2.12b). For example, two connected body parts can be allowed to rotate freely around the normal vector **n**, but are not permitted to rotate around tangential vectors **u** and **v** (the twist joint). Moreover, the order of non-constrained rotations is also important, because different order of torque application results in different joint behavior (this is the reason why both hinge-twist and twist-hinge joints exist). Table 2.1 defines individual joint types with respect to non-constrained rotations around vectors **n**, **u** and **v**.

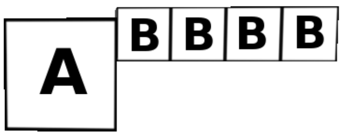
2.2.2 Control System

Creature's control system is distributed along its body. Each morphological node contains local sensors, effectors and a local controller. Besides local controllers, a single global controller (the *brain*) is also present to allow global coordination among organism parts. Global sensors and effectors could also be implemented, although they are not currently used. Local controller in a child node can also communicate with its parent node controller (neural connections in both directions are allowed). This way, a neural signal can spread through the organism body in a fashion similar to real organisms. An example of a creature with such distributed control system is shown in Figure 2.13.

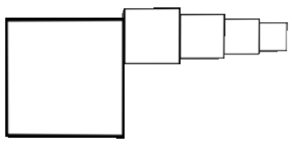
Controller is a processing unit, which receives input values and produces output values. Each controller has a set of *input ports* (which can receive signals from local sensors, output ports of parent/child controllers or output ports of the brain) and a set of *output ports* (which can send signals to local effectors, input



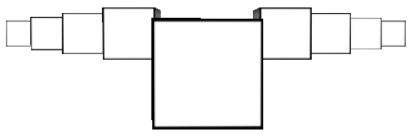
(a) Creature genotype. Recursive limits of nodes **A** and **B** are 1 and 4, respectively. Node **A** is the root node of the graph.



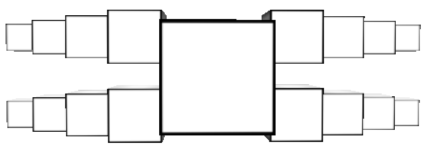
(b) Physical creature created from the genotype shown on the left. Four copies of node **B** are created during the recursive traversal of the genotype. Copies of node **B** are connected using connection **d**, while nodes **A** and **B** are connected using connection **c**.



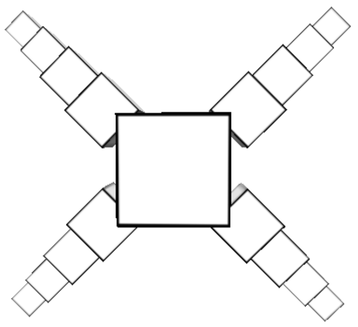
(c) Scaling factor of connection **d** has been changed from 1 to 0.8.



(d) Reflection flag for x axis of connection **c** has been enabled.

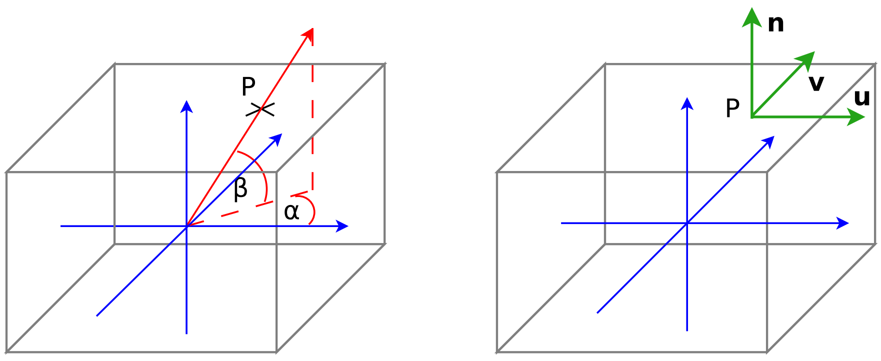


(e) Reflection flag for z axis of connection **c** has been enabled.



(f) One of three rotation angles of connection **c** has been changed from 0 to 45 degrees.

Figure 2.11: **Step-by-step construction of a creature.** The structure of the creature genotype (a) is fixed during all steps. Creature is constructed by successive application of scaling (c), reflection for x axis (d), reflection for z axis (e) and rotation (f).



(a) Position of an anchor point P on the surface of the body part is determined by two angles α and β (included in the genotype of the creature).

(b) Anchor point local coordinate system is specified by surface normal vector ($\mathbf{n}$) and two vectors tangent to the surface ($\mathbf{u}$ and $\mathbf{v}$).

Figure 2.12: **Joint placement.** Position of the anchor point is specified in polar coordinates (left). Local coordinate system is then defined for the anchor point (right).

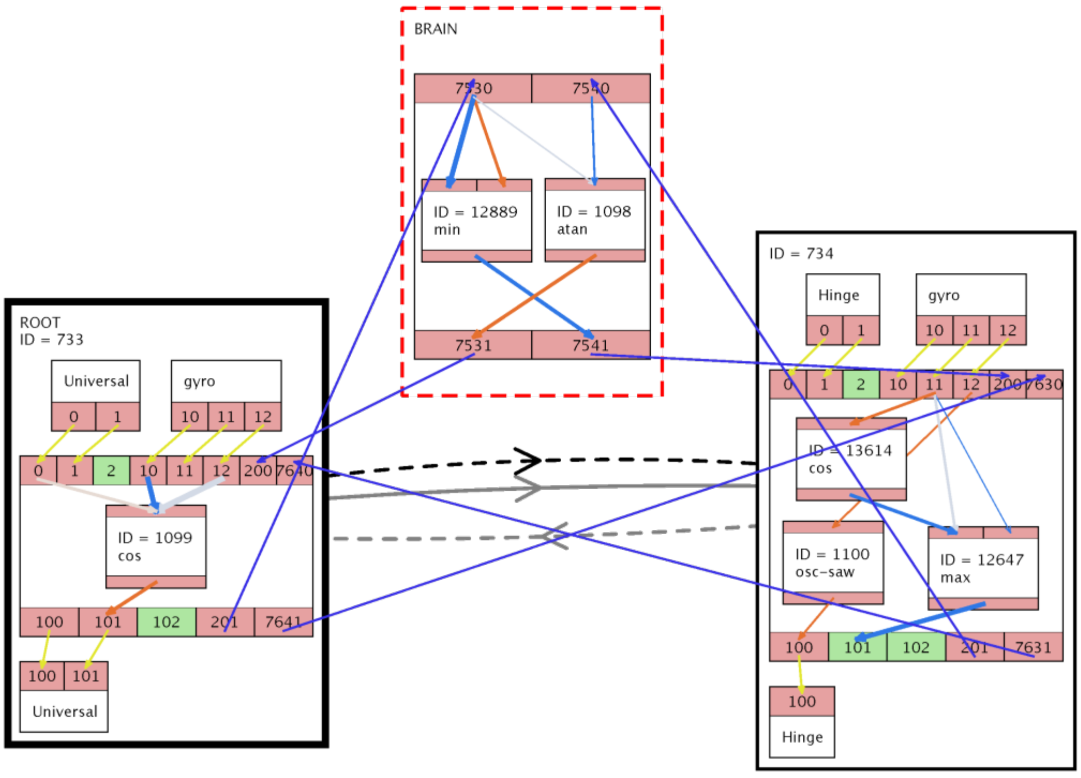


Figure 2.13: **Example of a creature genotype illustrating the distributed control system.** The brain (represented by the box on the top) is connected to local controllers of morphological nodes using neural connections (blue lines). Neural connections also connect the controller of the root node (bottom left) with controller of its child node (bottom right). Each morphological node contains local sensors (on the top of each box) and local effectors (on the bottom of each box).

ports of parent/child controllers and input ports of the brain).

In this thesis artificial neural network (ANN) controllers are used exclusively as controllers. Neurons in the ANN form a directed graph and are connected by weighted connections. Connection weights are limited to the range $[-2, 2]$ (maximum value of connection weight is referred to as $w_{\max}$ in the subsequent text). Furthermore, each connection can be enabled or disabled. Disabled connections are not used during the signal propagation (i.e. the only difference between disabling a connection and its removal is the possibility of re-enabling previously disabled connection). Disabled flag is subject to mutation and recombination. Each genetic operator must, however, ensure that each neuron of the resulting network has at least one enabled incoming neural connection and one enabled outgoing neural connection. This constraint is introduced in order to prevent genetic operators from disrupting the signal flow of the network. Only feed-forward networks are permitted. Feedback loops are prohibited and signal propagation evaluates neurons in the order given by a topological sort.

Sensors

Sensor is a part of the creature control system, which measures some property of a virtual world. Each sensor is contained in a specific morphological node. Sensors allow a creature to respond to changes in its environment. The value of each sensor is measured during each simulation step and propagated to the input port of the corresponding controller. While many sensors are possible, experiments in this thesis only use *the joint angle sensor*, which measures the current angle value for each degree of freedom of a joint. Each node contains a single joint sensor measuring the joint connecting a node with its parent.

Neurons

Neuron is the basic computing unit in an artificial neural network. Each neuron has a transfer function, which is one of *sigmoid, osc-saw, osc-wave* (see Table 2.2 for description). The set of transfer function has been selected based on experiments with different sets of transfer functions described in [40]. The neuron receives input values either from a controller input port (which can be connected to a sensor, the brain or another controller), another neuron's output or it simply receives a constant value. Neuron's output is connected to a controller's output port (which, again, can be connected to an effector, the brain or another controller) or to another neuron's input. Multiple connections can be connected to a neuron input. Neuron computes its value by summing weighted values of all enabled connections for each input and computing transfer function value on these inputs.

Effectors

Effector is a part of the creature control system which allows the creature to change some aspect of the virtual world. Only one type of effector, *the joint effector*, is currently used although other effector types could also be implemented. Joint effectors allow the creatures to move in the physical world by applying torques to the physical joints of the creature body parts. Each degree of freedom

Transfer function	**Description**
osc-saw(x)	$\varphi = \varphi + x \cdot dt/1.5$, return $2 \cdot \varphi_{frac} - 1$
osc-wave(x)	$\varphi = \varphi + x \cdot dt/1.5$, return $cos(2\pi\varphi)$
sigmoid(x)	$1/(1 + e^{-5(x-b)})$, where b is the bias

Table 2.2: **Transfer functions.** The output of all transfer functions is limited to the range $[-1, 1]$, to prevent signal congestion. The following notations are used in the descriptions: x denotes value of the input variable in the current simulation step and dt denotes elapsed simulation time in seconds since the last simulation step. φ_{frac} is the fractional part of φ.

of a joint is controlled by one effector output value. Effector receives its values from a controller.

The straightforward application of the effector values to physical joints has shown to be impractical, because unconstrained values often cause undesirable effects and instability in the simulation. Therefore, several transformations are applied to effector values before their application.

Effector values are first clipped to the range $[-1, 1]$ and then scaled by a factor proportional to the mass of the smaller of two connected body parts. This transformation limits the maximum size of a force to some reasonable value and consequently improves simulation stability.

Effector values are then smoothed by averaging previous ten clipped values. The average value is used as the torque applied to a joint of the creature. This modification eliminates sudden large forces and also improves stability of the simulation.

2.2.3 Genetic Operators

Algorithms and parameters used for random generation and mutation of virtual creatures are provided in Appendix A. An algorithm for performing recombination of two creatures into an offspring creature is part of HierarchicalNEAT algorithm and is described in Section 2.4.

2.2.4 Fitness Evaluation

This section describes the process of evaluating fitness of an individual creature. This process is common to all fitness functions (e.g. swimming or locomotion on the ground). Specific fitness functions are described in sections describing individual experiments (see Chapter 3 and Chapter 4).

Fitness of a creature is evaluated by first performing a validity test. The purpose of the validity test is to detect invalid creatures early, so that they do not consume computational resources during full-scale physical simulation. Validity testing is described in detail in section 2.2.5. If the creature is valid, it is placed in a virtual 3D world. ODE library [86] is used to simulate rigid body dynamics in the virtual world. Simulation proceeds in discrete simulation steps at the rate of 60 steps per simulated second. During each time step the following phases occur for each simulated creature:

1. Sensor values are set according to the current creature environment.

2. Neural signals are propagated throughout the distributed control system of the creature.
3. Torque is applied to each joint between body parts, according to the current value of the corresponding effector.
4. Physics simulation step is taken and the positions of all objects in the world are updated.

The simulation runs for a specified amount of time during which the creature is evaluated.

Physical Simulation

Open Dynamics Engine (ODE) is used to simulate the rigid body dynamics (including collision detection and friction approximation). ODE provides reasonably robust physical simulations and it proved to be sufficient for the purposes of the simulation of evolving creatures. Evolved creatures, however, often exploit errors and instabilities in the physics engine to their advantage. The following mechanisms were used to prevent such instabilities:

1. *Simulation watchdog.* During the fitness evaluation, relative displacement of each two connected parts of each creature is watched and when the maximum displacement rises above a specified threshold (which suggests that the creature starts to behave unrealistically), the creature is immediately assigned zero fitness and its simulation is stopped. Individual body parts are also prohibited to reach angular or linear velocity above the specified threshold. Creature with such body parts is assigned zero fitness. Finally, a simple oscillation detector is used to prevent creatures from moving using unrealistically rapid oscillations (e.g. when position of creature's body part rapidly alternates between two states). If an oscillating creature is detected, it is also discarded.

2. *Validity testing.* Validity test has been introduced to quickly remove organisms, which are likely to abuse the physics engine. The validity test is described in detail in Section 2.2.5.

3. Physical forces used by creatures to move, are carefully controlled so that creatures cannot apply forces, which would result in unrealistic simulation (as described in Section 2.2.2).

4. *ODE configuration.* ODE was configured to be as robust as possible to different kinds of creature behavior, while retaining realism of the simulation. In ODE physical engine, this was accomplished by the proper setting of the CFM (set to 0.01) and the ERP (set to 0.2) parameters (see [86] for details).

The water environment is simulated by adding viscous forces and turning off gravity. ODE engine does not include a simulation of the viscous force. The viscous force is, therefore, computed by a simple approximative method: the viscous force is added for each face of each box in the direction opposing the surface normal, proportional to the surface area and the velocity of the face projected to the face normal vector.

2.2.5 Validity Testing

As mentioned in the previous section, many organisms tend to exploit properties of the physical simulation to their advantage. However, several pre-simulation tests can be carried out to discover such abusive creatures early. The following creature properties are tested before the simulation starts:

1. The count of nodes in the phenotype graph must not exceed specified limit (organisms with a large number of nodes tend to be unstable).
2. In their initial position, none of the body parts is allowed to overlap with any other body part except for body parts connected by joints.
3. The volume of each body part must be larger than the specified threshold (extremely small body parts also cause instability in the physical engine).

Such simple validity tests are much faster to compute than the entire physical computation. It is therefore preferable to exclude unsuitable creatures as early as possible, so that they do not consume computer resources later. Therefore, every newly introduced genotype (created by the creature generation, mutation or recombination) is tested immediately and if the test fails, the genotype is discarded and a new genotype is introduced followed by the same testing procedure. This process repeats until a genotype passing the test is introduced or a specified number of unsatisfactory genotypes is created, in which case the last created genotype is returned.

The process of testing slows down genetic operators, but it significantly increases the overall computation speed, since genotypes do not have to be evaluated by the computationally expensive fitness function in order to be discarded.

Another approach would be to completely avoid generating faulty creatures. This approach results in a more effective and faster computation and it is used often (e.g. the scaling of effector forces described in Section 2.2.2). There are few cases, however, when avoiding faulty creatures would inadequately complicate the operations of generation, mutation and crossing, making them less flexible (e.g. avoiding self-penetrating creatures). In such cases, the use of validity testing is a reasonable compromise.

2.3 ERO Framework

ERO (Evolution of Robotic Organisms) is an evolutionary framework developed as a student software project at Faculty of Mathematics and Physics, Charles University in Prague. Since its development, ERO has been used in several publications and theses by teams at Charles University [40, 39, 43, 44, 50, 45, 51, 46, 47, 48, 49] and also outside of the university [58, 53, 63].

At its core ERO is a generic Java-based parallel computation system. A single parallel computation in ERO is called *a project*. For each project, ERO provides a user-friendly way to configure the project and an interactive environment for observing progress of the computation.

ERO consists of four separate applications:

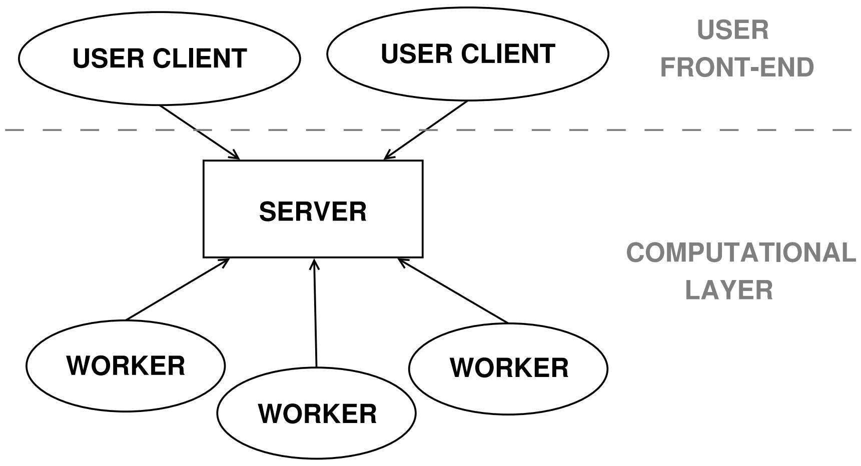


Figure 2.14: **Overview of main ERO components.**

- a server application (or *the server*) responsible for keeping the state of the project and synchronizing work among all computational clients,
- a computational client (or *the worker*) responsible for performing a single computation,
- a graphical user interface (GUI) application (or *the user-client*), and
- a command-line client which can be used to automate starting and stopping of projects and downloading project results for further analysis.

The relationship between these components is illustrated in Figure 2.14 (user-client refers to either command-line or GUI user client). Server and workers together form the computational layer. The goal of the computational layer is to receive a project from the user-client and to compute its result in a parallel fashion, while continuously providing preliminary computation results to all user-clients currently watching the project. The computational layer is described in more detail in the following section. User clients (either command-line or graphical) provide interactive interfaces to ERO. Graphical user-client can be used to configure new projects, send configured projects to a server, view project results, pause/unpause project computation or remove existing projects from a server. Each user-client can connect to any project on any server.

While ERO can perform arbitrary parallel computations, primary use of ERO has been to serve as a tool for experiments with evolution of neural networks and virtual creatures. In this case the evolutionary algorithm runs on the server and CPU-intensive tasks (such as fitness computation or complex genetic operators) are sent to the workers for computation.

2.3.1 Parallel Computation

The computational layer consists of the server and workers. The server runs projects and coordinates user-clients and workers. Several projects may run on a server simultaneously, sharing the same computational resources. Worker is a computational client, which receives a small part of computation from the server, computes it, and returns the result to the server (server provides the result to

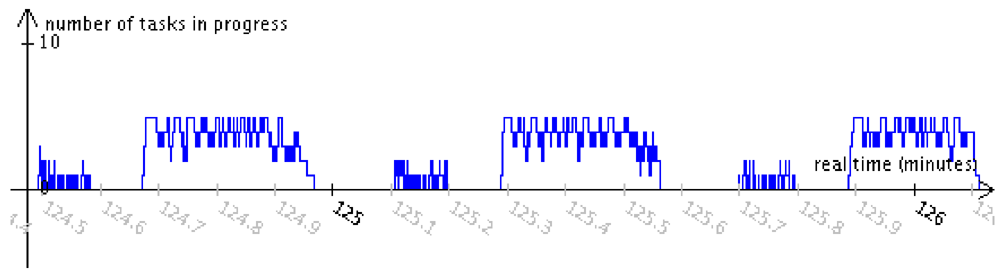


Figure 2.15: **Computational efficiency of evolutionary computation using 5 workers.**

the project which created the task). Number of workers connected to the server is not limited. To take advantage of multi-core processors, multiple workers can be started on the same computer (one per CPU core).

User-client offers various statistics about the parallel computation. Besides basic information (number of generated/finished tasks, total runtime, etc.) a graph showing how the number of tasks concurrently computed by workers changes over time is shown. The graph has proved to be an invaluable tool for detecting problems with the parallel computation and for improving the efficiency of the computations. Parts of the project which cannot run in parallel (such as organism selection or fitness transformation), are computed directly by the server and are not shown in the graph (see Figure 2.15 for an example of the efficiency graph).

Experiments presented in this thesis have been performed using a cluster of 80 computers. To prevent communication overhead caused when large number of workers are connected to a single server, computers in the cluster were divided into 15 groups of 5 or 6 computers (analysis used for selecting the optimum number of computers per group is described in more detail in Krcah [40]). One computer in each group was dedicated for running the server process for the group and the remaining computers were running 4 worker processes each (one worker per CPU core). Each of the 15 servers was therefore using between 20 and 24 workers and the total number of worker processes across all machines in the cluster was 260. Typical run of a single evolution using this setup took between 30 and 60 minutes (depending on the complexity of the physics simulation required by the fitness function). The entire cluster could therefore typically finish between 15 and 30 evolutionary runs in one hour (each run using one of 15 groups of computers).

2.3.2 Implementation Notes

ERO is implemented in the Java programming language. Open Dynamics Engine (ODE) [86] is used for rigid body dynamics simulation using modified OdeJava for Java bindings. OpenGL is used for real-time 3D rendering using JOGL for Java bindings. RMI interface is used for communication between all components of the system. XML file format is used for saving project results (such as evolved creatures) and project configuration.

Choosing Java as the primary programming language brings the advantage of the cross platform software. The usage of ERO is therefore only limited to platforms supported by ODE/JavaODE and OpenGL/JOGL (native C/C++ libraries). ERO was tested (and fully works) on recent Linux and Windows operating systems on 32-bit architectures.

Java's ability to dynamically load classes is used to automatically download code for projects from the user-client to the server and from the server to the workers. New versions of projects can be thus tested without the need to restart neither the server nor the workers.

2.4 Evolving Virtual Creatures Using HierarchicalNEAT

This section introduces HierarchicalNEAT – a method for evolving virtual creatures introduced in my master thesis [40, 43]. The main innovation of HierarchicalNEAT is the use of historical markings to track all structural elements comprising a virtual creature (body parts, joints, neurons, neural connections) through the evolution. Such markings make it possible to define an efficient recombination algorithm for virtual creatures and to quantify similarity of any two virtual creatures in a robust way. HierarchicalNEAT uses similarity values to assign creatures to *species* which are used to maintain diversity of the population. This helps to prevent evolution from getting trapped in local optima of the fitness landscape.

The algorithm takes its name from NEAT (NeuroEvolution of Augmenting Topologies) – an algorithm for efficient evolution of neural networks by Stanley [91, 92] who first introduced the concept of historical markings. HierarchicalNEAT extends concepts introduced in NEAT to the nested (or hierarchical) structure of virtual creatures where body parts of creatures form a graph and each node in this graph (i.e. each body part) contains a nested graph describing structure of the neural network. HierarchicalNEAT has been shown to outperform standard genetic algorithm in evolution of virtual creatures [40, 43] and it has therefore been chosen as basis for all experiments in this thesis.

This section consists of five parts. The first part (Section 2.4.1) provides more detailed motivation for using HierarchicalNEAT for evolving virtual creatures. The second section describes how historical markings can be used to evolve structure (regardless of what the structure represents) and provides details on how the general part of the algorithm (i.e. part independent of the specific organism type) has been implemented in ERO framework. The third and the fourth sections provide a description of how historical markings are applied to controllers and bodies of virtual creatures, respectively. Section 2.4.5 then provides visualizations of speciation and similarity matrix in HierarchicalNEAT in typical runs.

2.4.1 Motivation

Standard genetic algorithm traditionally used for evolution of virtual creatures (e.g. by Sims [84]) have been observed to have the following shortcomings [40]:

1. Recombination operators work only on the level of morphology. Examples are *grafting* and *crossover* operators used by Sims [84], both of which create an offspring by first copying a subset of body parts from each parent and then ensuring that all selected body parts remain connected in the offspring. Such operators often result in an invalid genotype. Such genotype is assigned zero fitness, or it is discarded by the validity testing procedure in ERO framework (as described in Section 2.2.5). This suggests that these recombination methods do not combine properties of parental genotypes but instead act more like disruptive mutation operators.

2. Recombination operators described above do not recombine internal parameters of morphological nodes and connections. Instead, both operators work on the morphology level and recombination thus works only with entire morphological nodes and connections. Internal parameters include all morphological parameters (scale, rotation, reflection, joint type, etc.) and also local controllers present in each body part. Recombination has been shown to improve the performance of the artificial neural network evolution [89]. Its absence in evolution of virtual creatures might therefore retard the evolutionary search.

3. Evolution of the creatures often gets trapped in a local optimum. Maximum fitness value stops increasing after reaching only a small value and new generations of organisms converge towards homogeneity.

HierarchicalNEAT addresses the first observation by introducing a new method of creature recombination. While grafting and crossover might be beneficial for exploration, they do not appear to work as robust methods of recombination. Also while both grafting and crossover do combine properties of the parents, the resulting offspring is very likely to be invalid (e.g. self-penetrating). The recombination method used in HierarchicalNEAT ensures that the resulting offspring combines properties of both parents in a less disruptive way.

The second observation could be addressed by introducing a new recombination method. This method should be able to combine internal properties of individual morphological structure elements (i.e. nodes and connections). At a high level of abstraction, an analogy with natural crossover method can be drawn. In the nature, the genetic information of an animal is contained in several chromosomes (specific number of chromosomes varies from tens to hundreds among different life forms). During crossover in nature, matching chromosomes are aligned, and matching parts of matching chromosomes are randomly exchanged. Crossover is thus performed on individual base pairs of each chromosome, instead of on entire chromosomes. Grafting and crossover methods, however, exchange entire "chromosomes" (nodes and connections), instead of individual "DNA base pairs" (internal properties of nodes and connections).

To illustrate further disadvantage of recombination on the level of morphology, consider the following example. Each of two creatures, both descendants of the same common ancestor, accidentally discover an innovation of one of their local neural networks. Innovations are different, but both of them are advantageous. Moreover, innovations occur in the corresponding body parts (i.e. affected body parts are descendants of the same ancestral body part in the common ancestor of the creatures). Using only grafting and crossover, these two innovations can never be combined in a single local neural network of their offspring. The absence of recombination on the level of internal parameters (e.g. neural networks local to each body part), hampers the evolutionary search significantly, because advantageous innovations of neural networks discovered by different creatures cannot be combined in their offspring.

In order to be able to perform genetic recombination of internal parameters of each structural element, a correspondence of body parts of the parents has to be found. Before HierarchicalNEAT was introduced, this was a difficult problem because body plans of parent creatures often differ in their topology. Finding correspondence of body parts without any a priori knowledge then requires application of one of the topology matching algorithms. These algorithms are computationally expensive and do not guarantee correct results.

One of the possible workarounds to this problem is to perform recombination only between creatures with the same topology of their morphological graphs. Evolution could, for instance, work in two repeating phases. During the first phase, the morphology of the creatures would evolve using only the mutation operator, without recombination (avoiding the problem of topology matching during recombination altogether). After a specified number of generations, all creatures except the best one would be discarded, and the second phase would start with a population filled with copies of the best creature from the previous phase. During the second phase, only internal parameters of each node and connection would be subject to mutation. Morphological structure would not change in this phase and the recombination would, therefore, work on graphs with the same topology (again avoiding the problem of topology matching). There are, however, two major drawbacks. The first one is that the absence of recombination is still present during the first phase. The second is the loss of diversity during each transition from the first to the second phase (in order to make easy recombination possible in the second phase).

It would be advantageous if the two phases (the first one evolving morphology without recombination, while the second one evolving internal parameters using recombination on structures with the same topology) could be combined to run simultaneously, instead of one after another. This would be possible, using a concept of *species*. Species would be defined as a group of creatures with the same topology (thus capable of recombination with one another without a need for topology matching). A new, improved, algorithm would start with a population filled with copies of the same initial organism, each copy belonging to the same species. The next generation would be assembled using both mutation and recombination. However, if the mutation changes the morphology of a creature, the new creature would be placed in a new species. Each new generation would therefore include several new species. Problem of topology matching would be avoided by allowing recombination only within a species (where all organisms have the same topology of their morphological graphs). This algorithm would solve both issues of the previous algorithm.

HierarchicalNEAT algorithm is a further extension of these ideas. Using the concept of *historical markings*, correspondences of the parental body parts can be found even if their morphological graphs differ in topology (the concept of historical markings is described in Section 2.4.2). This correspondence then allows recombination algorithm to exchange genetic information in a sensible way (including the internal properties of the structural elements). HierarchicalNEAT, therefore, solves the problem of recombination even without the need of species, which would group together creatures with the same topology, as proposed in the previous algorithm. Historical markings thus offer an elegant solution to both the first and the second observed problems.

HierarchicalNEAT does use the concept of species, but to address a different issue. The primary reason for species is borrowed from the original NEAT algorithm, to “protect innovation through speciation” [91]. Each population is divided into several species and each organism competes only against organisms in the same species. This allows new structural innovation (which might be disadvantageous initially) to optimize in a separate species, instead of being immediately dominated by currently better neural networks in the entire population. A structural innovation is thus being *protected* by speciation.

An interesting side-effect of speciation is that the evolution is less likely to “get trapped” in a local optimum. Each species attempts to solve a problem in a different way and if one species fails (i.e. stops improving its fitness value), another one can take its place. This aspect of speciation addresses the third observation mentioned at the beginning of this section.

To summarize, HierarchicalNEAT addresses all three observations mentioned at the beginning of this section. New method of recombination (based on historical markings) makes it possible for creatures to combine their advantageous innovations in their offspring (even at the level of internal parameters of individual nodes and connections). Speciation makes the evolution less likely to “get trapped” in a specific solution too early.

2.4.2 Evolving Structure Using Historical Markings

The central concept used in HierarchicalNEAT is the concept of *historical markings*. This section describes how historical markings can be used for evolution of any organisms with changing internal structure. In subsequent sections we will show how can this generic approach be used for evolution of neural networks and virtual creatures. Historical markings bring the possibility of tracing individual structure elements (e.g. neurons and neural connections in an artificial neural network, or body parts and joints in virtual creatures) throughout the evolution. Each structural element is assigned a unique identifier (i.e. a historical marking) upon its creation (either during the construction of the initial population or during mutation). Historical markings are inherited, so each node and connection can be traced back to its oldest ancestor.

Another way to look at the historical markings is as an age indicators. The higher value of the marking means that the marking has been introduced later in the evolution and is therefore more recent. This also brings an interesting possibility of determining the relative age of different parts of an organism after it has evolved.

Historical markings are also computationally inexpensive. The genetic algorithm only needs to keep track of the value of the most recent marking used in any virtual creature so far. Upon each request for a new marking (e.g. when the mutation creates a new node or a new connection), the value of the most recent marking is incremented and returned.

Historical markings play a key role in many parts of the HierarchicalNEAT algorithm. Next section describes how they are used to recombine creatures with different topology. Section 2.4.2 then describes the process of speciation and what is the role of markings in this process. Section 2.4.2 summarizes motivation for another key concept – starting the evolution with organisms with a minimal structure.

Recombination

Historical markings offer an elegant method of performing recombination of structures with different topology. Parental structures are first scanned for the presence of structure elements with matching historical markings (these are the ones which share the common ancestor and are thus considered compatible). Since most organisms during the evolution are typically close relatives, the number of matching structure elements is expected to be fairly high. An offspring is constructed by first copying the parent with a higher fitness value, followed by random exchange of internal parameters of all matching nodes and connections with another parent. This way, new offspring inherits all non-matching nodes and connections from its better parent, while all matching elements are formed by a random recombination of properties of both parents.

Since matching structural elements (one from each parent) are descendants of a common ancestral element, it is very likely that both elements serve the same "purpose" in both parents (or at least occupy the same position in the genotype graph). It is therefore reasonable to expect that the recombination of their internal properties is likely to sensibly mix genetic information, without introducing destructive changes in the offspring (which is often the case with other recombination methods such as grafting and crossover).

Speciation

Speciation is the evolutionary process by which new species arise. Speciation in HierarchicalNEAT serves two purposes:

- to maintain diversity of the population,
- to protect structural innovation.

The mechanism for speciation is based on the idea of fitness sharing, as proposed by Goldberg [23]. Fitness sharing is inspired by natural systems. In the nature, different species of animals occupy different ecological niches, taking advantage of different sets of resources. Organisms in the same niche inevitably reach a point when the demand for resources overgrows the amount of resources available. In this situation, a conflict arises and organisms are forced to *share* their resources. With more and more organisms taking advantage of the given niche, it becomes harder for any given organism to succeed (its "fitness" value is lowered). The number of organisms occupying a given niche is thus limited by the amount of available resources in that niche.

In genetic algorithms there is not a clear definition of a resource or a niche. Fitness sharing is, therefore, based solely on similarity of organisms instead of the niche they occupy (assuming that similar organisms occupy the same niche and different organisms occupy different niches). A similarity measure is defined between each pair of organisms. While several schemes of speciation based on fitness sharing exist, *explicit fitness sharing* is used in HierarchicalNEAT algorithm (this method is also used in the original NEAT algorithm [91]).

Genetic algorithm with speciation using explicit fitness sharing is outlined in Algorithm 2.4.1. Genetic algorithm is controlled by a series of parameters. The parameters are listed (along with the descriptions and default values) in

Algorithm 2.4.1 Genetic algorithm with speciation

```
1: divide initial population into species
2: while not terminated do
3:     evaluate fitness function for all organisms in the current generation
4:     copy the organism with the highest fitness (the superchampion) to the next generation
5:     compute the size of each species in the next generation
6:     for all species in the current population do
7:         copy the champion of the species to the next generation
8:         produce offspring by recombination and mutation of selected organisms until the size allocated in step 5 is reached
9:     end for
10:    assign newly created organisms into species (create new species if needed)
11: end while
```

Parameter name	**Description**	**Default**
Interspecies recomb. prob.	Probability that the recombination partner will be selected from another species	0.1
Mutation only %	Portion of organisms created by using mutation operator only.	25%
Recombination %	Portion of organisms created by recombination.	75%
Mutation after recomb. prob.	Probability that an offspring created by recombination will be mutated afterwards.	0.8
Maximum stagnation	Maximum allowed stagnation (in generations) of a species. After this period elapses the species is discarded.	15
Youth age threshold	Species younger than this value are protected by multiplying their fitness by c_{youth}.	10
c_{youth}	The fitness of young species is increased by this factor.	1.5
Initial c. d. threshold	Initial value of the compatibility distance threshold.	0.5
Dynamic c. d. threshold	Specifies whether dynamic thresholding is used for compatibility distance.	yes
Desired number of species	Desired number of species used for dynamic compatibility distance thresholding.	10

Table 2.3: **Parameters of the genetic algorithm with speciation.**

Table 2.3. These default values are used in all experiments in this thesis, unless stated otherwise. The default method of selecting organisms for reproduction is the truncation selection, i.e. parents are selected randomly from the elite $r\%$ organisms of the given species ($r = 20$ is the default value).

The algorithm starts by dividing the initial population of organisms into species according to their similarity. The measure of similarity is called *the compatibility distance* and is computed in such way that more similar organisms have lower value of compatibility distance. If the compatibility distance is zero the two organisms are identical from the point of view of the genetic algorithm. One more constraint is imposed on the compatibility distance: the maximum compatibility distance of any pair of organisms must not exceed one. This constraint is not necessary, but simplifies visualization of compatibility distance and also the construction of a compatibility distance for virtual creatures (see Section 2.4.4 for details).

The algorithm for computing compatibility distance of two organisms takes advantage of historical markings of individual structure elements. Non-matching nodes and connections increase the compatibility distance and matching nodes and connections decrease it. The exact implementation of compatibility distance depends on the specific organism type (procedures of computing compatibility distance for neural networks and the virtual creatures are outlined below in Section 2.4.3 and Section 2.4.4, respectively).

Organisms are assigned to species using the following algorithm: each organism is compared to a representative of each species one at a time (representative is chosen randomly, e.g. as the first member of a species). If the value of compatibility distance is smaller than the specified compatibility threshold, the organism is placed in that species (the organism is placed in the first species that satisfies the condition). If none of the species satisfies the condition, a new species is created. To ensure the continuity of the species over generations, representatives of the species are chosen from the *previous* generation.

The choice of the compatibility threshold greatly influences the progress of the evolution. Lower values make it easier for mutations to shift an organism from one species to another, while large threshold may prevent new species from forming at all. For better control over the compatibility threshold, a dynamic thresholding has been introduced [89] and is also used in this thesis. Dynamic thresholding tries to keep the average size of a species constant, by adaptively changing the compatibility threshold. Desired size of a species (or, alternatively, the desired number of the species in the population) is provided by the experimenter in advance. During the genetic algorithm, compatibility threshold is automatically increased if the average size of the species is below the desired value and vice versa. To change the value of the compatibility threshold, its current value is divided or multiplied by the adjustment factor (value of 0.7 is used for all experiments in this thesis). The threshold is not permitted to exceed the value of one.

The number of species in the next generation is computed (during step 5 in Algorithm 2.4.1) using the following series of steps. First, shared fitness value is computed for each organism as follows:

$$f'_{ij} = \frac{f_{ij}}{N_j} \tag{2.5}$$

where f_{ij} and f'_{ij} are the original (i.e. measured using the fitness function) and shared fitness of an organism i in species j, respectively and N_j is the number of organisms in species j. In order to provide larger portions of the next generation to more fit species (and vice versa), free slots in the next generation are divided among species proportionally, according to the fitness of the species. Fitness F_j of a species j is defined simply as a sum of the shared fitness values of its organisms or, equivalently, as an average of their original fitness values:

$$F_j = \sum_{i=1}^{N_j} f'_{ij} = \frac{\sum_{i=1}^{N_j} f_{ij}}{N_j}. \tag{2.6}$$

Therefore, if N positions in the new generation are to be divided among S species, then the number of organisms in a species j will be:

$$N'_j = N \frac{F_j}{\sum_{i=1}^{S} F_i}. \tag{2.7}$$

Sizes of all species (N'_j) are then rounded to integer values, such that the following condition also holds for the rounded values:

$$\sum_{j=1}^{S} N'_j = N.$$

This speciation scheme ensures that any single species is unlikely to dominate the population, which is the key concept in maintaining the diversity.

Another key role of the speciation is to protect innovation. The speciation algorithm described above already protects innovation, but its ability to do so can be enhanced even more by directly "helping" new species. This is accomplished by replacing Equation 2.6 with the following equation in case when the species is "young" (i.e. it has existed for less than the given number of generations – specified by parameter "youth age threshold"):

$$F_j = c_{\text{youth}} \sum_{i=1}^{N_j} f'_{ij} \tag{2.8}$$

The parameter c_{youth} determines how much help will be given to the new population (the value of 1.5 has been used in all experiments in this thesis).

To further eliminate the problem of stagnation, a species which fails to improve its maximum fitness for a specified number of generations is discarded (i.e. its fitness value F_j is set to zero).

Incremental growth from minimal structure

Many approaches to the evolution of structure start with a population of organisms with randomly generated structure (Sims [84], Grau [26, 25]). Authors of NEAT algorithm argue (and also prove their arguments experimentally [89]) that starting from a complex randomly-generated structure might decrease the performance of the evolution, because the random generation introduces a lot of unjustified structure, not tested by a single fitness evaluation. Starting with a small structure and increasing complexity as the evolution proceeds also minimizes the dimensionality of the search space during the early stages the search, which is beneficial. These positive effects have also been confirmed in experiments with HierarchicalNEAT [40].

2.4.3 Evolving Control System of Virtual Creatures

This section describes application of historical markings to the evolution of neural network controllers of virtual creatures described in Section 2.2.2.

Historical markings

NEAT algorithm, as proposed in [92], uses historical markings for tracing neurons and neural connections. In this thesis, a modified version of the algorithm is used. Historical markings are tracked for each neuron. Neural connections, however, do not need historical markings, because they are identified by their position in the network. This simplification is possible because of the prohibition of multiple connections between an output of one neuron and an input of another neuron. For the purposes of finding matching neural connections during recombination, each connection is identified by a triplet of integer values: $(m_{source}, m_{target}, i)$, where m_{source} and m_{target} are the historical markings of the source node and the target node, respectively, and i is an identifier of an input port of the transfer function. This kind of identification does not provide information about common ancestry of two connections (because connections created by different mutations can have the same identifier if they appear at the same position in the network), but it is sufficient for the topology matching.

Recombination

Recombination of two neural network controllers follows the process described in Section 2.4.2. Matching pairs of neural connections are determined using connection identifiers (described in the previous section) instead of using historical markings.

There are three internal parameters which are subject to recombination: transfer function of a neuron, the weight of a neural connection and the disabled flag of a neural connection.

Transfer functions differ in the number of inputs. Therefore, in order to be able to safely exchange transfer functions of matching nodes, both transfer functions must have the same number of inputs. This is, however, easily guaranteed by allowing the mutation operator to only exchange the transfer function with a compatible one.

Disabled flags of neural connections are recombined in such way that at least one enabled incoming connection and one enabled outgoing connection exist in each neuron of the offspring. This is achieved by using one of the enabled connections of a better parent (at least one always exists, thanks to the constraint imposed on the network structure in Section 2.2.2) in case when recombination results in a neuron with all connections disabled.

Compatibility distance

Measuring the compatibility distance of two neural networks is again based on historical markings. This section presents a hierarchical method of compatibility distance measurement, which slightly differs from the method proposed in NEAT. Advantages of this approach become apparent when computing compatibility distance on a complex morphological structure.

The hierarchy is set up in the bottom-up approach. To compose a compatibility distance of two neural networks, two other compatibility distances are first defined: one between nodes (δ_{n}) and another one between connections (δ_{c}).

$$\delta_{\mathrm{n}}(a,b) = \begin{cases} 1 & \text{if transfer functions of nodes } a \text{ and } b \text{ differ} \\ 0 & \text{otherwise} \end{cases}$$

The compatibility distance between neural connections d and e is computed as follows, if their disabled flags are the same:

$$\delta_{\mathrm{c}}(d,e) = s \cdot \frac{|w_d - w_e|}{2 \cdot w_{\max}}$$

and if their disabled flags differ, the following equation is used:

$$\delta_{\mathrm{c}}(d,e) = (1-s) + s \cdot \frac{|w_d - w_e|}{2 \cdot w_{\max}}$$

where $w_{\max}$ is the maximum absolute weight of a connection (the maximum difference of two weights is thus $2 \cdot w_{\max}$), w_d and w_e are weights of connections d and e, respectively, and $s \in [0,1]$ is the parameter used by experimenter to adjust the relative impact of disabled flags and connection weights on the outcome of the compatibility distance. Parameter $w_{\max}$ has been set to 2 for all experiments in this thesis (as discussed in Section 2.2.2).

To compute compatibility distance δ for neural networks A and B, the distance of all nodes $D_{\mathrm{n}}(A,B)$ and the distance of all connections $D_{\mathrm{c}}(A,B)$ is first computed:

$$D_{\mathrm{n}}(A,B) = |N_{\mathrm{n}}(A,B)| + \sum_{(n_1,n_2)\in N_{\mathrm{m}}(A,B))} \delta_{\mathrm{n}}(n_1,n_2)$$

$$D_{\mathrm{c}}(A,B) = |C_{\mathrm{n}}(A,B)| + \sum_{(c_1,c_2)\in C_{\mathrm{m}}(A,B)} \delta_{\mathrm{c}}(c_1,c_2)$$

where $N_{\mathrm{n}}(A,B)$ and $C_{\mathrm{n}}(A,B)$ are the sets of all non-matching nodes and connections of A and B (i.e. nodes and connections, which do not have counterpart with the same value of historical marking in the other parent), respectively, and $N_{\mathrm{m}}(A,B)$ and $C_{\mathrm{m}}(A,B)$ are the sets of pairs of matching nodes and connections of A and B (i.e. each of two nodes or connections in the pair comes from a different parent, but they have the same value of historical marking), respectively. Finally, the compatibility distance δ between two neural networks is computed as follows:

$$\delta(A,B) = \frac{t \cdot D_{\mathrm{n}}(A,B) + (1-t) \cdot D_{\mathrm{c}}(A,B)}{t \cdot (|N_{\mathrm{n}}(A,B)| + |N_{\mathrm{m}}(A,B)|) + (1-t) \cdot (|C_{\mathrm{n}}(A,B)| + |C_{\mathrm{m}}(A,B)|)}$$

where t is the parameter used by experimenter to adjust the relative impact of nodes and connections on the resulting compatibility distance. In the case of small networks (i.e. $|N_\mathrm{n}(A,B)|+|N_\mathrm{m}(A,B)| \leq N_\mathrm{max}$ or $|C_\mathrm{n}(A,B)|+|C_\mathrm{m}(A,B)| \leq C_\mathrm{max}$), the total number of nodes and connections in denominator is replaced by N_max and C_max, respectively. The range of all compatibility distances (δ, δ_n, δ_c) is $[0,1]$.

2.4.4 Evolving Morphology of Virtual Creatures

This section describes how historical markings can be used in the evolution of both morphology and control system of virtual creatures. The generic algorithm described in Section 2.4.2 requires definition of two components to adapt it to virtual creatures: compatibility distance measure (which the speciation is based on) and recombination of virtual creatures.

In order to evolve both morphology and control of the virtual creatures, historical markings are needed on both levels of the creature structure. Therefore, a new – hierarchical – approach is taken. Markings on the level of morphology work the same way as markings in the evolution of neural networks described in Section 2.4.3. In addition to that, historical markings are also assigned to all newly created neurons. Markings on the morphological level are independent of markings on the control system level (i.e. they are never compared to each other). Moreover, so are independent markings of neurons in different morphological nodes (i.e. they are not compared to each other, neither during recombination nor during compatibility distance measurement). Therefore, separate counters of historical markings (counter keeps track of the last used marking, in order to assign unused markings to newly created structure elements) could be kept for each morphological node, along with a single global counter used on the morphology level. Thus, a hierarchy of counters could be used. The hierarchy is not strictly necessary since a single counter can be used to create historical markings for all structure elements on both levels. The disadvantage of using a single counter is lower readability of the genotype graph, because the value of the counter would increase very fast, leading to very high values of historical markings. Other than readability, the number of counters does not affect the progress of evolution in any way. This thesis uses a compromise solution of three counters: one counter for morphological nodes, one for morphological connections and one for neural nodes (neural connections do not need to use historical markings at all, as described in Section 2.4.3).

Speciation, as a mechanism by which new species arise, is entirely independent of virtual creatures once the compatibility distance measure is given. The method of measuring compatibility distance of two creatures is similar to the method introduced in Section 2.4.3 for measuring the compatibility distance of two neural networks. However, there is one important difference between evolution of neural networks and evolution of virtual creatures. Neural networks are included in their genotype *directly.* There is no transcription phase from genotype into phenotype; the phenotype and the genotype of each creature is identical. Therefore, it does not matter whether the compatibility distance measures distances between genotypes or between phenotypes. On the other hand, creatures are grown from a genotype recursively, using a method described in Section 2.2.1. Genotype and phenotype of a creature are typically two very different structures.

HierarchicalNEAT algorithm takes the approach of measuring compatibility distance of the creatures using their genotype. Historical markings offer a simple and straightforward way of measuring compatibility distance on genotypes. Measuring distances between phenotype graphs would be a lot more complex, because phenotype graphs might contain copies of nodes (thus several nodes in the phenotype might share the same historical marking), which makes historical markings useless for finding correspondences. This, again, opens up the problem of topology matching which has been successfully solved by historical markings in the case of genotypes. Measuring compatibility distance of genotypes is also much faster, simply because phenotypes do not need to be built. Measuring compatibility distance on genotypes is described in detail in Section 2.4.4.

Recombination

Recombination of virtual creatures proceeds hierarchically, based on the correspondence given by historical markings. Recombination on the level of morphology works in the same way as described above in Section 2.4.2. The morphology graphs of parents are first searched for the presence of nodes and connections with the same values of their historical markings. The first parent is then copied to become the first draft of an offspring. The values of all internal properties of all matching nodes and connections in the offspring are then randomly chosen either from the first or from the second parent.

The only exception from this rule are local neural networks contained in each morphological node. When a pair of corresponding morphological nodes is found, the neural network of the offspring is *not* created by copying one of the parental neural networks. Instead, neural networks themselves are combined internally, based on the historical markings of individual neurons. Recombination of neural networks works in the same way as described in Section 2.4.3.

The following internal properties are subject to recombination in morphological nodes: joint type, shape (including its size) and recursive limit. And the following ones in connections: terminal flag, source anchor, scale factor, reflection flags and rotation angles.

Thanks to the historical markings, recombination is capable of sensibly mixing genetic information even on the level of individual weights of neural connections.

Compatibility distance

The compatibility distance between two creatures is defined in a similar way as the compatibility distance between two neural networks defined in Section 2.4.3. Thanks to the historical markings, which provide a correspondence between nodes and connections of two morphological graphs, the compatibility distance can be computed hierarchically, measuring even a small difference in weights of two corresponding neural connections. Nodes and connections of the morphological graphs of both creatures can be divided into four sets: a set of all pairs of matching nodes (N_m), a set of all pairs of matching connections (C_m), a set of all non-matching nodes from both parents (N_n) and a set of all non-matching connections from both parents (C_n). Non-matching nodes and connections are those which do not have counterpart with the same value of historical marking in the other parent, while matching pairs nodes or connections consist of elements coming from different parents having the same value of historical marking. To compute the final compatibility distance, a hierarchy of several simple compatibility distance functions is defined first.

The compatibility distance between two morphological nodes a and b is defined as a weighted sum of normalized differences between individual properties of two morphological nodes:

$$\delta_{\mathrm{n}}(a,b) = \frac{w_{\mathrm{p}} \cdot \frac{|a^{\mathrm{r}} - b^{\mathrm{r}}|}{r_{\max}} + w_{\mathrm{m}} \cdot \delta_{\mathrm{s}}(a^{\mathrm{s}}, b^{\mathrm{s}}) + w_{\mathrm{m}} \cdot \delta_{\mathrm{j}}(a^{\mathrm{j}}, b^{\mathrm{j}}) + w_{\mathrm{c}} \cdot \delta^{'}(a^{\mathrm{c}}, b^{\mathrm{c}})}{w_{\mathrm{p}} + 2 \cdot w_{\mathrm{m}} + w_{\mathrm{c}}}$$

where $a^{\mathrm{r}}, b^{\mathrm{r}}$ are recursive limits of nodes a, b; $r_{\max}$ is a maximum possible difference between recursive limits of two nodes; $a^{\mathrm{s}}, b^{\mathrm{s}}$ are geometric shapes of nodes (e.g. a box, a sphere, etc.); $a^{\mathrm{j}}, b^{\mathrm{j}}$ are joints and $a^{\mathrm{c}}, b^{\mathrm{c}}$ are local neural network controllers of nodes a and b, respectively. A set of parameters specified by the experimenter is used to set relative weights of morphological properties (w_{m}), control properties (w_{c}), and properties affecting the structure of the phenotype graph (w_{p}). The compatibility distance of neural networks $\delta^{'}$, as defined in Section 2.4.3, is used to compare local neural networks of corresponding nodes. Compatibility distance between two joints is defined simply as

$$\delta_{\mathrm{j}}(a^{\mathrm{j}}, b^{\mathrm{j}}) = \begin{cases} 1 & \text{if joints } a^{\mathrm{j}} \text{ and } b^{\mathrm{j}} \text{ differ} \\ 0 & \text{otherwise} \end{cases}$$

and compatibility distance of shapes is defined as:

$$\delta_{\mathrm{s}}(a^{\mathrm{s}}, b^{\mathrm{s}}) = \begin{cases} 1 & \text{if } a^{\mathrm{s}} \text{ and } b^{\mathrm{s}} \text{ differ} \\ \text{normalized difference in volume of shapes} & \text{otherwise} \end{cases}$$

The compatibility distance between two morphological connections d and e is defined as a weighted sum of normalized differences between individual properties of two morphological connections:

$$\delta_{\mathrm{c}}(d,e) = \frac{w_{\mathrm{m}} \cdot \sum_{i=1}^{6} \delta_{\mathrm{c}}^{i}(d,e) + w_{\mathrm{p}} \cdot \sum_{i=7}^{11} \delta_{\mathrm{c}}^{i}(d,e)}{6 w_{\mathrm{m}} + 5 w_{\mathrm{p}}}$$

where w_{p} and w_{m} are the same parameters as used in constructing δ_{n} function (i.e. used by experimenter to adjust the relative weight of morphological parameters and parameters affecting the structure of the phenotype graph). Each of eleven functions $\delta_{\mathrm{c}}^{1}, \cdots, \delta_{\mathrm{c}}^{11}$ compares values of a specific internal parameter of two morphological connections. First six functions compare morphological parameters (two source anchor coordinates, three angles of rotation and the scale factor) and next five functions compare parameters affecting the structure of the phenotype graph (the terminal flag, the recursive limit and three reflection flags). The range of these functions is $[0,1]$ and each of them outputs zero if the properties are the same and one if the properties are as different as they can be.

Finally, the compatibility distance between two morphological graphs A and B is defined similarly as for the neural networks in Section 2.4.3. The distance of all morphological nodes $D_{\mathrm{n}}(A,B)$ and the distance of all morphological connections $D_{\mathrm{c}}(A,B)$ is first computed using distance functions defined above:

$$D_{\mathrm{n}}(A,B) = |N_{\mathrm{n}}(A,B)| + \sum_{(n_1,n_2) \in N_{\mathrm{m}}(A,B)} \delta_{\mathrm{n}}(n_1,n_2)$$

$$D_{\mathrm{c}}(A,B) = |C_{\mathrm{n}}(A,B)| + \sum_{(c_1,c_2)\in C_{\mathrm{m}}(A,B)} \delta_{\mathrm{c}}(c_1,c_2)$$

where $N_{\mathrm{n}}(A,B)$ and $C_{\mathrm{n}}(A,B)$ are the sets of all non-matching nodes and connections of morphological graphs A and B, respectively, and $N_{\mathrm{m}}(A,B)$ and $C_{\mathrm{m}}(A,B)$ are sets of pairs of matching nodes and connections of morphological graphs A and B, respectively. Finally, the compatibility distance δ between two creatures A and B is computed as follows:

$$\delta(A,B) = \frac{t \cdot D_{\mathrm{n}}(A,B) + (1-t)\cdot D_{\mathrm{c}}(A,B)}{t\cdot(|N_{\mathrm{n}}(A,B)| + |N_{\mathrm{m}}(A,B)|) + (1-t)\cdot(|C_{\mathrm{n}}(A,B)| + |C_{\mathrm{m}}(A,B)|)}$$

where t is the parameter used by experimenter to adjust the relative impact of nodes and connection on the resulting compatibility distance. In the case of small morphological graphs (i.e. $|N_{\mathrm{n}}(A,B)| + |N_{\mathrm{m}}(A,B)| \leq N_{\mathrm{max}}$ or $|C_{\mathrm{n}}(A,B)| + |C_{\mathrm{m}}(A,B)| \leq C_{\mathrm{max}}$), the total number of nodes and connections in denominator is replaced by N_{max} and C_{max}, respectively.

In summary, this section introduced a method of hierarchical comparison of two complex morphological graphs based on correspondences given by historical markings. Behavior of the compatibility distance function can be controlled by a set of weights. Weights can be conveniently used to put more emphasis on a specified set of properties of the creatures (e.g. morphological properties, or properties of a control system). Also, the range of all compatibility distances defined in this section is $[0,1]$, which makes them modular and easy to visualize (see the next section).

2.4.5 Visualizing HierarchicalNEAT

HierarchicalNEAT has been shown to evolve creatures with a given target fitness value significantly faster than the standard genetic algorithm. The speed-up factor varied between 1.5 and 2.5 for different tasks (swimming, jumping, light following and land locomotion). Further experiments have shown that most of the performance gain is provided by speciation, although recombination based on historical markings also has a significant positive effect [40].

This section presents visualizations of various parts of HierarchicalNEAT algorithm. Three visualizations are introduced: compatibility distance map (which provides visualization of compatibility distances among a set of organisms), graph of the fitness values (showing an overview of fitness values of all organisms in all generations) and speciation graph (which visualizes development of species over time).

Compatibility Distance Map

Compatibility distance is a function, which measures similarity of any two organisms. Given a sequence of organisms $s_1, \ldots, s_n$ and a compatibility distance function δ, the matrix M of distances among all organisms can be constructed: $M_{i,j} = \delta(s_i, s_j)$ for each (i,j), where $1 \leq i \leq n$ and $1 \leq j \leq n$. Since δ is symmetric, matrix M is also symmetric. Also, each organism is equal to itself and therefore $\forall i\ M_{i,i} = 0$.

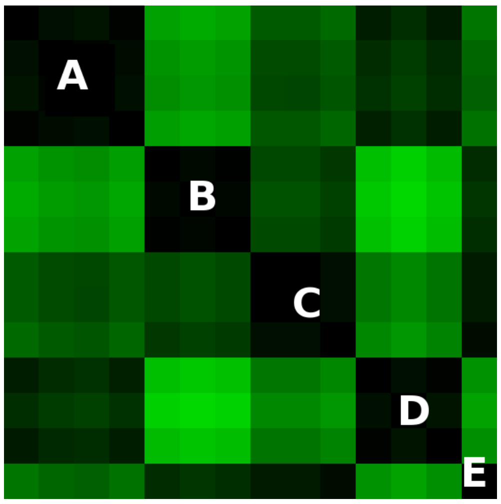


Figure 2.16: **Visualization of similarities among 14 individuals.** Black color corresponds to the zero distance and bright green corresponds to the distance of one. The figure shows individuals ordered by the species they belong to. Species are marked with letters A, B, C, D, E.

Compatibility map is a visualization of matrix M. All compatibility functions used in this thesis are constructed in such way that their range is $[0, 1]$. Their values can, therefore, be mapped to a range of colors for visualization purposes. Black color is assigned to the value of zero and bright green is assigned to the value of one. Other shades of green are assigned proportionally to the values between zero and one.

Compatibility map is also a useful tool for visualization of the species assignment (and thus, the correctness of the species assigning algorithm). Figure 2.16 shows compatibility map of individuals sorted by their species. Species should group together organisms, which are very similar to each other. The map shows that this is exactly the case, since large black square areas are formed on the diagonal for individuals belonging to the same species.

Graph of Fitness Values

The fitness graph shows an overview of fitness values of all organisms created during the process of evolution. Example of a fitness graph is shown in Figure 2.17. Horizontal axis shows increasing generations, while the vertical axis corresponds to the fitness values. Each individual created during the evolution is marked on the graph using a semi-transparent red mark. The darkness of red thus represents number of individuals at the same position on the graph. Semi-transparent marks allow experimenter to see the fine structure in the fitness graph, where there would be a solid area otherwise. Best individuals from each generation are connected by a blue line.

Fitness chart can also be used to visualize speciation. Since each species contains individuals that are similar to each other, fitness values of individuals from the same species tend to cluster around the same value in the chart. As a result, individuals from the same species form “strands“ in the fitness chart. Figure 2.17 illustrates how innovation introduced in one species is improved over

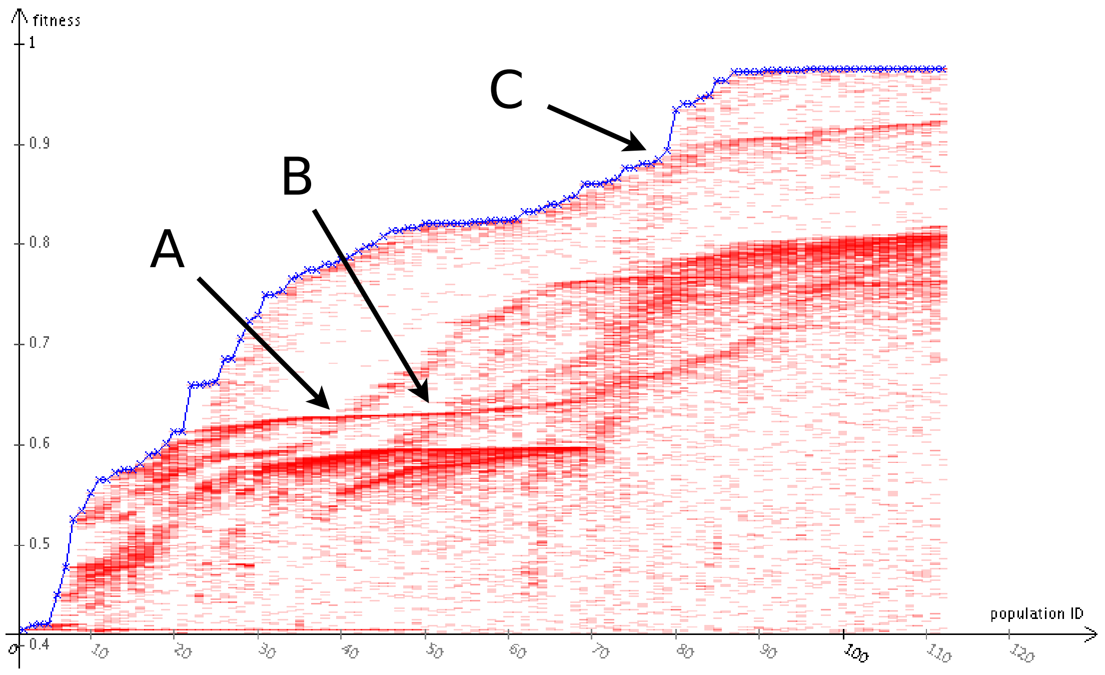


Figure 2.17: **Graph of the fitness values.** Species correspond to red "strands" in the fitness graph. Points A, B and C highlight the moments in the evolution, when the fitness of one species overgrows the fitness of another species.

successive generations, eventually overtaking older well-established species. This is the process by which speciation helps evolution maintain diversity and escape local optima in the search space.

Speciation Graph

A more direct way of visualizing speciation is to show how size of each species changes across generations (this visualization was first used in [91]). Figure 2.18 shows an example of species development over time. Vertical axis represents time (i.e. generations), increasing from the top to the bottom, and horizontal axis shows individuals. Each row represents all organisms in a fixed-size population, sorted by their species. Each species is assigned gray color with random darkness to distinguish it visually from other species. New species appear on the right side of the graph. The red triangles represent the extinction of a species, while green triangles represent birth of new species.

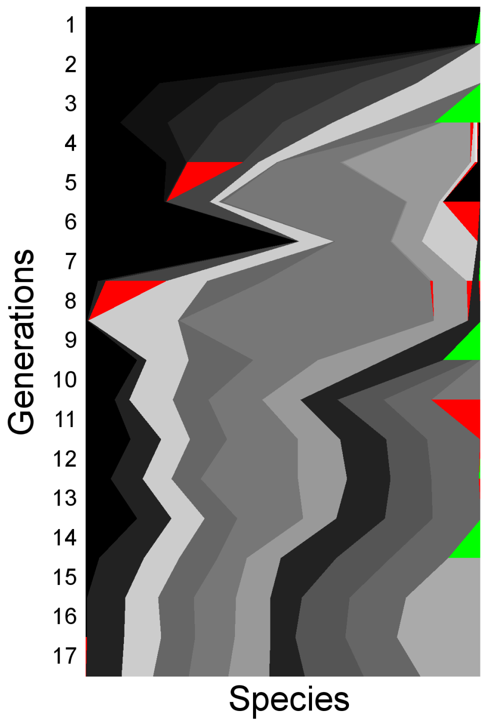


Figure 2.18: **Visualization of the speciation process**. Vertical axis shows generations increasing from the top to the bottom, while horizontal axis shows the sizes and distribution of individual species. Each species is colored using a random shade of gray to distinguish it from other species. Red triangles represent the extinction of a species and green triangles represent birth of new species.

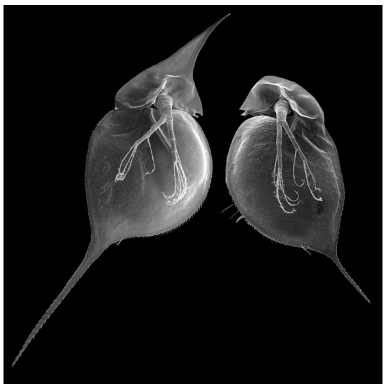
Chapter 3

Adaptation Through Morphological Plasticity

Phenotypic plasticity, defined as the capacity of a single genotype to exhibit a range of phenotypes in response to changes in the environment [99], is a well studied phenomenon in biology. Some of the most striking examples of such plasticity occur in animals that are capable of modifying their morphology during their lifetime. Water flea (see Figure 3.1a) has been shown to grow protective spines when exposed to chemicals released into water by its predators. Wasps and ants adjust size and shape of their bodies depending on which caste they are born into [76] (see Figure 3.1b). Morphological changes in these cases are not a result of mutations, but occur purely in response to some property of the environment experienced by the individual during its lifetime. Such morphological plasticity can allow individuals to adapt to different environments quickly, without having to wait for slow mutations to discover the necessary changes [99, 1, 76].

In some cases morphological traits discovered during animal's lifetime can, over generations, be assimilated into the genotype of the animal. Evolutionary mechanism for such transfer has first been proposed by Baldwin in 1896 [2] and is since referred to as the *Baldwin effect*. The Baldwin effect occurs in a situation when individuals in the population are capable of acquiring some beneficial trait during their lifetimes (e.g. by learning new behaviors). Since the beneficial trait increases their chances for survival and reproduction, such individuals are preferred by natural selection. Moreover, individuals that are capable of acquiring such trait faster than the rest of the population will be promoted by natural selection (since they will be able to start making use of the new trait earlier in their lifetime) and as a result selection pressure will lead evolution towards increased ability to acquire the trait. Such selection pressure can lead to a point where the trait becomes available to individuals automatically without the need to acquire it (i.e. the trait is assimilated into the genotype). While on the surface this process might resemble Lamarckian evolution (which assumes that traits acquired during the lifetime of an individual are passed on to its offspring directly), the transfer of acquired traits to the genotype in this case happens through a purely Darwinian process (all changes to the genotype are introduced through random mutations and are acted on by natural selection).

A classic example in nature of such assimilation of morphological traits into a genotype can be found in ostriches [96]. Ostrich skin (same as skin of many other


(a)

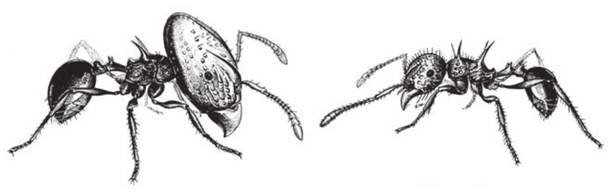
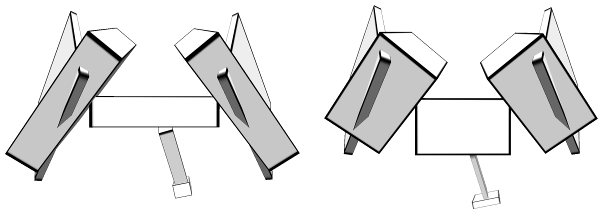

(b)

Figure 3.1: **Examples of morphological plasticity exhibited by animals:** (a) *Daphnia Lumholtzi* when exposed to chemicals produced by a predatory fish (long spines in the individual on the left reduce predation compared to individual on the right that has not been exposed to the chemicals) [1] and (b) minor vs. major worker ant of *Acanthomyrmex* species [76] (drawn by Turid Hölldobler).


(a) Presence of the sea floor induces increased body mass allowing creature to gain more traction against the ground (right) compared to morphology used for swimming in open water (left). Video is available at `https://youtu.be/GX1BNduCvmo`.

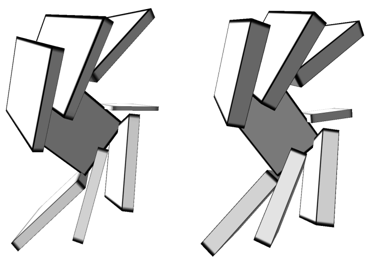

(b) Decreased viscosity of the surrounding fluid (left) induces reduction in the body mass of a swimming creature. Video is available at `https://youtu.be/hFzeowtPmmk`.

Figure 3.2: **Examples of morphological plasticity in evolved robots.** Robots are adapted for swimming in different environments using the method presented in this thesis.

vertebrates) can become thickened and hardened as a result of repeated pressure and friction applied to it during the lifetime of an individual. The ability of the skin to harden is itself an adaptive trait – when ancestors of an ostrich first started to sit down on the ground for longer periods of time (to avoid predators or to sit on their eggs), the ability to develop calluses (patches of hardened skin) on their rump provided an evolutionary advantage because it enabled them to sit on the ground for longer periods of time without developing sores or tears in their skin. Individuals that could develop calluses more rapidly in the correct locations had an advantage over individuals that developed calluses more slowly. This provided selection pressure which over large number of generations lead to calluses being present already in an ostrich embryo, before the skin was exposed to environmental stimuli that normally cause skin to harden (such as friction or pressure on affected skin areas). A trait which initially required input from the environment to be expressed was instead expressed automatically before any input from the environment was available, which shows that the trait has been assimilated into the genotype.

Cases of such genetic assimilation in nature have also been demonstrated experimentally (e.g. in *Drosophila Melanogaster* by Waddington [97]). It has been hypothesized that Baldwin effect may have played a major role in the evolution of instinctive herding skills in dogs bred for that purpose (herding behavior initially acquired through learning becomes assimilated into the genotype over generations) or development of language and lactose tolerance in humans [16].

The importance of the Baldwin effect lies in the fact that it makes it possible for evolution to discover traits that would otherwise be unreachable or very hard to reach. In the words of Dennett [15] (p. 186): “Thanks to the Baldwin effect, species can be said to pretest the efficacy of particular different designs by phenotypic (individual) exploration of the space of nearby possibilities. If a particularly winning setting is thereby discovered, this discovery will *create* a new selection pressure: organisms that are closer in the adaptive landscape to that discovery will have a clear advantage over those more distant. This means that species with plasticity will *tend* to evolve faster (and more ”clearsightedly”) than those without it.”

Motivated by these examples from nature, we pose the following questions in the context of evolutionary robotics (in the following text, terms virtual robots and virtual creatures will be used interchangeably):

1. Can virtual creatures benefit from the ability to adapt their morphology to different environments when solving a given task? (Similar to water flea modifying its body when it detects presence of a predator, i.e. specializing its morphology to the environment in which it was placed.)

2. Can morphological plasticity achieved through learning make evolution of virtual creatures faster even in a single environment, i.e. decrease the number of generations required to evolve a creature with a given level of fitness? (Similar to how Baldwin effect can speed up evolution in nature.)

3. Can learning make evolution more efficient even in a strictly computational sense, when the extra computational cost required for learning is compared to evolution without learning? I.e. can evolution of virtual creatures which can change their morphology during their lifetimes reach a given target fitness value with less total computational resources than evolution without learning?

To answer these questions, we propose a method of evolving virtual creatures that can adapt their morphology during their lifetime to the current environment. Adaptation is performed using a hill-climbing learning rule (described in Section 3.2). Creatures used in all experiments are composed of blocks connected by joints, and represented in evolution using genotype-to-phenotype mapping described in Section 2.2 (an example of evolved creatures is shown in Figure 3.2). Morphological changes during the learning phase are limited to adjustments to the size of each block while the overall structure of the creature (number of blocks and positions of joints used to connect them) remains unmodified. While such changes may not be sufficient to adapt the creature to a completely new task (which might require learning new behaviors or making more radical changes to the morphology), we show that they are sufficient to allow virtual creatures to adapt to significantly different environments and perform given tasks more efficiently than creatures not capable of such adaptation. Limiting morphological plasticity to only adjustments of block sizes also makes it more feasible to apply this technique to construction of robots capable of such adaptations in reality—constructing a robot that can adjust sizes of its parts is likely to be more practical than building a robot that can restructure its morphology more dramatically (a simple physical robot consisting of a single resizable box has already been demonstrated by Roper et al. [81] in Voxbot system).

The rest of this chapter is organized as follows. Section 3.1 provides an overview of previous works that study interaction of evolution and learning. Section 3.2 describes a generic learning algorithm that can be used to optimize morphology of a creature during its lifetime and Section 3.3 describes how it can be applied to the swimming task. Section 3.4 then describes experiments demonstrating the ability of creatures to adapt to different environments and Section 3.5 show how learning can be used to improve evolution even in a single environment. Finally Section 3.6 and Section 3.7 provide ideas for future work and conclusion.

3.1 Previous Works

Interaction of evolution and learning has first been investigated by Hinton and Nowlan [28] and by John Maynard Smith [85] in 1987, who have demonstrated that ability of individuals to learn can lead to faster evolution. Hinton and Nowlan [28] have shown how Baldwin effect can speed up evolution using a simple model: the subjects of evolution were neural networks consisting of 20 potential neural connections (each connection can be either present or absent and connection weights were not subject to optimization). One of the 2^{20} possible neural networks was arbitrarily chosen to be *the good network* and all other networks were *bad networks*. The goal of the search was to find the one good network. While such setup is quite artificial, it can be used to clearly demonstrate possible benefits of the Baldwin effect. The problem of finding the one good network is too difficult for a standard genetic algorithm, because the fitness landscape does not provide any gradient which the genetic algorithm can use. Evolutionary algorithm therefore has to resort to randomly testing different networks. To combine evolution with learning, Hinton and Nowlan used genetic encoding consisting of 20 genes where each gene specified whether a given neural connection was present, absent or whether its presence was undecided (left to be determined by learning during the lifetime of each individual). Fitness evaluation in the genetic algorithm was modified to test a fixed number of different random combinations of all undetermined genes (a simple simulation of lifetime learning), and the fitness of the individual was set to the number of combinations remaining when the good network was found. This method of evaluating fitness meant that genotypes which were "closer" to the good network (i.e. it was easier to the lifetime learning to guess the correct value of all undetermined genes) received higher fitness score than individuals further away. Over generations, the number of unspecified genes was reduced because genes that were guessed correctly during learning were (over time) assimilated into the genotype by mutation and crossover. Evolutionary search with learning therefore converged on the good network unlike evolution without learning. While their model was limited, Hinton and Nowlan have shown that learning can make evolution faster by providing a gradient which can be used by the genetic algorithm to find the solution.

One of the main limitations of Hinton and Nowlan's model is that learning uses exactly the same objective function as evolution. As argued by Nolfi and Floreano [72] this is rarely the case in nature, where the environment rarely provides direct hints on how an individual can improve its evolutionary success. Clues from the environment can instead be used only as an indirect guide for learning that *might* lead to increased ability to reproduce. Nolfi et al. [74] address this by studying a model which uses different tasks for learning and for evolution. The subject of evolution are artificial agents living on a two-dimensional grid. The goal of each agent is to collect food tokens scattered randomly on the grid. Each agent is controlled by a neural network with the following inputs: direction and distance to the nearest food token and the motor action planned in the next simulation step (possible actions are: turn 90° left, turn 90° right, go forward, remain still). Neural network outputs the next planned motor action and a prediction of what the inputs will be in the next step. While the goal of each agent (from evolutionary point of view) is to collect as many food tokens as possible, the goal of learning is different: to predict the state of sensors in the next simulation step. Learning is performed in each simulation step using a standard back-propagation algorithm. Weights of neural connections are not copied back to the genotype after learning (evolution is Darwinian, not Lamarckian). Nolfi et al. show that even though learning in this case does not attempt to directly improve the ability of agents to collect food, evolution with learning significantly outperforms evolution without learning.

Since the goals for learning and for evolution in this experiment are different, the explanation provided by Hinton and Nowlan (i.e. learning provides the gradient for evolution) is on its own not sufficient to explain increased performance. Nolfi et al. explain the performance increase by using a concept of *dynamically correlated surfaces*. In this explanation, they first imagine all individuals exist on two different search surfaces: learning surface and evolutionary surface. Each individual can have different fitness values on these two surfaces (robot can collect many food tokens even if it cannot predict its sensor values well and vice versa). The process of learning leads each individual towards higher areas in the learning surface, while evolution leads individuals towards higher areas in the evolutionary surface. Nolfi et al. propose that when learning and evolution are used together, evolution will tend to select individuals for which learning increases not only their fitness in the learning surface, but also their fitness on the evolutionary surface (i.e. individual is in an area where the two surfaces are dynamically correlated).

Both works described so far have concentrated on studying interaction of learning and evolution in the context of neural networks. Other works have studied interaction and learning using control systems of simple robots with fixed morphology [72, 19, 75]. Robots can, however, benefit from being able to adapt other components of their phenotype to the environment. As we have shown in the introduction to this chapter, many species of animals benefit greatly from morphological adaptations. In this thesis we study plasticity of robot's body as opposed to plasticity of its control system.

Previous works in evolutionary robotics have traditionally focused on optimizing morphology of robots purely by evolution [84, 62, 33, 5, 59] (also see Section 2.1 for an overview) while investigations of phenotypic plasticity have been limited to the control system of robots with fixed morphology [72, 19, 75]. One exception is work by Bongard [6] which demonstrates that modifying morphology during the lifetime of the creature according to a predetermined plan can lead to evolution of more robust creatures with higher fitness. However, morphological changes in this case were predefined and did not change in response to the environment or behavior of the robot. The purpose of the study was not to study learning of morphology during individual's lifetime, but to improve optimization of the control system by evolution through changes in morphology. Effects of morphological plasticity achieved through lifetime learning have not been studied yet to the author's knowledge.

3.2 Learning Algorithm

Motivated by examples of morphological plasticity in nature (described at the start of this chapter) and by previous works that studied plasticity of the control system of a robot, we propose a new method where morphological plasticity of a robot is achieved through lifetime learning. The main insight which this learning method is based on is that for many problems, the final performance of a given morphology can be predicted (at least to some extent) using a significantly shorter (and therefore less expensive) test. Examples of such problems include any problems where the evolved behavior needs to be sustained by the robot for a longer period of time. Common cases of such problems in evolutionary robotics include various forms of locomotion, such as walking on the ground, swimming in water or flying in the air.

The motivation for using shorter tests during learning can be further illustrated using the following analogy. Imagine a circus owner needs to find out who from a group of tightrope walkers has the best chance of successfully walking over the entire length of a rope. The most reliable method is to allow each tightrope walker to walk the full length of the rope and select the winner from the ones who successfully reach the end of the rope. However, if the time for such a test is limited, the owner might decide to ask each tightrope walker to make only a small number of steps on the rope before climbing down. Although such short test will not have as much predictive power as test on the full rope (it will not distinguish between walkers who will fall in the middle of the rope from the ones who successfully walk the entire length), it can still give the owner valuable information about the capabilities of different test participants. Co-evolution of morphology and control system of a robot can be in many ways similar to the tightrope walking example. Virtual robots are subject to the laws of physics and can get into situations which they cannot recover from (e.g. flip on one side preventing any further locomotion, or cause an unrecoverable instability in the physics simulation). While the short test cannot predict the outcome of a full test perfectly, the information can be sufficient to help guide evolution towards more promising areas of the search space.

To combine evolutionary search with learning, we modify each fitness evaluation to start with a *learning phase* during which the robot attempts to discover

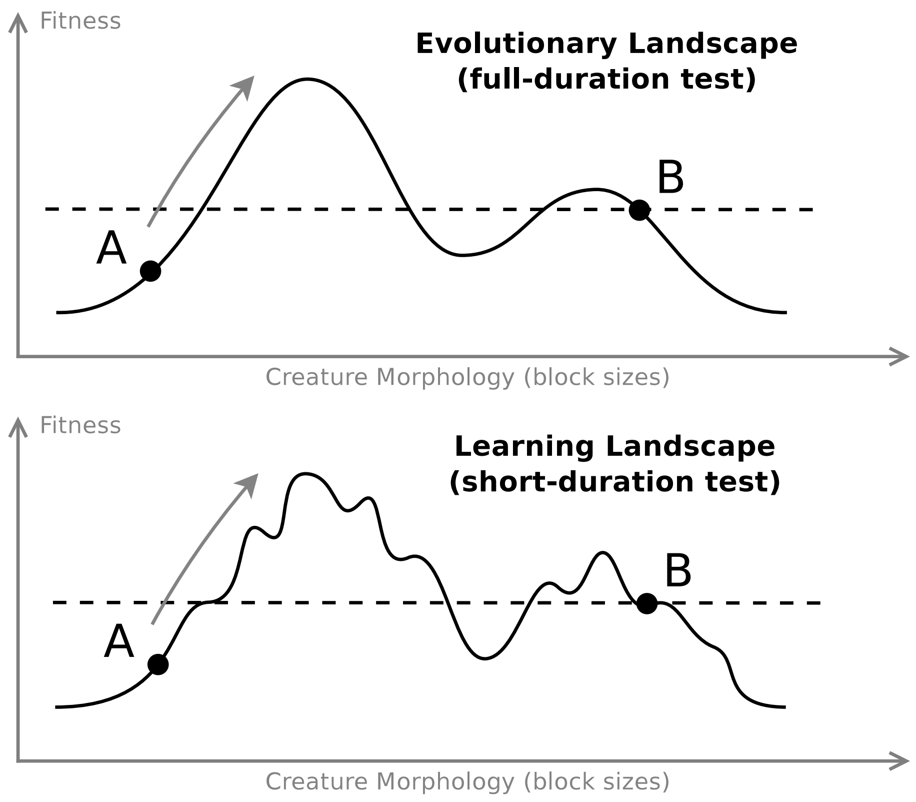


Figure 3.3: **Learning and evolutionary landscapes.** Learning based on short-duration experiments searches a different landscape (bottom) than evolution where the landscape is based on full-duration experiments (top). Even when the information used by learning is less accurate, learning can still aid evolution by searching the local neighborhood of each individual as long as there is some correlation between gradients of the two landscapes. Evolution without learning would (incorrectly) allocate more offspring to individual B compared to individual A (since B has higher fitness than A), while evolution with learning will allocate more offspring to individual A when learning during the lifetime of individual A increases its fitness value.

better morphology through experimentation. The learning phase consists of a fixed number of short-duration experiments designed to test if the selected morphology adjustment has the potential to improve the fitness of the individual. After the learning phase, full-duration fitness test is performed to verify if the robot can sustain discovered improvements for a longer period of time. Result of the full-duration test is used as the final fitness score.

The hypothesis tested in this chapter is that if the computational cost of learning is reduced (by reducing duration of individual learning experiments) the benefits gained from learning during evolution can outweigh the cost of learning (despite the fact that learning acts on less accurate information). As a result, evolution with learning might be able to outperform evolution without learning not just in terms of the number of generations required to evolve creature with a given fitness, but also in a strict sense—in terms of the total computational cost required.

The mechanism through which learning can influence evolution is illustrated in Figure 3.3. Since all dimensions of all blocks of the creature are being optimized

Algorithm 3.2.1 Lifetime Learning of Morphology.

```
Require:
    $N$, the number of learning steps
    $t_{\text{learn}}$, duration of each learning step
    $t_{\text{test}}$, duration of the final fitness evaluation
    $L_{\text{geno}}$, array of block sizes encoded in the genotype of a virtual creature
 1: Initialize $L_{\text{best}} = L_{\text{geno}}$, $f_{\text{best}} = 0$
 2: for all $i \in \{1, ..., N\}$ do                          ▷ Hill-climbing learning phase
 3:     $L_{\text{current}} \leftarrow$ RandomlyAdjust($L_{\text{best}}$)
 4:     $f_{\text{current}} \leftarrow$ TestMorphology($L_{\text{current}}, t_{\text{learn}}$)
 5:     if $f_{\text{current}} > f_{\text{best}}$ then
 6:         $f_{\text{best}} \leftarrow f_{\text{current}}$
 7:         $L_{\text{best}} \leftarrow L_{\text{current}}$
 8:     end if
 9: end for
10: $f_{\text{final}} \leftarrow$ TestMorphology($L_{\text{best}}, t_{\text{test}}$)      ▷ Final fitness evaluation
11: return $f_{\text{final}}$
```

by learning at the same time, learning occurs in $3n$-dimensional space (where n is the number of blocks forming creature's body, excluding any mirrored and repeated blocks). We can therefore imagine that both evolution and learning are searching through a multidimensional space of all morphologies [1]. To illustrate the benefits of learning, imagine that a population contains individuals A and B. Individual B has higher fitness than individual A, but individual A lies in a more promising area of the search space. Evolution without learning will focus on exploring the neighbourhood of individual B over individual A (i.e. individual B will product more offspring due to its higher fitness value). In evolution with learning, the process of learning for individual A can discover higher fitness values by exploring the local gradient around individual A (grey arrow in the figure). The hypothesis is that this can occur even if the information used by learning is imperfect (because of shorter test durations). The gradient can still be preserved sufficiently for learning to make use of it and transmit information about higher fitness values to evolution (similar to the cases of the Baldwin effect discussed at the start of this chapter).

As described in Section 3.1, Nolfi et al. in [74] used back-propagation algorithm to learn the weights in a neural network during the lifetime of a robot. In the case of morphological plasticity, there are no known analytical methods that can be used to predict the best morphology for a given creature and a given task. The process of learning therefore has to rely on black-box optimization methods. While any black-box optimization method could be used, hill-climbing has been chosen due to its focus on exploitation of the local gradient around the individual.

Pseudocode for the learning process is shown in Algorithm 3.2.1. Morphology changes during the learning phase are limited to the sizes of individual blocks, while the number of blocks and their connections remain unchanged. The learning phase (lines 2-9) performs a hill-climbing search over the space of all possible

[1]This is only an approximation because unlike learning, evolution can add and remove blocks as well which changes the dimensionality of the search space.

sizes of all blocks comprising the creature. In the first learning step, sizes of all blocks are set to the values encoded in the individual's genotype (L_{geno} in the pseudocode). In each subsequent step, the best block sizes found so far are modified by a small random amount (RandomlyAdjust(L_{best}) function call on line 3 randomly adjusts the size of each block along each dimension). Both magnitude and direction of the adjustment are selected uniformly from a predefined range of values using a standard pseudorandom number generator. The sequence of pseudorandom numbers is never reset (new pseudorandom values are generated in each learning step through the entire evolution run). Block sizes are not permitted to exceed predefined upper and lower limits. Sizes of blocks generated using recursive limits and reflection are adjusted by the same amount in each step to preserve symmetry (see Section 2.2.1 for a description of reflection flags and recursive limits). After each learning step, the relative positions of all blocks are reset to their initial positions to ensure that early unsuccessful learning steps do not interfere with subsequent learning steps.

Each generated morphology is tested by TestMorphology(L, t) function (parameter L contains morphology encoded as an array of block sizes and parameter t contains the duration of the test). The test can measure any aspect of the creature during the allocated time. As an example, a test designed for evolution of swimming virtual creatures returns the distance travelled by the creature during the test (swimming task will be described in more detail in the following section).

3.3 Learning Applied to the Task of Swimming

In this section we describe how is the generic learning algorithm described above applied to the task of swimming. Swimming is a task ideally suited for the learning algorithm described above since it requires the robot to sustain an efficient swimming behavior for a longer period of time to be considered successful.

To test robot's swimming ability, robot is placed in a water environment simulated by turning off gravity and adding viscosity forces. Fitness of the robot is set to the average speed achieved over the course of the simulation by the center of mass of the robot. Forward motion is favored over circular motion by computing fitness as a distance between starting and final position of the robot divided by the duration of the test. For more details about fitness evaluation in general, see description in Section 2.2.4. Swimming task can be modified in various ways by changing properties of the environment such as viscosity or by adding a sea floor and enabling a weak downward force.

The same fitness calculation method is used during learning and for the final fitness evaluation. The effect of shortening the test on the search space can be seen in Figure 3.4 which shows how fitness changes for a specific evolved creature when varying the length of two sides of one of its blocks. While the landscapes are different, the main features remain similar. For example, creatures with large block width and large block depth (top right corner of each chart) achieve low fitness while creatures with small block depth tend to achieve larger fitness.

Learning landscapes shown in Figure 3.4 and Figure 3.7c provide clear gradients, suggesting that the problem might be well suited for a hill-climbing learning algorithm.

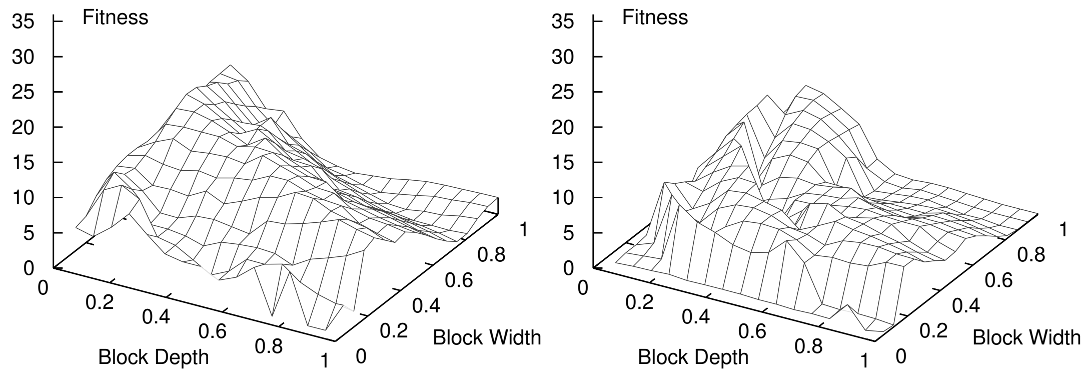


Figure 3.4: **Cross-section of a learning landscape (left) and evolutionary landscape (right).** Learning test is limited to two seconds while full fitness evaluation takes 48 seconds. The main result is that while the landscapes are different, they are sufficiently similar in order for the learning to inform evolution about promising areas worth exploring. See Figure 3.7c for an image of the corresponding robot (evolved for swimming near the sea floor).

The learning rule described in the previous section has three parameters: (1) the duration of the final fitness evaluation, (2) the duration of each learning step and (3) the number of learning steps. The rest of this section describes how the value for each of these parameters was selected.

The first parameter (duration of the fitness evaluation after learning) significantly impacts the behavior of evolved creatures. With short duration of fitness evaluation, evolution favours creatures which simply perform a strong initial push at the start of the simulation, but often fail to continue moving afterwards. Longer fitness evaluations are needed to prevent such behaviors, however long durations also incur extra computational cost. A set of preliminary experiments was performed to find the minimum duration of the fitness evaluation that results in creatures that maintain their speed for a long time period. Each evaluation duration was tested using 50 independent evolutionary runs each ending after 200 generations. Average speeds of the best evolved creatures for each duration are shown in Figure 3.5 (left). The fastest creature evolved in each run has then been re-evaluated using a long (64 second) fitness evaluation to see if the average speed was maintained. Results (shown in Figure 3.5, right) show that very short durations of fitness evaluation result in creatures that fail to maintain their fitness (average speed) in the longer simulation[2]. Based on these preliminary experiments, 48 second duration of the fitness evaluation was chosen for all experiments in this chapter – the smallest value that reliably results in creatures that perform at least as well as creatures evolved using the longest duration.

The second parameter (duration of the learning step) was set to 2 seconds – a time in which most evolved creatures complete their first swimming stroke. On its own, learning based on such short tests is not guaranteed to produce creature good at swimming for long periods of time (in fact, short learning steps can fail to correctly predict the outcome of the full test), but as results of our

[2]All boxplots in this thesis use the whisker bars for minimum and maximum value, box boundaries for 1st and 3rd quartile, horizontal line for the median and black dot for the mean.

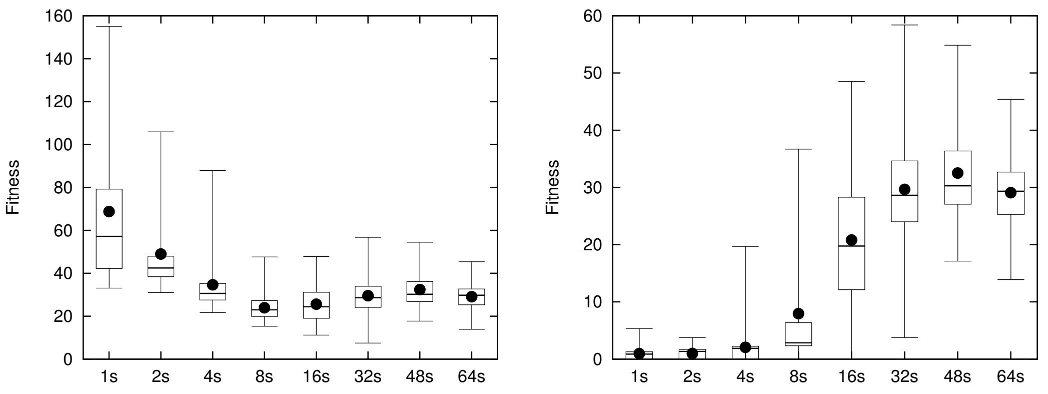


Figure 3.5: **Evolution of stable swimming behavior requires long fitness evaluation.** Fitness of the best creatures evolved for swimming using different fitness evaluation durations (left) compared to the fitness of the same creatures evaluated using a long (64 second) fitness evaluation (right). Note the difference in the ranges of the vertical exes between the two graphs. Short fitness evaluation durations result in creatures that achieve high average speed initially but fail to maintain the speed during longer period.

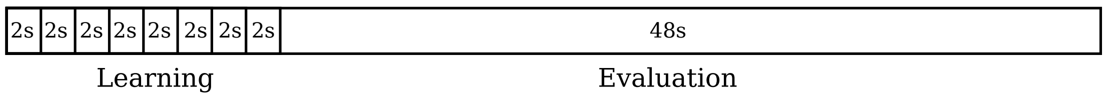


Figure 3.6: **Duration of learning compared to fitness evaluation.** Each fitness evaluation starts with a hill-climbing learning phase in which different morphology changes are tested.

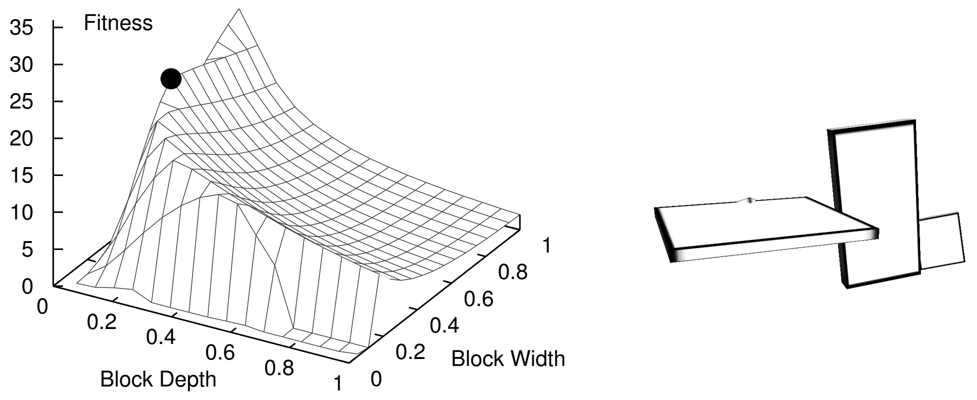
experiments show, even such a short test provides enough information to guide evolution towards more promising areas of the fitness landscape if each creature is also evaluated in a long-duration test after learning.

The third parameter controls the number of learning steps performed for each individual before it is evaluated in a long-duration final fitness test. Larger values allow hill-climbing to explore neighborhood of the individual more extensively, while also increasing the computational cost. To keep the computational cost of learning low, 8 learning steps have been used in all experiments in this chapter. Diagram visualizing the amount of time used for learning steps and for the final fitness evaluation is shown in Figure 3.6.

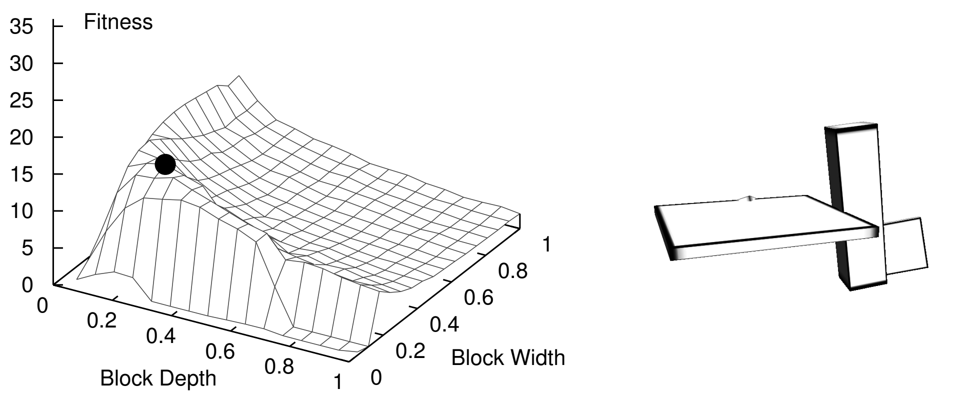
3.4 Adaptation to Different Environments

In this section we attempt to answer the first question posed at the start of this chapter. We show how learning can be used to evolve virtual creatures capable of taking advantage of differences in the environment by adjusting their morphology to increase their fitness (similar to water flea growing spikes when it detects presence of a predator or ants growing bodies of a different size and proportions depending on which caste they are born into).

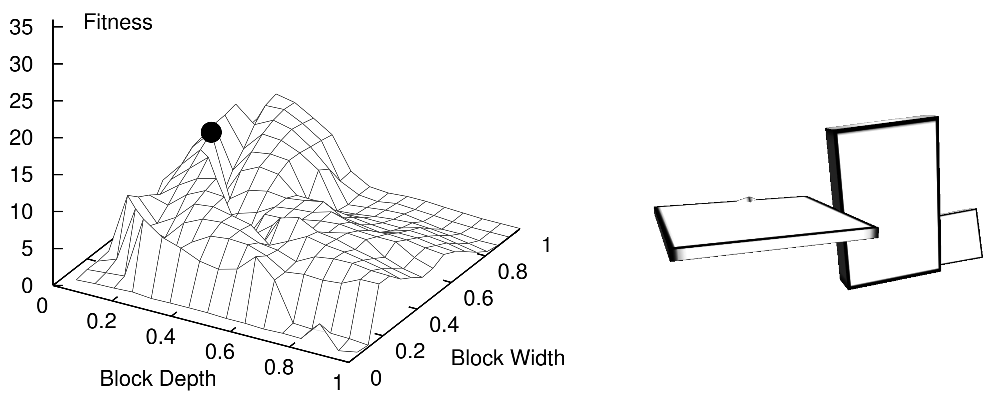
To evolve virtual creatures capable of adapting their morphology to different



(a) Open Water



(b) Low Viscosity



(c) Sea Floor

Figure 3.7: **Learning landscapes.** Two-dimensional cross-sections of nine-dimensional learning space for a simple evolved swimming creature (each dimension of each block contributes one dimension). Each plot shows swimming speeds achieved in a given environment for varying widths and depths of the central block of the creature. Creature displayed next to each plot shows the best size of the central block (also marked using black dot in the plot). Size of the central block is varied along two axes: depth (into the page) and width (horizontal across the page).

environments we use HierarchicalNEAT algorithm as described in Section 2.4 with a modified fitness evaluation. The new fitness evaluation consists of two independent tests of a given virtual creature, each in a different environment. The minimum per-environment fitness value is used as the final result. An individual therefore needs to perform well in both environments in order to receive a high fitness value. Morphological changes discovered by learning are not copied back into the genotype (the evolution is Darwinian, not Lamarckian).

Three different environments were used to test the ability of virtual creatures to adapt their morphology to the environment through learning:

Open Water Virtual creature is placed in a simulated water environment without any obstacles. No extra gravity or buoyancy forces are added so the creature is suspended in water unless it actively performs movement. Forces simulating viscosity of water are enabled.

Sea Floor A single plane is added to the environment simulating the sea floor. At the start of the simulation, creature is positioned such that its lowest point is in contact with the plane. Reduced gravity of 2 m/s^2 is enabled to simulate density of blocks higher than the surrounding water. The environment is otherwise identical to the open water environment.

Low Viscosity Fluid Viscosity forces applied to the virtual creature are reduced by 50% to simulate a low viscosity fluid surrounding the creature (such as water at moderately higher temperatures). The environment is otherwise identical to the open water environment.

Two different experimental setups are used, each using a different pair of environments. The goal of the first experiment is to evolve creatures capable of swimming in both open water and in the sea floor environment. In the second experiment open water and low viscosity fluid environments are used instead.

While creatures can generally use the same movements to propel themselves forward in all three environments with some degree of success, the differences in environments can be exploited to achieve higher fitness in each of them. For example, since the sea floor is immovable, creatures can push against it more effectively than against the water to achieve movement. Since the floor is perfectly smooth such push must rely on the surface friction forces to be effective, which in turn depend on the weight of the creature. Heavier creatures thus might be able to push against the floor more effectively and achieve higher speeds. Figure 3.7 illustrates the differences between the three environments in more detail. The shape and ruggedness of the learning space are different in each environment, as is the position of the optimum.

The following configurations have been tested for each pair of environments:

Evolution with learning Learning was performed during evolution as part of each fitness evaluation as described in Section 3.2 and Section 3.3. This configuration allows creatures to learn morphology specific to each environment.

Evolution without learning No learning was performed as part of fitness evaluation. In this configuration, creatures were forced to use the same morphology in both environments.

Evolution in one environment only (baseline) Creatures were evolved using only one of the environments and without learning. After evolution finished, the best creature from each generation was evaluated in both environments and the smaller fitness was used as the resulting fitness (the same as in the first two configurations). This configuration serves as a baseline to see how do the creatures evolved in one of the environments perform in the other environment without any previous exposure to it.

3.4.1 Parameter Settings

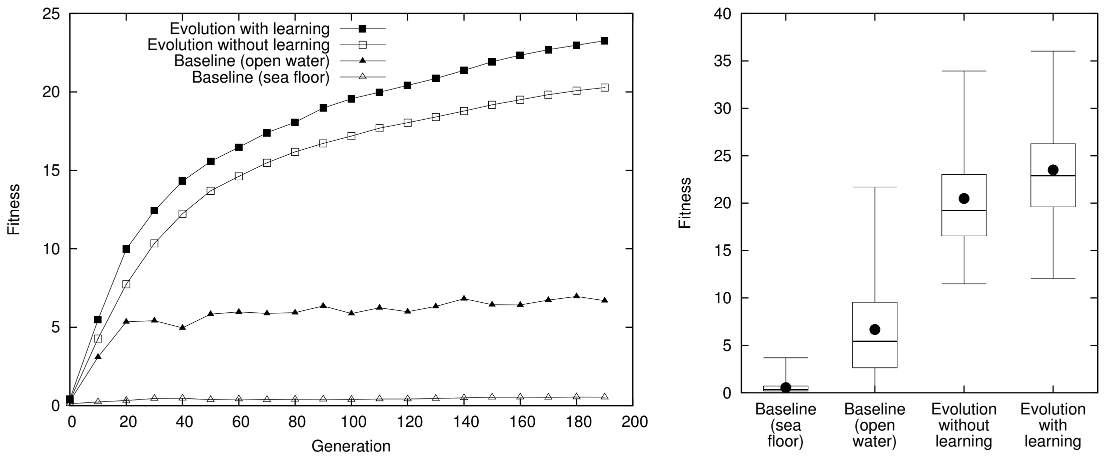
HierarchicalNEAT algorithm described in Section 2.4 was used in all experiments, using the same configuration parameters unless stated otherwise. Population size was set to 300 in all experiments. All evolutionary runs were stopped after 200 generations. Each configuration was tested 50 times. Significance levels were calculated using Student's t-test. Maximum size of each block along each dimension was set to 1. Volume of any block was not permitted to decrease below 0.008 during learning or during mutation. Random changes to each side of each block were selected uniformly from range [-0.05, 0.05] in each learning step.

3.4.2 Results and Discussion

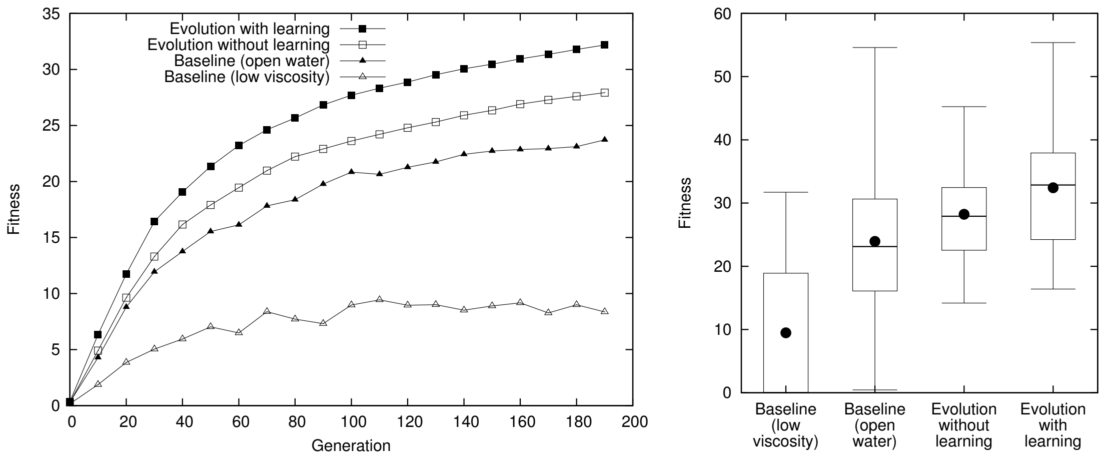
In both experiments, evolution combined with learning achieved higher average fitness values in the last generation compared to evolution without learning (an improvement of 15% in both cases, $p < 0.01$, see Figure 3.8). Moreover, evolution with learning outperforms evolution without learning not just in terms of fitness achieved after a fixed number of generations, but also when accounting for the extra computational cost that learning incurs. Since learning increases the total cost of each fitness evaluation by one third (the total simulation time required for a single evaluation increases from 48s to 64s, see Figure 3.6), the extra computational cost was accounted for by comparing results from generation 150 of evolution with learning with results from generation 200 of evolution without learning. Such comparison shows that evolution with learning outperforms evolution without learning by 9% in open water/sea floor experiment and by and 7% in lower/higher viscosity experiment ($p < 0.05$ in both cases). These results show that allowing creatures to specialize for each environment through learning allows them to reach higher fitness values compared to creatures that are constrained to use the same morphology in both environments.

Large diversity of creatures has been evolved in all experiments. The most common type was a snake-like creature swimming in a sinusoidal pattern. Snake-like creatures were more common in evolution with learning (14% of all runs) than in evolution without learning (6% of all runs), suggesting that this type of creature benefits from being able to adapt to different environments (see Figure 3.9 for an example of a snake-like creature). In general, evolved creatures made significant use of features provided by the encoding, such as symmetry and repetition. Examples of evolved creatures are shown in Figure 3.2 at the start of this chapter and in Figures 3.12 and 3.13.

Morphological adaptations learned through creature's lifetime strongly reflect the environment (see Figure 3.10). Learning in low viscosity environment on

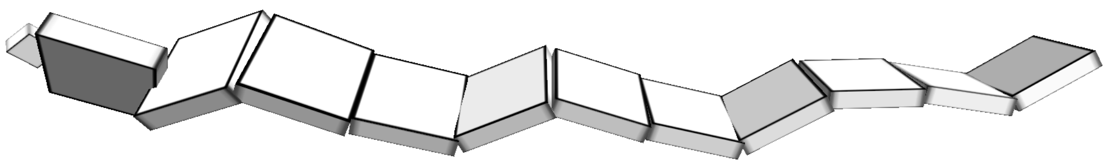


(a) Evolution of creatures capable of swimming in both open water and on the sea floor

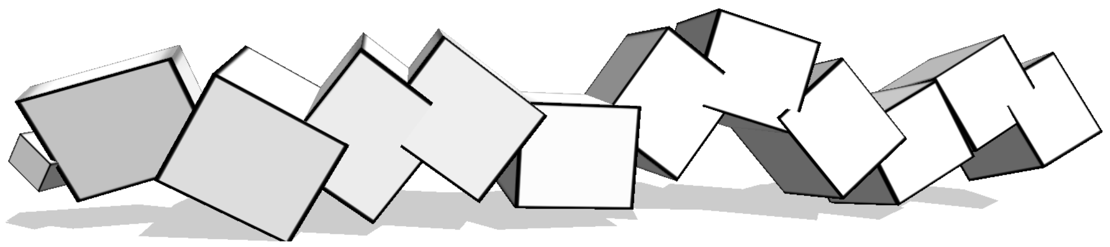


(b) Evolution of creatures capable of swimming in fluids with different viscosities

Figure 3.8: **Results of experiments with creatures adapting to a pair of environments.** Allowing creatures to adapt their morphology to each environment improves performance compared to evolution without learning.



(a) Open Water



(b) Sea Floor

Figure 3.9: **Different phenotypes of a snake-like robot.** Different phenotypes resulting from adaptation of a single virtual robot to different environments. Video is available at `https://youtu.be/VbP8CN53nWk`.

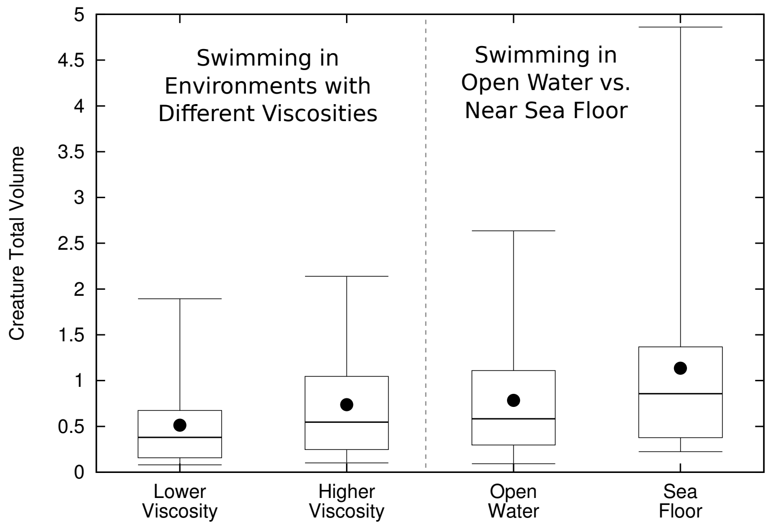


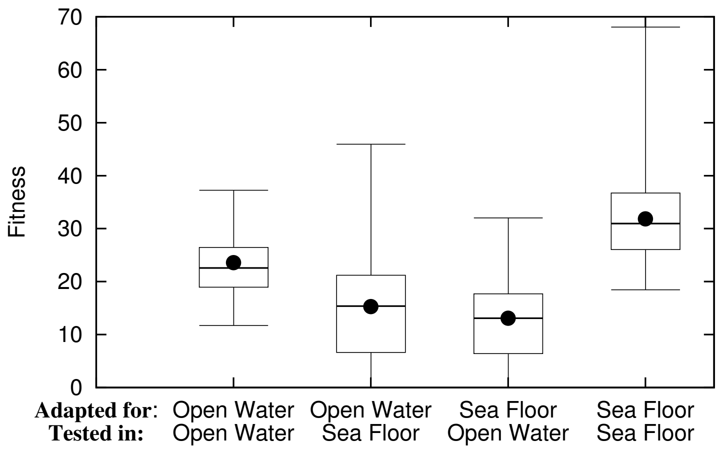
Figure 3.10: **Volume of phenotypes expressed in different environments.** The main result is that in the viscosity experiment, phenotypes expressed in low-viscosity environments are significantly smaller than phenotypes expressed in higher-viscosity environments. The relationship is reversed with open water and sea floor environments.

average lead to reduction of the total volume (and therefore mass) of creature's body by 30.1% compared to environment with higher viscosity (0.51 vs. 0.73, $p < 0.02$). Conversely, in the open-water/sea-floor experiment the volume of phenotypes expressed in the sea floor environment was on average 44% larger than for phenotypes swimming in open water (1.13 vs. 0.78, $p < 0.05$). As discussed above, the difference likely comes from the fact that larger blocks provide more weight, and thus more friction against the sea floor allowing the creatures to make better use of the floor for locomotion.

Results of baseline experiments confirm that evolution of creatures exposed to only one of the environments results in creatures that perform significantly worse in the other environment compared to creatures exposed to both environments ($p < 0.01$ in all cases). For instance, creatures evolved for swimming near the sea floor without being exposed to open-water environment often rely on sea floor friction to such an extent that they are not able to move forward at all without the sea floor (see Figure 3.8a). On the other hand creatures evolved for swimming in higher-viscosity fluid without being exposed to lower-viscosity fluid performed quite well in low-viscosity fluid without any further adaptation (although still significantly worse than creatures exposed to both environments).

To see the effect of morphological plasticity more clearly, the best creature discovered in each run of evolution with learning was subjected to learning in one of the environments and then placed in each of two the environments with no further learning. Results (shown in Figure 3.11) show that morphological adaptations discovered through learning are specific to one environment and cause degradation of performance in the other environment (in other words each creature performs better in the environment it used for learning than in the other environment).

Morphological plasticity thus allows creatures to increase their fitness in each environment independently. The same effect is also demonstrated in Figure 3.12



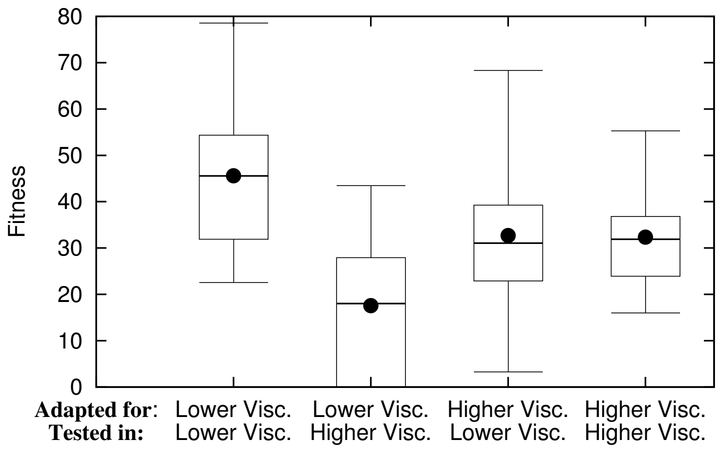


Figure 3.11: **Performance of robots when tested in a different environment than the environment used for learning.** The main result is that morphological changes discovered through learning are beneficial in the environment used for learning but detrimental when used in the other environment.

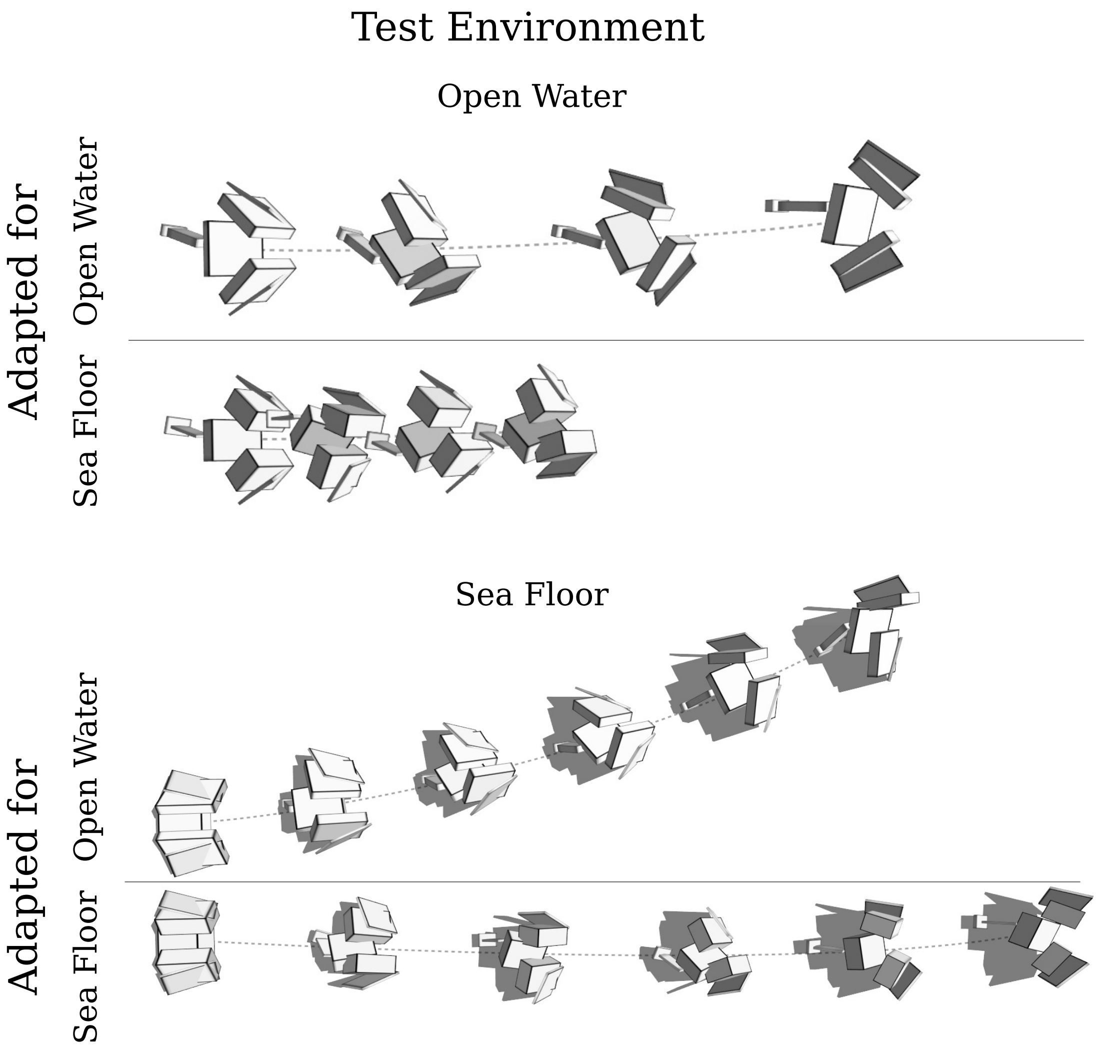


Figure 3.12: **Environment-specific phenotype adaptations (open water vs. sea floor).** Comparison at the top shows that phenotype adapted for swimming in open water performs better in open water than phenotype adapted for swimming near the sea floor. Performance of these two phenotypes is reversed in the sea floor environment (bottom). Both phenotypes are a result of learning algorithm applied to the same genotype. Phenotypes are also shown in Figure 3.2a. Video is available at `https://youtu.be/GX1BNduCvmo`.

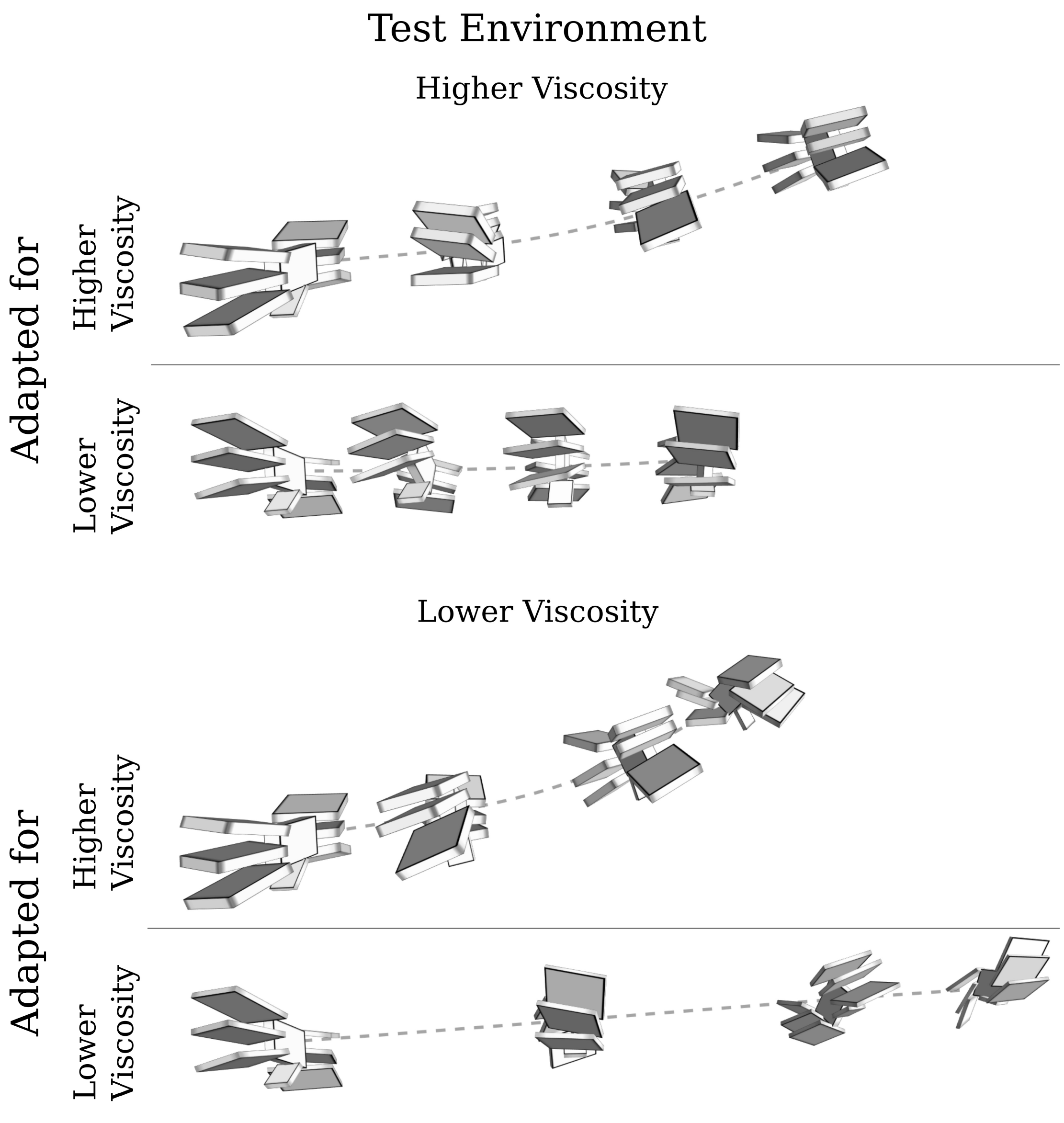


Figure 3.13: **Environment-specific phenotype adaptations (lower and higher viscosity).** Comparison at the top shows that phenotype adapted for swimming in higher viscosity fluid performs better in higher viscosity fluid than phenotype adapted for swimming in lower viscosity fluid. Performance of these two phenotypes is reversed in the lower viscosity environment (bottom). Both phenotypes are a result of learning algorithm applied to the same genotype. Phenotypes are also shown in Figure 3.2b. Video is available at `https://youtu.be/hFzeowtPmmk`.

and Figure 3.13. Each of these figures shows two morphological adaptations of a creature encoded by the same genotype, and the performance of each of those two adaptations in both environments. Performance in the environment that was used for learning is better than performance in the other environment.

Results presented so far do not make it clear whether learning improves evolution by altering the paths evolution takes through the fitness landscape, or whether learning is *orthogonal* to evolution, i.e. final result of evolution with learning would be the same if learning was performed only at the end of evolution. To answer this question, the best creature from each run without learning was subjected to a learning phase consisting of 100 learning experiments each 48 seconds long (in each environment separately) and its fitness was measured afterwards. The average improvement from this extended learning was only 0.8% in the sea floor experiment and 0.9% in the low viscosity experiment. Large proportion of creatures failed to improve at all (42% and 47% respectively). This suggests that when learning is performed during evolution as opposed to after evolution, it helps evolution guide the search to more promising areas in the fitness landscape and it cannot be replaced by performing learning only after evolution.

3.5 Learning in a Single Environment

In previous section we have shown that evolution with learning can outperform evolution without learning when each creature uses learning to adapt to each of the two different environments. Performance improvement observed in previous section can be explained by two processes:

Specialization Being able to adjust morphology for each environment gives learning creatures wider set of options when searching for high fitness values than options available to "generalist" creatures which are constrained to use the same morphology in both environments. As shown in previous section, the best evolved creatures take advantage of the fact that they can use learning to adapt their morphology and moreover, adaptations are systematic—all creatures tend to reduce their body mass in one of the environments compared to the other environment.

The Baldwin Effect As discussed at the beginning of this chapter, learning alone can make evolution faster by efficiently exploring the local neighbourhood of the individual in the learning landscape. Even when the changes discovered by learning are not passed to the genotype of the creature, the information discovered by learning can inform evolution about promising areas of the search space.

In this section we attempt to separate the effects of Baldwin effect from the effects of specialization by running evolution with learning using single environment only. Since fitness evaluation only occurs in one environment, performance gains cannot come from the ability of creatures to specialize, but must come only from the interaction of evolution with learning.

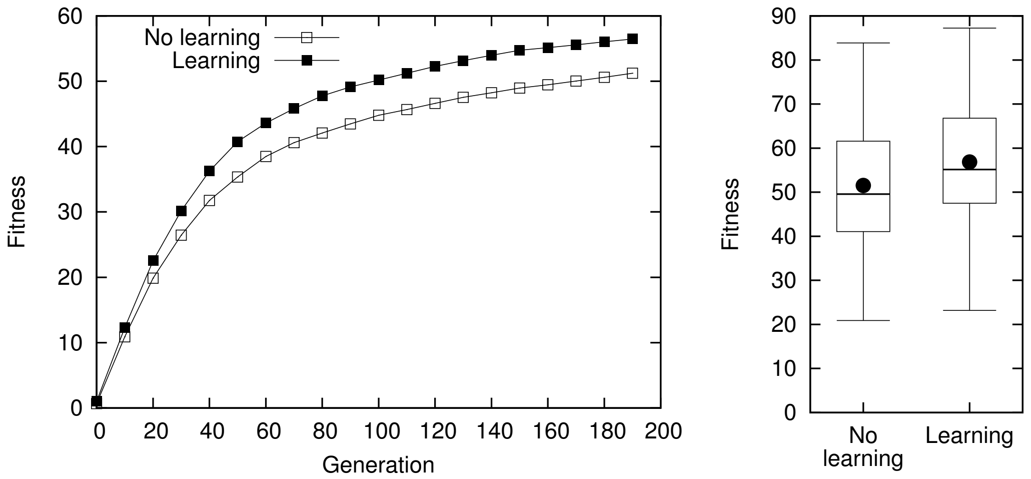


Figure 3.14: **Evolution with and without learning in a single environment.** Evolution combined with learning outperforms evolution without learning even when creatures are evolved for one environment only (swimming in low viscosity fluid).

3.5.1 Experiment Setup

All methods and configuration parameters used to run the experiment are identical to experiments described in the previous section with two exceptions. First, to evaluate fitness of an individual, the individual is subjected to learning (using Algorithm 3.2.1) in only one environment and the fitness achieved after learning in this environment is returned to the evolutionary algorithm. The environment used for the experiment has been chosen arbitrarily to be the low viscosity fluid. Second, the number of runs was increased to 200 to take into account smaller magnitude of the observed effect.

3.5.2 Results and Discussion

Results of experiments (averaged over all runs) are shown in Figure 3.14. Evolution without learning achieved average fitness value of 56.04 in in the last generation compared to 51.52 for evolution without learning (an improvement of 8.7%, $p < 0.03$). When extra computational cost used by learning is taken into account (i.e. results of generation 150 of evolution with learning are compared to generation 200 of evolution without learning, the same as in Section 3.4.2), evolution with learning still outperforms evolution without learning by 4.0%. These results show that learning has a significant positive effect on evolution and they suggest that Baldwin effect plays a significant role even in multi-environment tests presented in the previous section. Examples of several evolved creatures are displayed in Figure 3.15 (note examples of repetition of segments and two-fold and four-fold symmetry).

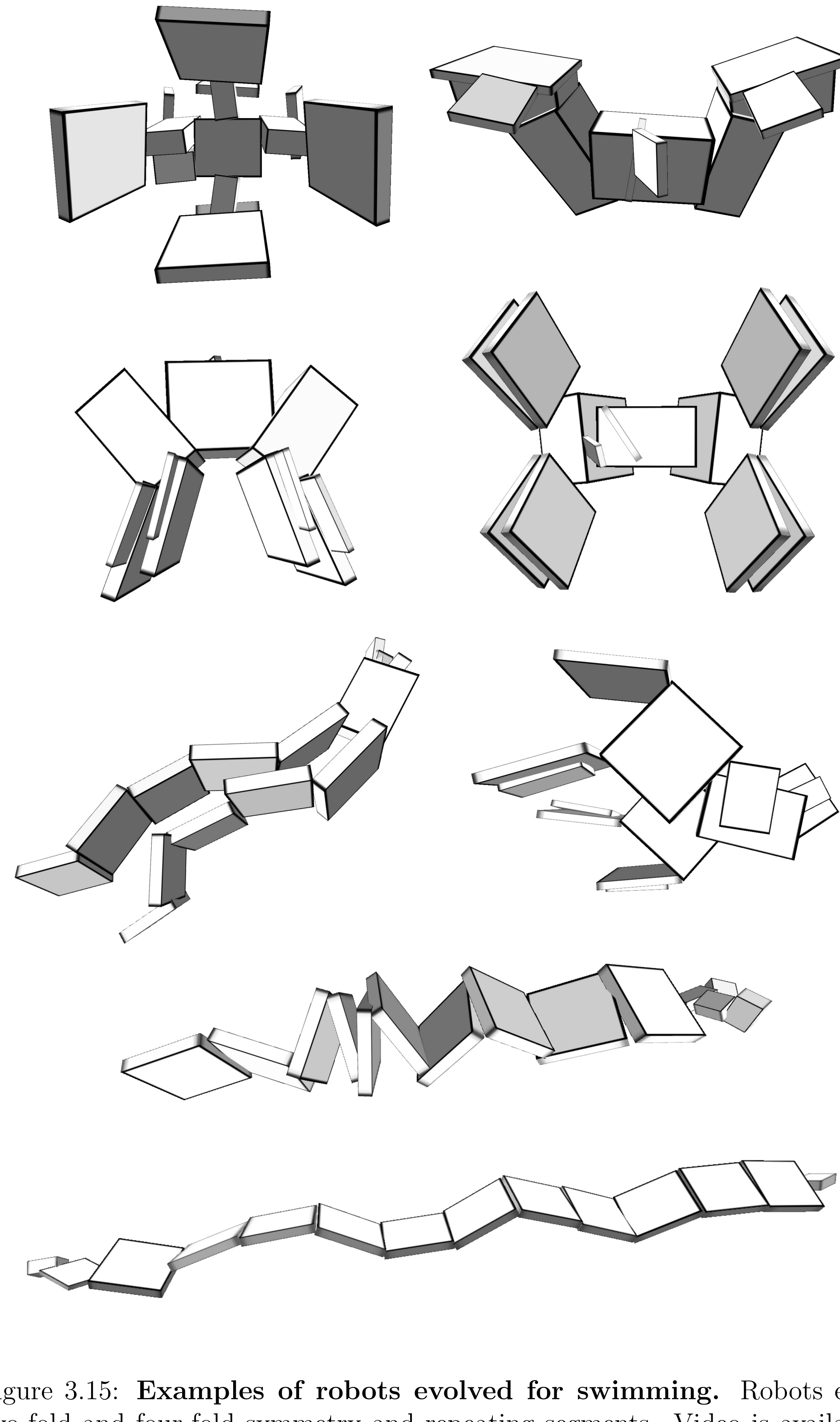


Figure 3.15: **Examples of robots evolved for swimming.** Robots exhibit two-fold and four-fold symmetry and repeating segments. Video is available at `https://youtu.be/A9gj7taU0SM`.

3.6 Future Works

In this chapter we have shown that morphological plasticity (achieved through lifetime learning) can improve performance of evolutionary search and it can allow robots to adapt their morphology to different environments. We have shown that the benefits provided by the new method generalize across different pairs of environments. One interesting area of future research is to investigate which tasks other than swimming and which other environments can benefit from such morphological plasticity. The method should be applicable to any problems where the robot needs to maintain its performance for a longer period of time and where future performance of the robot is correlated with its performance at the start of the test. Examples of such problems are flying, following a distant light source or terrestrial locomotion.

This work focused solely on morphological plasticity which has not been studied before in the context of body-brain co-evolution (to the author's knowledge). While we have shown that such plasticity can provide significant benefits, the fact that learning cannot modify the behavior of the robot or make more significant changes to its morphology limits the extent of adaptation achievable using this method. One natural extension of this work would be to apply the same method to the control system of the robot or to allow learning to experiment with more significant changes to robot's morphology. Examples of such morphological changes could be adding or removing a block or changing a recursive limit (resulting in new copy of an existing block being added to the creature). With regards to plasticity of the control system, preliminary experiments with Hebbian learning (where weights of neural connections are updated based on how are they used by the individual during its life) and with learning based on black-box optimization of neural networks (using similar hill-climbing algorithm as presented here) have been conducted but they have not yielded interesting results yet.

Due to high demands on computational resources, each evolutionary run in this thesis had to be terminated after 200 generations. It would also be useful to compare performance of different configurations of evolutionary algorithms in terms of the maximum *achievable* fitness. However, this would require running each experiment to the point where the fitness stops improving. In practice this can take thousands of generations and such experiments were not feasible for this thesis due to their high computational cost.

High computational cost was also the primary reason for the following issue: many of the creatures evolve behavior which is sensitive to the exact conditions creature starts in. This problem can be solved by evaluating each creature many times, in different starting configurations, and returning the mean (or minimum) fitness. This approach has not been used for evolution of virtual creatures in this thesis due to its high computational cost. This effect also required resetting the creature to its starting configuration after each learning step. While this approach complicates transfer of evolved creatures to real robots, it is used commonly in evolutionary robotics [7, 14] due to high computational cost of making the behavior of robots less sensitive to starting conditions.

Learning method presented in this thesis uses hill-climbing algorithm to search the local neighborhood of the individual. Other variations of hill-climbing (or even other black-box optimizations methods) could be used instead. However, preliminary experiments have shown that hill-climbing provides significant advantage over random search (where morphology encoded in the genotype is randomly modified in every step instead of modifying the best morphology discovered by learning so far, i.e. L_{best} on line 3 in Algorithm 3.2.1 is replaced by L_{geno}). Preliminary experiments have also shown that increasing the learning rate (i.e. magnitude of random changes introduced in each learning step) hurts performance of the algorithm suggesting that it is more advantageous to gradually climb the local gradient as opposed to attempting to guess a larger step. Preliminary experiments with Lamarck-style evolution (where improvements discovered by evolution are copied back to the genotype) have not shown significant improvements in the achieved performance, but further research is required to confirm the result conclusively.

Another interesting avenue of future research would be to provide more direct evidence of different components of the Baldwin effect. As an example, assimilation of acquired traits into genotype (as described in the introduction to this chapter) could be shown to occur in virtual creatures in a similar way as in animals, i.e. morphological changes discovered through learning become part of the genotype of the creature later in evolution.

3.7 Conclusion

In this chapter we studied interaction of evolution and learning in the context of body-brain co-evolution of virtual robotic creatures. We introduced a method which allows evolution of robots capable of adjusting their morphology to different environments—the method is inspired by several cases of such morphological adaptations found in nature. We show that the evolution of learning creatures outperforms evolution without such morphological plasticity, both in terms of the fitness value reached after a fixed number of generations and also in a more strict sense, when taking into account extra computational cost of learning during each fitness evaluation. The main insight of the new method is that the computation cost of learning can be significantly reduced by shortening the learning phase. While this reduces accuracy of learning (i.e. learning can result in morphology which fails to maintain good performance during a full-duration test), the gradient of the search space is preserved sufficiently for evolution to use it efficiently.

We show that resulting performance improvements generalize across different sets of environments (swimming in fluids with different viscosities and swimming near the sea floor vs. in open water) and that evolved creatures tend to adjust their bodies in similar ways in the same environments (e.g. swimming near the sea floor leads to increased body mass compared to swimming in open water). Further experiments have demonstrated that when learning is performed during evolution (as opposed to *after*) it results in improved performance of the search, showing that learning is not *orthogonal* to evolution, but has significant impact on the evolutionary search itself (even when improvements discovered by learning are not copied back to the genotype in the next generation).

In the second part of this chapter we have shown that learning improves performance of evolution not just when robots can learn different morphology in each of several environments, but also when robots are evolved in one environment only. The improvement in this case cannot be explained by ability of robots to *specialize* for each environment. Instead, it is explained by the fact that learning explores the neighborhood of the individual in the learning landscape and this exploration can guide evolution towards more promising areas of the search space.

In summary, we have shown that morphological plasticity can make evolution of robotic creatures more efficient and can therefore be a valuable tool in the field of evolutionary robotics.

Chapter 4

Overcoming Deceptiveness with Novelty Search

Evolutionary algorithms are an often used technique for designing morphology and controller of a robot (an overview of methods is provided in Chapter 2). The advantage of using evolutionary algorithms compared to other optimization methods is that only a high level fitness function and a set of genetic operators (such as mutation and crossover) need to be specified. While such fitness-driven search can be very successful, a common problem in evolutionary algorithms is premature converge to local optima. Premature convergence occurs when the search finds the locally best solution in some part of the search space, however the search stagnates because none of the genetic operators can produce an individual better than the locally best individual found so far. A typical symptom is that the population converges to a point where all individuals are very similar to each other, i.e. diversity is lost.

The first solution to this problem might be to try to increase mutation rate to produce individuals further away from the local optimum. However, such technique on its own is unlikely to succeed. While large random mutations can move the search to a completely different part of the search space, they seldom produce good solutions without further optimization (which often does not have a chance to occur before the individual is pruned by selection). A large mutation applied to an individual which has already been finely tuned by a long series of successful mutations is more likely to break the functionality discovered so far than to improve it further. One solution to this issue is to *shield* individuals created by such large mutations from direct competition with previously best individuals for a number of generations to see if they can be significantly improved by further mutations (i.e. if the performance lost after the large mutation can be recovered through further search in the neighbourhood of the newly created individual). This idea forms the basis of the speciation algorithm used in HierarchicalNEAT algorithm presented in Section 2.4. While it has been shown to delay evolution from getting trapped in local optima (see Figure 2.17 for an example of evolution escaping a local optimum), HierarchicalNEAT does not solve the underlying problem completely.

In general, premature convergence to local optima is caused by too much focus on exploitation of already discovered areas of the search space as opposed to exploration of yet unknown solutions. The trade-off between exploration and exploitation is well understood and a number of methods for addressing this issue have been proposed (overview is provided in [22]). The standard approach to prevent convergence to a local optimum is to use methods that attempt to increase the rate of exploration by maintaining the diversity of individuals in a population through techniques such as fitness sharing [23, 91], age-layered population structure [32], multi-objectivization [36], hierarchical fair competition [34] or fitness uniform selection [35]. Several of such diversity maintaining methods have also been applied to evolutionary robotics [69, 8], including shielding of novel individuals from competition mentioned in the previous paragraph.

While such diversity-preserving methods can delay convergence to local optima, recent works suggest that the underlying problem lies not just in the balance of exploration and exploitation, but in the deceitfulness of the fitness function itself [54, 80, 17, 90]. Novelty search, a method recently proposed by Lehman and Stanley [54], introduces a radical idea that convergence to local optima can be completely avoided by simply ignoring the original objective altogether and instead only searching for *any* novel behaviors regardless of their quality with respect to the original fitness function. While seemingly counter-intuitive, this approach has already been successfully applied to problems in several domains: evolving neuro-controllers for robot navigation in a maze [56, 54], evolving genetic programs [17, 55, 64, 95], evolution strategies [13], evolving *learning* artificial neural networks for maze navigation [80], swarm robotics [24] and data clustering [71].

In this work, we use search for behavioral novelty as a method for avoiding premature convergence to local optima in the evolution of body and controller of a virtual robotic creature. Both body and controller of the robot are subject to optimization by evolution, forming a larger and more complex search space than in domains where novelty search was previously tested. We demonstrate the advantages and disadvantages of novelty search in this domain in two experiments: one with a deceitful fitness function with constrained space of possible behaviors and one with a large behavioral space and a less deceitful fitness function.

We start the chapter by describing the novelty search algorithm (Section 4.1). Section 4.2 then describes the setup of experiments with virtual creatures. Results of experiments along with examples of evolved behaviors are provided in Section 4.3. Chapter concludes with the discussion of results (Section 4.4), a summary of future works and a conclusion.

4.1 Novelty Search

Instead of following the gradient of the fitness function, novelty search (originally introduced by Lehman and Stanley [54] in the context of evolution of neural controllers) directs the search towards any yet unexplored parts of the behavior space. This is achieved by modifying the search algorithm (in this case HierarchicalNEAT described in Section 2.4) to use the measure of individual's behavioral novelty instead of the fitness function. No other modifications to the underlying search algorithm are required.

Pseudocode for the algorithm calculating behavioral novelty is shown in Algorithm 4.1.1. As a first step, robot's behavior is measured (MeasureBehavior(α)

Algorithm 4.1.1 Measuring Behavioral Novelty.

```
Require:
    $\alpha$, phenotype of the individual being measured
    $\epsilon$, threshold for adding an individual to the archive
    $M$, archive of previously discovered behaviors
1: $x \leftarrow$ MeasureBehavior($\alpha$)
2: $\delta \leftarrow$ CalculateNovelty($x, M$)
3: if $\delta > \epsilon$ then
4:     AddToHistory($x, M$)
5: end if
6: return $\delta$
```

function call on line 1) by placing robot in its simulated 3D environment, letting it perform the given task and measuring a vector of real values representing robot's behavior.

Measure of individual's novelty is then calculated (function CalculateNovelty(x, M)) using an archive of previously discovered behaviors M, as an average distance of the individual's behavior from k closest behaviors recorded in the archive:

$$\delta(x) = \frac{1}{k}\sum_{i=1}^{k} dist(x, \mu_i) \tag{4.1}$$

where μ_i is the ith-nearest neighbor of x and with respect to distance metric *dist* (in this work Euclidean distance was used, but other metrics could be used as well). Such measure provides an estimate of local sparseness in the vicinity of the behavior being measured. Novelty search thus promotes individuals which are further away from already discovered behaviors.

If novelty of an individual exceeds a threshold value, individual is added to the archive (function AddToHistory(x, M)). Threshold is used to reduce the size of the archive while keeping the archive sufficiently accurate. In this work the size of an archive was unlimited (previous works have successfully used limited archive [55] as well).

4.2 Experiments

By searching only for novelty in behavior, novelty search is less prone to falling into traps set up by the objective-based fitness function in the form of local optima. However if the behavior space is too large and unconstrained, novelty search may spend most of the time exploring behaviors that are uninteresting with respect to the objective and never find any useful solutions. To test both ends of this scale in the domain of body-brain co-evolution, we perform two experiments: one non-deceptive with a large unconstrained space of behaviors (the swimming experiment) and one highly deceptive with a constrained behavioral space (the barrier avoidance experiment).

In each experiment, the performance of novelty search was compared to the performance of standard fitness-based search. Random search was used as a

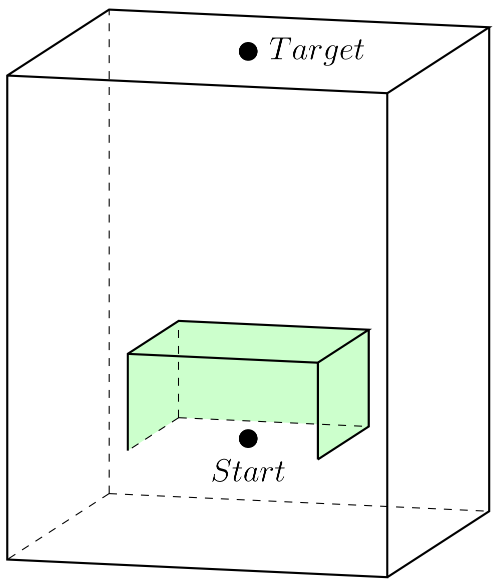


Figure 4.1: **Barrier avoidance experiment.** Dimensions of the environment are 20m x 20m x 25m. The barrier is located between the starting position of the robot and the target and is formed by a hollow 10m x 10m x 5m box (shown in green) with the bottom face left out (front face of the barrier has been left out from the figure only for visualization). The target is located at the top of the container at the initial distance of 20m from robot's initial position.

baseline when comparing performance in each experiment (random search was implemented by performing selection randomly, regardless of the fitness of an individual). HierarchicalNEAT as described in Section 2.4 was used at the underlying evolutionary algorithm in all three experiments (with fitness evaluation replaced by novelty measure in the novelty search setup and with selection performed randomly for the random search experiment).

4.2.1 Swimming Experiment

The objective in the swimming experiment is to evolve robots capable of moving in a water environment the largest distance from the starting position in the allocated time. The difficulty in this task is that the robot must evolve useful morphology *and* control to successfully move through water. At the start of each test, robot is placed at the origin of the coordinate system. Robot is then free to move in any direction for 60 seconds after which the position of its center of mass is recorded. Fitness function is defined as the distance between the final and starting positions of the robot. Behavior of a robot is defined as the position of the robot at the end of the test and thus consists of a vector of 3 real values. Behavior metrics is defined as Euclidean distance between the final positions of two robots. There were no barriers for the robot to avoid and no constraints on the behavior of the robot apart from the limits of the physics simulation.

4.2.2 Barrier Avoidance Experiment

In the barrier avoidance experiment, the robot is placed at one end of a box-shaped container filled with water (size of the container is 20m x 20m x 25m) while the target is placed at the other end of the container (see Figure 4.1). Robot's goal is to reach the target within the time limit of 60 seconds. The task is made deceptive by placing the barrier directly in front of the robot, so it obstructs the direct path to the target. Moreover, the shape of the barrier only makes it possible to reach the target if the robot first moves *away* from it, which makes this task highly deceptive. Fitness function for a robot with final position x is defined as

$$f(x) = \max(0, d(p_{\text{start}}, p_{\text{target}}) - d(x, p_{\text{target}})) \tag{4.2}$$

where $d(p_{\text{start}}, p_{\text{target}})$ is the constant distance from the starting position of the robot to the target (in this case 20m) and $d(x, p_{\text{target}})$ is the distance from the final position of the robot to the target. Behavior of the robot is defined as robot's final position (the same as in the swimming experiment) and behavior metrics is defined as the Euclidean distance between final positions of two robots. Target position is fixed for all experiments and robot has no information about the position of the target. As in the swimming experiment, barrier avoidance experiment is performed in a simulated water environment.

4.2.3 Parameter Settings

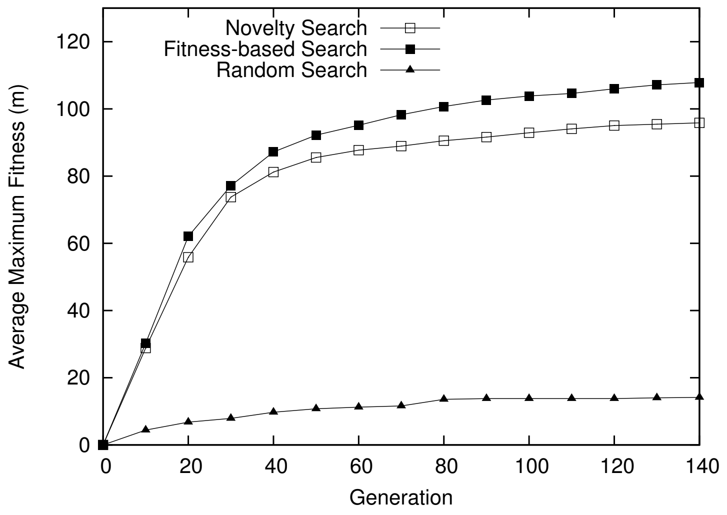
The number of individuals in a population is set to 300, with 150 generations per run. In all experiments, initial generation was initialized with uniform robots consisting of a single box with no neurons except for inputs/outputs. All parameters of the underlying HierarchicalNEAT algorithm were set to the same values in both the standard fitness-based configuration and in the novelty-based configuration. The configuration of HierarchicalNEAT was the same as in other experiments presented in this thesis. Parameter k for computing the novelty of an individual (equation 4.1) was set to 15 and the novelty threshold for adding an individual to the archive was set to 0.1. Archive size was not limited. Each configuration was tested independently 25 times. All significance levels were computed using Student's t-test.

4.3 Results

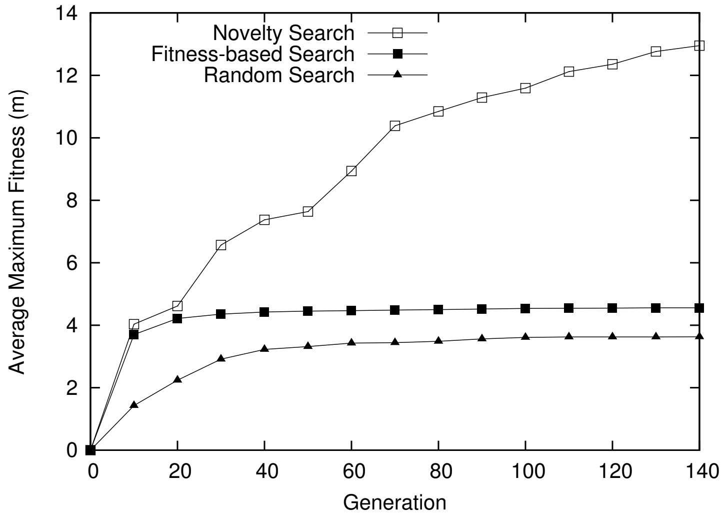
In the swimming experiment, both novelty search and fitness-based search were able to consistently find effective solutions (see Figure 4.2a). Average maximum distance of the robot from the starting position after 150 generations was 108.32m for fitness-based search, 96.23m for novelty search and 14.46m for random search. Fitness-based search thus outperformed novelty search by 12.47% ($p < 0.05$), confirming that unconstrained behavior space without deceptive fitness function is favourable to the standard fitness-based approach[1].

In the barrier avoidance experiment, fitness-based search never successfully found a way out of the barrier, reaching an average maximum fitness of 4.56m (see

[1]Video of evolved swimming robots is available at `https://youtu.be/0xvabl1DRyw`.



(a) Swimming



(b) Barrier Avoidance

Figure 4.2: **Comparison of maximum fitness achieved by novelty search and fitness-based search.** The average maximum fitness is shown for the swimming experiment (4.2a) and for the barrier avoidance experiment (4.2b) averaged over 25 runs. Results show that novelty search significantly outperforms fitness-based search in constrained deceptive task (barrier avoidance), while reaching similar performance in unconstrained non-deceptive task (swimming).

Figure 4.2b). Performance of the fitness-based search was in this case only marginally better than random search which achieved maximum average fitness of 3.68m (although the difference was statistically significant; $p < 10^{-8}$). Novelty search was the only method that consistently escaped the trap and reached an average maximum fitness of 13.16m, significantly outperforming both fitness-based and random search ($p < 10^{-10}$).

Figure 4.3 shows examples of creatures evolved in barrier avoidance experiment. Best creatures evolved using novelty search were consistently able to navigate around the barrier. Common strategy employed by successful creatures was to move along one face of the enclosure box (using the enclosure box as a guide) to move closer to the goal. Best creatures evolved in the fitness-based search, on the other hand, typically simply moved forward towards the goal until they hit the barrier and then stopped. To further improve the fitness value (i.e. reduce distance to the goal), several evolutionary runs discovered that flattening the body of the creature will bring its center of mass closer to the barrier wall and therefore closer to the goal. Examples of such creatures are shown in Figure 4.4. Creatures with such flat bodies evolved as a consequence of the fact that the position of each creature was defined as the center of mass of the creature's body. If the position was instead defined as the point on the creature's body which is closest to the goal, there would be no selection pressure for flat body structure and creatures with flat morphology most likely would not evolve (however, evolution might find other ways to exploit the fitness function in that case, such as creating long thin bodies, resulting in creatures that get closer to the goal even without moving).

4.3.1 Analysis of Behavioral Diversity

To provide better insight into how individual search methods explore the space of behaviors, the coverage of behavior space was computed for each experiment. The coverage was computed as a total number of cells from a 1m x 1m x 1m grid that contained the final position of at least one robot. This statistic was computed after each generation using all behaviors seen since the start of the run. Resulting values were averaged over 25 runs (see Figure 4.5).

Comparison of the amount of behavior space covered by individual methods shows that novelty search was exploring the behavior space faster than the fitness based search in both experiments ($p < 10^{-10}$). In swimming experiment the results demonstrate that, although novelty search achieved on average lower fitness values than fitness-based search, it was able to explore the behavior space more efficiently. The difference in how the two methods explore the behavior space can best be seen in the set of typical runs shown in figures 4.6a and 4.6b. While novelty search thoroughly and evenly explores the search space progressing in an ever-expanding sphere (Figure 4.6a), fitness-based search tends to exhaustively search a small number of promising directions (4.6b).

In the barrier avoidance experiment, coverage of the behavior space for fitness based search never exceeded $500m^3$, which corresponds to the volume of the interior space of the barrier. A typical distribution of behaviors for such run is shown in Figure 4.6d. While standard fitness-based approach always failed to escape the barrier, novelty search consistently explored the entire container,

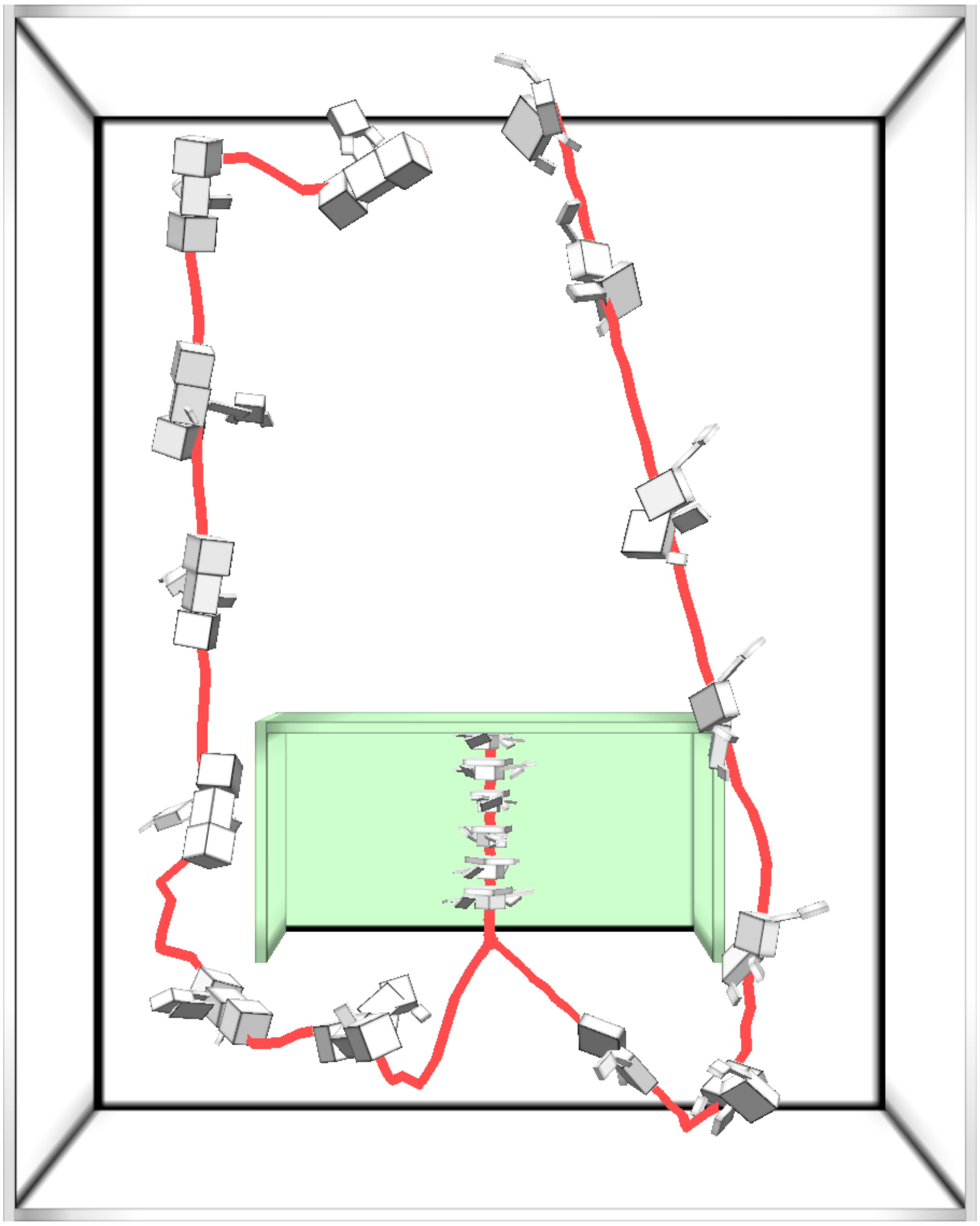


Figure 4.3: **Examples of three creatures evolved in barrier avoidance experiment.** The barrier is shown in green color and the enclosing box in white (see Figure 4.1 for a diagram of the experiment). Front face of the barrier and of the enclosure have been removed for the purpose of visualization. Red lines show how the position of the center of mass of each creature changed during simulation. All creatures start at the same starting point at the bottom of the barrier. Two creatures that successfully navigate around the barrier (creatures on the left and right) were evolved using novelty search. The third creature (center) has been evolved using the standard fitness-based approach—this creature moves directly forward towards the barrier and stops when it reaches the barrier, without ever reaching the goal at the top of the enclosure box. Video is available at `https://youtu.be/I9ggqjVp6Po`.

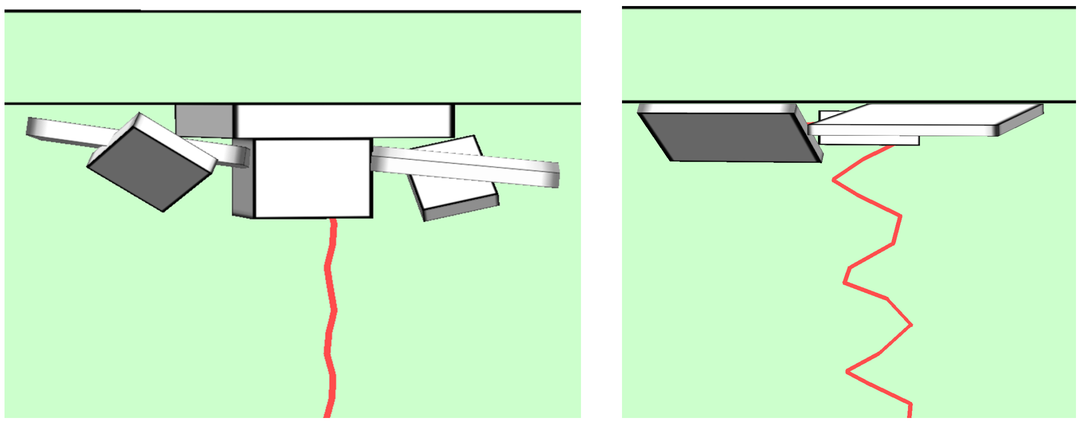


Figure 4.4: **Close-up view of two of the creatures evolved using fitness-based search at the moment when they touch the barrier.** Red lines show positions of the center of mass of each creature prior to touching the barrier. Since neither of the creatures can swim around the barrier, the only option for evolution to increase their fitness value further is to bring their center of mass as close to the barrier as possible (and therefore closer to the goal) by modifying the structure of their bodies. As a result, many creatures evolved using fitness-based search evolve flat bodies resting as close to the barrier as possible. Video is available at `https://youtu.be/Uc5wDbHz_-I`.

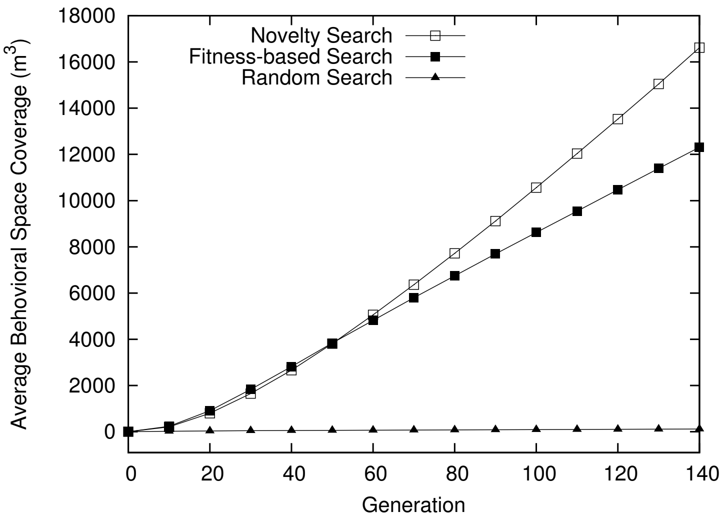
eventually reaching the target as well. Example of a typical run is shown in Figure 4.6c.

4.3.2 Combining Novelty Search with Fitness-based Search

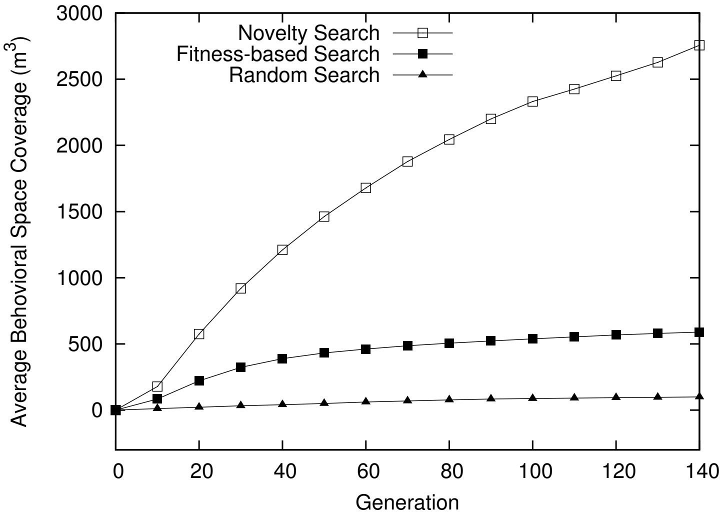
To further analyze both search methods, additional experiments were performed using a combination of the novelty search and the fitness-based search. The combined search starts with novelty search and switches to fitness-based search after 20, 40, 60 or 80 generations. The motivation for switching the methods in the middle of the search is to more efficiently exploit strengths of each method in the barrier-avoidance experiment. Once novelty search overcomes the barrier (by focusing on the exploration part of the search), fitness based search may be able to quickly converge to a solution (by focusing on the exploitation).

Results of experiments with combination of novelty search and fitness-based search (shown in Figure 4.7) show that the later the switch was made, the higher the final fitness value was achieved. The best average fitness values were reached by the novelty search (13.16m), and the combined method switching at generation 60 (12.26m) and generation 80 (13.65m). The differences between these best methods were not statistically significant ($p > 0.25$). Other two combined search methods achieved average fitness value of 10.39m (switch after 40 generations) and 6.81m (switch after 20 generations).

Comparison of the amount of behavior space covered by individual methods confirms that novelty search explores the behavior space faster than the fitness based search. In combined experiments, later switch to fitness-based search con-

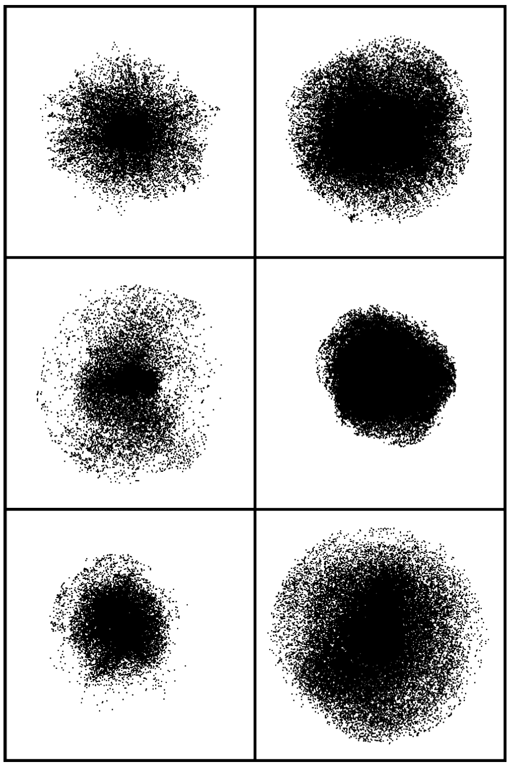


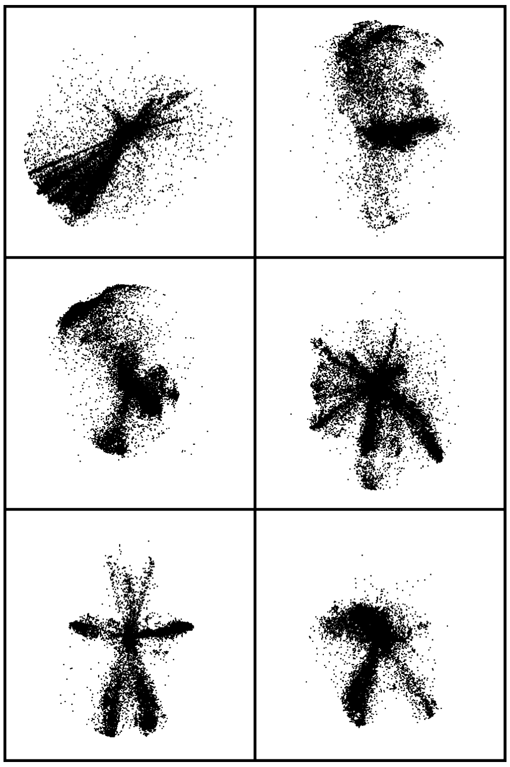
(a) Swimming

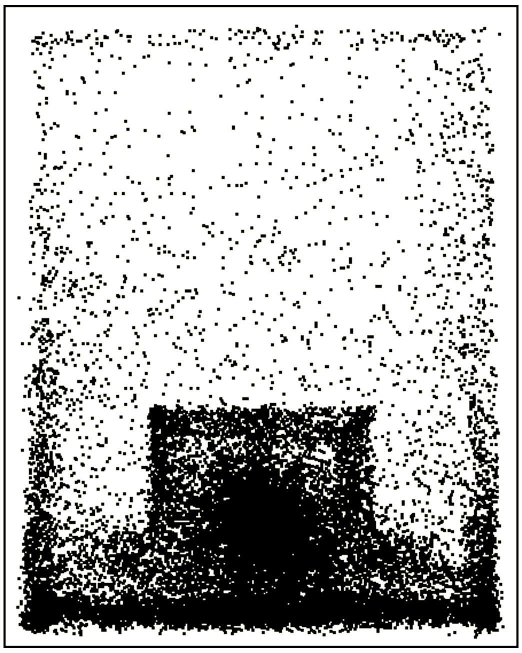


(b) Barrier Avoidance

Figure 4.5: **Comparison of behavioral diversity in novelty search and fitness-based search.** Behavioral diversity of a single evolutionary run is measured as a number of cells in 1m x 1m x 1m grid that contain at least one robot behavior discovered during the run. The cumulative behavioral diversity per generation averaged over 25 runs is shown for the swimming task (4.5a) and for the barrier avoidance task (4.5b). The main conclusion is that in both tasks, novelty search discovers significantly more behaviors than fitness-based search.


(a) Swimming (Novelty)


(b) Swimming (Fitness)


(c) Barrier Avoidance (Novelty)

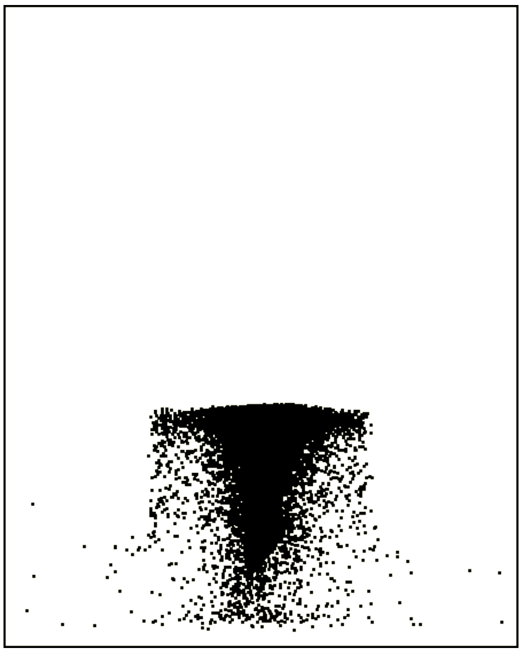
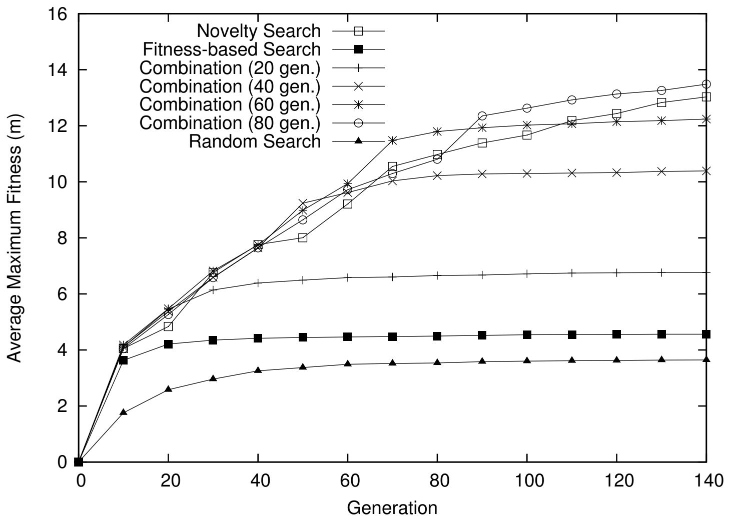
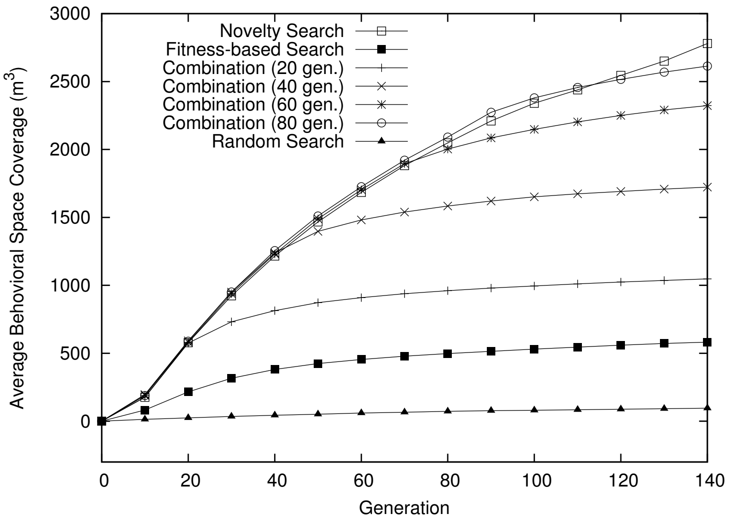

(d) Barrier Avoidance (Fitness)

Figure 4.6: **Final positions of the robot visited in typical runs.** Final positions of all robots discovered in selected typical runs is shown for the swimming experiment (4.6a, 4.6b) and for the barrier avoidance experiment (4.6c, 4.6d). Novelty search in both experiments explores the space of behaviors more evenly, while fitness-based search focuses on optimization of few chosen areas. For swimming experiment (4.6a, 4.6b) a 260m x 260m x 260m part of the behavior space is shown. Final positions are three-dimensional and are shown using a two-dimensional orthogonal projection.



(a) Fitness Value (see equation (4.2))



(b) Behavior Space Coverage

Figure 4.7: **Comparison of behavioral diversity and maximum fitness for different combinations of novelty search and fitness-based search.** Combined algorithm performs fitness-based search for the first fixed number of generations and then switches to using novelty search for the rest of the search. Figure 4.7a shows the average maximum fitness averaged over 25 runs and Figure 4.7b shows the cumulative behavioral diversity per generation averaged over 25 runs.

sistently resulted in higher coverage of behavior space. Combined search methods switching at generation 20, 40, 60 and 80 reached 10.57%, 17.33%, 23.48% and 26.49% of the container, respectively. Standalone novelty search reached the highest behavior space coverage of all methods: 28.87%. All differences in behavior coverage are statistically significant ($p < 0.05$) except the difference between novelty search and combined search switching after 80 generations ($p < 0.06$).

The hypothesis that switching to fitness-based search after the barrier is overcome will improve the performance of the search was not confirmed by the experiments. Experiments with combined search switching at generations 60 and 80 indicate that switching to fitness search may provide a speedup. However, the speedup was only temporary and the difference from novelty-search was not sufficiently significant ($p > 0.1$). Moreover, behavior space analysis shows that after switching to fitness-based search, the exploration rate slowed significantly.

4.4 Discussion

In the swimming experiment the behavior space was only constrained by the limits of the physical simulation (the speed at which a robot can move is limited by the maximum amount of torque it can exert by its joints). Such large space of possible behaviors together with absence of any obvious deceptiveness makes this problem unfavorable to the novelty search. This intuition was confirmed by the results of the swimming experiment, where novelty search was outperformed by the fitness-based search, although it was able to explore more of the behavior space (figures 4.2a and 4.5a).

However, the results of the swimming experiment do not confirm the expectation that in such unconstrained environment the novelty search will perform just as poorly as a random search. On the contrary, novelty search performed almost as well as the fitness-based search and it significantly outperformed random search. The reason is that even though novelty search does not explicitly optimize solutions towards the main objective, it still explores behavior space in a structured way (even when the behavior space is practically unlimited) progressing incrementally from simple to more complex behaviors. This is unlike the random search which often revisits the same solutions repeatedly.

In the barrier avoidance experiment, the deceptiveness of the task lies in the fact that fitness-based search leads the robot directly to the target until it reaches the barrier. At that point, in order to find better solutions, the search needs to explore behaviors that are temporarily further away from target. In the standard objective-based approach, getting further away from the target decreases the fitness value; fitness function thus has a local optimum inside the barrier where fitness-based search is likely to be trapped. This was confirmed by the results of the barrier experiment with fitness-based configuration, where individuals never fully escaped the barrier. Evolution gets trapped in the local optimum because single mutations of robots that reach this local optimum are very unlikely to produce a robot which can (1) continue to be able to successfully swim and at the same time (2) change direction to get around the obstacle. In the same environment, robots discovered by novelty search were able to explore the entire container and were capable of consistently finding the target. The effectiveness of the novelty search in this experiment can be explained by the fact that the space of all possible behaviors was constrained to the size of the container; the boundaries of the container in this case served in a sense as a guide directing the search towards the target.

4.5 Future Works

Experiments presented in this chapter have been selected to test novelty search using two extremes on the spectrum of deceptive tasks—at one extreme we tested novelty search in an unconstrained environment with large space of possible behaviors and fitness function with no obvious deceptiveness. On the other extreme we tested novelty search in a constrained environment with a highly deceptive fitness function. One area of further research is to investigate the effects of different ways to define behavior of an individual on the progress of the search. For example, the space of possible behaviors in the unconstrained swimming experiment could be reduced by simply collapsing large section of the behavior space to the same point, so that novelty search would not treat individuals in that part of the behavior space as novel (a related technique has been tested by Lehman and Stanley in [56] for evolution of neural controllers). For instance, in the unconstrained swimming task all behaviors with negative value of the x coordinate could be replaced with zero vector. Such change would remove half of the behavior space, potentially resulting in novelty search being able to explore the remaining half of the behavior space at a higher rate and achieve higher fitness values more quickly. A similar technique could be used in the barrier avoidance experiment. In this case a physical container was added to the environment to limit the size of the behavior space. It is possible that similar search performance could be achieved by replacing the container with a *virtual* container, created by replacing all behaviors outside of the original container with zero (and therefore discouraging novelty search from exploring area outside of the container, even without a physical container being present). Alternatively, it might be possible to accelerate the search by replacing behaviors in one half of the container with zero, same as in the swimming experiment.

Novelty search is a generic search method that could be applied to many problems in evolutionary robotics. However, as we have shown not all tasks are suitable for novelty search. Further investigation is needed to investigate which problems are deceptive enough to warrant the use of novelty search and what is the most suitable definition of the behavior for each of them (examples of such tasks are locomotion on the ground, flying, following a light source, fleeing from a predator or jumping).

Experiments presented in this chapter focus on behavior measures based on the position of the evolved robot at the end of the simulation. However, behavior could include other properties of robot's phenotype. For example, including properties of robot's morphology (number of blocks, types of joints connecting the blocks) could encourage novelty search to explore different morphologies more aggressively. Other researches have started exploring this idea, e.g. Lehman and Stanley [58], who have shown that defining behaviors as height and weight of the robot increases the diversity of evolved robots, or Cully et al. [14] who used a related technique to create a catalogue of hexapod robots with their morphology altered in different ways (this work has been described in more detail at the end of Section 2.1.2).

Novelty search as presented in this chapter could also be combined with the learning algorithm presented in the previous chapter. The goal of learning in that case would be not to increase the fitness of an individual, but to maximize individual's *novelty*. An open question in this approach is whether the target behavior can be predicted from a short experiment the same way as fitness can be predicted—a property required for efficient combination of both methods.

In Section 4.3.2 we have shown that combining novelty search with fitness-based search by switching from one to the other after a specified number of generations does not provide measurable benefits. Other researchers have also investigated using different methods of combining fitness-based and novelty-based search. Lehman and Stanley used fitness-based competition only among robots with similar morphologies, while novelty was used to encourage diversity [58]. Mouret has shown that Pareto-based multiobjective evolutionary algorithm can be used to optimize both novelty and fitness at the same time [68], using simulated Khepera-like robots with fixed morphology (only the control system was optimized). It would be interesting to know if the same benefits would apply in body-brain co-evolution of virtual creatures.

4.6 Conclusion

In this chapter we have shown that evolution of virtual creatures can greatly benefit from disregarding the objective and the searching for any previously unseen behaviors instead (a method called novelty search). We demonstrated advantages and disadvantages of novelty search in evolution of virtual creatures on two tasks: (1) swimming in an environment without obstacles and (2) barrier avoidance. Results from the swimming experiment have confirmed that novelty search does not provide an advantage for tasks where behavior space is unconstrained and fitness function is not deceptive. For such tasks, novelty search may still explore the space of possible behaviors more effectively than fitness-based search, but due to the large space of possible behaviors it takes longer to optimize solutions towards the objective. However, the barrier avoidance experiment has shown that if the range of possible behaviors is limited (in this case by putting the robot inside a container) and fitness function is deceptive then the novelty search algorithm significantly outperforms fitness-based search. In summary, we show that the benefits of novelty search can be successfully leveraged in the co-evolution of body and brain of robots, provided that the fitness function is deceptive. Results presented in this work demonstrate that directing the search towards behavioral novelty can help in solving deceptive problems in body-brain co-evolution that standard fitness-based search methods often struggle with.

Conclusion

This thesis studies methods of automatic design of both bodies and controllers of simulated robots using evolutionary algorithms. The thesis has successfully fulfilled its goals by making two contributions to the field of evolutionary robotics:

In the first part of the thesis (Chapter 3), a novel nature-inspired algorithm that combines lifetime learning of robot morphology with evolutionary search is presented. The algorithm is motivated by two processes found in nature: (1) many animals use morphological plasticity to increase their chances of survival and reproduction by adapting their bodies to different environments during their lifetime and (2) animal's ability to learn and adapt during its lifetime can accelerate evolution by exploring alternative phenotypes more rapidly than evolution alone which must rely on discovering beneficial changes only through random mutations (changes discovered during animal's lifetime can also, over time, be assimilated into the genotype of an animal—through a mechanism called the Baldwin effect). The algorithm proposed in this thesis reproduces these effects in the domain of body-brain co-evolution of simulated robots.

The proposed algorithm automatically designs robots capable of achieving high performance in different environments by making environment-specific alterations to their bodies. Robots evolved using this method outperform robots which cannot alter their morphology and must therefore use the same generic morphology in all environments. We show that the performance increase is achieved across different types of environments—when varying viscosity of the fluid surrounding a swimming robot, or when varying density of the robot so that it either swims near the sea floor or it is suspended in open water. Significant performance improvements are achieved even when changes to the morphology of the robot are relatively minor (when only sizes of the individual building blocks of the robot are modified)—making it feasible that it will be possible to construct such adapting robots in the near future.

We also demonstrate that morphological plasticity achieved through learning can accelerate evolution even for robots that need to perform well only in a single environment. Morphological plasticity is in this case used purely to make evolutionary search more efficient and it is not required in the final evolved robot. The performance improvement comes not from the robot being able to adjust its morphology differently in different environments, but from the interaction between evolution an learning. We show that learning reduces the computational cost required to evolve a robot with a given minimum performance compared to evolution without learning. The performance gain is based on an insight that shortening the learning phase in each fitness evaluation can significantly reduce the cost of learning, while maintaining the accuracy of learning required to accelerate evolution—as a result the benefits of learning outweigh its computational cost. Morphological plasticity achieved through learning is therefore a generic new tool in evolutionary robotics toolbox which can be used to facilitate more efficient evolution of robot morphology and behavior.

The second part of the thesis (Chapter 4) addresses the problems of fitness function design and premature convergence to local optima in the context of evolutionary robotics. Fitness function lies at the core of an evolutionary algorithm—its purpose is to guide the search towards the objective. However, for many problems objective-based fitness functions can be inherently deceptive. For example, a robot navigating a maze does not always get closer to reaching the target by moving directly towards it (since moving directly towards the target can lead to a dead end in the maze). Similarly, the most direct way for evolution to start optimizing the speed of a swimming robot might be to learn how to perform a single powerful swimming stroke, although a more effective long-term strategy would be to use a series of less powerful periodic swimming strokes. Such evolutionary dead-ends correspond to local optima in the fitness landscape and are a common problem in evolutionary robotics.

This thesis shows that for some problems convergence to local optima in evolutionary robotics can be avoided by replacing the objective-based fitness function with a measure of behavioral novelty of the robot (a technique called Novelty Search introduced by Lehman and Stanley [54]). By searching for any novel behaviors (with the behavior definition supplied by the user) and ignoring the original objective, evolution explores the space of all possible behaviors in a structured way and by doing so can also reach the original objective without the risk of getting trapped in a local optimum of the fitness function. We demonstrated the effectiveness of this method experimentally and we have shown that when the fitness function is deceptive, novelty search successfully avoids the trap present in the fitness function. We also studied the limits of this method on a problem which is not deceptive, and although it improved the objective more slowly, it still discovered larger diversity of behaviors than objective-based search.

With the recent advances of automated manufacturing techniques (such as 3D printing), methods for automated design of both morphology and control of a robot are becoming more and more relevant. This thesis contributes novel insights into the field of evolutionary robotics, forming a basis for future research in this area. In particular, a combination of both techniques presented in this thesis (lifetime learning of robot morphology and novelty search) could be a promising area for new investigations.

Bibliography

[1] Anurag A Agrawal. Phenotypic plasticity in the interactions and evolution of species. *Science*, 294(5541):321–326, 2001.

[2] J. Mark Baldwin. A new factor in evolution. *The American Naturalist*, 30(354):441–451, 1896.

[3] Peter J. Bentley, editor. *Evolutionary Design by Computers.* Morgan Kaufmann, San Mateo, California, 1999.

[4] Josh Bongard. Evolving modular genetic regulatory networks. In *Proceedings of the 2002 Congress on Evolutionary Computation, CEC '02*, volume 2, pages 1872–1877, 2002.

[5] Josh Bongard. Behavior chaining: incremental behavioral integration for evolutionary robotics. In S. Bullock, J. Noble, R. Watson, and M. A. Bedau, editors, *Artificial Life XI: Proceedings of the Eleventh International Conference on the Simulation and Synthesis of Living Systems*, pages 64–71. MIT Press, Cambridge, MA, 2008.

[6] Josh Bongard. Morphological change in machines accelerates the evolution of robust behavior. *Proceedings of the National Academy of Sciences*, 108(4):1234–1239, 2011.

[7] Josh Bongard, Victor Zykov, and Hod Lipson. Resilient machines through continuous self-modeling. *Science*, 314(5802):1118–1121, November 2006.

[8] Josh C. Bongard and Gregory S. Hornby. Guarding against premature convergence while accelerating evolutionary search. In *GECCO '10: Proceedings of the 12th annual conference on Genetic and evolutionary computation*, pages 111–118, New York, NY, USA, 2010. ACM.

[9] Nicholas Cheney, Jeff Clune, and Hod Lipson. Evolved electrophysiological soft robots. In *ALIFE 14: The Fourteenth Conference on the Synthesis and Simulation of Living Systems*, volume 14, pages 222–229, 2014.

[10] Nick Cheney, Josh Bongard, and Hod Lipson. Evolving soft robots in tight spaces. In *Proceedings of the 2015 on Genetic and Evolutionary Computation Conference*, pages 935–942. ACM, 2015.

[11] Nick Cheney, Robert MacCurdy, Jeff Clune, and Hod Lipson. Unshackling evolution: evolving soft robots with multiple materials and a powerful generative encoding. In *Proceedings of the 15th annual conference on Genetic and evolutionary computation*, pages 167–174. ACM, 2013.

[12] Dave Cliff, Phil Husbands, and Inman Harvey. Explorations in evolutionary robotics. *Adapt. Behav.*, 2(1):73–110, 1993.

[13] Giuseppe Cuccu, Faustino Gomez, and Tobias Glasmachers. Novelty-based restarts for evolution strategies. In *Evolutionary Computation (CEC), 2011 IEEE Congress on*, pages 158–163. IEEE, 2011.

[14] Antoine Cully, Jeff Clune, Danesh Tarapore, and Jean-Baptiste Mouret. Robots that can adapt like animals. *Nature*, 521(7553):503–507, 2015.

[15] Daniel C. Dennett. Consciousness explained. *Boston: Little, Brown*, 1991.

[16] Daniel C. Dennett. The Baldwin Effect: A Crane, not a Skyhook. In B. H. Weber and D. J. Depew, editors, *Evolution and learning: The Baldwin Effect Reconsidered*, pages 60–79. MIT Press, Bradford Books, 2003.

[17] John Doucette and Malcolm I. Heywood. Novelty-Based Fitness: An Evaluation under the Santa Fe Trail. In *Genetic Programming, 13th European Conference, EuroGP 2010*, volume 6021 of *Lecture Notes in Computer Science*, pages 50–61. Springer, 2010.

[18] Peter Eggenberger. Evolving morphologies of simulated 3D organisms based on differential gene expression. In *Proceedings of the Fourth European Conference on Artificial Life*, pages 205–213, Cambridge, MA, 1997. MIT Press.

[19] Dario Floreano and Francesco Mondada. Evolution of plastic neurocontrollers for situated agents. In *From Animals to Animats 4, Proceedings of the 4th International Conference on Simulation of Adaptive Behavior (SAB" 1996)*, number LIS-CONF-1996-001, pages 402–410. MA: MIT Press, 1996.

[20] Masahiro Fujita and Hiroaki Kitano. Development of an autonomous quadruped robot for robot entertainment. *Auton. Robots*, 5(1):7–18, 1998.

[21] Pablo Funes and Jordan Pollack. Computer evolution of buildable objects. In P. Husbands and I. Harvey, editors, *Fourth European Conference on Artificial Life*, pages 358–367. MIT Press, 1997.

[22] David E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley Professional, January 1989.

[23] David E. Goldberg and Jon Richardson. Genetic algorithms with sharing for multimodal function optimization. In *Proceedings of the Second International Conference on Genetic Algorithms on Genetic algorithms and their application*, pages 41–49, Mahwah, NJ, USA, 1987. Lawrence Erlbaum Associates, Inc.

[24] Jorge Gomes, Paulo Urbano, and Anders Lyhne Christensen. Evolution of swarm robotics systems with novelty search. *Swarm Intelligence*, 7(2-3):115–144, 2013.

[25] Frederic Gruau. Cellular encoding for interactive evolutionary robotics. In *Fourth European Conference on Artificial Life*, pages 368–377, Cambridge, MA, USA, 1997. MIT Press.

[26] Frederic Gruau, Darrell Whitley, and Larry Pyeatt. A comparison between cellular encoding and direct encoding for genetic neural networks. In John R. Koza, David E. Goldberg, David B. Fogel, and Rick L. Riolo, editors, *Genetic Programming 1996: Proceedings of the First Annual Conference*, pages 81–89, Stanford University, CA, USA, 28–31 1996. MIT Press.

[27] Jonathan Hiller and Hod Lipson. Automatic design and manufacture of soft robots. *Robotics, IEEE Transactions on*, 28(2):457–466, 2012.

[28] Geoffrey E Hinton and Steven J Nowlan. How learning can guide evolution. *Complex systems*, 1(3):495–502, 1987.

[29] John H. Holland. *Adaptation in natural artificial systems.* University of Michigan Press, Ann Arbor, 1975.

[30] G. S. Hornby, M. Fujita, and S. Takamura. Autonomous evolution of gaits with the sony quadruped robot. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1297–1304. Morgan Kaufmann, 1999.

[31] G. S. Hornby, H. Lipson, and J. B. Pollack. Generative representations for the automated design of modular physical robots. *IEEE Transactions on Robotics and Automation*, 19(4):703–719, 2003.

[32] Gregory S. Hornby. Alps: the age-layered population structure for reducing the problem of premature convergence. In *GECCO '06: Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pages 815–822, New York, NY, USA, 2006. ACM.

[33] Gregory S Hornby, Jordan B Pollack, et al. Body-brain co-evolution using L-systems as a generative encoding. In Lee Spector et al., editor, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, pages 868–875, San Francisco, USA, 2001. Morgan Kaufmann Publishers.

[34] Jianjun Hu, Erik Goodman, Kisung Seo, Zhun Fan, and Rondal Rosenberg. The hierarchical fair competition (hfc) framework for sustainable evolutionary algorithms. volume 13, pages 241–277, Cambridge, MA, USA, 2005. MIT Press.

[35] Marcus Hutter and Shane Legg. Fitness uniform optimization. *Evolutionary Computation, IEEE Transactions on*, 10(5):568–589, 2006.

[36] Joshua D. Knowles, Richard A. Watson, and David W. Corne. *Evolutionary Multi-Criterion Optimization: First International Conference, EMO 2001 Zurich, Switzerland, March 7–9, 2001 Proceedings*, volume 1993 of *Lecture Notes in Computer Science*, chapter Reducing Local Optima in Single-Objective Problems by Multi-objectivization, pages 269–283. Springer Berlin Heidelberg, Berlin, Heidelberg, 2001.

[37] Maciej Komosinski and Szymon Ulatowski. Framsticks: Towards a simulation of a nature-like world, creatures and evolution. In Jean-Daniel Nicoud Dario Floreano and Francesco Mondada, editors, *Proceedings of 5th European Conference on Artificial Life (ECAL99)*, volume 1674 of *Lecture Notes in Computer Science*, pages 261–265. Springer-Verlag, 1999.

[38] John R. Koza, Martin A. Keane, Matthew J. Streeter, William Mydlowec, Jessen Yu, and Guido Lanza. *Genetic Programming IV: Routine Human-Competitive Machine Intelligence*. Kluwer Academic Publishers, 2003.

[39] Marcel Krčah. Evolution of springy organisms. Bachelor's Thesis, Department of Software and Computer Science Education, Faculty of Mathematics and Physics, Charles University, Prague, Czech Republic, 2008.

[40] Peter Krčah. Evolutionary development of robotic organisms. Master's thesis, Department of Software and Computer Science Education, Faculty of Mathematics and Physics, Charles University, Prague, Czech Republic, 2007.

[41] Peter Krčah. Evolving virtual creatures revisited. In *GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, page 341, New York, NY, USA, 2007. ACM.

[42] Peter Krčah. Towards efficient evolution of morphology and control. In *GECCO '08 Proceedings of the 10th annual conference on Genetic and evolutionary computation*, pages 287–288, New York, NY, USA, 2008. ACM.

[43] Peter Krčah. Towards efficient evolutionary design of autonomous robots. In Gregory S. Hornby, Lukas Sekanina, and Pauline C. Haddow, editors, *ICES '08: Proceedings of the 8th international conference on Evolvable Systems: From Biology to Hardware*, volume 5216 of *Lecture Notes in Computer Science*, pages 153–164, Berlin, Heidelberg, 2008. Springer-Verlag.

[44] Peter Krčah. Evolučný návrh robotických organizmov. In V. Kvasnička, J. Pospíchal, Š. Kozák, P. Návrat, and P. Paroulek, editors, *Umelá inteligencia a kognitívna veda I*, pages 195–229. STU Press, Bratislava, Slovak Republic, 2009.

[45] Peter Krčah. Solving deceptive tasks in robot body-brain co-evolution by searching for behavioral novelty. In *10th International Conference on Intelligent Systems Design and Applications (ISDA)*, pages 284–289. IEEE, 2010.

[46] Peter Krčah. Effects of speciation on evolution of neural networks in highly dynamic environments. In Youssef Hamadi and Marc Schoenauer, editors, *Learning and Intelligent Optimization*, volume 7219 of *Lecture Notes in Computer Science*, pages 425–430. Springer, 2012.

[47] Peter Krčah. Solving deceptive tasks in robot body-brain co-evolution by searching for behavioral novelty. In Tauseef Gulrez and Aboul Ella Hassanien, editors, *Advances in robotics and virtual reality*, volume 26 of *Intelligent Systems Reference Library*, pages 167–186. Springer, 2012.

[48] Peter Krčah. Virtual creatures evolved using speciation and historical markings, 2014. Runner-up in Virtual Creatures Contest, GECCO 2014: conference on Genetic and evolutionary computation `http://www.cs.utexas.edu/~joel/virtual_creatures_contest/` [accessed online on May 1, 2016].

[49] Peter Krčah. Adaptation of virtual creatures to different environments through morphological plasticity. In *SAB 2016: The 14th International Conference on the Simulation of Adaptive Behavior*, Lecture Notes in Computer Science (Subseries: Lecture Notes in Artificial Intelligence). Springer, 2016. To appear.

[50] Peter Krčah and Daniel Toropila. Combination of novelty search and fitness-based search applied to robot body-brain co-evolution. In *Proceedings of the 13th Czech-Japan Seminar on Data Analysis and Decision Making in Service Science, Otaru, Japan*, pages 1–6, 2010.

[51] Peter Krčah and Daniel Toropila. Riešenie zavádzajúcich úloh v koevolúcii ovládania a morfológie robotov pomocou hľadania noviniek v správaní. In Jiří Jelínek and Radim Jiroušek, editors, *Znalosti 2011*. VŠB-Technická univerzita Ostrava, Fakulta elektrotechniky a informatiky, 2011.

[52] Nicolas Lassabe, Hervé Luga, and Yves Duthen. A new step for artificial creatures. In *2007 IEEE Symposium on Artificial Life*, pages 243–251. IEEE, April 2007.

[53] Joel Lehman. *Evolution through the Search for Novelty*. PhD thesis, Department of Electrical Engineering and Computer Science, College of Engineering and Computer Science at the University of Central Florida, Orlando, Florida, 2012.

[54] Joel Lehman and Kenneth O. Stanley. Exploiting open endedness to solve problems through the search for novelty. In *In Proceedings of the Eleventh International Conference on Artificial Life*, Cambridge, MA, 2008. MIT Press.

[55] Joel Lehman and Kenneth O. Stanley. Efficiently evolving programs through the search for novelty. In *GECCO '10: Proceedings of the 12th annual conference on Genetic and evolutionary computation*, pages 837–844, New York, NY, USA, 2010. ACM.

[56] Joel Lehman and Kenneth O. Stanley. Revising the evolutionary computation abstraction: minimal criteria novelty search. In *GECCO '10: Proceedings of the 12th annual conference on Genetic and evolutionary computation*, pages 103–110, New York, NY, USA, 2010. ACM.

[57] Joel Lehman and Kenneth O Stanley. Abandoning objectives: Evolution through the search for novelty alone. *Evolutionary computation*, 19(2):189–223, 2011.

[58] Joel Lehman and Kenneth O Stanley. Evolving a diversity of virtual creatures through novelty search and local competition. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, pages 211–218. ACM, 2011.

[59] Dan Lessin, Don Fussell, and Risto Miikkulainen. Adapting morphology to multiple tasks in evolved virtual creatures. In Hiroki Sayama, John Rieffel, Sebastian Risi, René Doursat, and Hod Lipson, editors, *Proceedings of The Fourteenth International Conference on the Synthesis and Simulation of Living Systems (ALIFE 14) 2014*, pages 247–254. MIT Press, 2014.

[60] Dan Lessin and Sebastian Risi. Soft-body muscles for evolved virtual creatures: The next step on a bio-mimetic path to meaningful morphological complexity. In Paul Andrews, Leo Caves, Rene Doursat, Simon Hickinbotham, Fiona Polack, Susan Stepney, Tim Taylor, and Jon Timmis, editors, *Proceedings of the European Conference on Artificial Life 2015*, pages 604–611. MIT Press, 2015.

[61] Aristid Lindenmayer. Mathematical models for cellular interactions in development: Parts i and ii. *Journal of Theoretical Biology*, 18(3):300–315, March 1968.

[62] Hod Lipson and Jordan B. Pollack. Automatic design and manufacture of robotic lifeforms. *Nature*, 406(6799):974–978, August 2000.

[63] Asger Lisborg, Dan Lessin, and Sebastian Risi. Evolving flying evcs with novelty search, 2015. Virtual Creatures Contest, GECCO 2015 conference on Genetic and evolutionary computation `http://www.sigevo.org/gecco-2015/competitions.html` [accessed online on May 1, 2016].

[64] Yuliana Martinez, Enrique Naredo, Leonardo Trujillo, and Edgar Galván-López. Searching for novel regression functions. In *Evolutionary Computation (CEC), 2013 IEEE Congress on*, pages 16–23. IEEE, 2013.

[65] Vittorio Megaro, Bernhard Thomaszewski, Maurizio Nitti, Otmar Hilliges, Markus Gross, and Stelian Coros. Interactive design of 3d-printable robotic creatures. *ACM Transactions on Graphics (TOG)*, 34(6):216, 2015.

[66] Thomas Miconi. In silicon no one can hear you scream: Evolving fighting creatures. In Michael O'Neill, Leonardo Vanneschi, Steven Gustafson, Anna Esparcia-Alcázar, Ivanoe De Falco, Antonio Della Cioppa, and Ernesto Tarantino, editors, *EuroGP*, volume 4971 of *Lecture Notes in Computer Science*, pages 25–36. Springer, 2008.

[67] Thomas Miconi and Alastair Channon. An improved system for artificial creatures evolution. In Luis Mateus Rocha, Larry S. Yaeger, Mark A. Bedau, Dario Floreano, Robert L. Goldstone, and Alessandro Vespignani, editors, *Artificial Life X: Proceedings of the Tenth International Conference on the Simulation and Synthesis of Living Systems*, pages 255–261, Bloomington, IN, USA, 3-7 June 2006. MIT Press.

[68] Jean-Baptiste Mouret. Novelty-based multiobjectivization. In Stéphane Doncieux, Nicolas Bredèche, and Jean-Baptiste Mouret, editors, *New horizons in evolutionary robotics*, volume 341 of *Studies in Computational Intelligence*, pages 139–154. Springer, Berlin, Heidelberg, 2011.

[69] Jean-Baptiste Mouret and Stéphane Doncieux. Overcoming the bootstrap problem in evolutionary robotics using behavioral diversity. In *CEC'09: Proceedings of the Eleventh conference on Congress on Evolutionary Computation*, pages 1161–1168, Piscataway, NJ, USA, 2009. IEEE Press.

[70] Satoshi Murata, Eiichi Yoshida, Akiya Kamimura, Haruhisa Kurokawa, Kohji Tomita, and Shigeru Kokaji. M-tran: Self-reconfigurable modular robotic system. *Mechatronics, IEEE/ASME Transactions on*, 7(4):431–441, 2002.

[71] Enrique Naredo and Leonardo Trujillo. Searching for novel clustering programs. In *Proceedings of the 15th annual conference on Genetic and evolutionary computation*, pages 1093–1100. ACM, 2013.

[72] Stefano Nolfi and Dario Floreano. Learning and evolution. *Autonomous robots*, 7(1):89–113, 1999.

[73] Stefano Nolfi and Dario Floreano. *Evolutionary Robotics. The Biology, Intelligence, and Technology of Self-organizing Machines.* MIT Press, Cambridge, MA, 2001. 2001 (2nd print), 2000 (1st print).

[74] Stefano Nolfi, Domenico Parisi, and Jeffrey L Elman. Learning and evolution in neural networks. *Adaptive Behavior*, 3(1):5–28, 1994.

[75] Stefano Nolfi and Jun Tani. Extracting regularities in space and time through a cascade of prediction networks: The case of a mobile robot navigating in a structured environment. *Connection Science*, 11(2):125–148, 1999.

[76] George F Oster and Edward O Wilson. *Caste and ecology in the social insects.* Princeton University Press, 1978.

[77] Marcin L. Pilat and Christian Jacob. Evolution of vision capabilities in embodied virtual creatures. In *GECCO '10: Proceedings of the 12th annual conference on Genetic and evolutionary computation*, pages 95–102, New York, NY, USA, 2010. ACM.

[78] Ingo Rechenberg. *Evolutionsstrategie: optimierung technischer systeme nach prinzipien der biologischen evolution.* Frommann-Holzboog, 1973.

[79] Joseph Reisinger and Risto Miikkulainen. Acquiring evolvability through adaptive representations. In *GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 1045–1052, New York, NY, USA, 2007. ACM.

[80] Sebastian Risi, Sandy D. Vanderbleek, Charles E. Hughes, and Kenneth O. Stanley. How novelty search escapes the deceptive trap of learning to learn.

In *GECCO '09: Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pages 153–160, New York, NY, USA, 2009. ACM.

[81] Mark Roper, Nikolaos Katsaros, and Chrisantha Fernando. Voxel robot: A pneumatic robot with deformable morphology. In Angel P. Del Pobil, Eris Chinellato, Ester Martínez-Martín, John Hallam, Enric Cervera, and Antonio Morales, editors, *From Animals to Animats 13 - 13th International Conference on Simulation of Adaptive Behavior, SAB 2014*, volume 8575 of *Lecture Notes in Computer Science*, pages 230–239. Springer, 2014.

[82] Wei-Min Shen, Maks Krivokon, Harris Chiu, Jacob Everist, Michael Rubenstein, and Jagadesh Venkatesh. Multimode locomotion via superbot reconfigurable robots. *Autonomous Robots*, 20(2):165–177, 2006.

[83] Karl Sims. Evolving 3D Morphology and Behavior by Competition. *Artificial Life*, 1(4):353–372, 1994.

[84] Karl Sims. Evolving virtual creatures. In *SIGGRAPH '94: Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 15–22, New York, NY, USA, 1994. ACM Press.

[85] John Maynard Smith. Natural selection: When learning guides evolution. *Nature*, 329:761–762, 1987. reprint in: Adaptive Individuals in Evolving Populations: Models and Algorithms, R. K. Belew and M. Mitchell (eds.), 1996, pp. 455–457, Reading, MA: Addison Wesley.

[86] Russell Smith. ODE: Open Dynamics Engine, 2005. http://www.ode.org [accessed online on May 1, 2016].

[87] Lee Spector. *Automatic Quantum Computer Programming: A Genetic Programming Approach*, volume 7 of *Genetic Programming*. Springer-Verlag, New York, USA, 2004.

[88] Lee Spector, David M. Clark, Ian Lindsay, Bradford Barr, and Jon Klein. Genetic programming for finite algebras. In *GECCO '08: Proceedings of the 10th annual conference on Genetic and evolutionary computation*, pages 1291–1298, New York, NY, USA, 2008. ACM.

[89] Kenneth O. Stanley. *Efficient Evolution of Neural Networks Through Complexification*. PhD thesis, Department of Computer Sciences, The University of Texas at Austin, August 2004. Technical Report AI-TR-04-314.

[90] Kenneth O Stanley and Joel Lehman. *Why Greatness Cannot Be Planned - The Myth of the Objective*. Springer, 2015.

[91] Kenneth O. Stanley and Risto Miikkulainen. Efficient reinforcement learning through evolving neural network topologies. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2002)*, San Francisco, CA, 2002. Morgan Kaufmann.

[92] Kenneth O Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary computation*, 10(2):99–127, 2002.

[93] Kenneth O. Stanley and Risto Miikkulainen. A taxonomy for artificial embryogeny. *Artificial Life*, 9(2):93–130, 2003.

[94] Gregory Studer and Hod Lipson. Spontaneous emergence of self-replicating structures in molecube automata. In Luis Mateus Rocha, Larry S. Yaeger, Mark A. Bedau, Dario Floreano, Robert L. Goldstone, and Alessandro Vespignani, editors, *Artificial Life X: Proceedings of the Tenth International Conference on the Simulation and Synthesis of Living Systems*, pages 227–233, Bloomington, IN, USA, 3-7 June 2006. MIT Press.

[95] Leonardo Trujillo, Enrique Naredo, and Yuliana Martínez. Preliminary study of bloat in genetic programming with behavior-based search. In *EVOLVE - A Bridge between Probability, Set Oriented Numerics, and Evolutionary Computation IV*, volume 227 of *Advances in Intelligent Systems and Computing*, pages 293–305. Springer, Berlin, Heidelberg, 2013.

[96] Conrad H Waddington. Canalization of development and the inheritance of acquired characters. *Nature*, 150(3811):563–565, 1942.

[97] Conrad H Waddington. Genetic assimilation of an acquired character. *Evolution*, 7:118–126, 1953.

[98] Hongxing Wei, Youdong Chen, Jindong Tan, and Tianmiao Wang. Sambot: A self-assembly modular robot system. *Mechatronics, IEEE/ASME Transactions on*, 16(4):745–757, 2011.

[99] Douglas W Whitman and Anurag A Agrawal. What is phenotypic plasticity and why is it important? In Douglas W. Whitman and T. N. Ananthakrishnan, editors, *Phenotypic Plasticity of Insects: Mechanisms and Consequences*, volume 10, pages 1–63. Science Publishers: Enfield, NH, USA, 2009.

[100] Victor Zykov, Efstathios Mytilinaios, Bryant Adams, and Hod Lipson. Self-reproducing machines. *Nature*, 435(7039):163–164, 2005.

[101] Victor Zykov, P William, Nicolas Lassabe, and Hod Lipson. Molecubes extended: Diversifying capabilities of open-source modular robotics. In *IROS Workshop on Self-Reconfigurable Robots,Systems and Applications*, 2008.

List of Tables

List of Figures

Appendix A

Mutation and Generation of Creatures

This appendix describes algorithms and parameters used for generating random virtual creatures and for performing mutation of virtual creatures. First section presents algorithms operating on the morphology level. The following section then presents algorithms operating on neural networks within each morphological node.

A.1 Morphology

Algorithms used for random generation and mutation of morphology of virtual creatures are provided in Algorithm A.1.1 (random generation) and Algorithm A.1.2 (mutation). Algorithm parameters are provided in Table A.1 (random generation) and Table A.2 (mutation).

Algorithm A.1.1 Random generation of a creature morphology.

```
1: generate root node of the morphological graph
2: generate the brain of the creature (i.e. the global controller)
3: repeat
4:     create new random node (use neural network generator described in Algorithm A.2.1 to generate neural networks)
5:     create new random connection pointing from randomly chosen existing node to the new node
6: until given number of nodes has been generated
7: while graph contains less connections than the given number of connections do
8:     create new random connection between two randomly chosen nodes
9: end while
```

Parameter name	**Description**	**Default**
Node count	Number of newly created morphological nodes.	3
Connection count	Number of newly created morphological connections.	3
Terminal flag prob.	Probability that newly created connection will be terminal.	0.2
Reflections per connection	Desired average number of enabled reflection flags per connection.	0.1

Table A.1: **Parameters of the algorithm for generation of random virtual creatures (Algorithm A.1.1).**

Algorithm A.1.2 Mutation of a creature morphology.

```
1: add a new random node with given probability and connect it immediately to the network either using one incoming randomly generated connection, or using one incoming and one outgoing randomly generated connection
2: add a new random connection with given probability, connecting two randomly chosen nodes
3: count the number of all parameters $n_p$ of the creature (in all nodes and connections), which affect the structure of a phenotype graph (such as terminal flag, reflection flag or recursive limit)
4: mutate each parameter from step 3 with probability of $k_p/n_p$, where $k_p$ is parameter provided by the user, which determines how many parameters affecting the structure of a phenotype graph should be mutated per run
5: repeat steps 3 and 4 for all non-structural morphological parameters using value $k_m$ (number of morphological parameters to mutate per run) provided by the user
6: repeat steps 3 and 4 for all controllers using value $k_c$ (number of controllers to mutate per run) provided by the user
```

Parameter name	**Description**	**Default**
Value replacement prob.	Probability that mutated value will be replaced by new randomly picked value. This probability applies to all mutated values.	0.1
Value perturb. amount (x)	Each mutated value will be perturbed by the random value with normal distribution with standard deviation of $x/2$.	0.25
Add node prob.	Probability of adding a new node to the neural network.	0.05
Add connection prob.	Probability of adding a new connection to the neural network.	0.02
Morphology mut. amount.	Specifies an average number of morphology parameters to mutate per run.	1
Phenotype graph structure mut. amount.	Specifies an average number of parameters affecting the structure of a phenotype graph (such as terminal flag, reflection flag or recursive limit) to mutate per run.	5
Controllers mut. amount.	Specifies an average number of controllers to mutate per run.	10

Table A.2: **Parameters of the algorithm for mutating virtual creatures (Algorithm A.1.2).**

Parameter name	**Description**	**Default**
Neuron count	Number of neurons created. This may differ from the number of neurons after garbage collection.	0
Saturation %	Number of connections to create, expressed as a percentage of the number of all neuron inputs. Value of 200%, for example, results on average in two incoming connections per neuron input.	60%
Garbage collection	If enabled, all nodes which, are unreachable from inputs of the neural network, will be deleted.	yes
Transfer functions	A set of transfer functions to use in the generated neurons.	{sigmoid}

Table A.3: **Parameters of the algorithm for generation random neural networks (Algorithm A.2.1).**

A.2 Controllers

Algorithms used for random generation and mutation of controllers inside each body part of a creature are provided in Algorithm A.2.1 (random generation) and Algorithm A.2.2 (mutation). Both algorithms are controlled by a set of parameters provided in Table A.3 (for random generation) and Table A.4 (for mutation).

Algorithm A.2.1 Random generation of a neural network controller.

```
1: generate specified number of neurons with the transfer function chosen randomly from a specified list of transfer functions
2: for all inputs of all generated neurons do
3:     choose random number of new connections for this input according to the saturation parameter (described in Table A.3)
4:     for all new connections do
5:         use current node as a target node for the new connection
6:         pick a random source node
7:         if recurrent connections are not permitted, repeat previous step until a non-recurrent connection is found
8:         create new connection with random weight between the source and the target node
9:     end for
10: end for
11: perform garbage collection on the graph, if requested
```

Parameter name	**Description**	**Default**
Add node prob.	Probability of adding a new node to the neural network.	0.03
Add connection prob.	Probability of adding a new connection to the neural network.	0.1
Trials for new connection	Number of attempts to create a new non-recurrent connection.	5
TF change prob.	Probability, for each node, of picking a new transfer function.	0.1
Disabled flag mutation prob.	Probability, for each node input, of enabling/disabling a random connection.	0.1
Weight mutation prob.	Probability, for each connection, that a weight of a connection will be mutated (either by perturbation or by replacement with a new random value).	0.3
Weight replacement prob.	Probability, for each mutated weight, that its value will be replaced by a new random value (otherwise, it will be perturbed).	0.1
Weight perturb. amount (x)	If a weight is perturbed, then it is changed by a random value with the normal distribution with standard deviation $x/2$.	0.25
Bias mutation prob.	Probability, for each node with sigmoidal transfer function, that its bias will be mutated.	0.3
Bias mutation amount (y)	Each mutated bias will be perturbed by the random value with normal distribution with standard deviation of $y/2$.	0.1

Table A.4: **Parameters of the algorithm for mutating neural networks (Algorithm A.2.2).**

Algorithm A.2.2 Mutation of a neural network controller.

```
1: for all neurons do
2:     pick a new transfer function for each node with given probability
3:     if this neuron has sigmoidal transfer function, mutate its bias with given probability
4: end for
5: for all connections do
6:     mutate connection weights of each connection with given probability
7: end for
8: for all inputs of all neurons do
9:     choose a random incoming connection
10:    if chosen connection is not the last enabled incoming or last enabled outgoing connection then
11:        invert disabled flag of chosen connection
12:    end if
13: end for
14: with given probability, add a new random node by splitting an existing connection and adding two new connections in its place (old connection is disabled)
15: add random new connection with given probability
```

Appendix B

Supplementary Material

The following supplementary material is provided on the CD attached to the thesis:

1. Electronic version of the thesis
2. Software used for experiments in this thesis (ERO framework) including the source code
3. Configuration files for experiments used in this thesis
4. Videos and XML-encoded genotypes of selected robots evolved using algorithms presented in this thesis

For more information please refer to the README.pdf file on the CD.

Videos of evolved robots are also available online at the following URLs:

Usage	**Description**	**Online Video**
Figure 3.9 (page 61)	Snake-like robot adapted differently for swimming in open water and near the sea floor.	`https://youtu.be/VbP8CN53nWk`
Figure 3.2a (page 48)	Crab-like robot adapted differently for swimming in open water and near the sea floor.	`https://youtu.be/GX1BNduCvmo`
Figure 3.2b (page 48)	Lionfish-like robot adapted differently for swimming in fluids with different viscosities.	`https://youtu.be/hFzeowtPmmk`
Figure 3.15 (page 68)	Robots evolved for swimming using morphological plasticity.	`https://youtu.be/A9gj7taU0SM`
Section 4.3 (page 76)	Robots evolved for swimming using novelty search and fitness-based search.	`https://youtu.be/0xvabl1DRyw`
Figure 4.3 (page 79)	Robots evolved for barrier avoidance using novelty search.	`https://youtu.be/I9ggqjVp6Po`
Figure 4.4 (page 80)	Robots evolved for barrier avoidance using fitness-based search.	`https://youtu.be/Uc5wDbHz_-I`