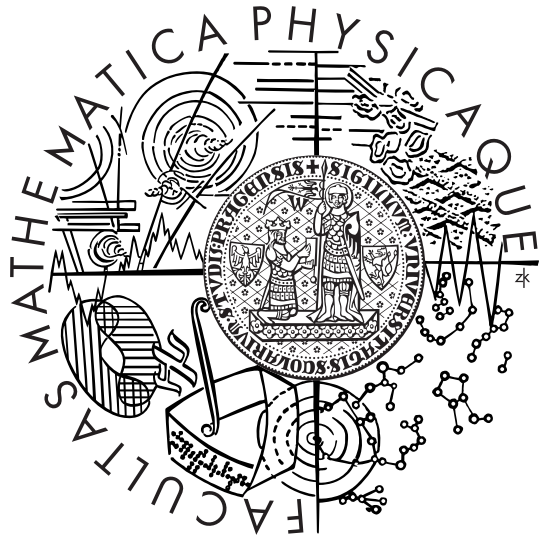
Charles University in Prague

Faculty of Mathematics and Physics

MASTER THESIS



Peter Krčah

Evolučný Vývoj Robotických Organizmov

Evolutionary Development of Robotic Organisms

Institution: Department of Software and Computer Science Education

Supervisor: RNDr. František Mráz, CSc.

Study branch: Artificial intelligence

I would like to express my sincere gratitude to my supervisor RNDr. František Mráz, CSc., who patiently guided this study and his advice significantly improved quality of this thesis.

Prohlašuji, že jsem svou diplomovou práci napsal samostatně a výhradně s použitím citovaných pramenů. Souhlasím se zapůjčováním práce.

V Praze dne 10.8.2007 Peter Krčah

Contents

Abstract

Název práce: Evolučný vývoj robotických organizmov
Autor: Peter Krčah
Obor: Softwarové systémy
Vedoucí diplomové práce: RNDr. František Mráz, CSc.
E-mail vedoucího: Frantisek.Mraz@mff.cuni.cz
Abstrakt: Od zveřejnění práce Karla Simse o evoluci virtuálních organizmů uplynulo již třináct let. Od té doby bylo publikováno několik přístupů k řešení genetických algoritmů a k evoluci neuronových sítí. Tato práce navrhuje nový algoritmus pro evoluci virtuálních organizmů. Představovaný algoritmus se nazývá Hierarchický NEAT a je rozšířením algoritmu NEAT (NeuroEvolution of Augmenting Topologies), který je schopen efektivně vyvíjet umělé neuronové sítě. Hierarchický NEAT aplikuje všechny tři hlavní složky algoritmu NEAT (ochranu evolučních inovací zavedením druhů, ohleduplné křížení organizmů a postupný vývin z minimální stavby těla) na evoluci stavby těla a kontrolního systému organizmů. Navíc algoritmus umožňuje citlivější křížení kontrolních systémů než původní metody. Experimenty prokázaly, že navrhovaný algoritmus výrazně zvyšuje výkon evoluce pro všechny testované úkoly. Další experimenty prokázaly, že každá ze složek algoritmu je přínosem pro výkon evoluce, centrální řídící mechanizmus není nezbytný pro úspěšný vývoj strategií pro sledování světla a že výběr konkrétní sady přenosových funkcí neuronu významněji neovlivňuje evoluci.
Klíčová slova: Evoluce virtuálních organizmů, Evoluce neuronových sítí, Umělý život

Title: Evolutionary development of robotic organisms
Author: Peter Krčah
Branch of study: Software Systems
Supervisor: RNDr. František Mráz, CSc.
Supervisor's email address: Frantisek.Mraz@mff.cuni.cz
Abstract: Thirteen years have passed since Karl Sims published his work on evolving virtual creatures. Since then, several approaches to neural network evolution and genetic algorithms have been introduced. This thesis proposes a novel algorithm for the evolution of virtual creatures. The algorithm – Hierarchical NEAT – is inspired by NeuroEvolution of Augmenting Topologies (NEAT) algorithm which efficiently evolves artificial neural networks. Hierarchical NEAT applies all three main components of NEAT algorithm (protecting evolutionary innovation through speciation, sensible mating of the creatures and incremental growth from minimal structure) to the evolution of morphology and control system of the virtual creatures. Furthermore, the algorithm also allows sensible mating of control systems of the creatures, as opposed to original mating methods. Experiments have shown that the proposed algorithm significantly increases the performance of the evolution on all tested tasks. Several supplementary experiments have also been conducted to confirm that each component of the algorithm is beneficent for the evolution, that central coordination in not necessary for successful evolution of light-following strategies and that the choice of neuron transfer functions does not have significant impact on the evolution of the creatures.
Keywords: Evolution of virtual creatures, Evolution of neural networks, Artificial life

Chapter 1

Introduction

Many problems seem to require human creativity to be successfully solved. Finding solutions often involves searching large multidimensional spaces of all possible solutions, which, luckily, humans can often do intuitively. However, the process of invention can be challenging, time-consuming and tiring. Therefore, several methods for automation of such processes have been proposed in the last few decades. One such approach is inspired by Darwinian theory of evolution. Several effective evolutionary search methods have been proposed (survey of methods can be found in [9, 19]) and have even led to automatic invention of several human-competitive results recently [20].

Artificial evolutionary search methods are based on genetic algorithms, which in many ways imitate the process of evolution in the nature. Possible solutions of the problem are compactly coded into a *genotype*, analogous to the genetic description of animals. The genotype is then translated to a *phenotype* (analogous to the physical body of an animal), which competes for survival in virtual arenas. Fitness function is defined to evaluate each phenotype according to its success in solving the problem. The fittest organisms are then allowed to reproduce, with the number of offspring proportional to their fitness value. Random mutations are introduced in the newly created offspring to maintain genetic variability.

The field of evolutionary computation is expanding very fast and evolutionary search methods have already been successfully applied to variety of problems (including many real-world optimization problems).

This thesis focuses on evolutionary search methods for automated design of robotic organisms, where both morphology and control system of an organism are automatically invented and optimized during the course of evolution. Experiments with virtual creatures are conducted using ERO (Evolution of Robotic Organisms) – an evolutionary framework developed as a student software project at Faculty of Mathematics and Physics, Charles University in Prague. Virtual creatures in ERO are evolved towards ability to perform a given task (i.e. swimming, walking, jumping or following). Representation of the creatures is inspired by the work of Karl Sims [31, 30].

ERO has been successfully used to evolve simple locomotion strategies for 3D virtual creatures. The work of Karl Sims was, however, published in 1994 and since then, much research has been done in the field of artificial evolution (Section 2.1 provides a review of methods). Especially, several novel algorithms for evolution of neural networks have been proposed.

One such algorithm is NEAT (NeuroEvolution of Augmenting Topologies) algorithm for neural network evolution developed by Kenneth O. Stanley [36]. NEAT simultaneously evolves both the neural connection weights and the topology of the neural network. NEAT is able to efficiently evolve topology by introducing three main novel ideas. First, it proposes a method of sensible crossover of networks with different topologies. Second, it introduces a method of "protecting innovation through speciation". 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 networks in the entire population. Finally, NEAT starts the evolution of neural networks from a minimal, simplistic topology and complexity of the network increases later, as the evolution proceeds. NEAT (along with its many variants) has been successfully applied in many domains and is a state-of-the-art algorithm for several problems (see Section 2.1 for review of methods of artificial neural network evolution).

Besides being very effective in evolving neural networks, all three main ideas of NEAT algorithm are also very general. They are not bound to the evolution of neural network, but can also be used to efficiently evolve topology of other structures as well. This thesis proposes a method of applying all three main ideas of NEAT algorithm (sensible crossover, speciation and minimal start) to the evolution of both morphology and control of the virtual creatures. Results of massive experiments (taking several months of CPU time to complete), conducted to measure the impact of NEAT on the original algorithm of evolving creatures are also provided.

Application of NEAT to morphology of the creatures also provides (as a side-effect) the possibility of several very promising algorithm improvements. For example, in order to sensibly recombine properties of individual structure elements in the organism, NEAT finds a correspondence between structure elements of both parents. When applied to virtual creatures, NEAT provides a correspondence of individual body parts of the parents. This correspondence then allows recombination of local neural network controllers present in each body part of each parent. Recombination of local neural networks is not possible with original creature mating methods proposed by Sims (i.e. grafting and crossover), because these methods work exclusively on the morphology level.

Neural networks of the creatures (as proposed by Sims) also differ from classical neural networks by having a large set of possible complex transfer functions, as opposed to a single sigmoidal transfer function used in classical neural networks [1]. It is not clear whether a large set of transfer functions helps the evolution, or retards the search by increasing the size of the search space. This thesis provides several experiments measuring the effect of larger set of transfer function on the ability of networks to solve given sample problem (XOR and function approximation problems are used).

[1] Classical neural networks are sometimes also composed of different types of transfer functions (e.g. linear transfer functions). However, control system of the creatures uses 23 different complex transfer functions. Such large amount of complex transfer functions is not used in any of the classical neural networks (to the author's knowledge).

Performance of algorithm with and without NEAT is also measured on these sample problems.

In summary, this thesis has two primary goals:

- To increase efficiency of the evolution of virtual creatures by proposing a novel method of evolving both morphology and control system of the creatures using methods adopted from NEAT algorithm and to experimentally investigate its impact on the process of evolution of the creatures.
- To apply NEAT algorithm to neural networks with a large set of possible transfer functions and compare its performance to NEAT algorithm applied on classical neural networks (with a sigmoidal transfer function).

Preliminary results achieved during the development of this thesis have been presented at Genetic And Evolutionary Computation Conference 2007 [21].

1.1 Structure of the thesis

The thesis is divided into three main parts. Chapter 2 provides reader with a background information on evolving virtual creatures. It starts with a section summarizing existing approaches to the evolution of virtual creatures and artificial neural networks. The chapter continues with a description of the key concepts in the evolution of virtual creatures (including detailed description of the genotype and the phenotype, genetic operators and the process of fitness evaluation) and ends with a brief description of ERO framework, which is used for all experiments in this thesis. Chapter 3 provides a detailed description of the application of NEAT algorithm to the morphology and control systems of virtual creatures. Motivation and summary of expected results are also provided. Chapter 4 summarizes results of large-scale experiments conducted to compare the performance of the proposed algorithm to the performance of the standard genetic algorithm. Results of several ablation experiments are also provided to show the importance of each part of the proposed algorithm. Results of the experiments show, that Hierarchical NEAT significantly outperforms standard genetic algorithm in all tested cases. Chapter 5 proposes several directions for future research and describes several methods, which could be used to further improve the performance of the algorithm.

Chapter 2

Background

This chapter provides an introduction to the area of evolving virtual creatures. Chapter is divided into three sections. First section summarizes existing methods of artificial neural network evolution and evolution of virtual creatures (along with several successful attempts of transferring virtual creatures to the real-world). Second chapter describes basic concepts of evolving virtual creatures (representation, genetic operators, etc.). Third section briefly describes ERO – a framework for evolutionary experiments, which is used for all evolutionary experiments in this thesis.

2.1 Review of existing methods

Autonomous robots have always attracted attention of researchers around the world. The main challenge of traditional autonomous robotics is to design a software for the given physical robot in such way, that the robot equipped with this software is able to autonomously perform given tasks. Methods of robotic control range from offline (state of the world is known in advance, thus the strategy can be created in advance) to online (state of the world changes and the robot needs to react and adapt to the changes).

Several approaches have been introduced for designing robotic control systems. In the area of manually-designed robotic controllers, methods range from classical planning techniques (e.g. STRIPS [7]) to reactive planning (behavior based robotics; Brooks subsumption architecture [5]). Another approach is to create robotic controllers automatically, training them for a given task in a simulated environment. Examples include back-propagation learning algorithm for artificial neural networks or evolutionary search methods, inspired by the natural evolution.

Simulated environments are a good foundation for performing experiments with the physical structure of the robot (it is easier to alter the body of a robot in a simulated environment than in the reality). Therefore, several methods optimizing both robot morphology and control system have been proposed. Some works have even accomplished the transfer of simulated robots into the real world.

This thesis studies automated design of artificial robotic organisms controlled by neural networks using evolutionary search methods. The following two sections briefly

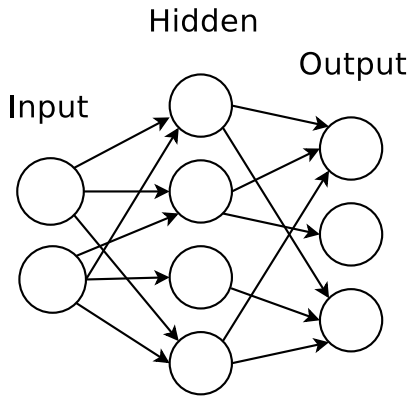


Figure 2.1: An example of a feed-forward artificial neural network topology.

summarizes existing methods in the evolution of artificial neural networks and the evolution of morphology and control systems of robotic organisms.

2.1.1 Methods of artificial neural network evolution

Artificial neural network (ANN) is an interconnected group of artificial neurons (see Figure 2.1.1 for an example of a neural network topology), inspired by real-world neural networks found in many complex living organisms. Neural network model found in the nature has proved to be a very powerful and adaptive computational model, capable to process robustly large amounts of data. It would be, therefore, very useful to imitate neural networks in a simulated computer environment.

Several models of ANN have been proposed in the past. The main challenge for an ANN algorithm is to optimize network properties, in such way that the neural network will be able to solve a given task. Traditional methods use a fixed network topology, which is provided by the experimenter in advance and only optimize weights of individual neural connections. However, recent methods have shown, that optimizing the topology of the neural network as well can increase the performance of the learning algorithm.

Two main categories of traditional learning techniques exist: supervised learning and unsupervised learning. In supervised learning, a "teacher" is present, who in the learning phase "tells" the network how well it is solving the task (reinforcement learning), or directly provides expected output values (fully supervised learning). In unsupervised learning, the neural network discovers some of the properties of the input data by itself and outputs data relevant to these properties. Survey of traditional ANN learning methods is beyond the scope of this thesis and can be found in [2, 16, 3].

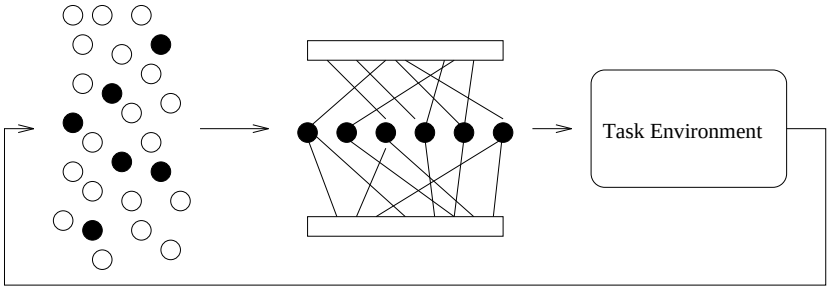
This thesis is focused on a specific class of non-traditional reinforcement learning techniques of optimizing ANN models - NeuroEvolution (NE) techniques. NE techniques optimize neural networks using evolutionary algorithms. Methods of ANN evolution range from those evolving only weights of neural connections with networks of fixed topology, to those evolving both structure of the network and weights of neural connections. Recently, several novel methods of indirect encoding of neural networks have also been proposed.

Evolution of neural networks has become a successful method of reinforcement learning. An often used benchmark problem for comparing different methods of reinforcement learning is a double pole-balancing problem. In this problem, a neural network controls a movement of a cart with two vertical poles attached to it. The goal is to keep both poles in the vertical position. The problem is suitable for benchmarking, because its difficulty is easily scalable by changing the relative lengths of the two poles (problem becomes more difficult as the difference in length of two poles gets smaller). The pole-balancing experiment was first introduced in 1991 by Wieland [42] for testing one of the first methods of evolving neural networks and it is used as a standard benchmark for reinforcement learning algorithms since.

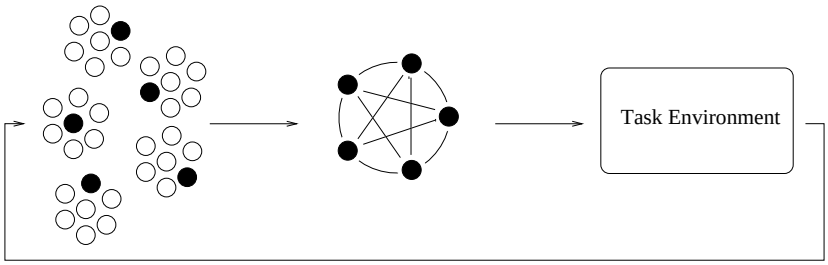
Several early approaches studied evolution of connection weights of a fixed topology neural networks [28, 41]. Wieland used both mating and mutation for evolving weight values of neural networks with fixed topology [42]. His approach is considered to be Conventional NE. An example of a more advanced method of evolving fixed topology neural networks is SANE (Symbiotic, Adaptive Neuro-Evolution) [25]. SANE works by evolving individual hidden neurons instead of entire networks. SANE evolves networks with a single hidden layer. Each hidden neuron in the population is described by a set of its input and output weights. During fitness evaluation, a random fixed-size set of neurons is chosen from the entire population to form a hidden layer in a neural network. This network is then evaluated on a given problem (i.e. double pole-balancing). This test is performed repeatedly for different random sets of neurons, until each neuron had appeared in at least 10 tests. Fitness of a neuron is the average fitness of all the tests it participated in (see Figure 2.2(a)). SANE has been shown to outperform other reinforcement learning techniques, such as Q-learning on the problem of double pole-balancing [25], but it has been surpassed by more recent NE methods.

First NE method evolving network topology was Cellular Encoding (CE) [12]. CE is motivated by biological cell growth, which occurs by cell division. CE uses a set of grammar rules to compactly represent topology of a neural network. Grammar rules represent a grammar-tree, which describes the cell growth process. Each cell contains a reading head pointing to a node in the grammar-tree, which instructs the cell how to divide (several types of cell division are used). Growth starts with a single cell. After each division, reading head of the first child advances to the left subtree and reading head of the second child advances to the right subtree. Finally, when the development of a cell terminates, cell transforms into a neuron. CE was one of the first methods to include a developmental phase in the process of phenotype construction (similar to development of embryo in natural organisms). CE was also the first method to successfully solve the double pole-balancing problem.

A direct follower of SANE is ESP (Enforced Sub-populations) method [11]. ESP addresses a problem of difficult specialization of neurons in SANE. Neurons in SANE compete and mate with neurons from entire population and thus it is unlikely, that different groups of neurons develop for different functions. ESP solves this problem by explicitly dividing population of neurons into n species, where n is the desired number of neurons in the hidden layer. Neurons are allowed to recombine only within their own species and competition also occurs only within the species (see Figure 2.2(b)). Besides



(a) SANE (Symbiotic, Adaptive NeuroEvolution)



(b) ESP (Enforced Sub-populations)

Figure 2.2: A comparison of SANE and ESP algorithms. Both algorithm evolve neurons instead of entire networks. SANE evolves all neurons in a single population, while ESP divides population into separate species. Each species contains candidate neurons for a single hidden node in a network [11].

allowing neurons to specialize during the process of evolution, speciation also allows recurrent networks to be evolved. This is possible in ESP (as opposed to SANE), because specialization of neurons allows neuron in one species to *expect* values from another neuron in another species. ESP evolves fixed topology networks. However, when the evolution stagnates (i.e. the search gets stuck in a local optimum), ESP is restarted with a random number of hidden nodes. ESP has been shown to outperform SANE on the task of double pole-balancing, and has been a state-of-the-art algorithm for double pole-balancing task [36], until NEAT (NeuroEvolution of Augmenting Topologies) algorithm has been proposed.

NEAT automatically evolves topology of the network, while optimizing connection weights at the same time. NEAT is able to efficiently evolve topology by introducing three main novel ideas. First, it proposes a method of sensible crossover of networks with different topologies. Each node and each connection is assigned a unique innovation number upon its creation during mutation. Innovation numbers are inherited and thus can be used to determine corresponding structure elements in parent networks during crossover (see Figure 2.3 for an example of a crossover operation). Second, it introduces a method of protecting innovation through speciation. Each population is divided into several species and each organism competes only against organisms in the same species (explicit fitness sharing mechanism is used for reproduction [10]). This allows new structural innovation (which might initially be disadvantageous) to optimize in a separate species, instead of being immediately dominated by currently better net-

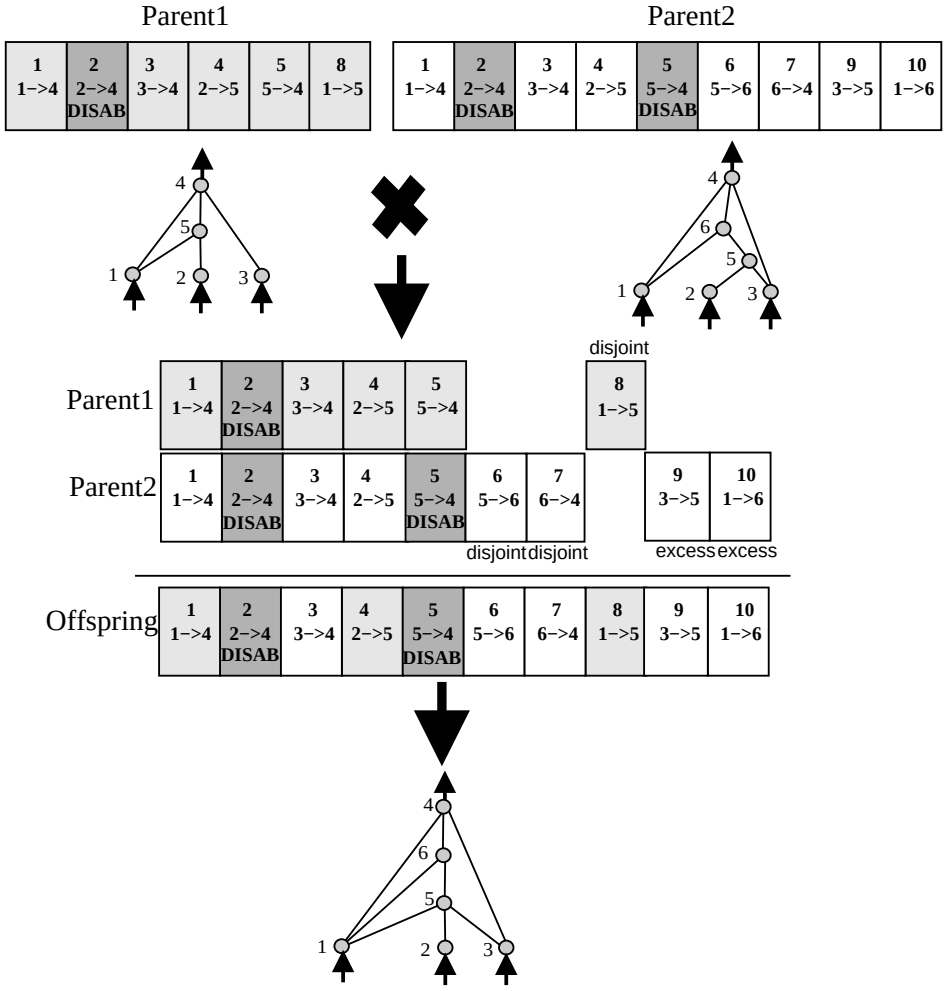


Figure 2.3: An example of a crossover of two networks with different topology in NEAT algorithm [36].

works in the entire population. Finally, NEAT starts the evolution of neural networks from a minimal, simplistic topology and complexity of the networks increases later, as the evolution proceeds. NEAT method is discussed in more detail in Chapter 3. NEAT has been shown to outperform all other methods of evolving neural networks (SANE, ESP, CE, Conventional NE) [36].

Since its introduction, several modifications and extensions of the original NEAT algorithm have been proposed. Real-time, online version of NEAT (rtNEAT) has been introduced and applied successfully to interactive learning of agents in a computer video game NERO [35]. FS-NEAT (FeatureSelection-NEAT) is another modification of the original algorithm, which has been used as a method of automatic selection of the best subset of inputs appropriate for solving a given task [40]. NEAT has also been recently used to improve the performance of temporal difference algorithms such as Q-learning (NEAT+Q) and Sarsa (NEAT+Sarsa) [39, 38].

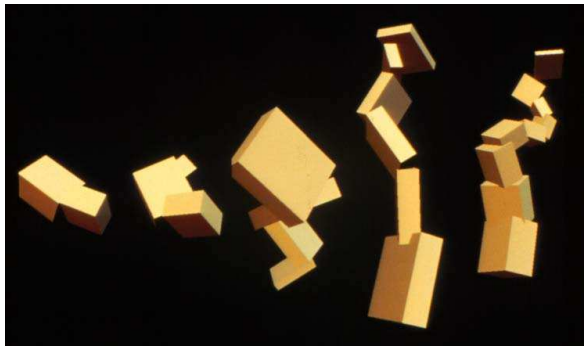


Figure 2.4: An example of the evolution of a creature by Karl Sims [31]. Rightmost creature is the final evolved creature; its ancestors are shown on the left.

The ultimate goal of artificial neural network evolutionary algorithms is to achieve the amazing complexity of biological neural networks through artificial evolution in computers. As it turns out, this goal cannot be approached simply by using a direct ANN representation combined with brute-force large-scale computation. In order to represent such a complex neural network as a human brain, while still being able to optimize such a structure efficiently using evolutionary search techniques, a very compact indirect encoding of a neural network must be introduced. Several methods of indirect encodings have been proposed. Taxonomy of existing approaches to indirect encodings and promising directions of future research can be found in [37]. Two most recent and most promising implicit encoding methods are implicit encoding based on genetic regulatory networks (GRN) [27] and HyperNEAT algorithm [6, 8].

2.1.2 Evolving morphology and control system of virtual creatures

One of the most interesting applications of evolutionary algorithms is an evolution of 2D or 3D self-controlling virtual creatures. Research in this area was pioneered by Sims [31]. His three-dimensional virtual creatures successfully evolved various locomotion, swimming, jumping and following strategies, often resembling behaviors found in the nature (Figure 2.4 shows an example of the creatures evolved by Sims). Sims also simulated a competition of the creatures [30], successfully producing the arms race among competing species. Creatures are represented by a directed graph, where each node represents a body part of the creature and each connection represents a physical joint of two parts. Creature control system is distributed over the creature's body. Phenotype is constructed by recursively traversing connections of genotype graph (which may contain cycles). This process resembles a system of rewrite rules, where nodes represent terminal symbols and connections represent non-terminal symbols (see Figure 2.8 for an example of phenotype construction). More details about the morphology and control system of the creatures will be given in Chapter 2.2.

Sims' work has inspired several other works on 3D virtual creatures. Shim and Kim have evolved various flying creatures with different types of wings [29]. Miconi, in one

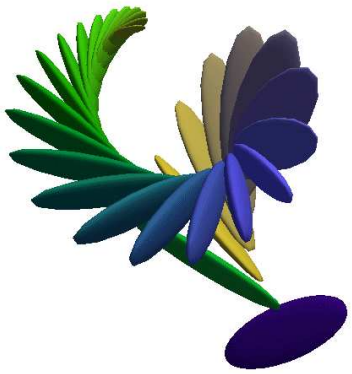


Figure 2.5: A creature evolved by an interactive evolution based on aesthetic selection [26].

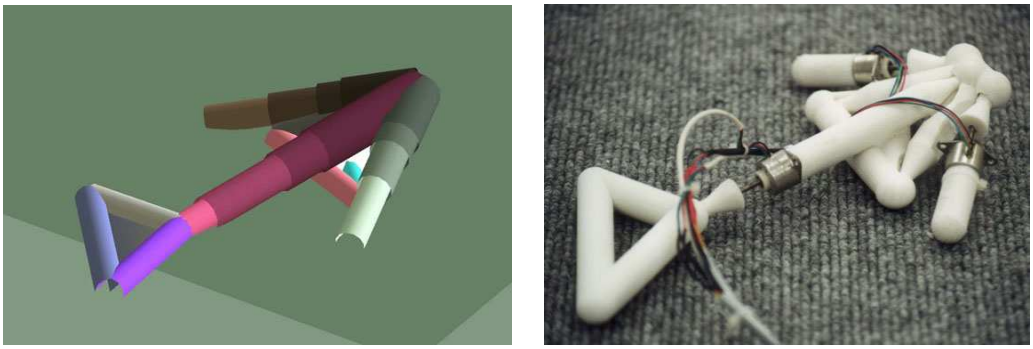
Figure 2.6: A creature evolved in a Framsticks project [18].

of the most recent works, successfully replicated original creatures of Sims, along with co-evolutionary arms race [24].

Ray has created a system for interactive evolution of a colorful three-dimensional virtual pets simulated in a physical environment [26]. The aim of Ray's work was to evolve creatures using emotional and aesthetic selection provided interactively by the user. Genotype of a creature was represented in the same way as in Sims' original proposal. Ray has interactively evolved various interesting creatures (see Figure 2.5 for an example of an evolved creature).

Spector has recently introduced a framework for studying open-ended evolution in a simulated 3D ecosystem [33]. Organisms start as a simple blocks (i.e. *Division Blocks*) controlled by an artificial neural network. All organisms are placed in a single virtual arena, where they collect energy from the virtual sun using photosynthesis, grow, shrink, divide, form joints and exchange resources with other organisms. Although this study is inspired by Sims [31], no explicit genetic algorithm is used to evolve creatures and no explicit behavioral fitness test is provided. Instead, organisms must evolve optimal reproduction and mutation parameters in order to survive in a challenging ecosystem. Preliminary results show, that organisms repeatedly develop cooperative strategies for resource transactions.

Komosinski and Ulatowski have created the *Framsticks* project for evolving virtual creatures [18, 17]. In Framsticks, both morphology and control system of the creatures are evolved. Each creature is composed of contractile "sticks", each having a wide range of properties inspired by natural organisms (e.g. initial energy level, muscle strength, friction, weight). Each stick is assigned an initial energy level at the start of each simulation (amount of initial energy is encoded in the genotype of the creature). Energy of a stick is decreased by muscle activation (energy consumption depends on the strength of the muscle) and increased by food intake. Three types of genotype representations are used: direct (genotype maps directly to phenotype), recurrent direct (phenotype is a tree structure encoded by a recursive genotype structure, similar to encoding used by Sims [31]) and developmental indirect encoding (both control system and morphology are grown from a single cell using growth instructions coded in genotype, similarly to



(a) A creature in a simulated environment. (b) Automatically constructed physical creature.

Figure 2.7: Example of a creature evolved in GOLEM project [22, 23].

Cellular Encoding [12] described in the previous section). Framsticks has been used to successfully evolve various swimming and walking virtual creatures.

Lipson and Pollack have evolved simulated robots capable of locomotion in simulated 3D environment and have also introduced a method of their automatic real-world fabrication [22, 23] (see Figure 2.7 for an example of an evolved creature). Lipson and Pollack have used a direct encoding for representing their creatures. A simple genetic algorithm was used with mutation and without mating.

Hornby and Pollack have built on the work of Lipson and Pollack and designed a developmental encoding based on L-systems for both morphology and control system of the creatures. Developmental encoding was compared to a non-developmental encoding based on a sequence of explicit build commands [14, 15]. Two different encodings were compared on a task of land locomotion in 3D environment. Results have shown, that developmental encoding outperformed non-developmental encoding and evolved more-fit creatures faster. Hornby and Pollack have also built the creatures in the real world, taking advantage of a modular nature of L-system encodings [13].

Bongard et al. [4] has recently proposed a method of automatic recovery from unexpected damage for physical robots, through adaptive self-modelling. Subject of experiments was the four-legged physical robot. During the experiments, robot uses its sensor and actuator values to infer its (possibly damaged) morphology and then uses this self-model to move forward. Self-model is updated continuously, so when the robot is suddenly damaged, its self-model adapts and robot can use it to resume forward movement. It has been experimentally shown, that self-modelling improves locomotion abilities of a damaged robot.

In summary, several works besides Sims' have successfully evolved interesting virtual creatures and there have been several successful attempts to transfer simulated creatures to the real world. In this thesis, new results in the field of neural network evolution are combined with previous works on evolving virtual creatures to yield a novel method of evolving both morphology and control of the creatures using NEAT algorithm. NEAT performs very well on evolving structure of artificial neural net-

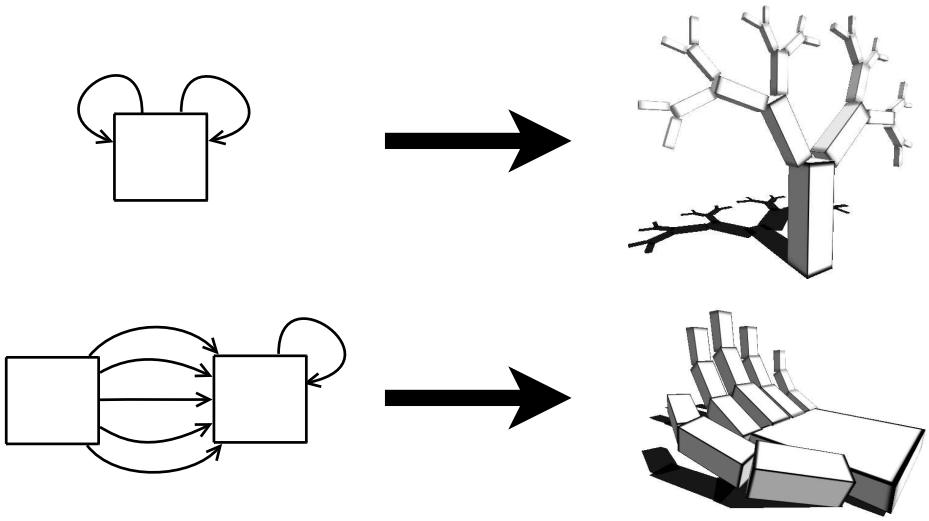


Figure 2.8: Manually designed examples of a transcription of genotype to a physical creature.

works. It is, therefore, a promising method for evolving structure of virtual creature morphology as well (for further motivation on using NEAT see Chapter 3).

2.2 Virtual creatures: Main concepts

This section introduces main concepts in the evolution of virtual creatures. Creature representation is inspired by the work of Sims [31, 30]. In the following text, the term *genotype* refers to a compact representation of a virtual creature (analogous to DNA in animals) and the term *phenotype* refers to its physical body.

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 summarizes fitness evaluation. Finally, section 2.2.5 introduces a testing mechanism for early removal of faulty creatures.

2.2.1 Creature morphology

A physical creature (i.e. creature *phenotype*) 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 2.8 for examples).

The 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. The sequence of steps taken upon entering a node in the genotype graph is described in Algorithm 2.2.1.

Algorithm 2.2.1 Processing a node during phenotype construction

1: Current genotype graph node (G) is copied and the copy is added to the phenotype graph.
2: Recursive limit of G is decreased.
3: All outgoing non-terminal connections of G are processed (as described in Algorithm 2.2.2).
4: If no copy of the node G has been created during the previous step (by processing the outgoing non-terminal connections of G), then all terminal connections are processed.
5: Recursive limit of G is increased.

Algorithm 2.2.2 Processing a connection during phenotype construction

1: If the recursive limit of the target node is zero, the current connection is not traversed.
2: A copy of the connection is added to the phenotype graph.
3: Target genotype node is entered and processed (as described in Algorithm 2.2.1).

Transition from a genotype graph with a terminal connection to corresponding phenotype graph is illustrated in Figure 2.11.

Both genotype node and genotype connection have various other 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 anchor point must be on the surface of the corresponding 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 2.10(a)). Local coordinate system is defined at each anchor point using the surface normal vector and two vectors tangent to the surface (see Figure 2.10(b)).

When creating a phenotype, 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

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: Description of joint types. 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 2.10(b)). Numbers in parentheses show the order of non-constrained rotations. Last column shows the number of degrees of freedom for a given joint type.

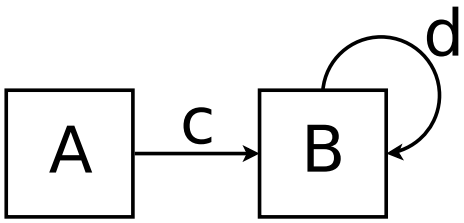
flags are enabled, two, four or eight mirrored copies of a child node are created in the phenotype graph, respectively).

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 is further illustrated in Figure 2.9.

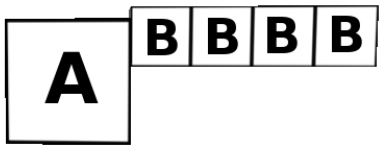
Each genotype node contains information about the *shape* and *size* of the resulting morphological node (in the case of a box, its dimensions are specified) and a *joint-type*. A joint-type defines physical properties of a joint connecting the current node and its parent in the phenotype graph. The 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 2.10(b)). 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**. All possible joint types are used, except for different permutations of the spherical joint type (two possible permutations of the universal joint type are not treated as different joint types, because they result in the same behavior).

2.2.2 Control system

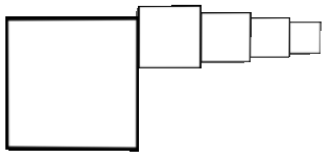
Creature's control system is distributed over its entire body. Each morphological node has its local sensors, effectors and a local controller. Besides local controllers, a single global controller (i.e. the "brain") is also present to allow global coordination among organism parts. Global sensors and effectors could also be implemented, although they



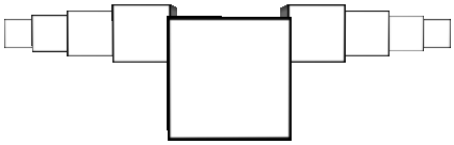
(a) Creature genotype. Recursive limits of nodes **A** and **B** are 1 and 4, respectively. Node **A** is the root node of the graph



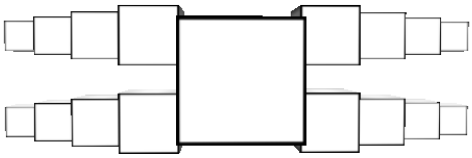
(b) Physical creature created from the genotype shown on the left. Four copies of node **B** are created during the recursive traversal of the genotype. Copies of node **B** are connected using connection **d**, while nodes **A** and **B** are connected using connection **c**.



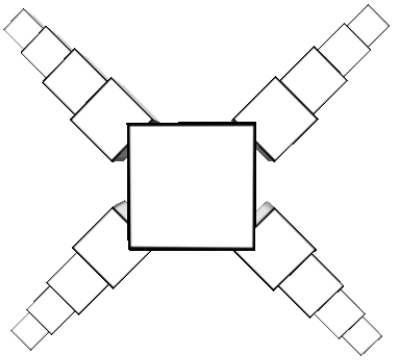
(c) Scaling factor of connection **d** has been changed from 1 to 0.8.



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

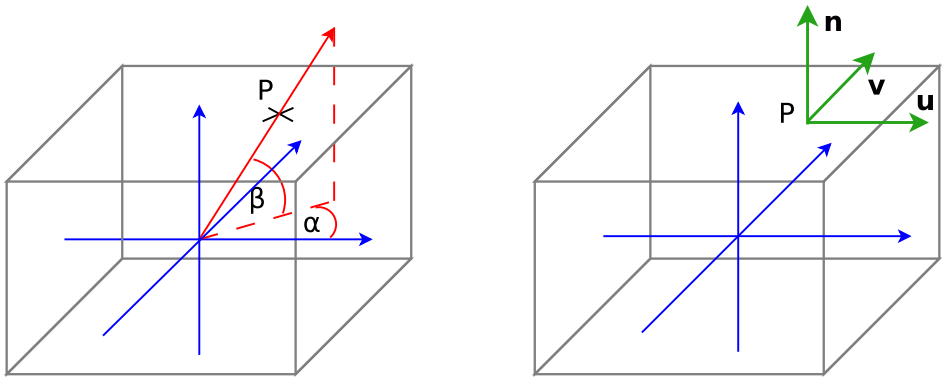


(e) Reflection flag for z axis of connection **c** has been enabled.



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

Figure 2.9: 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.

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

Figure 2.10: Position of the anchor point is specified in polar coordinates (left). Local coordinate system is then defined for the anchor point (right).

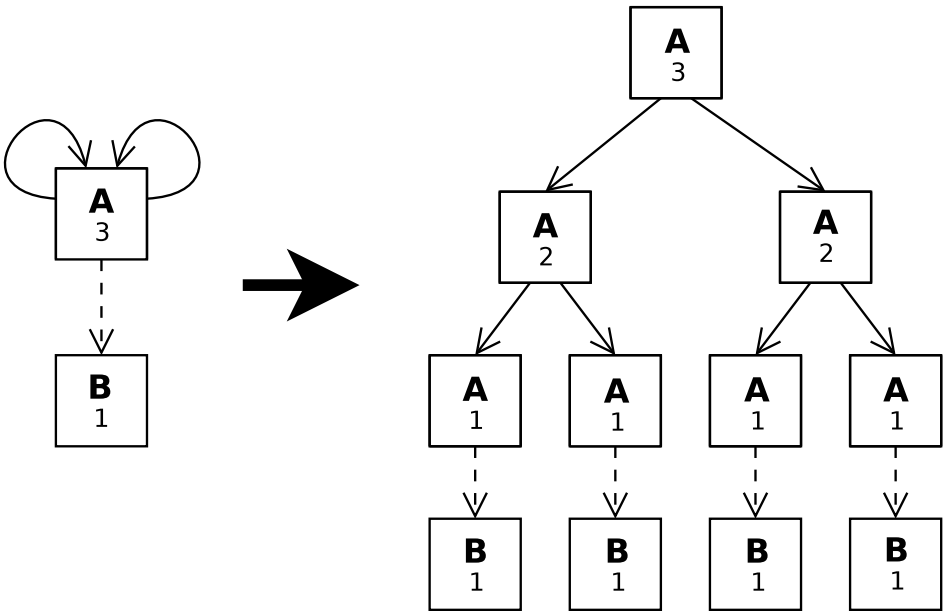


Figure 2.11: Creature genotype (left) and phenotype (right). Dashed line represents a terminal connection. Recursive limit of each node is shown under the node mark (A, B).

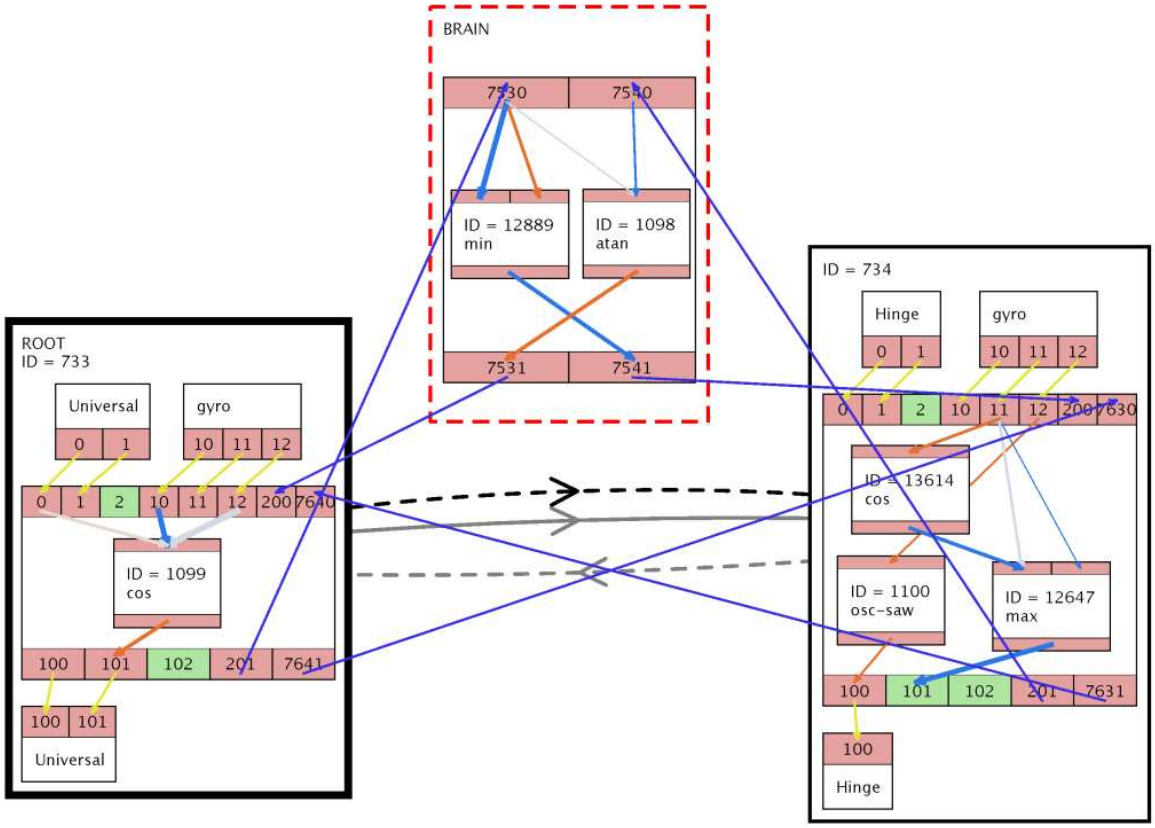


Figure 2.12: Example of a creature genotype, illustrating the distributed control system of the creature. The brain (represented by the box on the top) is connected to local controllers of morphological nodes using neural connections (blue lines). Neural connections also connect the controller of the 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).

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 the real organisms. An example of such creature distributed control system is shown in Figure 2.12.

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

In this thesis artificial neural network (ANN) controllers are used exclusively. Neurons in the ANN form a directed graph and are connected by weighted connections. Connection weights are limited to the range $[-2, 2]$. 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.

The topology of the graph can be configured to work in one of two modes: *recurrent* or *feed-forward*. In recurrent mode, there are no constraints put on the topology of the neural graph and connections can form feedback loops. Signal propagation works by evaluating neurons in a random order. In feed-forward mode, feedback loops are prohibited and signal propagation evaluates neurons in the order given by a topological sort.

2.2.2.1 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. Following sensor types were used in experiments with the creatures:

1. *The joint angle sensor* measures the current angle value for each degree of freedom of a joint. Each node contains a single joint sensor attached to a joint connecting a node with its parent.

2. *The gyroscopic sensor* measures relative orientation of a morphologic node. The orientation is provided in the form of a vector pointing upwards, relative to the reference frame of the body part containing the sensor.

3. *The touch sensor* is located on each face of each box and returns 1 if a contact occurred and -1 if there was no contact (no distinction is made between self-contact and contact with other objects in the virtual world).

4. The directional *photo sensor* provides information about the closest light source. Sensor values form a vector representing direction to the closest light source relative to the reference frame of the body part containing the sensor. Creatures use this information, for example, to approach the light source in the light following experiments.

2.2.2.2 Neurons

Neuron is the basic unit in an artificial neural network. Each neuron has a transfer function, which is one of *abs, atan, cos, differentiate, exp, integrate, log, memory, osc-saw, osc-wave, sigmoid, sign-of, sin, smooth-linear, divide, max, min, product, sum, greater-than, if, interpolate, sum-threshold* (see Table 2.2 for detailed information about the transfer functions). Set of transfer functions is inspired by those used by Sims in [31]. Each neuron has one, two or three inputs, depending on the transfer function. 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

Transfer function	Description
abs(x)	$\|x\|$
atan(x)	$atan(7x)/atan(7)$
cos(x)	$cos(2\pi x)$
differentiate(x)	$x(t) - x(t-1)$
exp(x)	e^{2x}/e^2
integrate(x)	$s = min(max(s + x, -1), 1)$, return s
log(x)	$max(-1, 0.2 \cdot ln\|x\|)$
memory(x)	keep last 30 samples in memory, output oldest
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
sign-of(x)	1 if $x > 0$, otherwise -1
sin(x)	$sin(2\pi x)$
smooth(x)	keep last 8 samples, output weighted sum of the samples, weights ascend linearly from past to the present
divide(x, y)	y/x if $\|x\| > 0.01$, otherwise $y/(0.01 \cdot sgn(x))$
max(x, y)	$max(x, y)$
min(x, y)	$min(x, y)$
product(x, y)	$x \cdot y$
sum(x, y)	$x + y$
greater-than(x, y, z)	z if $x > y$, otherwise $-z$
if(x, y, z)	y if $x > 0$, otherwise z
interpolate(x, y, z)	$w = (z + 1)/2$, $x \cdot w + y \cdot (1 - w)$
sum-threshold(x, y, z)	1 if $x + y > z$, otherwise -1

Table 2.2: Description of all transfer functions. The output of all transfer functions is limited to the range $[0, 1]$, to prevent signal congestion. Following notations are used in the descriptions: $x(t)$ and $x(t-1)$ denote values of input variable in the current and previous time-steps, respectively. dt denotes elapsed simulation time is seconds since the last time-step. φ_{frac} is the fractional part of φ and $sgn(x)$ is the signum function.

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.

2.2.2.3 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 effector allows the creatures to move in the physical world by applying torques to the physical joints of the creature body parts. Each degree of freedom of a joint is controlled separately, by a single effector value. Effector receives its values from a controller.

The straightforward application of the effector values to physical joints has shown

to be inconvenient, 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 physical 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

This section describes random creation, mutation and mating of both neural network controllers and virtual creatures. Algorithms used for generating and mutating neural networks are provided in Algorithm 2.2.3 (random generation) and Algorithm 2.2.4 (mutation). Both algorithm are controlled by a set of parameters provided in Table 2.3 (for random generation) and Table 2.4 (for mutation).

Algorithm for the mating of neural networks is not provided, because grafting and crossover (mating methods for creatures proposed by Sims [31]) do not allow mating of neural networks. Neither grafting nor crossover combines internal properties of individual nodes, making mating of neural networks (which are themselves internal parameters of morphological nodes) impossible. However, it is possible to mate neural networks, when creatures are evolved using NEAT algorithm. This form of mating will described in Chapter 3.

Algorithms used for generating and mutating virtual creatures are provided in Algorithm 2.2.5 (random generation) and Algorithm 2.2.6 (mutation). Algorithm parameters are provided in Table 2.5 (random generation) and Table 2.6 (mutation). Methods of mating virtual creatures will be described in the following text.

Creature mating assures that advantageous genetic information discovered by different individuals can be combined in their offspring. The same methods as those proposed by Karl Sims, i.e. grafting and crossover, were used for creature mating. The third, novel method is also proposed in this thesis. The proposed method of the creature mating is based on NEAT algorithm and will be introduced in detail in Chapter 3. Here, we describe the two methods introduced by Sims [31].

Both methods combine morphology graphs of the parents. During mating, the control system within each body part is copied along with that body part. Local control systems are thus not affected by mating and they are changed only during mutation. A global controller (i.e. the brain) is simply copied from the first parent. Neural connections between the parent and child node controllers, whose source or target node have been changed during reproduction, are pointing to the same relative locations or are randomly reassigned if this is not possible (i.e. if a corresponding input or output port is not present or it is already used by another neural connection).

Algorithm 2.2.3 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 2.3)
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**
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 2.3: Description and default values for parameters of the algorithm for generation random neural networks (Algorithm 2.2.3).

Algorithm 2.2.4 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

Algorithm 2.2.5 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 2.2.3 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
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 2.4: Description and default values for parameters of the algorithm for mutating neural networks (Algorithm 2.2.4).

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 2.5: Description and default values for parameters of the algorithm for generation random virtual creatures (Algorithm 2.2.5).

Algorithm 2.2.6 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 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 2.6: Description and default values for parameters of the algorithm for mutating virtual creatures (Algorithm 2.2.6).

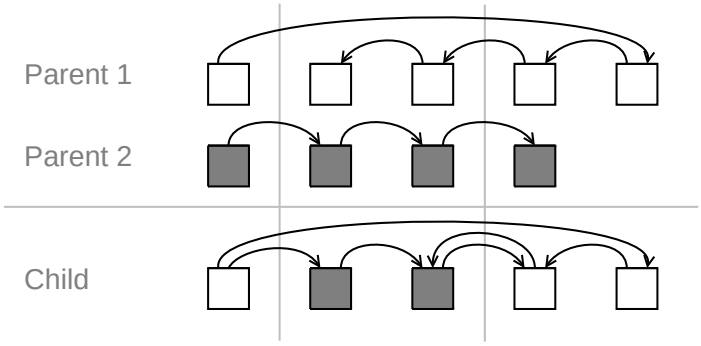


Figure 2.13: An example of the crossover mating.

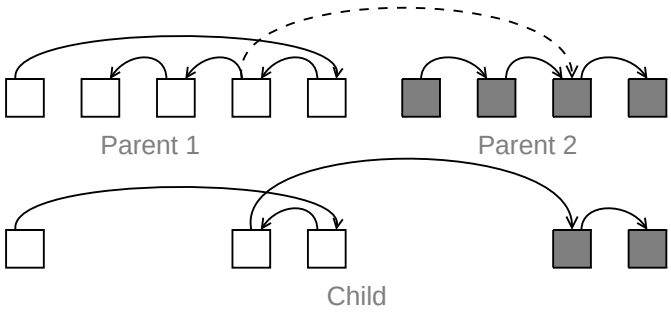


Figure 2.14: An example of the grafting mating.

2.2.3.1 Crossover

During crossover, nodes of each of the two parent genotypes are first aligned in a row. One or more crossover points are then randomly chosen. An offspring is created by copying nodes from the first parent until the first crossover point is reached. Copying then continues in the second parent and the copying source switches at each of the following crossover points. Connections are copied along with the nodes, pointing to the same relative locations (if a corresponding location does not exist, connection is randomly redirected). Example of the crossover operation is shown in Figure 2.13.

2.2.3.2 Grafting

Grafting works by first copying nodes from both parents to the resulting offspring, setting the root-node of the offspring to be the root-node of the first parent. Then, a single connection from the first parent is randomly chosen and redirected to point to a random node in the second parent. Finally, all nodes, which are unreachable from the root-node are deleted (both from the first and second parent). Example of the grafting operation is shown in Figure 2.14.

2.2.4 Fitness evaluation

Fitness of a creature is evaluated by first preforming a validity test. Validity test is introduced 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. The physical engine is used to simulate rigid body dynamics in the virtual world. Simulation proceeds in discrete simulation steps at the rate of 30 steps per simulated second. During each time step, following phases occur for each simulated creature:

1. Sensor values are filled according to the current creature environment.
2. Neural signal is 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. Simulation step is taken and the positions of all objects in the world are updated.

The simulation runs for a specified amount of time (60 seconds of simulated time), during which the creature is evaluated. Walking, swimming, jumping and following fitness evaluations are defined as proposed by Sims in [31]. Individual fitness functions are described in the following subsections.

2.2.4.1 Swimming

The water environment is simulated by turning off gravity and adding a viscosity effect. An organism fitness is determined as the distance traveled by an organism during the simulation. Forward motion is favored over circular motion by computing fitness as a distance between starting and final position of the creature.

2.2.4.2 Walking

During the walking fitness evaluation, the simulation is started with gravity turned on and a single infinite plane representing the ground added to the virtual world. A creature is then placed on the ground plane. A special initial phase of simulation is needed to prevent the creatures to evolve tall, tower-like morphology. These creatures achieve high fitness values by simply falling on the ground. To prevent this strategy from taking over the evolution, organisms are first simulated for a specified amount of time without a fitness measurement. After the initial phase, the creature's current position is marked as the starting position and the simulation continues. The creature's fitness value is the distance between its starting and final position.

2.2.4.3 Jumping

The simulation is set up in the same way as for the walking fitness. The lowest point of the creature is measured during each simulation time step. The largest height of all lowest points is returned as the fitness value.

2.2.4.4 Following

Following fitness takes place in the water environment. Six independent tests are conducted for each fitness evaluation. During each test, a light source is placed in a large distance along one of three major axes of the world coordinate system (for both positive and negative directions). Velocity of the creature's center of mass at each point of the simulation is projected to the vector pointing from the creature to the light source. The projection represents speed of the creature towards the light source. All speed values from all six tests are finally averaged to provide the final fitness value of the creature. Negative fitness values are prohibited (zero is returned in such case).

2.2.4.5 Physical simulation

Open Dynamics Engine (ODE) is used to simulate the rigid body dynamics (including collision detection and the friction approximation). ODE provides reasonably robust physical simulations and has 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. For example, without the simulation watchdog (described in the following text), several creatures evolved to change position using small explosions. Creature initiates an explosion by hitting one box with another in just the right way. Physics engine manages to reassemble the creature after the explosion. However, the creature is reassembled in a slightly different position. The following mechanisms were introduced to prevent this and other similar behaviors:

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 small oscillations (which is another way of exploiting inaccuracies in ODE). 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 2.2.2.3).

4. *ODE configuration.* ODE was configured to be as robust as possible to all kinds of creature behavior, while retaining a realistic 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 [32] 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 abusive creatures early. The following creature properties are tested before the simulation starts:

1. The count of nodes in the phenotype graph must be in a specified range (organisms with a large number of nodes tend to be highly unstable, while organisms with a small number of nodes are uninteresting).

2. Organisms must not interpenetrate themselves in their initial position.

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).

Simple organisms validity tests are much faster to compute than the entire physical computation and moreover, it is convenient 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 crossing) 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 the creature generation, but significantly increases the overall computation speed, because genotypes do not have to be evaluated by the computationally-expensive fitness function 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 2.2.2.3). 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 validity testing is a good 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. Primary use of ERO is to serve as a tool for experiments with evolving virtual creatures. ERO is, however, useful not just for the evolutionary computations, but for other parallel computations as well.

Parallel computation in ERO is called *a project*. For each project, ERO provides a user-friendly project configuration and an environment for watching the project results.

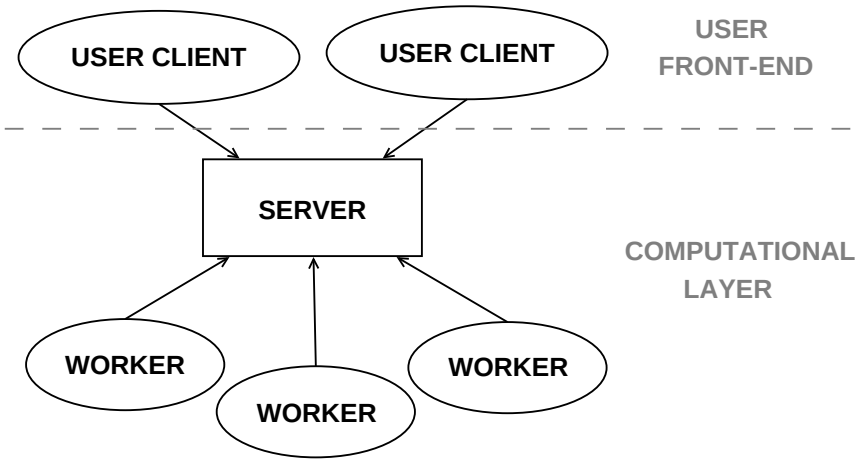


Figure 2.15: Overview of main ERO components.

Current projects in ERO include a testing project which computes an approximation of π and two evolutionary projects: one for evolving real-numbers coded as binary strings (see Section 4.2 for details) and other for evolving virtual creatures.

ERO is composed of three different applications: *the user-client*, *the server* and the computational client (or *the worker*). The relationship among these components is illustrated in Figure 2.15. The 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 the following section. The user-client is an interactive user-friendly interface to ERO. User can 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.

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 the project which created the task). Number of workers connected to the server is not limited.

The user-client offers various statistics about the parallel computation. Besides basic information (number of generated/finished tasks, running time, 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 included in the graph (see Figure 2.16 for an example of the efficiency graph).

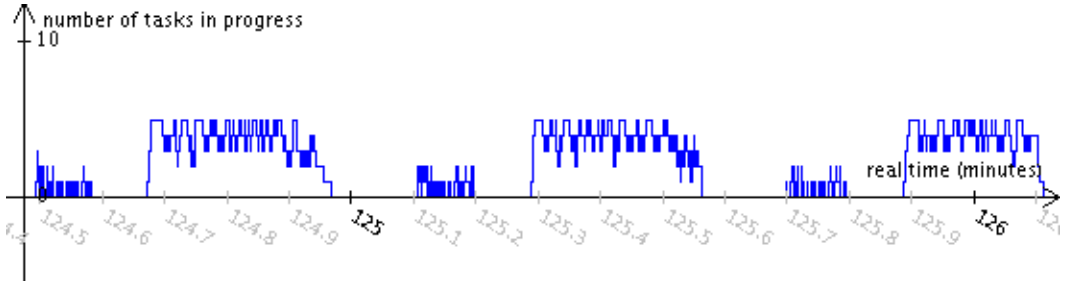


Figure 2.16: Computational efficiency of evolutionary computation using 5 workers.

2.3.2 Implementation notes

ERO is implemented in the Java programming language. ODE [32] 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. The RMI is used for the TCP/IP communication. The XML file format is used for saving organisms and a 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, which are 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.

Chapter 3

Evolving creatures with Hierarchical NEAT

This chapter introduces Hierarchical NEAT – a novel method for evolving virtual creatures. The proposed algorithm is inspired by NEAT – an algorithm for efficient evolution of neural networks. NEAT is an algorithm proposed by Kenneth O. Stanley in 2002 [36]. What makes NEAT unique among other algorithms for evolving artificial neural networks is its ability to efficiently evolve topology of the network along with weights of individual neural connections. All main ideas, which NEAT is based on, are also very general and can be applied to evolution of other structures as well. Proposed algorithm takes advantage of this fact by applying these main ideas to the morphology of the creatures.

This chapter consists of four sections. The first section provides motivation for using NEAT for evolving virtual creatures and summarizes expected improvements. The second section describes how NEAT evolves 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. The third and the fourth sections provide a detailed description of application of NEAT to controllers and virtual creatures, respectively.

3.1 Motivation

During the experiments with virtual creatures in the original ERO framework, several observations have been made:

1. Grafting and crossover (mating methods proposed by Sims, described in Section 2.2.3) often result in an invalid genotype, which is discarded by the validity testing procedure (as described in Section 2.2.5). This suggests, that both mating methods do not combine properties of parental genotypes very sensibly, but instead act more like violent mutation operators, exploring new areas of the fitness landscape. The lack of robust and sensible method for mating creatures might have a negative impact on the process of evolution.

2. Mating operators also do not recombine internal parameters of morphological nodes and connections. Instead, both mating 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. Mating has been shown to improve the performance of the artificial neural network evolution [34]. Its absence in evolution of virtual creatures might, therefore, retard the evolutionary search.

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

To address the first observation, a new method of creature mating needs to be proposed. While grafting and crossover might be beneficial for exploration, they do not appear to work as robust methods of recombination. 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 new method should ensure that the resulting offspring combines properties of both parents, while still being functional.

The second observation could also be addressed by introducing a new mating 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, 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 "chromosomes", 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. It is possible, that the absence of recombination on the level of internal parameters (e.g. neural networks local to each body part), retards 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. This is, however, a difficult problem because body plans of parental creatures might 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, however, computationally expensive and do not guarantee correct results.

One of the possible workarounds to this problem is to perform mating only between creatures with the same topology of their morphological graphs. Evolution could, for example, 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 mating 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 (first one evolving morphology without recombination, while 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. The second generation would, therefore, probably 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.

NEAT algorithm is a further extension of the previous algorithms. NEAT offers a much more elegant way of structure recombination. NEAT introduces a concept of *historical markings*, which makes it possible to easily find correspondence of the parental body parts, even if their morphological graphs differ in topology (the concept of historical markings is described in Section 3.2). This correspondence then allows the mating algorithm to exchange genetic information in a sensible way (including the internal properties of the structural elements). NEAT, 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 are thus a very promising solution to both the first and the second observed problems.

NEAT, however, does introduce the concept of species, but to address a different issue. The primary reason for species in NEAT is to "protect innovation through speciation". Each population in NEAT 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 in NEAT is, that the evolution is less likely to "get stuck" 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, NEAT algorithm is a promising method for addressing all three observations mentioned at the beginning of this section. New method of recombination (based on historical markings) has a potential of being sensible, producing meaningful offspring. It should also make 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 should make the evolution less likely to "get stuck" on a solution too early. It should, however, also make it easier for creatures to complexify their morphological structure during the course of the evolution.

3.2 Evolving structure with NEAT

The central concept in NEAT is the concept of *historical markings*. Historical markings bring the possibility of tracing individual structure elements (e.g. neurons and neural connections in an artificial neural network) 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. 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 NEAT algorithm. Next section describes how they are used to mate organisms with different topology. Section 3.2.2 then describes the process of speciation in NEAT and what is the role of markings in this process. Section 3.2.3 summarizes motivation for another key concept in NEAT – starting the evolution with organisms with a minimal structure.

3.2.1 Recombination

Historical markings offer an elegant method of sensibly mating 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 very 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). Therefore, it is 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 grafting and crossover).

3.2.2 Speciation

Speciation is the evolutionary process by which new species arise. Speciation in NEAT 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 [10]. 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, however, inevitably reach the 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 an 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, however, 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 NEAT algorithm.

Genetic algorithm with speciation using explicit fitness sharing is outlined in Algorithm 3.2.1. Genetic algorithm is controlled by a series of parameters. Parameters are listed (along with descriptions and default values) in Table 3.1. These default values are used in all experiments in this thesis, unless stated otherwise. The default method

Algorithm 3.2.1 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 mating and mutating 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 mating prob.	Probability, that the mating partner will be selected from another species	0.1
Mutation only %	Portion of organisms, which are created by using mutation operator only.	25%
Mating %	Portion of organisms, which are created by mating.	75%
Mutation after mating prob.	Probability, that an offspring created by mating 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 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 3.1: Description and default values for parameters of the genetic algorithm.

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 initial population of organisms into species according to their similarity. The measure of similarity in the NEAT algorithm is called *the compatibility distance* and is computed in such way, that more similar organisms have lower value of compatibility distance. Moreover, if the compatibility distance is zero, 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 should 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 3.4.2 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 in Section 3.3.3 and Section 3.4.2, 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 [34] 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 the Algorithm 3.2.1) using the following series of steps. First, shared fitness value is computed for each organism as follows:

$$f'_{ij} = \frac{f_{ij}}{N_j} \tag{3.1}$$

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:

$$F_j = \sum_{i=1}^{N_j} f'_{ij} = \frac{\sum_{i=1}^{N_j} f_{ij}}{N_j}. \tag{3.2}$$

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:

$$N'_j = N \frac{F_j}{\sum_{i=1}^{S} F_i}. \tag{3.3}$$

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 algorithm of speciation 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 3.2 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"):

$$F_j = c_{youth} \sum_{i=1}^{N_j} f'_{ij} \tag{3.4}$$

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).

3.2.3 Incremental growth from minimal structure

Many approaches to the evolution of structure start with a population of organisms with randomly generated structure (Sims [31], Grau [12]). Authors of NEAT algorithm argue (and also prove their arguments experimentally [34]), 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.

NEAT comes with an idea of starting with a minimal, simplistic structure and increasing complexity as the evolution proceeds. Starting the search in this way also minimizes the dimensionality of the search space during the early stages the search. Minimal structure in the case of a neural network consists of a network without any hidden nodes, with neural connection randomly connecting the inputs directly to the outputs of the network.

3.3 Applying NEAT to controllers of virtual creatures

This section describes application of NEAT to the evolution of neural network controllers as described in Section 2.1. Application of NEAT to controllers in ERO is quite straightforward. Neural networks in ERO, however, differ from neural networks used in the original NEAT algorithm [36] in the neuron architecture.

Each neuron in ERO has a large set of possible transfer functions. Furthermore, these transfer functions differ in the number of inputs (the number of their inputs ranges from one to three). Each input receives values from one or more neural connections. However, multiple neural connections between an output of one neuron and an input of another neuron are prohibited. Example of a neural network is shown in Figure 4.8. To address this difference, a modification has to be made to the original NEAT algorithm.

3.3.1 Historical markings

NEAT algorithm, as proposed in [36], uses historical markings for tracing neurons and neural connections. In this thesis, a modified version of the algorithm is used. Historical markings are traced 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 (ranges from one to three, depending on 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.

3.3.2 Recombination

Neural network controllers are mated as described in Section 3.2.1. Matching pairs of neural connections are, however, determined using connection identifiers (introduced in the previous section), instead of 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.

3.3.3 Compatibility distance

Measuring the compatibility distance of two neural networks is, again, based on the 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).

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

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

$$\delta_c(d,e) = s \cdot \frac{|d_w - e_w|}{2 \cdot w_{max}}$$

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

$$\delta_c(d,e) = (1-s) + s \cdot \frac{|d_w - e_w|}{2 \cdot w_{max}}$$

where w_{max} is the maximum absolute weight of a connection (the maximum difference of two weights is thus $2 \cdot w_{max}$), d_w and e_w 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.

To compute compatibility distance δ for neural networks A and B, the distance of all nodes $D_n(A,B)$ and the distance of all connections $D_c(A,B)$ is first computed:

$$D_n(A,B) = |N_n(A,B)| + \sum_{(n_1,n_2) \in N_m(A,B))} \delta_n(n_1,n_2)$$

$$D_c(A,B) = |C_n(A,B)| + \sum_{(c_1,c_2)\in C_m(A,B)} \delta_c(c_1,c_2)$$

Where $N_n(A,B)$ and $C_n(A,B)$ 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 $N_m(A,B)$ and $C_m(A,B)$ are 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:

$$\delta(A,B) = \frac{t \cdot D_n(A,B) + (1-t) \cdot D_c(A,B)}{t \cdot (|N_n(A,B)| + |N_m(A,B)|) + (1-t) \cdot (|C_n(A,B)| + |C_m(A,B)|)}$$

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. $|N_n(A,B)| + |N_m(A,B)| \leq N_{max}$ or $|C_n(A,B)| + |C_m(A,B)| \leq C_{max}$), 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 $[0,1]$.

3.4 Applying NEAT to morphology of virtual creatures

This section describes a method of applying NEAT algorithm to the evolution of morphology of the virtual creatures. Most of the components of NEAT algorithm are very general and independent of a specific structure that is being evolved (these components are described in detail in Section 3.2). There are, however, two parts of NEAT, where the actual organism type matters. These are: the compatibility distance measure (which the speciation is based on) and the mating of the organisms. The application of each of these components to virtual creatures will be analyzed separately in this section.

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 3.3. 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, however, not necessary, since a single counter is perfectly capable of creating historical markings for

all structure elements on both levels. The only disadvantage of using a single counter is the 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 3.3.1).

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 3.3.3 for measuring the compatibility distance of two neural networks. There is, however, one important difference between evolution of neural networks and evolution of virtual creatures. Neural networks are coded in their genotype *directly*. There is no transcription phase; 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. It is not clear, whether the compatibility distance should compare genotype or the actual phenotype graphs of the creatures. There are no works (known to the author of this thesis), which compare speciation based on genotypes to speciation based on phenotypes. This thesis takes the approach of measuring compatibility distance of the creatures using their genotype. The historical markings offer a simple and straightforward way of measuring compatibility distance on genotypes. Measuring distances between phenotype graphs would be a lot more complicated, 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 3.4.2.

3.4.1 Recombination

Virtual creatures are mated hierarchically, based on the correspondence given by historical markings. The mating on the level of morphology works in the same way as described in Section 3.2.1. 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 3.3.2.

The following internal properties are subjects of recombination in 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, mating is capable of sensible recombination of genetic information even on the level of, for example, individual weights of neural connections.

3.4.2 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 3.3.3. 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 the morphological nodes:

$$\delta_n(a,b) = \frac{w_p \cdot \frac{|a^r - b^r|}{r_{max}} + w_m \cdot \delta_s(a^s, b^s) + w_m \cdot \delta_j(a^j, b^j) + w_c \cdot \delta'(a^c, b^c)}{w_p + 2 \cdot w_m + w_c}$$

where a^r, b^r are recursive limits of nodes a, b; r_{max} is a maximum possible difference between recursive limits of two nodes; a^s, b^s are geometric shapes of nodes (e.g. a box, a sphere, etc.); a^j, b^j are joints and a^c, b^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 δ', as defined in Section 3.3.3, is used to compare local neural networks of corresponding nodes. Compatibility distance between two joints is defined simply as

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

and compatibility distance of shapes is defined as:

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

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

$$\delta_c(d,e) = \frac{w_m \cdot \sum_{i=1}^{6} \delta_c^i(d,e) + w_p \cdot \sum_{i=7}^{11} \delta_c^i(d,e)}{6w_m + 5w_p}$$

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 $\delta_c^1, \cdots, \delta_c^{11}$ 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 3.3.3. The distance of all morphological nodes $D_n(A,B)$ and the distance of all morphological connections $D_c(A,B)$ is first computed using distance functions defined above:

$$D_n(A,B) = |N_n(A,B)| + \sum_{(n_1,n_2)\in N_m(A,B)} \delta_n(n_1,n_2)$$

$$D_c(A,B) = |C_n(A,B)| + \sum_{(c_1,c_2)\in C_m(A,B)} \delta_c(c_1,c_2)$$

Where $N_n(A,B)$ and $C_n(A,B)$ are the sets of all non-matching nodes and connections of morphological graphs A and B, respectively, and $N_m(A,B)$ and $C_m(A,B)$ 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:

$$\delta(A,B) = \frac{t \cdot D_n(A,B) + (1-t) \cdot D_c(A,B)}{t \cdot (|N_n(A,B)| + |N_m(A,B)|) + (1-t) \cdot (|C_n(A,B)| + |C_m(A,B)|)}$$

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. $|N_n(A,B)| + |N_m(A,B)| \leq N_{max}$ or $|C_n(A,B)| + |C_m(A,B)| \leq C_{max}$), 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 Section 4.1).

Chapter 4

Experiments and results

This chapter presents various experiments conducted to measure an impact of NEAT algorithm on the evolution of virtual creatures. The chapter starts by a section presenting various methods of visualizing NEAT algorithm which will be used in subsequent chapters. Other three sections describe conducted experiments.

Experiments were conducted in the order of their increasing complexity. The first category of experiments (presented in Section 4.2) is designed to test the functionality of the speciation and the fitness sharing algorithm. This part of NEAT is completely independent of a specific organism type used. Experiments are conducted using one of the simplest types of organisms – real numbers coded as binary strings. Organisms are evaluated by their ability to maximize the value of the given function. Experiments are designed to show, that evolution with speciation is capable of successfully finding multiple optima of a given function simultaneously, instead of focusing on a single optimum.

The second set of experiments (presented in Section 4.3) is designed to show the performance of NEAT algorithm on two neural network problems: XOR and function approximation. Experiments are designed to:

- compare the performance of implementation of NEAT presented in this thesis with the performance of the original implementation of NEAT, as described in [34],
- investigate impact of a larger set of transfer functions on the evolution of neural networks,
- perform a series of ablation experiments (where one or more components of NEAT are disabled) to confirm the usefulness of each of the components of NEAT.

Finally, the third set of experiments (Section 4.4) measures impact of the algorithm proposed in this thesis on the evolution of virtual creatures. The following sets of experiments are presented:

- a series of experiments with the proposed algorithm on different fitness functions (jumping, following, walking and swimming),

- a series of ablation experiments (where one or more components of the proposed algorithm are disabled) to confirm the usefulness of each of the components of the proposed algorithm,
- a series of experiments to determine the usefulness of communication among controllers and the presence of the brain,
- a series of experiments with different sets of transfer functions.

4.1 Visualizing NEAT

This section presents various tools used to visualize various parts of NEAT algorithm during experiments described throughout the rest of this chapter. Three tools 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).

4.1.1 Compatibility distance map

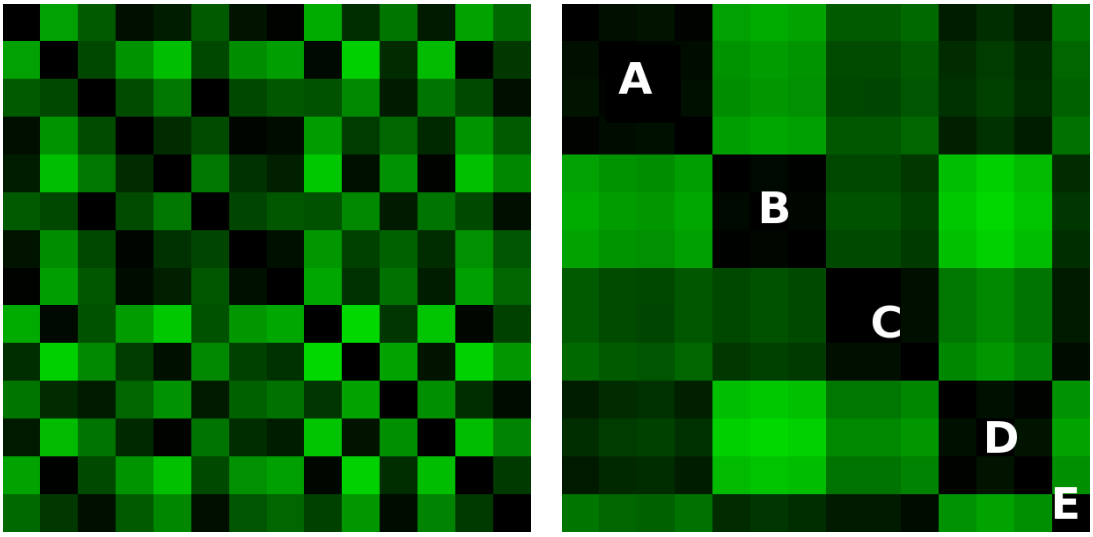
Compatibility distance is a function, which measures similarity of any two organisms. Given a sequence of organisms $s_1, \ldots, s_n$ and a compatibility distance function δ, the matrix M of distances among all organisms can be constructed: $M_{i,j} = \delta(s_i, s_j)$ for each (i, j), where $1 \leq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

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 4.1(a) shows an example of a compatibility map with an arbitrary ordered organisms, while Figure 4.1(b) shows the compatibility map of the same organisms 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 organisms in the same species.

4.1.2 Graph of the 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 4.2. Horizontal axis shows increasing generations, while the vertical axis corresponds to the fitness values. Each organism created during the evolution is marked on the graph using a semi-transparent red mark. The darkness of red, thus, represents number of organisms at the same position on the graph. Semi-transparent marks allow experimenter to see



(a) Unsorted compatibility distance map. Organisms are ordered by their identifiers: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13.

(b) Map showing compatibility distance between organisms sorted by their species (the order of organisms is 0, 3, 6, 7, 1, 8, 12, 2, 5, 13, 4, 9, 11, 10).

Figure 4.1: Visualization of similarities among 14 organisms. Black color corresponds to the zero distance and bright green corresponds to the distance of one. The left picture shows organisms in arbitrary order, while the right picture shows the same organisms, but reordered by the species they belong to. Species are marked with letters A, B, C, D, E.

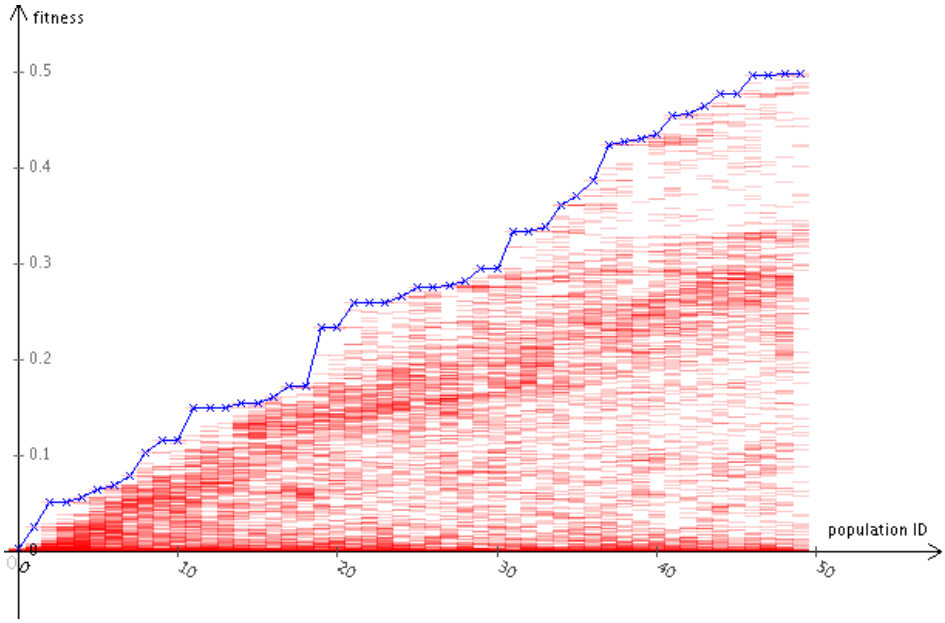


Figure 4.2: Graph of the fitness values of all organisms. Each organisms ever created during the evolution is marked by a semi-transparent red mark. Blue line connects the best organisms in each generation.

the fine structure in the fitness graph, where there would be a solid area otherwise. Best organisms in each generation are connected by a blue line.

4.1.3 Speciation graph

Visualization of speciation is inspired by a method proposed in [36]. Figure 4.3 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 the assignment of organisms to the species. Each row represents all organisms in a fixed-size population, sorted by their species. Each species is assigned a 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.

4.2 Testing speciation with binary strings

Speciation, one of the key components of NEAT, is a way of maintaining diversity throughout the evolution. Speciation in NEAT is based on fitness sharing, as introduced by Goldberg [10]. Goldberg has illustrated the advantages of speciated evolution on the problem of multimodal function optimization. This section attempts to repeat his experiments (as presented in [10]), in order to verify the functionality of the speciation algorithm used in this thesis.

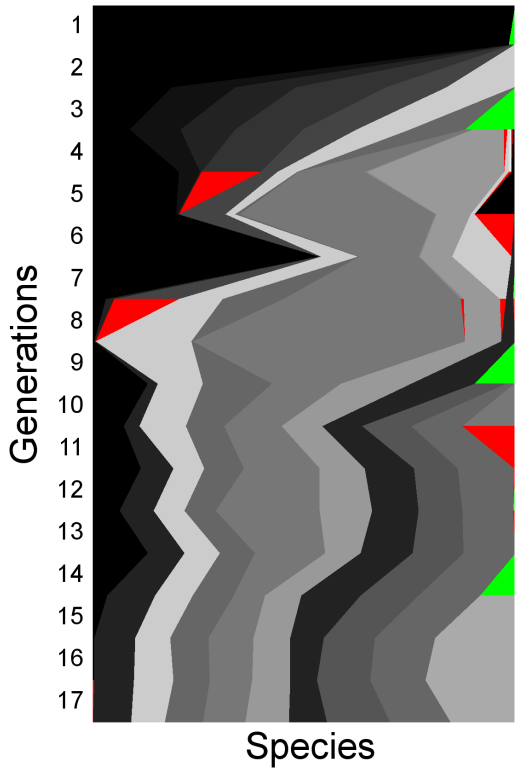


Figure 4.3: Visualization of the speciation. Vertical axis shows generations increasing from the top to the bottom, while horizontal axis shows the sizes and distribution of individual species. Each species is colored using a random shade of gray to distinguish it from other species. Red triangles represent the extinction of a species and green triangles represent birth of new species.

Parameter name	Value
Interspecies mating prob.	0.1
Mutation only %	50%
Mating %	50%
Mutation after mating prob.	0
Maximum stagnation	∞
Youth age threshold	0
c_{youth}	1.0
Initial c. d. threshold	0.1
Dynamic c. d. threshold	no
Desired number of species	10

Table 4.1: Parameter settings for the genetic algorithm. Descriptions of individual parameters are provided in Table 3.1.

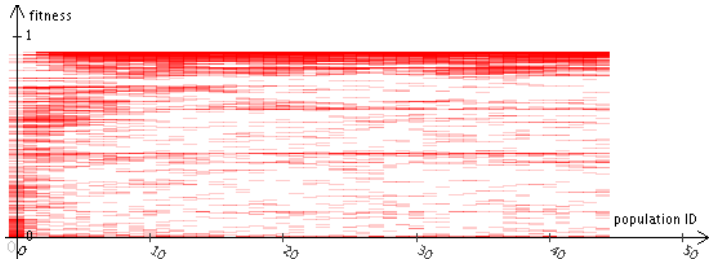
4.2.1 Setup of experiments

The subject of evolution are real numbers coded as binary strings. Genotype of each organism consists of 31 bits. Phenotype is a real number constructed by dividing a 31-bit integer represented by the binary string by $2^{31} - 1$. The resulting number thus resides in range $[0, 1]$.

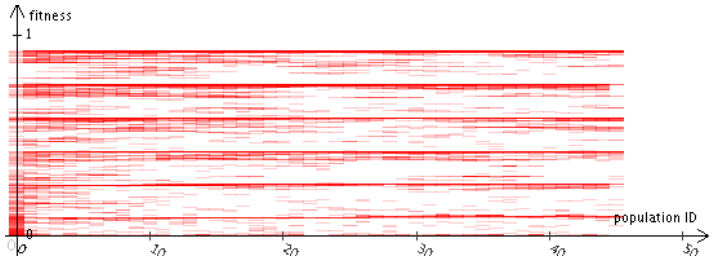
The first population of binary strings is generated randomly, with the uniform probability distribution. Binary strings are mutated by flipping one randomly selected bit. One-point crossover is used for mating (crossover point is chosen randomly).

Compatibility distance of two 31-bit strings is computed as the absolute difference of their phenotypes. Fitness function first constructs the phenotype x and then returns the value of $x \cdot sin^2 6\pi x$. The fitness function has six peaks of increasing size (see Figure 4.5 for illustration). Selection pressure imposed by this fitness function thus pushes organisms towards higher function values (i.e. towards the centers of the peaks).

In order to compare speciated and non-speciated evolution, two experiments have been performed. First experiment uses genetic algorithm (GA) with speciation based on fitness sharing, as described in Section 3.2.2. The second uses non-speciated GA (equivalent to the speciated GA starting with a single species, with births of new species prohibited). Otherwise, both GAs are configured in the same way. Population size of 300 organisms is used. Stochastic universal sampling [1] is used for selection. Half of each new generation is created by mutation and the other half by mating selected organisms. For speciated GA, compatibility threshold is set to 0.1, but other more advanced speciation techniques are disabled (such as stagnation tests, dynamic thresholding and additional protection of the new species using c_{youth} parameter), so that they do not interfere with the main purpose of this experiment which is comparing speciated and non-speciated evolution. List of all parameters is provided in Table 4.1.



(a) Fitness values in non-speciated GA. Majority of organisms is clustered on the highest peak, forming a single major "strand" on the graph of fitness values.



(b) Fitness values in speciated GA. Organisms are divided into six groups, each concentrating on a different local optimum.

Figure 4.4: Overview of the fitness values of all organisms created during the non-speciated (top) and speciated (bottom) evolution.

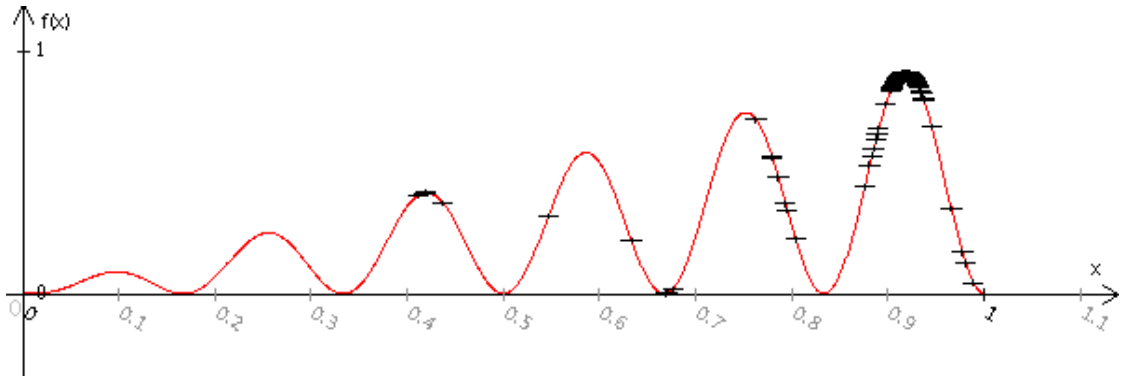
4.2.2 Results and discussion

Experiments have been terminated after 45 generations. Both experiments have successfully discovered the best solution (i.e. the highest peak). Speciated GA has, however, also discovered all other local maxima of the given function.

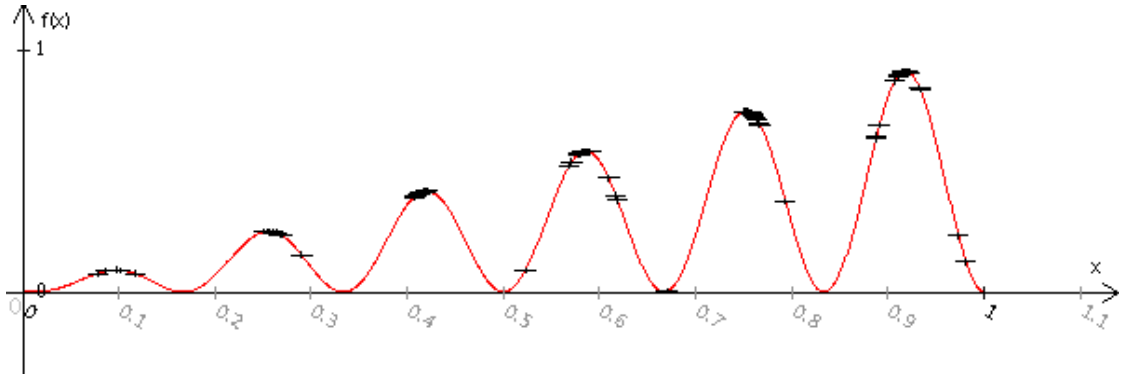
Results of both experiments are illustrated on a series of figures. Figure 4.4 shows an overview of fitness functions of all organisms created during the first 45 generations. Non-speciated evolution has very quickly (after roughly 10 generations) concentrated on the best solution, forming a single main "strand" of organisms near the optimum. Speciated evolution, however, did not concentrate on a single solution. Instead, six stable "strands" formed, each at the value corresponding to a different local maximum of the fitness function.

Figure 4.5(b) shows, that each of six "strands" corresponds to a group of organisms surrounding one of the peaks of the fitness function. Non-speciated GA has, however, concentrated only around the highest peak, neglecting all other peaks (as shown in Figure 4.5(a)).

Figures 4.6(a) and 4.6(b) further confirm, that speciated GA divides organisms into six species. Figure 4.6(a) shows the compatibility distance map of all organisms in the final generation of speciated GA. Organisms in the map are sorted by their species. Compatibility distance map confirms, that six independent species have been formed. Moreover, it shows, that the sizes of individual species are proportionally increasing. Figure 4.6(b) also demonstrates the relative sizes of species and also shows, that the

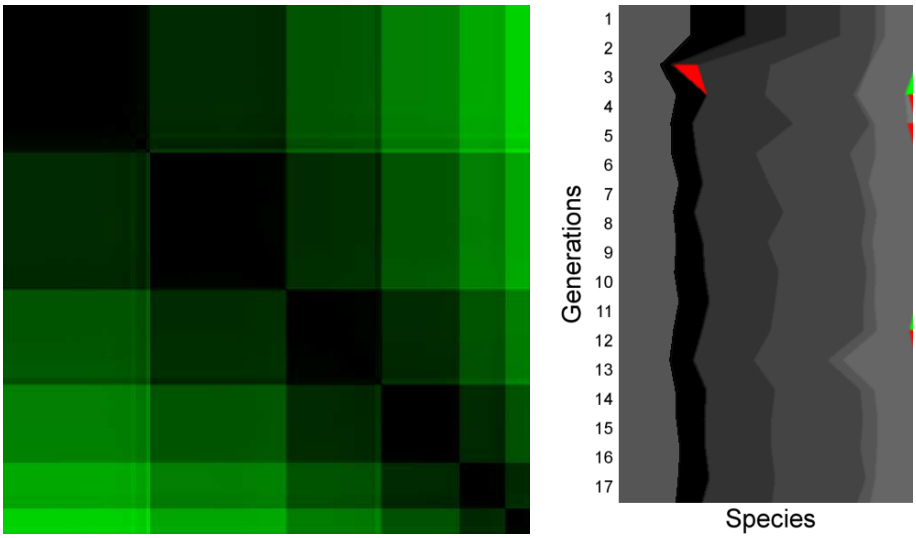


(a) Distribution of 300 organisms in the final generation of the non-speciated GA. 277 of 300 (92.3%) organisms are located on the highest peak and only 23 (7.7%) on other peaks.



(b) Distribution of 300 organisms in the final generation of the speciated GA. Organisms are distributed on all peaks, proportionally to the height of the peak. Distribution of organisms on individual peaks is (from the smallest to the largest peak): 14, 26, 44, 55, 77, 84.

Figure 4.5: Distribution of organisms in the final generation of experiments.



(a) Compatibility distance map of all organisms in the final generation, sorted by their species.

(b) Graph of species development over generations. Generations increase from top to bottom.

Figure 4.6: Visualization of speciation in multimodal function optimization experiments.

process of speciation is quite stable over time. All species in the figure manage to keep their sizes throughout the evolutionary process. Especially, the most fit species (one corresponding to the highest peak) does not dominate other species.

Speciation algorithm allocates space for individual species based on the average fitness value of all their organisms. Since organisms in speciated GA experiment are tightly clustered around the six peaks of the fitness function, the average fitness values of individual species should be approximately the same as the heights of individual peaks. Therefore, relative sizes of the populations should be the same as relative sizes of the peaks. Since the size of the peaks increases linearly, the ratio between heights of individual peaks is 1:2:3:4:5:6. Sizes of individual species (sorted in ascending order) in the final generation are 14, 26, 44, 55, 77, 84. This yields the following ratios: 0.98:1.82:3.08:3.85:5.39:5.88, which is close to the expected ratios of peaks sizes. The difference between real and expected ratios can be attributed to sampling errors and recent mutations.

This simple experiment has shown how the speciated GA explores multiple areas of the fitness landscape at the same time, giving each area as much attention as it "deserves", according to the average fitness function of all organisms exploring that area. On the other hand, in non-speciated GA, 277 out of 300 organisms (92.3%) are clustered on the highest peaks, while other 23 organisms (7.7%) organisms are scattered among other peaks.

In summary, multimodal function optimization experiments presented in [10] have been successfully repeated in this section and the functionality of the speciation algorithm used in this thesis has been thus verified.

4.3 Testing controllers

This section presents a set of experiments with neural network controllers (as described in Section 2.2.2). Experiments have been conducted with several goals in mind. The first goal was to verify the correctness of implementation of all components of NEAT algorithm and the neural network architecture without the interference of morphology. Results achieved by the implementation of NEAT in this thesis are compared to the published results achieved using the original NEAT implementation. Another goal was to perform a series of ablation experiments, confirming the usefulness of each of the components of NEAT. Finally, several experiments have been conducted to investigate impact of a larger set of transfer function on the evolution of neural networks.

Section is divided into four subsections. First subsections describes the setup of the experiments (e.g. description of fitness functions and settings of a genetic algorithm). Second subsection summarizes result of ablation experiments, which were aimed to test whether each major component of NEAT algorithm is necessary. Third subsection provides results of the performed experiments with a large set of transfer functions and a comparison of these results to the results obtained using a smaller set of transfer functions. Finally, fourth subsection presents discussion and summary of the achieved results.

4.3.1 Setup of experiments

In order to comprehensively test different configurations of NEAT algorithm, three problems of increasing difficulty are used. The simplest problem used in experiments is the XOR problem, in which the goal of the network is to compute logical operation of exclusive-or on its inputs. The two other, more difficult, problems are two variants of the function approximation problem. In these problems, the goal of the network is to closely approximate the graph of the given function. Two functions of different difficulty are used in the experiments. The following two sections describe fitness functions used for the XOR and function approximation problems. The configurations of genetic algorithms and genetic operators are provided in sections 4.3.1.3 and 4.3.1.4, respectively.

4.3.1.1 Fitness function for the XOR problem

Exclusive-or (or XOR) is a standard problem for neural network testing. Its difficulty lies in the fact that it cannot be solved without the presence of hidden neurons. It is, therefore, a good starting problem for testing the ability of NEAT to evolve structure of neural networks.

Neural networks used in XOR experiments have two inputs and a single output. Logical values of true and false are represented by values one and zero, respectively. Each test consists of four phases. During each phase, one of four possible combinations of input values is set (combinations are set always in the same order) and the neural signal is then propagated 75 times through the network. The difference of expected and real output value is measured after each propagation. The resulting fitness value

is computed as mean squared error of the differences in expected and real values in all phases.

Neural network is marked as *the winner*, if it computes the correct output in all steps (output value larger than 0.5 is interpreted as true, while other values are interpreted as false).

4.3.1.2 Fitness function for the function approximation problem

In the problem of function approximation, the neural network is evaluated according to its ability to approximate the given function. Functions of one input and one output variable are used. Each neural network thus has a single input neuron and a single output neuron.

Each test consists of 300 steps. During k^{th} step ($1 \leq k \leq 300$), the input value is set to $(k-1)/299$. Input value thus increases linearly from zero to one during each test. After setting an input value, neural signal is propagated through the neural network. As with the XOR problem, the difference of expected and real output value is measured after each step. The resulting fitness value is the mean squared error of the differences measured in each step.

The target function for approximation is defined piecewise. Function consists of a linear part, a constant part and an oscillating part and is defined in the following way:

$$f(x) = \begin{cases} 8x - 1 & \text{if } 0 \leq x < \frac{1}{4} \\ 1 & \text{if } \frac{1}{4} \leq x < \frac{1}{2} \\ cos(4\pi n(x - \frac{1}{2})) & \text{if } \frac{1}{2} \leq x \leq 1 \end{cases}$$

Parameter n in the oscillating part of the function is set by the experimenter and can be used to change the number of oscillations occurring in the last segment of the function. Changing the value of n changes the difficulty of the problem (functions with more oscillations are more difficult to approximate). In this thesis, the simpler version of the problem uses one oscillation (i.e. $n = 1$) and the more difficult version uses two oscillations (i.e. $n = 2$). Graphs of the functions are shown in Figure 4.7(a) (the simpler version) and Figure 4.7(b) (the more difficult version).

Neural network is marked as *the winner*, if the absolute difference between real and expected output value is less than 0.4 in at least 270 of all 300 steps.

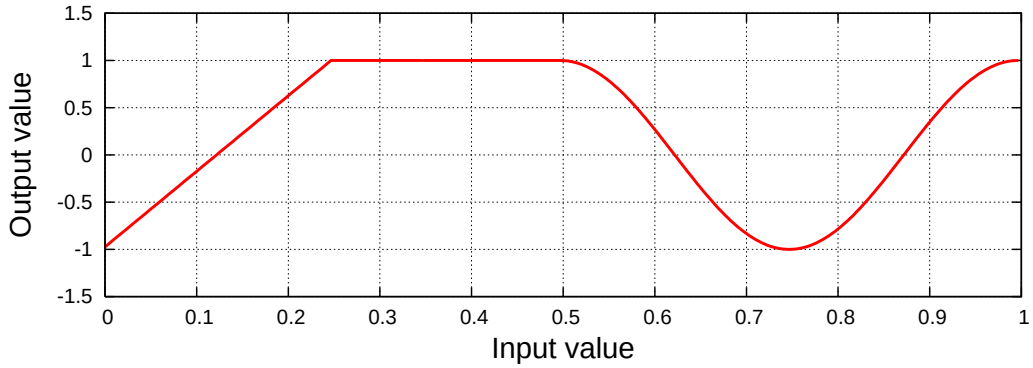
4.3.1.3 Genetic algorithm settings

All experiments used parameter values listed in Table 4.2, unless stated otherwise. Difference in population sizes (150 for XOR problem and 300 for function approximation problems) reflects the difference in difficulties of XOR and function approximation problems.

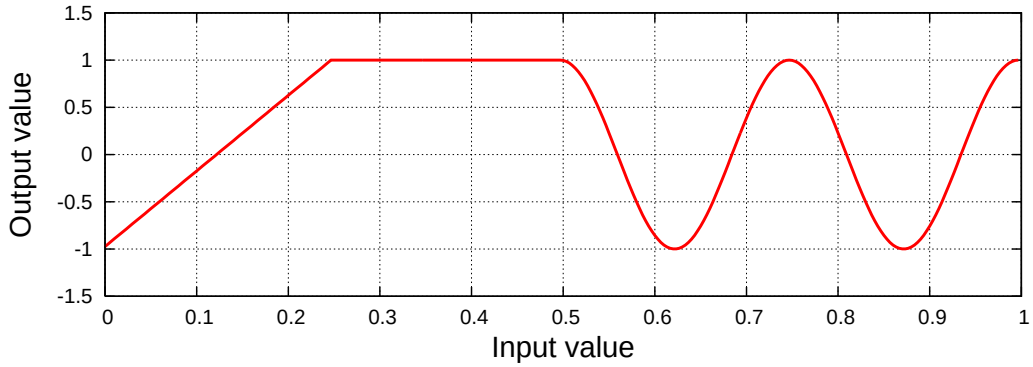
Each configuration has been tested 100 times. Each test consisted of running evolution with the given configuration for 200 generations.

4.3.1.4 Genetic operator settings

This section describes parameter settings for mutation, generation and mating operators. All genetic operators use default settings, as provided in Section 2.2.3, except



(a) The simpler version of the target function, with one oscillation ($n = 1$).



(b) The more difficult version of the target function, with two oscillations ($n = 2$).

Figure 4.7: Target functions used in the simpler (top) and in the more difficult (bottom) version of the function approximation problem.

Parameter name	**XOR**	**Approx**
Population size	150	300
Desired number of species	8	10
Interspecies mating prob.	0.1	0.1
Mutation only %	25%	25%
Mating %	75%	75%
Mutation after mating prob.	0.8	0.8
Maximum stagnation	15	15
Youth age threshold	10	10
c_{youth}	1.5	1.5
Initial c. d. threshold	0.5	0.5
Dynamic c. d. threshold	yes	yes

Table 4.2: Parameter settings for the genetic algorithm in XOR problem (second column) and function approximation problem (third column). Descriptions of individual parameters are provided in Table 3.1 in Chapter 3.

Parameter name	Value
s	1/3
t	2/3
N_{max}	10
C_{max}	20

Table 4.3: Parameter values for compatibility distance measurement. Descriptions of individual parameters are provided in Section 3.3.3.

Parameter name	Random init.	Minimal-start
Neuron count	5	0
Saturation %	150%	100%
Garbage collection	yes	no

Table 4.4: Parameter values of the algorithm for generation random neural networks. Descriptions of individual parameters are provided in Table 2.3.

those described in the following text. Recurrent neural connections have been prohibited in all experiments. Table 4.4 shows settings of network generation for experiments with random initial population and for experiments with minimal start. Settings used for compatibility distance function are provided in Table 4.3.

4.3.2 Ablation experiments

The purpose of ablation experiments presented in this section is to show specifically which components of NEAT improve the performance of the search and which do not. Set of ablation experiments was adopted from [34]. Ablation experiments were conducted for all three fitness functions (XOR and two function approximation problems). Only sigmoidal transfer functions have been allowed for all experiments. For each problem, the following set of configurations has been tested:

1. **Non-ablated NEAT algorithm.** All components of NEAT are enabled in this configuration: speciation, minimal-start and mating.

2. **Non-ablated NEAT algorithm with 100% probability of mutation after crossing.** Same configuration as 1, except for the probability of mutation after crossing, which is set to 100%.

3. **Nonmating NEAT.** Mating of the neural networks has been replaced with mutation in this configuration (otherwise, the configuration is the same as 1 above).

4. **Non-speciated NEAT.** Speciation is disabled in this configuration. All organisms are initially assigned to a single species and birth of new species is prohibited through the evolution. Otherwise, the configuration is the same as configuration 1.

5. **Non-speciated, initial-random NEAT.** The same configuration as configuration 4, but evolution is started with a population of randomly created organisms.

Id	Name	Failed	Evals-mean	Evals-std
A1	full NEAT	0%	4077.97	1522.02
A2	full NEAT with 100% mut.	0%	4293.55	1596.51
A3	nonmating NEAT	0%	4299.53	1678.88
A4	non-speciated NEAT	0%	9491.54	8621.35
A5	non-spec. rand. init. NEAT	1%	4039.85	4290.77
A6	random init. NEAT	0%	2760.38	1567.93
A7	original NEAT [34]	0%	4755.00	2533.00

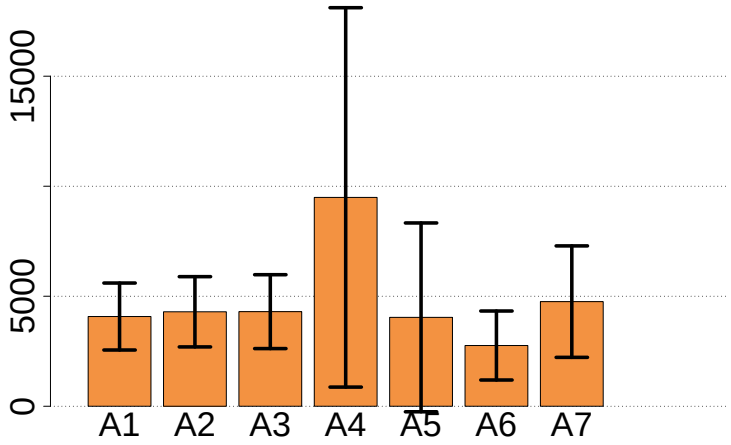


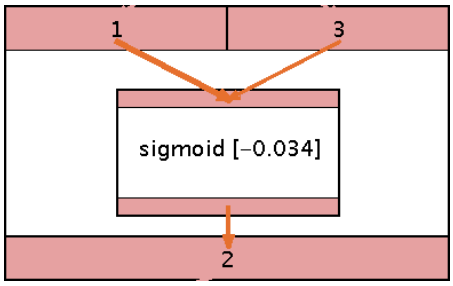
Table 4.5: Results of ablation tests with XOR fitness function.

6. **Initial-random NEAT.** Evolution is started with a population of randomly created organisms. Otherwise, the configuration is the same as configuration 1.

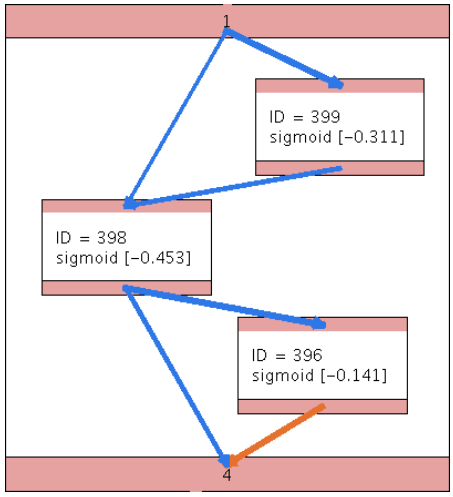
Configurations are designed to specifically show the following: that speciation speeds up the evolutionary search (configuration 1 should be faster than configuration 4), that starting from randomly generated organisms does not help the evolution (configuration 1 compared to configuration 6 for speciated case and configuration 4 compared to configuration 5 for non-speciated case) and that sensible mating using historical markings speeds up the evolution. However, in configuration 1, the probability of mutation after mating is 0.8, while in non-mating NEAT (configuration 3), all organisms are created using mutation. Organisms in configuration 3, therefore, receive on average more mutations than organisms in configuration 1. This prevents meaningful measurement of the effect of mating on the evolution. Configuration number 2 has been introduced to address this issue. In configuration 2, probability of mutation after crossing is 1.0. Therefore, configurations 2 and 3 differ only in mating and can thus be used to measure its relevance.

4.3.2.1 Results

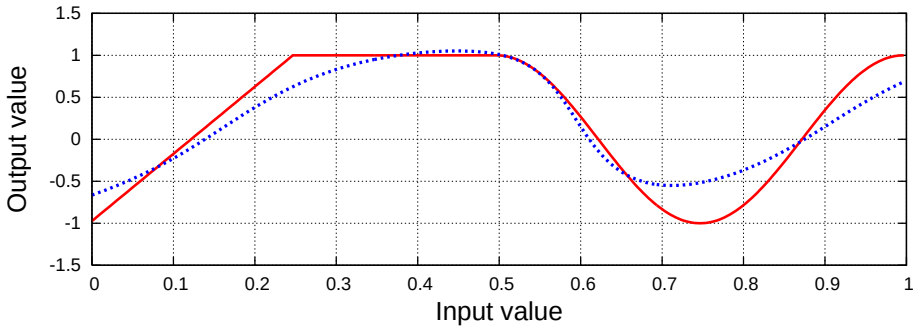
Results of the performed experiments are shown in Table 4.5 and Table 4.6. The last three columns of the table represent percentage of failed tests (where no winner has



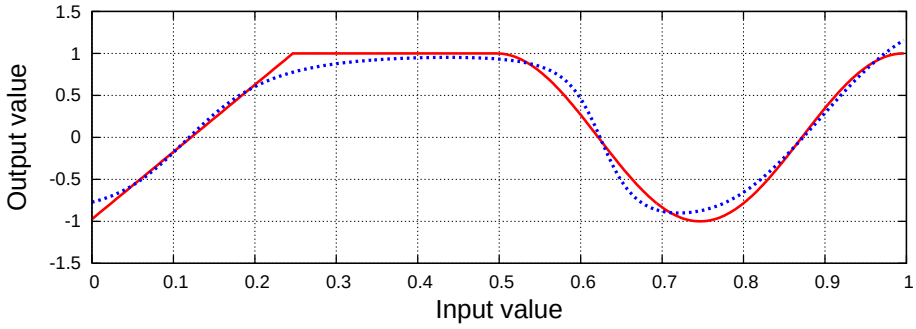
(a) Neural network evolved using configuration A1 (see Table 4.5) for solving the XOR problem.



(b) Neural network evolved using configuration B6 (see Table 4.6) for solving the SFA problem.



(c) Solution of the SFA problem by network shown in Figure 4.8(b). This network has been marked as the winner of the evolution.



(d) Solution of the SFA problem by neural network evolved further from network shown in Figure 4.8(b).

Figure 4.8: Examples of evolved neural networks for solving the XOR problem (left) and for solving SFA problem (right). Thickness of a neural connection represents the absolute value of weight of the connection, color represents the sign of the weight (red and blue colors represent positive and negative values, respectively). Number shown inside each neuron represents the bias of a neuron.

Id	Name	Failed	Evals-mean	Evals-std
B1	full NEAT	0%	21679.20	5578.11
B2	full NEAT with 100% mut.	0%	23189.09	5901.54
B3	nonmating NEAT	0%	21200.15	4819.76
B4	non-speciated NEAT	88%	57857.47	8307.80
B5	non-spec. rand. init. NEAT	23%	30679.63	19622.98
B6	random init. NEAT	0%	15258.58	4585.47

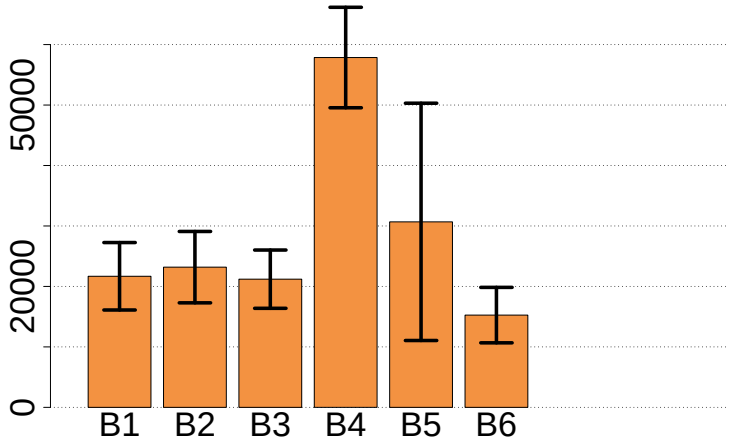


Table 4.6: Results of ablation tests with SFA fitness function.

been found during the first 200 generations) and mean and standard deviation of the number of fitness evaluations needed to find the first winner. If no winner has been found, maximum number of evaluations is used. Each function approximation evolution has taken about 10 minutes to finish, while evolution of XOR has taken about 5 minutes. Examples of evolved solutions for XOR problem and function approximation problem are shown on Figure 4.8(a) and Figure 4.8(b), respectively.

4.3.2.2 Discussion

Ablation tests have been designed to show that each component of NEAT algorithm is necessary. This section provides an analysis of the results with respect to the individual components of NEAT: speciation, sensible mating and minimal start.

None of the configurations has been able to solve the more difficult version of function approximation problem within the first 200 generations. This suggests, that the problem might not be solvable using only the sigmoidal transfer function. However, that is not the case because solutions *have* been discovered in several experimental longer runs, after about 300 generations. In the following analysis only XOR and the simpler version of the function approximation problems (SFA) are used. All p-values have been computed using Welch two sample t-test for unequal variances (as implemented in R Project for Statistical Computing). Significance level of 5% has been used. Configurations are referred to using either numbers from the list of configurations in the beginning of this section or by their identifiers listed in the result tables.

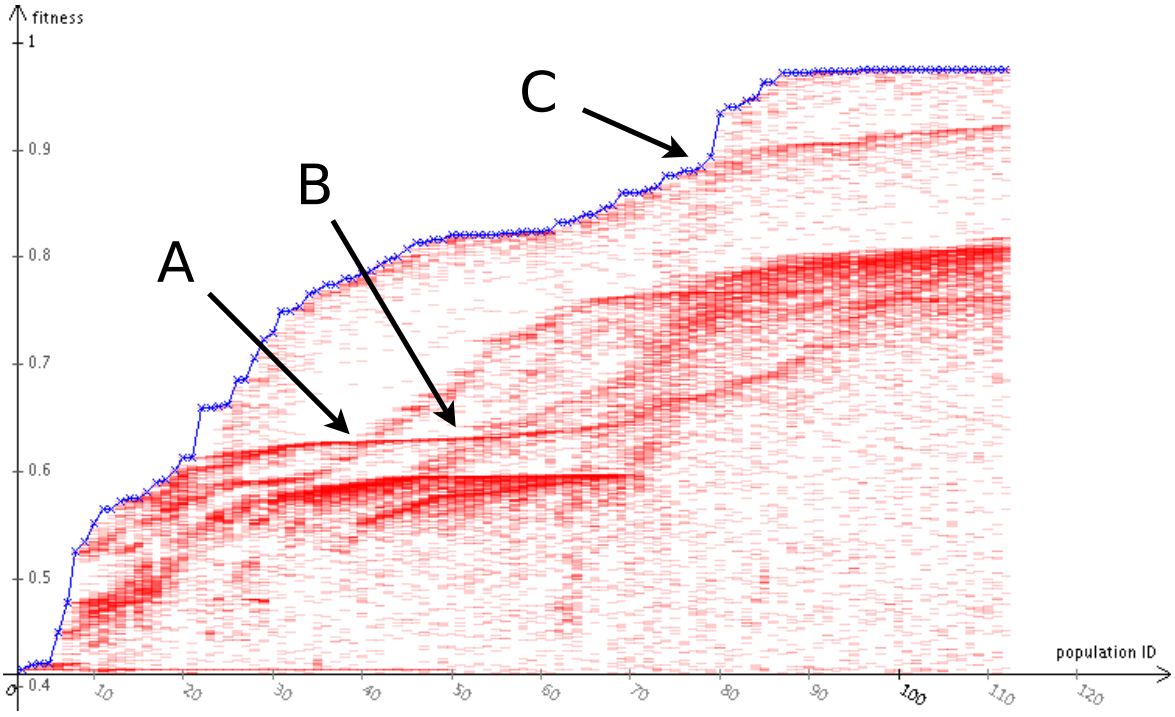


Figure 4.9: Graph of the fitness values (see Section 4.1 for description of the graph). Species correspond to red "strands" in the fitness graph. Points A, B and C highlight the moments in the evolution, when the fitness of one species overgrows the fitness of another species.

The effect of speciation can be studied using results of configurations 1 and 4. In both XOR and SFA, speciated search has surpassed non-speciated search in performance by a large margin (by 132% in XOR problem and by 166% in SFA problem). Moreover, non-speciated NEAT failed in 88% of experiments with SFA. The difference in mean values of the results is statistically significant in both cases ($p < 10^{-7}$). Large standard deviation in non-speciated XOR experiment (A4) compared to the much smaller standard deviation in speciated XOR experiment (A1) shows that speciation increases the stability of an algorithm. Smaller deviation in speciated XOR experiment reflects the ability of speciated NEAT to explore several different solutions simultaneously, avoiding the risk of "getting stuck". This effect is also illustrated in Figure 4.9. Since organisms in the same species achieve similar performance, their fitness values form a red "strand" in the fitness graph. The development of individual "strands" shows how speciation overcomes the problem of stagnation and premature convergence. If one of the species stagnates (or improves too slowly), another one takes its place in the evolution. This effect can be seen in various places in the fitness graph (points A, B and C highlight some of these moments). At the point C, previously best species is surpassed by a new fast-improving species, resulting in a large increase of the overall best fitness value.

The positive effect of mating is not apparent in the XOR experiments, where the difference between numbers of necessary fitness evaluations in configurations A2 and

A3 is too small to be statistically significant ($p = 0.9794$). However, the results on the SFA problem (B2 and B3) show, that the mating increases the speed of the search by 9.4% in this case. The difference is statistically significant ($p < 0.01$). Configuration 2 is used instead of configuration 1 to compensate for the increased mutation in configuration 4 (to ensure the fairness of the test).

The positive effect of minimal start has not been confirmed. Instead, all experiments with randomly generated initial population performed significantly better than their minimal-starting counterparts. Specifically, random initialization (A6, B6) increases performance of full NEAT (A1, B1) by 47.7% on the XOR problem and by 42% on the SFA problem ($p < 10^{-8}$ in both cases). The impact of random initialization has also been tested on non-speciated NEAT. Random initialization, again, speeds up the search significantly on both XOR and SFA problems. The reason for performance increase (instead of decrease) might be caused by different methods of random neural network generation. Random generation algorithm used in this thesis generates smaller networks (with at most 5 neurons), while algorithm used in [34] uses graphs with number of neurons ranging from 1 to 10. Also, original ablation experiments have been performed using the double-pole balancing method, while experiments in this thesis have been performed using function approximation problem. Experiments provided in this section have shown, that random initialization as performed in thesis is beneficial on XOR and SFA problems.

The performance of NEAT as implemented in this thesis has also been compared to the original NEAT implementation as presented in [34]. Results A1 and A7 suggest, that implementations have comparable performance. Implementation presented in this thesis appears to be faster on this particular problem. However, direct comparison can be misleading, because two implementations might differ in a variety of internal properties.

4.3.3 Experiments with different sets of transfer functions

This section presents experiments with different sets of transfer functions. Experiments are inspired by the work of Sims [31], who has used a wide range of transfer functions to evolve neural network controllers for his virtual creatures. This sections attempts to answer the question whether the large number of complicated transfer functions helps the evolution (by providing evolutionary "shortcuts"), or instead, retards it by enlarging the search space.

Three transfer function (TF) configurations have been tested:

1. **Sigmoidal TF.** Only the sigmoidal transfer function is enabled in this configuration.

2. **Oscillators.** Besides the sigmoidal transfer function, two types of oscillators are used: *wave-oscillator* and *saw-oscillator*. Oscillators are special in their ability to produce varying signal with constant input. This makes them suitable for usage in evolution of virtual creatures (e.g. they have been preferred in the evolution of flying creatures [29]).

Id	Name	Failed	Evals-mean	Evals-std
C1	Sigmoidal TF	0%	4077.97	1522.02
C2	Oscillators	0%	3386.23	1308.64
C3	All TFs	0%	1415.55	591.02

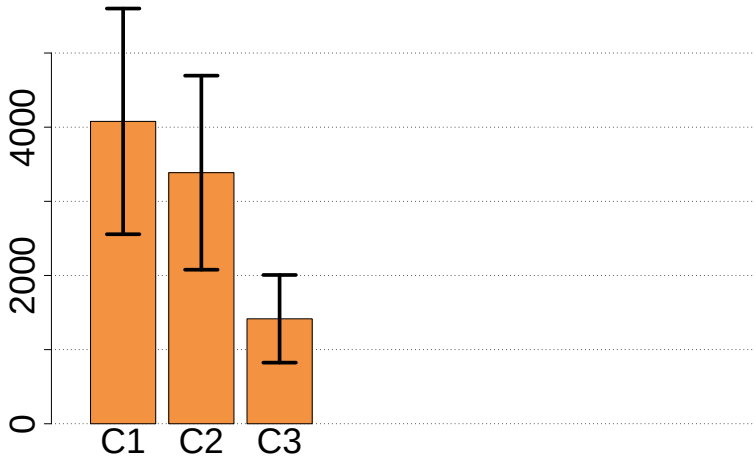


Table 4.7: Results of the tests with different sets of transfer functions with XOR fitness function.

3. **All TFs.** All 23 different transfer function are allowed to be used during this test. The transfer functions are described in Section 2.2.2.2.

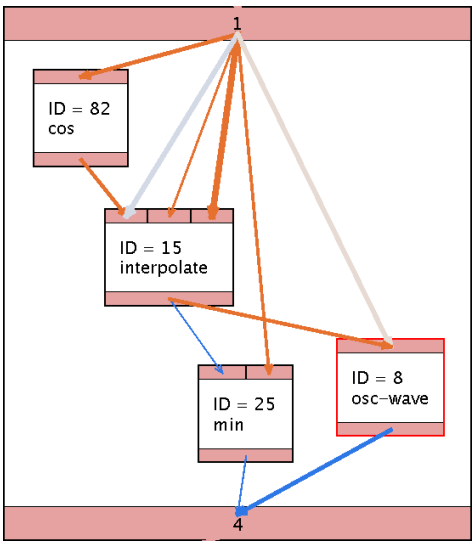
Experiments have been conducted using all three fitness functions: XOR and simple (SFA) and difficult (DFA) versions of function approximation problem.

4.3.3.1 Results and discussion

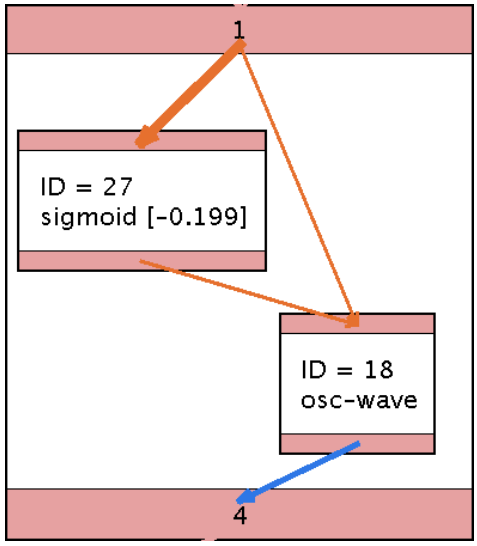
Results of the performed experiments are shown in Table 4.7 for the XOR problem, Table 4.8 for the SFA problem and Table 4.9 for the DFA problem. Columns of the tables have the same meaning as in the ablation experiments. Differences in means are statistically significant in all cases ($p<0.01$).

Larger set of transfer functions has increased NEAT performance in all three tested scenarios (XOR, SFA and DFA). The most significant increase in performance has been achieved in the DFA problem, where neural networks with only sigmoidal transfer function were unable to solve the problem at all, while other two configurations solved both problems in more than 95% of cases. Examples of solutions evolved for the DFA problem are shown on Figure 4.10.

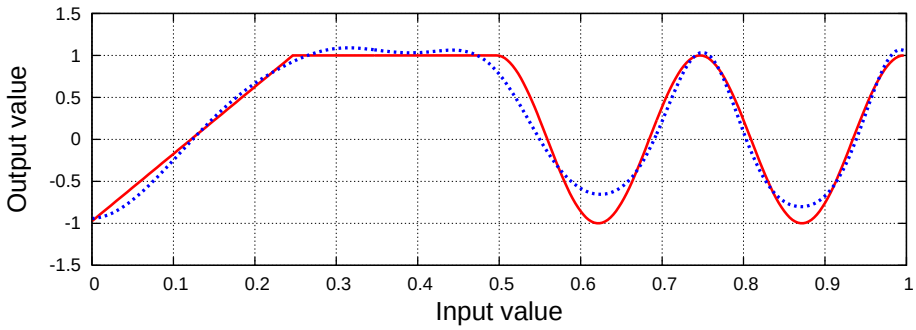
Results show, that NEAT is capable of efficiently utilizing a large set of transfer functions in the evolution of neural networks.



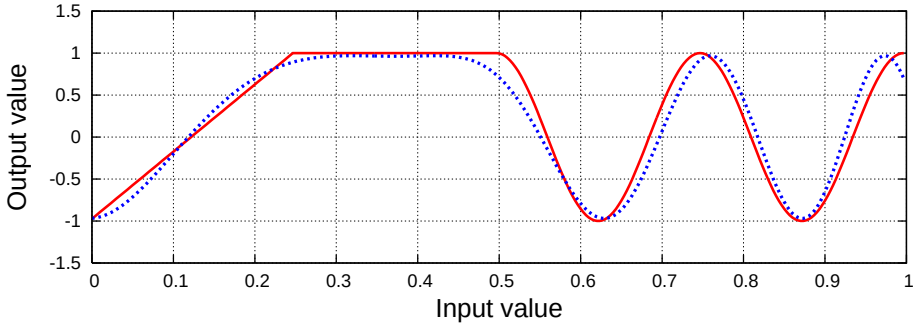
(a) Neural network evolved using configuration E3 (see Table 4.9) for solving the DFA problem.



(b) Neural network evolved using configuration E2 (see Table 4.9) for solving the DFA problem.



(c) Solution of the DFA problem by network shown in Figure 4.10(a).



(d) Solution of the DFA problem by network shown in Figure 4.10(b).

Figure 4.10: Examples of neural networks evolved for solving the DFA problem using configurations E3 (top left) and E2 (top right). Thickness of a neural connection represents the absolute value of weight of the connection, color represents the sign of the weight (red and blue colors represent positive and negative values, respectively). Number shown inside the sigmoidal neuron represents the bias of a neuron.

Id	Name	Failed	Evals-mean	Evals-std
D1	Sigmoidal TF	0%	21679.20	5578.11
D2	Oscillators	0%	8744.93	5418.25
D3	All TFs	0%	7206.41	2172.33

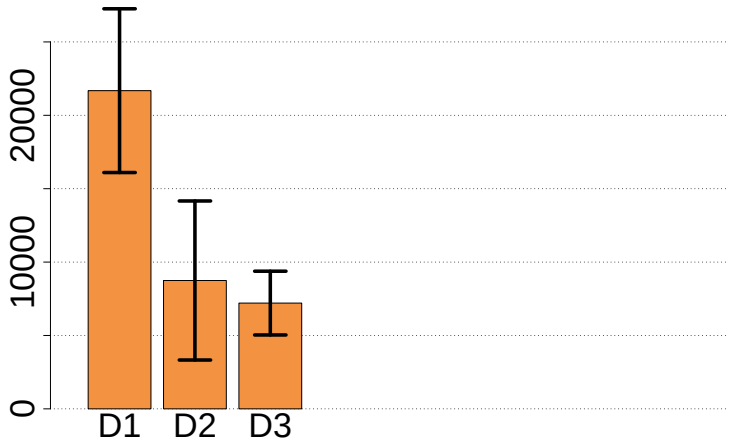


Table 4.8: Results of the tests with different sets of transfer functions with simpler function approximation (SFA) fitness function.

Id	Name	Failed	Evals-mean	Evals-std
E1	Sigmoidal TF	100%	60000.00	0.00
E2	Oscillators	5%	20503.13	16663.31
E3	All TFs	4%	27747.29	16538.44

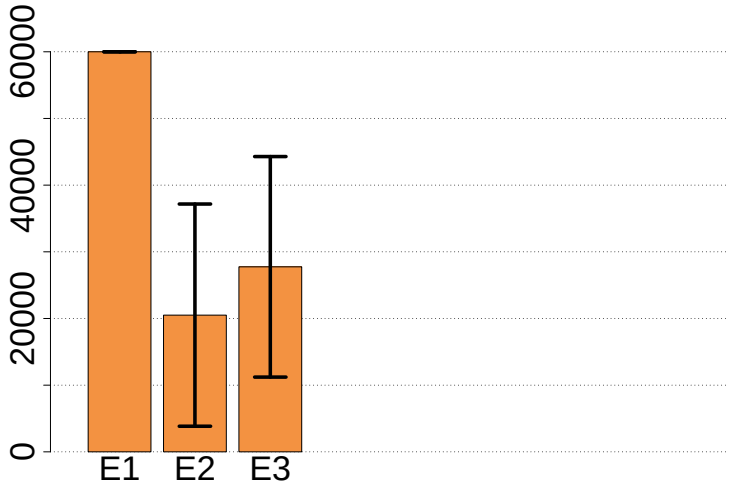


Table 4.9: Results of the tests with different sets of transfer functions with difficult function approximation (DFA) fitness function.

4.3.4 Summary

Experiments presented in this section have verified that the implementation on NEAT presented in this thesis is fully functional and it is in the same performance range than the original NEAT implementation (as presented in [34]). Ablation experiments have verified, that the speciation and mating in NEAT significantly improves the performance of the algorithm. However, starting minimally has been shown to decrease the performance (instead of increasing it, as shown in [36]). This might be caused by different algorithm used for random generation of neural networks and may also be influenced by different fitness functions used for experiments (double pole-balancing in [34] and XOR and SFA in this thesis).

Experiments with different sets of transfer function have shown, that NEAT can take advantage of new transfer functions and use them to build better solutions. Larger set of transfer function has increased performance of NEAT on all tested problems.

4.4 Testing creatures

This section describes experiments with evolving virtual creatures using a novel method introduced in Chapter 3. Experiments have been conducted to answer the following set of questions about the evolution of virtual creatures:

1. **Does hierarchical NEAT improve the performance of the search?** Previous section has confirmed, that speciation and sensible mating based on historical markings improve the performance of the evolution of neural networks significantly. However, NEAT is a method originally developed for the evolution of neural networks and it has never been applied to evolution of morphology before. Therefore, it is not clear whether the evolution of both morphology and the neural networks will be improved in the same way. To answer this question, experiments measuring the performance of hierarchical NEAT have been conducted on several fitness functions. These experiments are described in Section 4.4.2.

2. **Is *each component* of hierarchical NEAT improving the performance of the search?** It has been shown in Section 4.3.2, that speciation and sensible mating significantly improve the performance of the evolution of neural networks. Minimal start, however, performed much worse than expected and was outperformed by random initialization. Section 4.4.3 presents the same set of ablation experiments, but this time with the virtual creatures.

3. **Does the choice of transfer functions affect the performance of the search?** Oscillatory transfer functions (as opposed to, for example, sigmoidal transfer functions) generate changing signal even if their inputs do not change. This property might be very useful for generating movement of the creatures. Several other transfer functions are also present. Section 4.4.4 describes experiments designed to test the performance of the evolutionary search using creatures with different sets of transfer functions.

4. **Is the capability of central coordination beneficial to the evolution of the creatures?** Each creature is equipped with a single global controller (the brain), which provides the possibility of coordinated control. Individual body parts of the creature can also communicate with each other using inter-node neural connections. Section 4.4.5 attempts to answer the question whether the brain and inter-node neural connections increase the performance of the search.

Several fitness functions have been used to conduct experiments in this section. However, fitness function evaluation is a computationally expensive process. Computation of all experiments presented in this section would take 8 months and 12 days, if computed by a single computer (this is a lower bound, computed as a total CPU time used by all workers, not including idle time and server computation time). However, using parallel computation with 10 servers and 6 workers per server (total of 70 computers), all experiments have been computed in under a week.

Because of the high demand on the computational resources, most of the experiments presented in this chapter were carried out using a single fitness function. Fitness function for light following behavior has been chosen, because it is the only fitness

Parameter	Value
t	2/3
w_p	2
w_c	1
w_m	1
N_{max}	5
C_{max}	10

Table 4.10: Parameter settings for the compatibility distance measure for virtual creatures. Descriptions of individual parameters are provided in Section 3.4.2.

function, which explicitly requires organisms to use their sensory input and to evolve their neural network controllers. This makes the light following fitness function the best candidate for experiments with evolution of both the morphology and the neural networks of virtual creatures.

4.4.1 Setup of experiments

This subsection presents parameter settings of genetic algorithm and genetic operators used in experiments presented in this section (any changes from parameter settings presented in this section will be explicitly stated).

Virtual creatures are represented by a hierarchical structure, where neural networks are embedded inside morphological nodes. Each genetic operator processes the morphology level first, and then delegates processing of the neural network controllers to the corresponding neural network genetic operator. All morphological operators used in all experiments use default parameter settings, as presented in Section 2.2.3. Parameter settings for compatibility distance on the morphology level are shown in Table 4.10.

Setup of all neural network operators builds on the results obtained in experiments with neural network evolution. The same parameters were used for compatibility distance measure, mutation and random generation of neural networks as presented in Section 4.3.

Each configuration of the evolutionary algorithm has been tested at least 30 times. Each time, evolution has been run for 100 generations, after which evolution has been stopped and detailed statistics of the evolutionary run has been recorded. Parameters of the genetic algorithm are shown in Table 4.11.

In order to measure performance of individual configurations, *the winning fitness* is first defined for each fitness function. The goal of each experiment is to find the winning organism (i.e. *the winner*), which is the first organism with fitness value at least as large as the winning fitness. Number of fitness evaluations necessary to find the winner is used as the performance measure. Number of fitness evaluations also includes the number of failed validity tests, to handicap genetic operators which explore fitness landscape aggressively by generating large numbers of invalid creatures. Table 4.12 shows winning fitness values for all four fitness functions used for experiments in this section. Values have been chosen so that the large majority of the evolutionary runs is able to find the winner within the first n generations ($n = 30000$ has been used in

Parameter name	Value
Population size	300
Desired number of species	8
Interspecies mating prob.	0.1
Mutation only %	25%
Mating %	75%
Mutation after mating prob.	0.8
Maximum stagnation	15
Youth age threshold	10
c_{youth}	1.5
Initial c. d. threshold	0.2
Dynamic c. d. threshold	yes

Table 4.11: Parameter settings for the genetic algorithm for evolving virtual creatures. Descriptions of individual parameters are provided in Table 3.1 in Chapter 3.

Fitness function	Winning fitness
Light following	0.3
Walking	15
Jumping	0.45
Swimming	5

Table 4.12: Values of the winning fitness for individual fitness functions.

all experiments). In case when the search fails to find the winner during the first n evaluations, its performance is set to n.

All p-values have been computed using Welch two sample t-test for unequal variances. Significance level of 5% has been used.

4.4.2 Performance of Hierarchical NEAT

The goal of this section is to show whether the hierarchical NEAT algorithm proposed in this thesis improves the performance of the evolutionary search. Four sets of experiments have been performed, each set with one of the following fitness functions: light following, walking, jumping and swimming. For each fitness function, two configurations of genetic algorithm (GA) have been tested:

1. **Hierarchical NEAT.** Full hierarchical NEAT algorithm has been used. Both speciation and sensible mating have been enabled and the initial population has been generated randomly, using the default parameters of the creature generator.

2. **Standard GA with grafting and crossover.** Speciation has been disabled, and sensible mating based on historical markings has been replaced with grafting and crossover mating methods. Otherwise, all parameters have been configured in the same way as with the hierarchical NEAT.

Results of the performed experiments are presented in Table 4.13. Hierarchical NEAT outperforms standard GA with grafting and crossover in all tested scenarios.

Id	Behavior	Configuration	Evals-mean	Evals-std	Failed
A1	Following	Hierarchical NEAT	16704.53	6339.75	8%
A2	Following	Standard GA	27873.94	5002.57	72%
B1	Walking	Hierarchical NEAT	11033.09	3419.77	0%
B2	Walking	Standard GA	27230.64	4128.62	63%
C1	Jumping	Hierarchical NEAT	19221.51	7966.50	22%
C2	Jumping	Standard GA	29159.23	2068.91	80%
D1	Swimming	Hierarchical NEAT	10425.67	2707.13	0%
D2	Swimming	Standard GA	22438.95	5507.82	16%

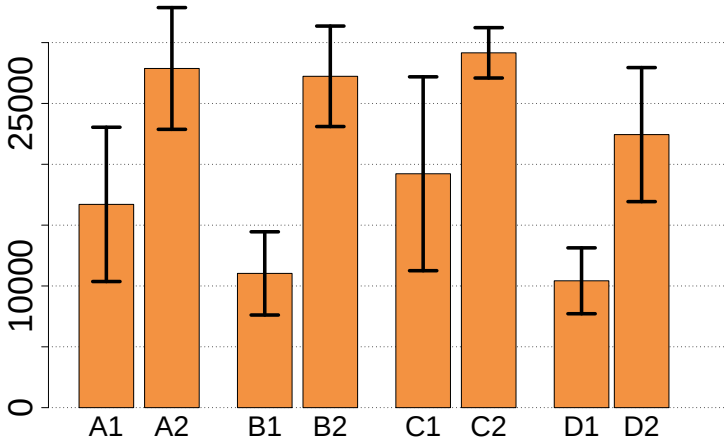


Table 4.13: Results of experiments measuring the performance of hierarchical NEAT algorithm.

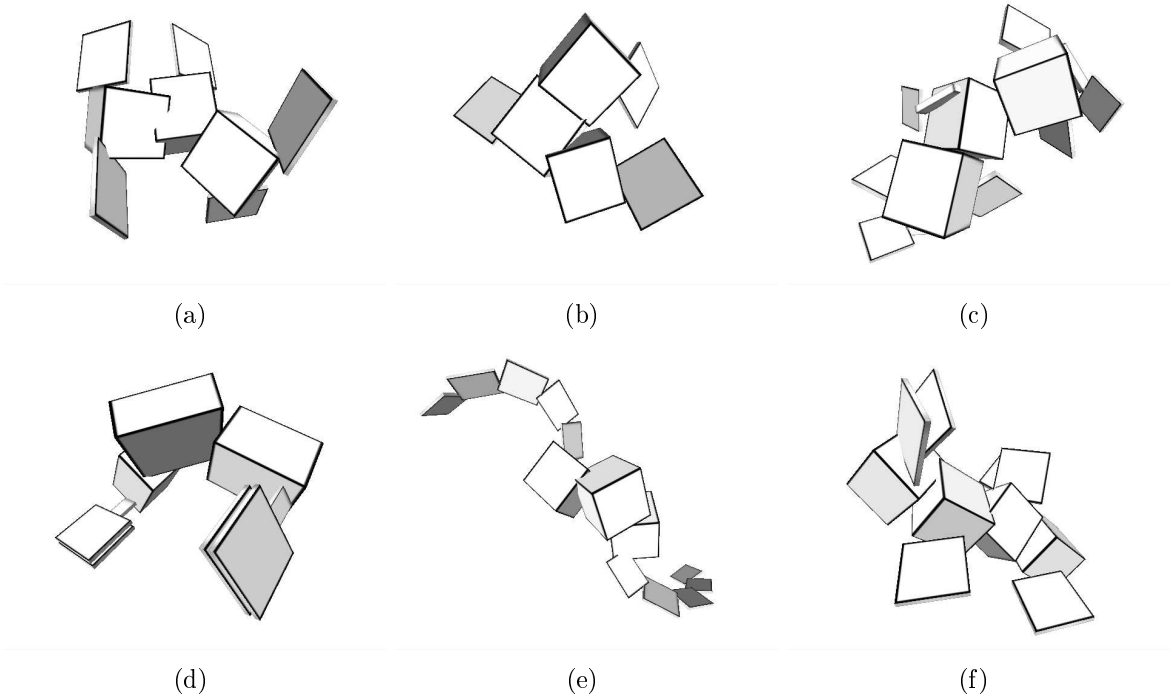


Figure 4.11: Examples of virtual creatures evolved for the task of following the light source.

The difference in mean number of evaluations is statistically significant for each fitness function ($p < 10^{-7}$). Hierarchical NEAT is 1.67 times faster than standard GA on the light following fitness function, 2.47 times faster on the walking fitness function, 1.52 times faster on the jumping fitness function and 2.15 times faster on the swimming fitness function. Moreover, standard GA fails much more frequently than hierarchical NEAT. Each failed attempt is included in number of evaluations as 30000, which is the lower bound on the actual number of evaluations, which would be needed to find the winner in cases when the search failed. Since standard GA fails much more frequently, increasing the duration of each experiment (i.e. maximum number of evaluations) would almost certainly result in even larger difference in performance between hierarchical NEAT and standard GA.

Several examples of creatures evolved using hierarchical NEAT algorithm are shown in Figure 4.11 (following the light source), Figure 4.12 (jumping), Figure 4.13 (swimming) and Figure 4.14 (walking). Movements of evolved creatures often resemble movements of natural animals. Figure 4.16 shows both the genotype and the phenotype of the light following creature. Gradual evolution of the snake-like creature is shown in Figure 4.15.

4.4.3 Ablation experiments

Ablation experiments are designed to show, how each component of hierarchical NEAT contributes to the overall performance of the algorithm. The following set of experi-

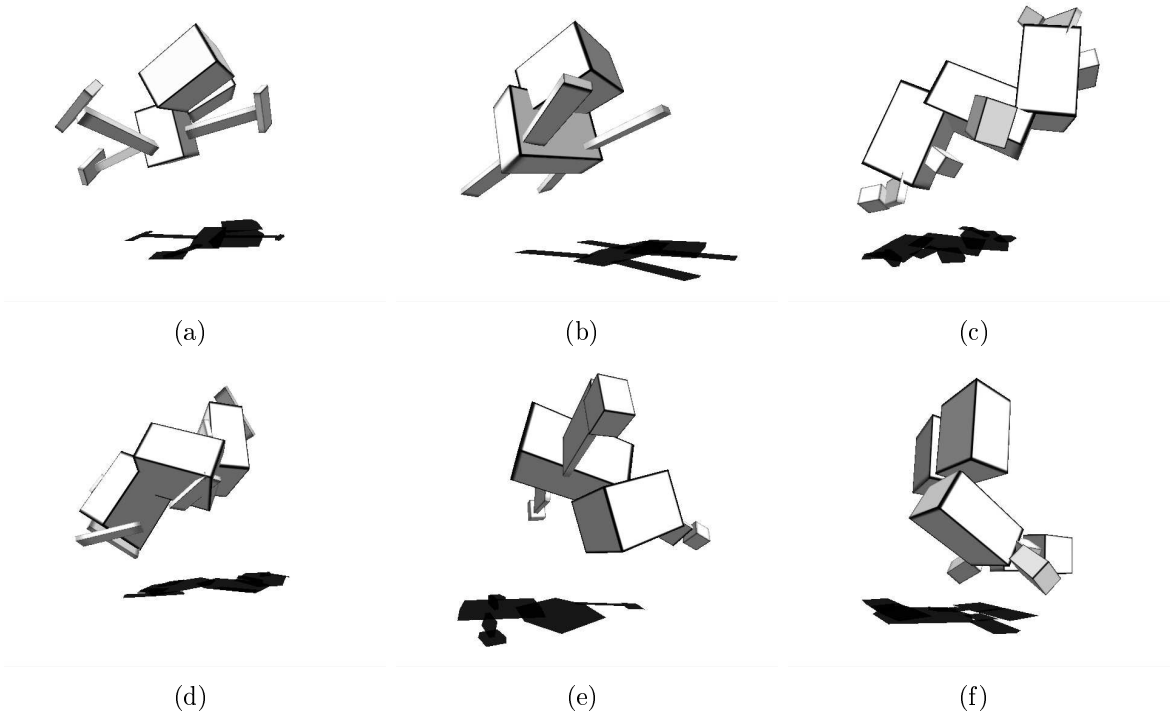


Figure 4.12: Examples of virtual creatures evolved for jumping.

ments has been conducted:

1. **Full Hierarchical NEAT.** Both speciation and sensible mating have been enabled and the initial population has been generated randomly, each new creature having 3 nodes and 3 connections.

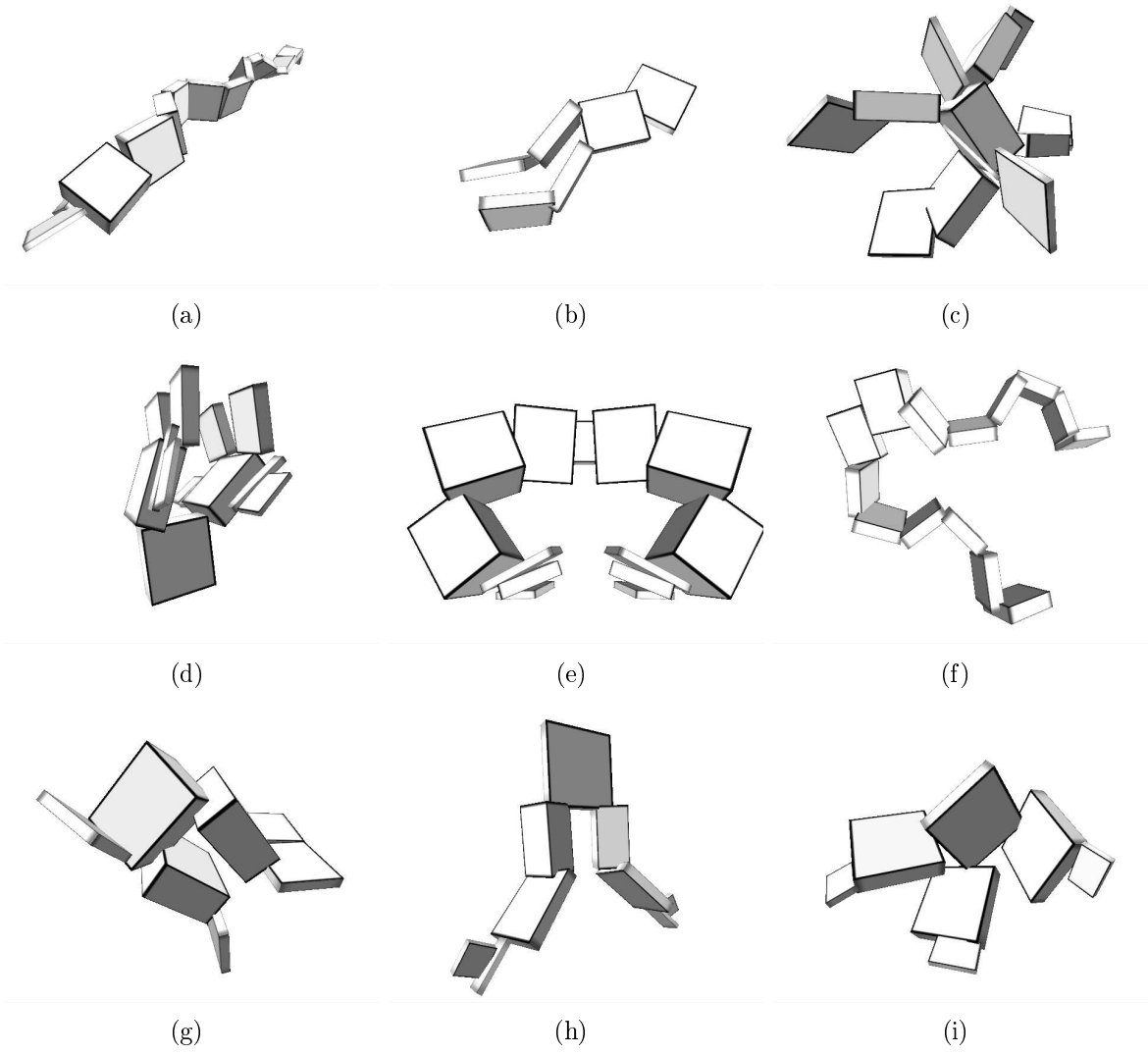
2. **Standard GA with grafting and crossover.** Speciation and sensible mating are disabled. Mating is performed using grafting and crossover (as described in Section 2.2.3). Otherwise, the configuration is the same as the configuration 1.

3. **Nonmating NEAT.** Sensible mating is disabled. Otherwise, the configuration is the same as the configuration 1.

4. **Non-speciated NEAT.** Speciation is disabled. Otherwise, the configuration is the same as the configuration 1.

5. **Uniform start with 1 node.** Initial population consists of copies of the same creature. The creature has minimal topology with only a single node and no connections. Otherwise, the configuration is the same as the configuration 1.

6. **Uniform start with 2 nodes.** Initial population consists of copies of the same creature. The creature has minimal topology with two morphological nodes connected by a single connection. Otherwise, the configuration is the same as the configuration 1.



(a) (b) (c)

(d) (e) (f)

(g) (h) (i)

Figure 4.13: Examples of virtual creatures evolved for swimming.

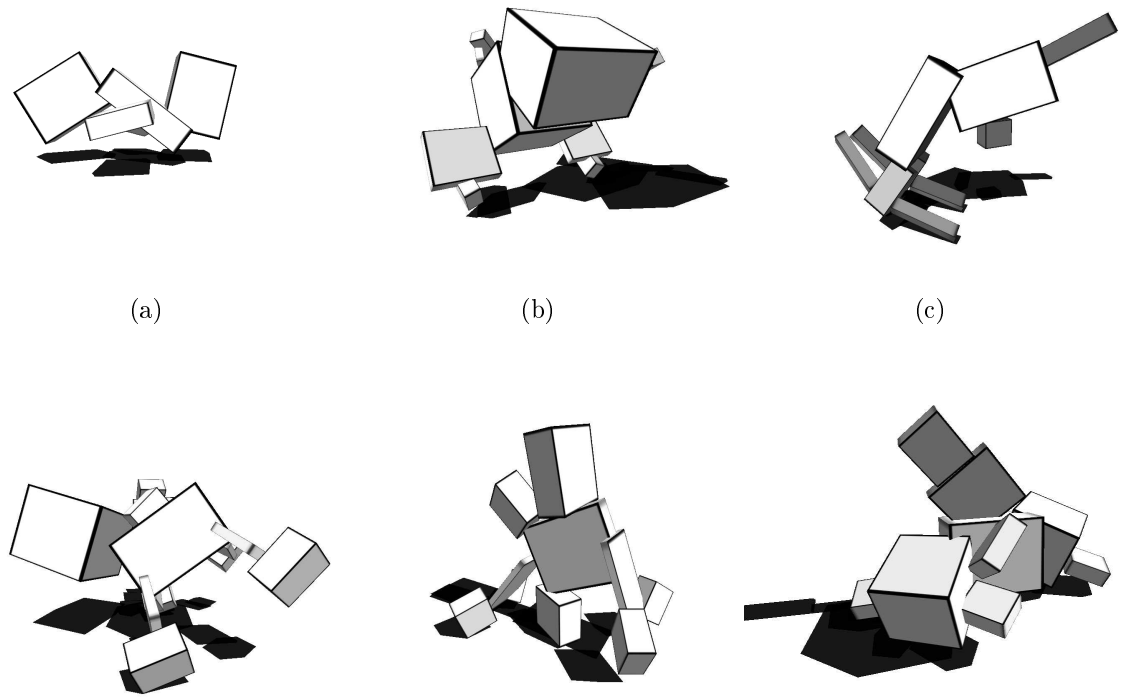


Figure 4.14: Examples of virtual creatures evolved for walking.

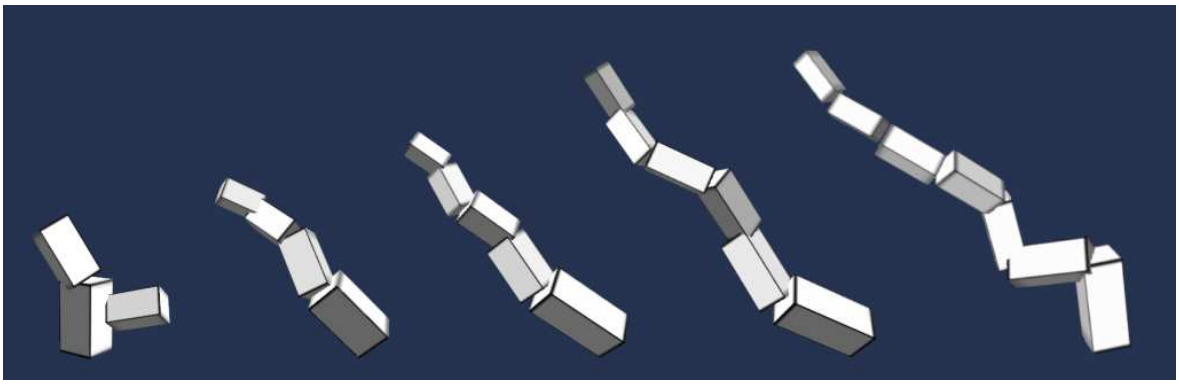
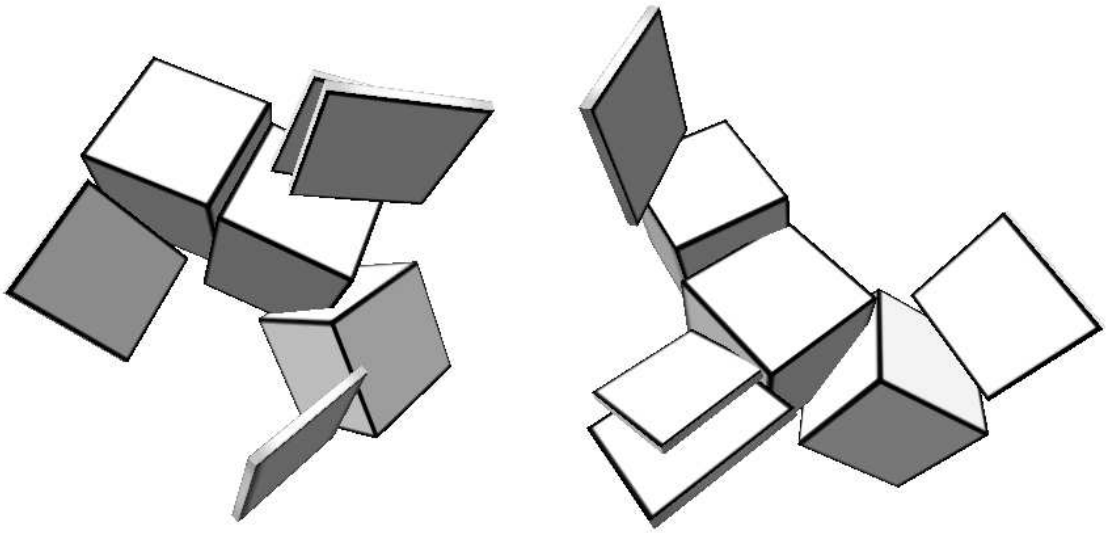
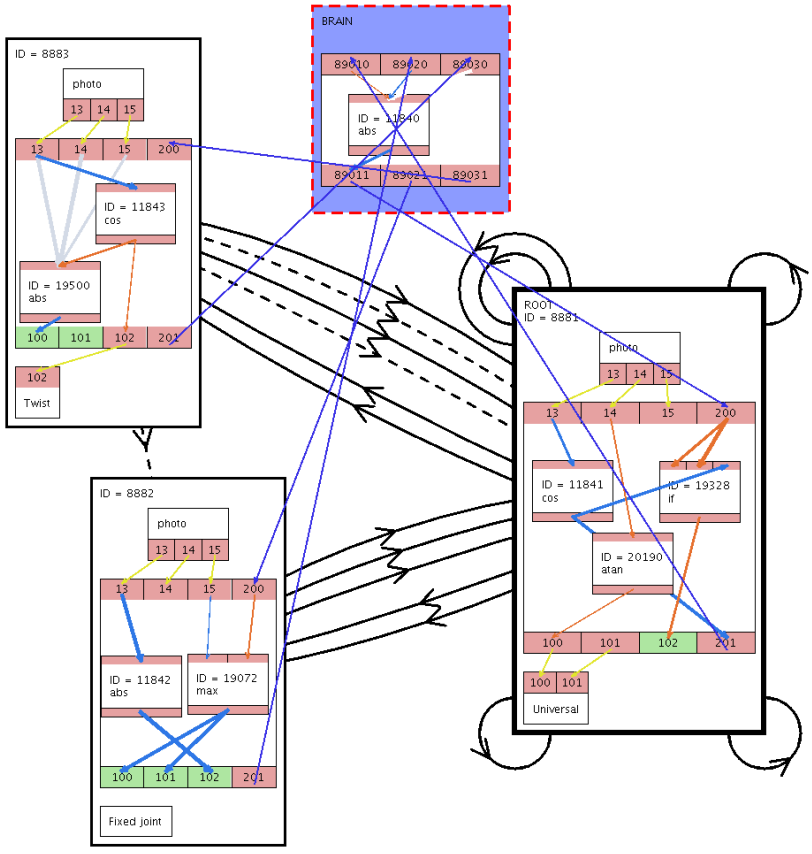


Figure 4.15: Evolution of a snake-like virtual creature.



(a) Two different pictures of the same phenotype following the light source.



(b) Genotype of the creature. For explanation of individual parts of the creature, see Section 2.2.2.

Figure 4.16: Phenotype (top) and genotype (bottom) of the creature evolved for following the light source.

7. **Random init. with 4 nodes.** Initial population is randomly generated. Each generated creature consists of 4 morphological nodes and 4 connections. Otherwise, the configuration is the same as the configuration 1.

8. **Random init. with 2 nodes.** Initial population is randomly generated. Each generated creature consists of 3 morphological nodes and 3 connections. Otherwise, the configuration is the same as the configuration 1.

Individual configurations are designed to test NEAT on morphology without one of its components. Configuration 1 is the full hierarchical NEAT algorithm, with all features enabled. Configuration 2 is the standard GA with grafting and crossover used for mating. Configurations 3 and 4 represent hierarchical NEAT with disabled sensible mating and speciation, respectively. Configurations 5-8 experiment with various method of creating initial population. These experiments are designed to show, what effect have various ways of initialization on the performance of the evolution.

4.4.3.1 Results and discussion

Results of the performed experiments are presented in Table 4.14. Experiments have been designed to show that each component of hierarchical NEAT algorithm increases the performance of the evolution of virtual creatures. This section provides an analysis of the results with respect to the individual components of the algorithm.

Experiments confirm the positive effect of sensible mating on the speed of the search. Experiments E1 and E3 show, that nonmating NEAT fails in 12% more cases than full NEAT. Full NEAT is also 1.34 times faster in finding the winner. Difference in results is statistically significant ($p < 0.0001$). The disrupting effect of grafting and crossover has also been confirmed. Evolution with sensible mating requires on average 298.2 failed validity tests per generation (approximately 0.99 failed tests per organism, including the mutation), while grafting and crossover require on average 1488.6 failed tests per generation (approximately 4.96 failed validity tests per organism including the mutation).

Experiments with various methods of generating the initial population have shown, that starting with small, randomly generated creatures increases the performance of the search. Size of the generated creatures greatly influences the outcome of the evolution. Evolution starting from a uniform population of creatures with only one morphological node (E5) has been unable to find the winner in any of the experiments. This is probably due to the fact, that the evolution has to spend several first generations finding an organism which consists of more than one body-part and is thus capable of at least a simple movement. In situation, when evolution starts with population of creatures with two nodes (experiments E6 and E8), random initialization is 1.46 times faster than uniform population ($p < 0.005$). Experiments with random initialization (E8 with 2 nodes per creature, E1 with 3 nodes per creature and E7 with 4 nodes per creature) show that the performance of the search decreases with increasing size of generated creatures ($p < 0.001$ in all cases). This fact supports the hypothesis that starting minimally helps the search by minimizing the dimensionality of the search space. In summary, it can be concluded that starting evolution with small randomly

Id	Configuration	Evals-mean	Evals-std	Failed
E1	Full hierarchical NEAT	16704.53	6339.75	8%
E2	Standard GA with G/C	27873.94	5002.57	72%
E3	Nonmating NEAT	22441.38	6146.31	20%
E4	Non-speciated NEAT	20456.40	6313.65	20%
E5	Uniform start NEAT with 1 node	30000.00	0	100%
E6	Uniform start NEAT with 2 nodes	16811.39	8190.81	21%
E7	Random init. NEAT with 4 nodes	29920.69	387.05	93%
E8	Random init. NEAT with 2 nodes	11503.77	5871.19	3 %

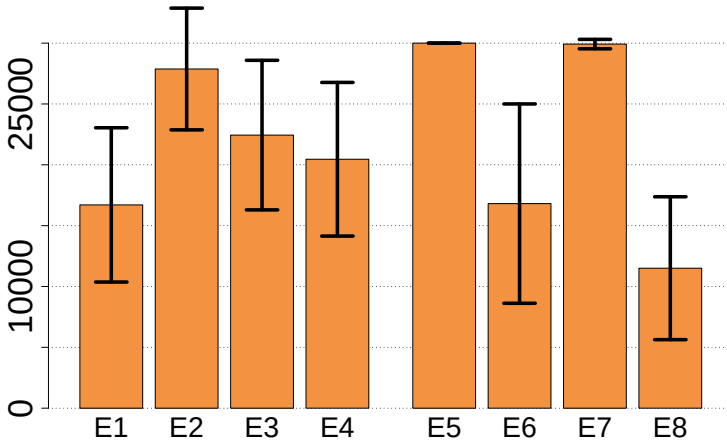


Table 4.14: Results of ablation experiments.

generated creatures is the more beneficent than either starting with uniform population, or starting with population of creatures with larger genomes.

The positive effect of speciation has also been confirmed. Results of experiments E4 (non-speciated hierarchical NEAT) and E1 (full version of the algorithm) show, that full NEAT is 1.22 times faster than non-speciated NEAT on the task of light-following ($p < 0.02$). The failure rate is also significantly higher in the non-speciated version of the algorithm.

Results of experiment E2 (standard GA with grafting and crossover) are also provided in Table 4.14 for comparison with ablated configurations. Results show, that standard GA performs significantly worse than both nonmating and non-speciated hierarchical NEAT.

4.4.4 Experiments with different sets of transfer functions

This section presents experiments comparing evolutions of virtual creatures with different sets of transfer functions. Experiments are designed to answer the question whether large number of transfer functions (as used by Sims [31] in his original proposal of virtual creatures) increases the performance of the evolution (by providing evolutionary "shortcuts"), or instead, retards it by enlarging the search space.

Id	Configuration	Evals-mean	Evals-std	Failed
F1	All TFs	16704.53	6339.75	8%
F2	Oscillators	20084.07	6746.83	13%
F3	Sigmoidal TF	18221.12	6093.09	7%

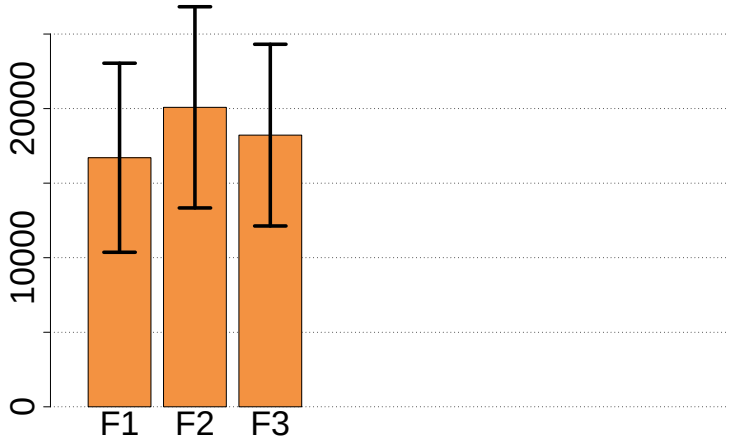


Table 4.15: Results of experiments with different sets of transfer functions.

Experiments presented in this section build on experiments with neural networks presented in Section 4.3.3. Experiments with neural networks have shown that using a variety of transfer functions significantly increases the performance of the evolution on the XOR problem and on the problem of function approximation.

Three transfer function (TF) configurations have been tested (configurations are the same as the configurations used in Section 4.3.3 for experiments with neural networks without morphology):

1. **Sigmoidal TF.** Only sigmoidal transfer function is enabled in all neural network controllers.

2. **Oscillators.** Besides sigmoidal transfer function, two types of oscillators are used: *wave-oscillator* and *saw-oscillator*.

3. **All TFs.** All 23 different transfer function are allowed to be used during this test. Transfer functions are described in Section 2.2.2.2.

All configurations use hierarchical NEAT algorithm for the evolution of the creatures. Light following fitness function is used for all experiments.

Results of the performed experiments are presented in Table 4.15. Overall, all configurations result in very similar performances. The only statistically significant difference in results is between oscillatory transfer functions (F2) and all transfer functions (F1). Configuration with oscillatory transfer functions is 1.2 times slower than configuration with all transfer functions ($p = 0.04$) and also fails more frequently.

While evolution with all possible transfer functions seems to slightly increase the performance of the evolution, the choice of the transfer functions does not seem to

Id	Configuration	Evals-mean	Evals-std	Failed
G1	With brain, with inter-conn.	16704.53	6339.75	8%
G2	Without brain, with inter-conn.	15769.15	3901.39	0%
G3	With brain, without inter-conn.	18743.47	7105.18	12%
G4	Without brain, without inter-conn.	18029.86	6429.15	8%

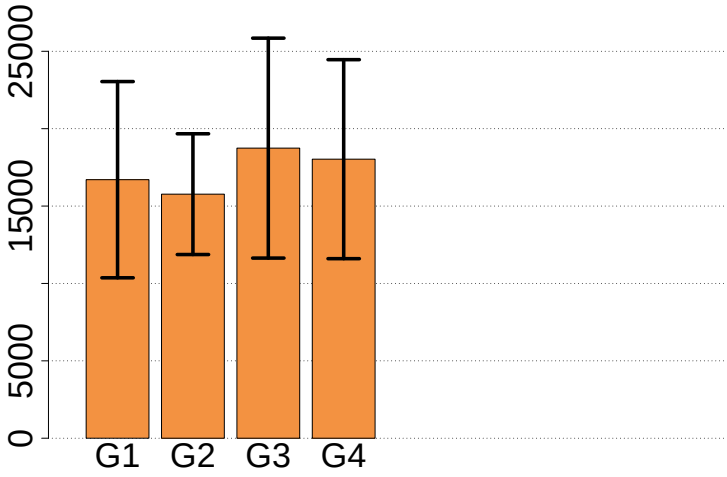


Table 4.16: Results of experiments with coordination.

have a large effect on the performance of the creatures. Preliminary analysis of several evolved creatures suggests, that creatures evolve as simple control as possible to solve the given task. For example, many light-following creatures simply apply a constant torque on one of the joint, creating a propeller-like structure, while using other body parts for steering. Body parts used for steering are also controlled by very simple controllers (often consisting of only direct neural connections between inputs and outputs). While complex transfer functions have been shown to significantly increase the performance on the problem of function approximation, they seem to be unnecessary for the task of light-following (evolution with only sigmoidal transfer function achieves comparable performance). An interesting area of future research is to propose a fitness function for the virtual creatures, which would allow creatures to utilize all complex transfer functions.

4.4.5 Experiments with centralized coordination

Experiments presented in this section are designed to show, whether the capability of central coordination is an advantage in the evolution of the creatures. All experiments have been conducted using the light following fitness function. In order to follow the light source successfully, creatures have to learn to adaptively change the direction of their movement, depending on the current position of the light source. This is the task, where central coordination of the movements could be beneficent.

Distributed control system of the creature can be coordinated using two different mechanisms. First mechanism is *the brain* of the creature. The brain is a global neural

network controller, which connects to local neural controllers in each body part using neural connections. Brain can thus coordinate individual local neural networks in different nodes. Second mechanism of coordination are inter-node neural connections. These connections connect neural network in those body-parts, which are directly connected by a physical joint. Neural signal can thus propagate through the body of the creature following its physical structure (similar to living organisms).

Four experiments have been conducted to test all possible combinations of enabling/disabling brain and inter-node neural connections. Results of all experiments are presented in Table 4.16.

The results show, that the absence of brain seems to increase the performance of the search, while the absence of inter-node neural connections seems to decrease the performance. However, the only statistically significant difference in results ($p < 0.04$) is between two complementary configurations: G2 (with the brain, without inter-node connections) and G3 (without the brain, with inter-node connections).

Overall, experiments have shown, surprisingly, that coordination is not a necessary component of creature control system. Successful following behavior can be evolved both with and without central coordination. Further analysis of the creatures could be done to investigate the exact participation of the brain and inter-node neural connection on the control of the creatures and to confirm the positive effect of inter-node neural connections and the negative effect of the presence of the brain. Experiments presented in this section can be used as a groundwork for further research in this area.

4.4.6 Summary

Experiments with evolving virtual creatures using hierarchical NEAT have shown, that it outperforms standard genetic algorithm (GA) with grafting and crossover mating methods on all tested fitness functions. Hierarchical NEAT found the solution on average 1.54 times faster than standard GA on the light following fitness function, 2.47 times faster on the walking fitness function, 1.52 times faster on the jumping fitness function and 2.15 times faster on the swimming fitness function.

A set of ablation experiments has also been performed. Results have confirmed, that each component of the algorithm (speciation, sensible mating and minimal start) is beneficial to its performance.

Results of experiments with different sets of transfer functions have shown, that using all 23 transfer functions slightly increases the performance of the search, compared to other transfer function configurations. Overall, however, the choice of the transfer functions does not seem to have a significant effect on the quality of evolved creatures.

Experiments with ablating coordination properties of virtual creatures suggest, that creatures without the central brain surprisingly perform better on the task of light following than creatures with the brain. On the other hand, absence of inter-node neural connections seems to have negative effect on the performance. Further research has to be done to investigate the exact impact of coordination in the evolution of the creatures.

Chapter 5

Future works

This chapter provides several possible directions for future research in the area of evolving virtual creatures. The first section is focused on the representation of the creatures. Representation used in this thesis is adopted from the work of Sims [31]. However, in recent years, several promising biologically inspired representations have been proposed. These new representations have a potential of allowing evolution to discover even more complex creatures. The second section presents a collection of ideas and preliminary data collected during the work on this thesis, which can be used to further increase performance of the evolution of virtual creatures. Next section focuses on control system of the creatures, and points out several directions of future research. Final section summarizes possible enhancements of the morphology of the virtual creatures.

5.1 Representation

Creatures presented in this thesis use an indirect encoding, based on recursive unfolding of the directed genotype graph. While this representation allows compact encoding of complicated structures, it is still far from reaching the complexity and evolvability of natural organisms. However, much research is being done in the field of generative and developmental indirect representations with the ultimate goal of achieving representations as powerful as those found in the nature. Summary of current methods and a unified perspective on indirect encodings is provided in [37]. Most of the indirect encoding methods are, however, introduced in the field of neural networks. A very promising area of the future research is to try to transfer some of the successful indirect encodings from the area of neural networks to the area of virtual creatures (as has been done with NEAT algorithm in this thesis). An example of one such recent promising method is indirect encoding based on genetic regulatory networks (GRN) [27].

Another area of future research is to compare evolution of direct encoding of virtual creatures (where genotype is identical to phenotype) and indirect encoding as used in this thesis. The expected outcome of this comparison is that symmetry and reuse of structure allowed by indirect encodings would dramatically enhance the performance of the search, outperforming direct encoding by a large margin. Explicit comparison could be conducted to confirm this hypothesis.

5.2 Increasing performance

Performance of the evolution of virtual creatures is mainly limited by two factors: computationally expensive fitness function evaluation and high number of invalid creatures generated by genetic operators.

Single fitness function evaluation takes typically several seconds of CPU time to compute. This means, that the evolution running for 100 generations with population size of 300 would take more than 8 hours to finish, assuming fitness function evaluation takes one second per organism. Large number of tests presented in this thesis (roughly 30 configurations of genetic algorithm, each running at least 30 times) were computed using the parallel computation engine, which allowed to compute all experiments in under a week using 70 computers. If computed by a single computer, fitness function evaluations alone would take 8 months and 12 days to compute (this does not include computation of genetic operators and idle time). Parallel computation thus makes large-scale experiments possible. However, parallel computation could be speeded up even more by reducing the network traffic between the workers and the server. Currently, each organisms is sent from the server to the worker to be evaluated by the fitness function and the resulting fitness value is sent back to the server. Server has to manage all traffic with all workers. This is not such a problem when individual tasks take a long time to compute, but with short tasks, server might not be able to serve tasks fast enough to the workers. In the case of short tasks, increasing number of workers increases load on the server to the point, when workers wait for tasks too long to be effective. The exact measurement of this effect is shown in Figure 5.2. Ratio of CPU time and real-time is shown for different numbers of workers. The figure shows, that for evolutionary computations (green and blue lines), adding more than about 12 workers does not increase the performance of the computational layer very much. Example task for computing the approximation of the value of π with long computation time (red line) does not suffer from this effect, because long computation times allow the server to be responsive even with high number of workers. To eliminate slow responsiveness, experiments in this thesis were conducted using 10 servers with 6 workers per server.

One direction of improving responsiveness of the server is by optimizing the implementation (e.g. use multi threaded management of individual workers, optimize synchronization among threads). Another approach to improve performance is to use island model of parallel genetic algorithm. Separate evolutions would be run on individual workers (representing islands), with occasional exchange of the best organisms among islands.

Another possible bottleneck in the performance of evolution is the high number of invalid creatures generated by genetic operators. In a highly constrained search space, it is often a difficult problem to generate valid solutions. Four different approaches can be taken:

1. Genetic operators can be designed very carefully in such way, that they generate only valid organisms.

2. A repair phase can be added, which "repairs" invalid organisms.

Figure 5.1: Distributed computation performance analysis. Ratio of total CPU time and real time for different numbers of workers is shown for two problems: evolutionary computation (green and blue lines) and approximation of π (red line).

3. Invalid organisms can be detected during fitness evaluation and assigned zero fitness.

4. Invalid organisms can be detected early and the generating procedure can be repeated until a valid organism is found.

Approaches 1, 3 and 4 are used for the virtual creatures in this thesis. The best approach with respect to the performance of the evolution is approach 1, where invalid creatures are avoided altogether. Approach 4 is implemented as a validity testing procedure, and is described in detail in Section 2.2.5. Genetic operators often generate invalid genotypes, which are then discarded by the validity testing procedure. Generation of the genotype is then repeated until a valid genotype is found. This process retards computation of genetic operators and therefore, it would be convenient to solve some of the prominent causes of rejections using approach 1. Table 5.1 and Table 5.2 show analysis of validity testing rejections for hierarchical NEAT algorithm and grafting and crossover algorithm (without NEAT), respectively, using the light following fitness function. Validity testing in hierarchical NEAT rejects an organisms on average 0.75 times per genetic operator run, while grafting and crossover (without NEAT) method rejects on average 5.93 times per genetic operator run.

The most common reason for rejection is self-penetration. Self-penetration rejection ratio could be lowered by allowing creatures to interpenetrate by some small amount. Physical simulation would be then run to "push apart" interpenetrating body parts of the creatures.

Some preliminary tests have also been done to eliminate rejections caused by the too small body-part volumes. Transcription procedure (as described in Section 2.2.1)

Reason for rejection	Percentage	Number of rej.
Self-penetrating body-parts	**50.05%**	7078
Small number of body-parts	**15.69%**	2219
Small size of body-part	**13.76%**	1946
Small volume of body-part	**20.50%**	2899
Total	**100.00%**	14142

Table 5.1: Distribution of reasons for refusal of invalid creatures during hierarchical NEAT evolution with light-following fitness function.

Reason for rejection	Percentage	Number of rej.
Self-penetrating body-parts	**49.25%**	3933
Small number of body-parts	**12.60%**	1006
Small size of body-part	**20.86%**	1666
Small volume of body-part	**17.29%**	1381
Total	**100.00%**	7986

Table 5.2: Distribution of reasons for refusal of invalid creatures generated during genetic algorithm without NEAT, using grafting and crossover mating methods, with light-following fitness function.

has been modified to terminate recursive traversal not only when the recursive limit is reached, but also when the volume of the body-part falls under the specified minimal value. Preliminary experiments with this modified method are encouraging, because the number of minimal volume rejections has been decreased to zero. However, an unwanted side-effect of this modification is excessive bloating of creature genotypes. Bloating occurs, because new nodes added to the graph are often not expressed at all because of their small size. Evolved creature thus often contains many unused structure elements, which increases the size of the search space, retarding the evolution.

Another way of increasing both performance and stability of the fitness evaluation would be to use different physics engine for simulation of a physical world. ODE engine is robust and stable, however, it occasionally suffers from inaccuracies. Several alternatives exist for ODE engine (e.g. Bullet engine). However, further investigation is needed to compare different physical engines and to choose which one is most appropriate for evolution of virtual creatures.

5.3 Control system

Creature control system is a complex system with many features (the brain, large set of transfer functions, neural connections between parent-child body parts). It is not clear, whether all features of the control system are necessary for successful evolution and how much individual features contribute to the performance of the evolutionary search. This thesis provides several experiments to answer some of these question (coordinated control in Section 4.4.5 and transfer functions in Section 4.4.4). It has been shown, surprisingly, that for the task of light following, coordinated control has little or no use for the creatures. One area for future research is to conduct experiments with coordi-

nation using different fitness functions, or to propose a fitness function, which would *require* the use of the coordinated control system. Coordination and the use of the central "brain" could allow more sophisticated behavior to arise during the evolution. Experiments comparing different transfer function have been also conducted, but have been inconclusive. This may be caused by the design of the fitness function, which does not require complex neural controllers to arise. However, large set of transfer functions has been shown to improve the performance of the evolution of controllers alone (see Section 4.3.3 for details). Further analysis of the contributions of individual transfer functions to the success of the search is another interesting future area of research.

NEAT method has been shown to be very powerful in evolving neural networks. However, in the evolution of virtual creatures, none of the fitness functions used in this thesis seems to provide enough challenge to drive the evolution of complex neural networks. Creatures seem to discover a very simple functional neural network early in the evolution (with either none or just one hidden neuron) and continue to optimize their morphology through the rest of the evolution without further enhancing their control system. For example, popular strategy of movement is to take advantage of a very simple oscillatory signal (created either directly by one of the oscillatory transfer function or creating a simple "pendulum" by applying force opposite and proportional to the current joint angle) or simply outputting a constant value, which then fuels the propeller. Even in the light following fitness function, creatures do not seem to require complex neural networks to sufficiently navigate towards the light source. Therefore, a very interesting challenge for the future is to devise a fitness function, which would force creatures to evolve complex neural networks and complex behavior. One of the promising directions is co-evolution, where evolutionary "arms race" between species might provide enough challenge to create more complex neural networks. Different species of organisms could compete for a box, as in the original work of Sims [31] or, for example, hunter could hunt the pray. Another approach would be to use a variation of double pole-balancing problem, which has been shown to be sufficiently difficult to solve. Pole-balancing is also a problem, which had been used for a long time for reinforcement learning benchmarking. However, designing pole-balancing experiment for evolution of both morphology and control system is an interesting open problem (standard pole-balancing problem uses fixed morphology).

Creatures are currently controlled by neural networks, but other controllers could be used as well. Moreover, the design of the creatures allows the co-existence of several controller types in a single creature. Another interesting area of future research would be to experiment with other types of controllers (for example, genetic programs) and compare the performance of the search using different controller types.

5.4 Morphology

Creature morphology used in this thesis is inspired by the work of Sims [31]. An interesting area of future research lies in extending morphology of the creatures in various ways. Some preliminary experiments have already been done with other geometric

shapes used for body parts (namely spheres and capped cylinders). The range of geometric shapes used for body-parts is limited only by the capabilities of the physics engine. An interesting possibility is to use arbitrary triangle meshes for body-parts of the creatures, incorporating the shape of the triangle mesh into the genome. This would allow the evolution to optimize the exact shape of individual body parts of the creatures.

Several works in the past have successfully transfered evolved morphology to the real-world (see Section 2.1.2 for survey of works). Another future area of research is to build on these works and attempt to manufacture virtual creatures evolved in this thesis. This would, however, require several changes in the creature morphology. Morphology of the creatures would have to be constrained in such ways, that evolved creatures would be easier to manufacture in the real world. For example, constraint could be imposed on the minimal size of the body-part of the creature, so that the servo could fit inside. Only physical joints with at most one degree of freedom would be used to be able to control each joint using a single servo. Effector semantics would also have to be changed from specifying the *torque* applied to the joint to specifying directly the *angle* of the joint.

Chapter 6

Conclusion

Automatic design and manufacture of autonomous robotic organisms is gradually being transferred from the area of science fiction to the present-day science. Work in this area was pioneered by Karl Sims, who has successfully evolved virtual creatures with movements strikingly similar to those of the real-world organisms. Recently, several other works have build on his work and some have even accomplished to manufacture evolved creatures in the real world.

The main contribution of this thesis is the proposal of a novel algorithm for evolving virtual creatures similar to those proposed by Sims. The proposed algorithm is inspired by NEAT – an algorithm for evolving complex neural networks. Large-scale experiments have been conducted to measure various properties of the algorithm. Results of experiments show, that the algorithm significantly increases the performance of the evolution on the tasks of swimming, walking, jumping and light-following.

Several sets of experiments are presented to test individual components of the algorithm. First, speciation algorithm has been tested on the evolution of real-numbers coded as binary strings on the task of maximizing the value of the given function. Experiments in this simple scenario have confirmed the expected role of the speciation in the evolution (reproducing the results obtained by Goldberg in [10]).

Second, experiments with neural networks have been conducted to test the properties of NEAT algorithm without the interference of morphology. Results have confirmed, that the implementation presented in this thesis is capable of evolving complex neural networks, achieving performance comparable (or better) to the original implementation of the NEAT algorithm (as presented in [34]). Several ablation experiments have also been performed to confirm that each component of NEAT algorithm is necessary for its performance. Another set of experiments has shown, that extended set of neuron transfer functions significantly increases the performance of NEAT algorithm on both XOR and function approximation problems.

Finally, several experiments have been conducted to test the performance of the proposed algorithm on the evolution of virtual creatures. Experiments have shown, that the algorithm significantly increases the performance of the evolution. Set of ablation experiments has also been conducted to confirm, that each component of the algorithm is beneficial to the evolution of the creatures. Another set of experiments

has shown, that the coordination among body-parts is not necessary for the evolution of successful light-following strategies (contrary to the intuition). Final set of experiments has shown, that the quality of evolved creatures is slightly increased when using all possible transfer functions. Overall, however, the choice of the neuron transfer functions used in the control system of the creatures does not seem to have a large impact on the performance of evolved creatures.

The final part of the study proposes several further improvements in the performance of the evolution of virtual creatures and also suggests various enhancements of control system and morphology of the creatures.

In summary, the goals of this thesis have been successfully fulfilled. The study contributes to the field of evolutionary robotics by proposing a novel effective method of evolving virtual creatures. Achieved results can be used as a groundwork for the future research in this area.

Appendix A

Attached CD contents

CD attached to this thesis contains movies of selected virtual creatures evolved using the algorithm proposed in this thesis.

The following movies are included on the CD:

- `walking.mpg` Virtual creatures evolved for walking.
- `swimming.mpg` Virtual creatures evolved for swimming.
- `jumping.mpg` Virtual creatures evolved for jumping.
- `following.mpg` Virtual creatures evolved for following the light source.

All movies can be played using the movie player included on the CD, on any Microsoft Windows operating system. To play movies using this movie player, run the file `play.bat`, which runs all movies one after another.

ERO framework is also included on the CD. ERO can be used for real-time simulation of the evolved creatures, which are also included in the form of XML files. For further instructions on how to run the simulation in ERO, please refer to file `ERO-howto.txt`.

Bibliography

[1] James E. Baker. Reducing bias and inefficiency in the selection algorithm. In *Proceedings of the Second International Conference on Genetic Algorithms on Genetic algorithms and their application*, pages 14–21, Mahwah, NJ, USA, 1987. Lawrence Erlbaum Associates, Inc.

[2] R. Beale and T. Jackson. *Neural Computing: An Introduction*. IOP Publishing, Bristol and Philadelphia, 1990.

[3] Ch. M. Bishop. *Neural Networks for Pattern Recognition*. Clarendon Press, Oxford, 1995.

[4] Lipson H. Bongard J., Zykov V. Resilient machines through continuous self-modeling. *Science*, 314(5802):1118–1121, 2006.

[5] Rodney A Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, 2(1):14–23, March 1986.

[6] David B. D'Ambrosio and Kenneth O. Stanley. A novel generative encoding for exploiting neural network sensor and output geometry. In *GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 974–981, New York, NY, USA, 2007. ACM Press.

[7] R. Fikes and N. Nilsson. Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208, 1971.

[8] Jason Gauci and Kenneth Stanley. Generating large-scale neural networks through discovering geometric regularities. In *GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 997–1004, New York, NY, USA, 2007. ACM Press.

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

[10] 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.

[11] Faustino J. Gomez and Risto Miikkulainen. Solving non-markovian control tasks with neuro-evolution. In *IJCAI*, pages 1356–1361, 1999.

[12] 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.

[13] Gregory S. Hornby, Hod Lipson, and Jordan B. Pollack. Evolution of generative design systems for modular physical robots. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2001.

[14] Gregory S. Hornby and Jordan B. Pollack. Body-brain co-evolution using L-systems as a generative encoding. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, pages 868–875, San Francisco, California, USA, 7-11 2001. Morgan Kaufmann.

[15] Gregory S. Hornby and Jordan B. Pollack. Evolving l-systems to generate virtual creatures. In *Computers and Graphics*, pages 1041–1048, 2001.

[16] Gallant S. I. *Neural Network Learning.* MIT Press, 1993.

[17] Maciej Komosinski. The world of framsticks: Simulation, evolution, interaction. In *VW '00: Proceedings of the Second International Conference on Virtual Worlds*, pages 214–224, London, UK, 2000. Springer-Verlag.

[18] Maciej Komosinski and Szymon Ulatowski. Framsticks: Towards a simulation of a nature-like world, creatures and evolution. In *ECAL '99: Proceedings of the 5th European Conference on Advances in Artificial Life*, pages 261–265, London, UK, 1999. Springer-Verlag.

[19] John R. Koza. *Genetic programming: on the programming of computers by means of natural selection.* MIT Press, Cambridge, MA, USA, 1992.

[20] 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.

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

[22] H. Lipson and J. Pollack. Evolving physical creatures. In *Artificial Life VII.* MIT Press, 2000.

[23] H. Lipson and J. B. Pollack. Automatic design and manufacture of robotic lifeforms. *Nature*, 406:974–978, 2000.

[24] T. Miconi and A. Channon. An improved system for artificial creatures evolution. In *Proceedings of the 10th conference on the simulation and synthesis of living systems (ALIFE X)*, Bloomington, Indiana, USA, 2006. MIT Press.

[25] David E. Moriarty and Risto Mikkulainen. Efficient reinforcement learning through symbiotic evolution. *Mach. Learn.*, 22(1-3):11–32, 1996.

[26] T. S. Ray. Aesthetically evolved virtual pets. In *Artificial Life 7 Workshop Procs*, pages 158–161, 2000.

[27] 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 Press.

[28] N. Saravanan and David B. Fogel. Evolving neural control systems. *IEEE Expert: Intelligent Systems and Their Applications*, 10(3):23–27, 1995.

[29] Yoon-Sik Shim and Chang-Hun Kim. Generating flying creatures using body-brain co-evolution. In *SCA '03: Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 276–285, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.

[30] Karl Sims. Evolving 3d morphology and behavior by competition. 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.

[31] 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.

[32] Russel Smith. ODE manual. Available at http://www.ode.org.

[33] Lee Spector, Jon Klein, and Mark Feinstein. Division blocks and the open-ended evolution of development, form, and behavior. In *GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 316–323, New York, NY, USA, 2007. ACM Press.

[34] 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.

[35] Kenneth O. Stanley, Bobby D. Bryant, and Risto Miikkulainen. Real-time evolution in the nero video game. In *CIG*, 2005.

[36] 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.

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

[38] Matthew E. Taylor, Shimon Whiteson, and Peter Stone. Comparing evolutionary and temporal difference methods in a reinforcement learning domain. In *GECCO '06: Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pages 1321–1328, New York, NY, USA, 2006. ACM Press.

[39] Shimon Whiteson and Peter Stone. Evolutionary function approximation for reinforcement learning. *J. Mach. Learn. Res.*, 7:877–917, 2006.

[40] Shimon Whiteson, Peter Stone, Kenneth O. Stanley, Risto Miikkulainen, and Nate Kohl. Automatic feature selection in neuroevolution. In *GECCO '05: Proceedings of the 2005 conference on Genetic and evolutionary computation*, pages 1225–1232, New York, NY, USA, 2005. ACM Press.

[41] Darrell Whitley, Stephen Dominic, Rajarshi Das, and Charles W. Anderson. Genetic reinforcement learning for neurocontrol problems. *Mach. Learn.*, 13(2-3):259–284, 1993.

[42] A. Wieland. Evolving neural network controllers for unstable systems. In *Proceedings of the International Joint Conference on Neural Networks*, pages 667–673, Piscataway, NJ, 1991. IEEE Press.