

CZECH TECHNICAL UNIVERSITY IN PRAGUE  
FACULTY OF ELECTRICAL ENGINEERING  
DEPARTMENT OF CYBERNETICS



## DIPLOMA THESIS

Use of HyperNEAT Encoding  
for Hybrid Artificial Neural Networks

Prague, 2015

Pavol Sekereš

## Declaration

I declare that I have written my Diploma Thesis myself and used only the sources (literature, project, SW etc.) listed in the enclosed bibliography and the Appendixes.

In Prague 11.5.2015 Yokaris  
signature

## **Acknowledgement**

I would like to thank Ing. Jaroslav Vítků for his time, significant help and guidance. In addition I would like to thank Prof. Assoc. Pavel Nahodil for help and experienced advices.

# Abstrakt

V posledních letech byl zaznamenán významný posun ve vývoji hybridních modulárních systémů, s cílem poskytnout elegantní řešení pro rozličné problémy z oblasti umělé inteligence. Tyto hybridní umělé neuronové sítě svou využitelností předčí umělé neuronové sítě v mnoha oblastech výzkumu umělého života, proto jsou v posledních letech předmětem studií vědců z této domény. K tomu, aby hybridní systém fungoval správně, je potřeba nalézt vhodnou a pokud možno optimální topologii propojení uzlů v systému. Na optimalizaci topologie umělých neuronových sítí existuje nesčetný počet optimalizačních algoritmů, ale jen část z nich byla navržena tak, aby fungovala i pro hybridní neuronové sítě. Analytická řešení jsou pro optimalizaci výpočetně náročná, a proto se vědci inspiroují přírodou a navrhují evoluční optimalizační algoritmy. Algoritmus HyperNEAT ( Hypercube-based Neuro-evolution of Augmented Topologies ) představuje jeden z nejnovějších algoritmů v oblasti neuro-evoluce založené na nepřímém kódování jedince v populaci.

Cílem této diplomové práce je navrhnout a implementovat rozšíření algoritmu HyperNEAT pro hybridní umělé neuronové systémy. Algoritmus bude implementován tím způsobem, aby jej bylo možné propojit s frameworkem hybridních umělých neuronových sítí, který navrhl a implementoval vedoucí této diplomové práce Ing. Jaroslav Vítků v spolupráci s doc. Pavlem Nahodilem z katedry kybernetiky Fakulty Elektrotechnické ČVUT v Praze. Implementovaný algoritmus bude ale zároveň plně funkční pro libovolnou hybridní umělou neuronovou síť s příslušnou evaluační funkcí definující optimální kvalitu zapojení dané hybridní neuronové sítě v konkrétním prostředí.

# Abstract

In the recent years there has been significant upturn in the development of hybrid modular systems, with the emphasis on solving various problems in the Artificial Intelligence domain. These hybrid neural network systems outperform artificial neural networks in distinct areas of the computer science research. In order for these systems to function properly, it is important to interconnect the system's nodes in valid, or in the best case optimal topology. Large amount of algorithms for optimization of the artificial neural networks exist, but not many operate, or have been modified to operate on these hybrid modular systems. The analytical solutions, are computationally complex and therefore scientists design and implement evolutionary optimization algorithms to achieve satisfactory results. The novel algorithm, representing the state of the art of the indirect encoding-based neuro-evolutionary algorithms is HyperNEAT algorithm.

The aim of this thesis is to design and implement extension of the HyperNEAT algorithm for evolution of the hybrid artificial neural network systems. The algorithm will be mainly designed to serve the systems developed using the framework of hybrid neural network systems proposed by my thesis supervisor, Ing. Jaroslav Vitku in cooperation with Prof. Assoc. Pavel Nahodil from the Faculty of Electrical Engineering of CTU in Prague. The algorithm will also operate with arbitrary hybrid modular system with respective evaluation function, which describes the performance of the system depending on the evolved topology of the given network.

Czech Technical University in Prague  
Faculty of Electrical Engineering  
Department of Computer Science and Engineering

## DIPLOMA THESIS ASSIGNMENT

Student **Bc. Pavol Sekereš**

Study programme: Open Informatics  
Specialisation: Artificial Intelligence

Title of Diploma Thesis: **Use of HyperNEAT Encoding for Hybrid Artificial Neural Networks**

### Guidelines

Study the principles of optimization of topology of Artificial Neural Networks based on the Hypercube-based NeuroEvolution of Augmenting Topologies (HyperNEAT).

Together with the diploma thesis supervisor propose a modification of this algorithm, which will be suitable of designing Hybrid Artificial Neural Networks

Implement the proposed algorithm. If possible, modify a suitable current implementation of the HyperNEAT for these purposes.

Test the ability of the proposed algorithm to optimize the topology of a Hybrid Artificial Neural Network on a selected experiment.

### Bibliography/Sources


- [1] Stanley, Kenneth O. Et al., A Hypercube-based Encoding for Evolving Large-scale Neural Networks, in Journal of Artificial Life, MIT Press, Cambridge, MA, USA, pp. 185-212, 2009
- [2] The Hypercube-based NeuroEvolution of Augmenting Topologies (HyperNEAT) Users Page, online at: <http://eplex.cs.ucf.edu/hyperNEATpage/>, cited 2014
- [3] P. Nahodil and J. Vitkú, "Evoluční architektura chování umělých bytostí - agentů v daném prostředí", in Kognice a umělý život (KUZ XIII), 2014.
- [4] Kenneth O. Stanley, Exploiting regularity without development, in Proceedings of the AAAI Fall Symposium on Developmental Systems, AAAI Press, Menlo Park, CA, 2006

Diploma Thesis Supervisor: Ing. Jaroslav Vitkú

Valid until the end of the summer semester of academic year 2015/2016

  
doc. Ing. Filip Železný, Ph.D.  
Head of Department



  
prof. Ing. Pavel Ripka, CSc.  
Dean

Prague, November 10, 2014

# Contents

<b>List of Abbreviations</b>	<b>viii</b>
<b>List of Figures</b>	<b>ix</b>
<b>List of Tables</b>	<b>xi</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Thesis Outline . . . . .	3
<b>2 Theoretical Foundation</b>	<b>4</b>
2.1 Hybrid Artificial Neural Network Systems . . . . .	4
2.1.1 Definition of the Hybrid Artificial Neural Network Systems . . . . .	4
2.1.2 Motivation for Modular Architectures . . . . .	5
2.2 Methods of Design of Artificial Neural Networks . . . . .	9
2.3 Compositional Pattern Producing Networks . . . . .	10
2.3.1 Developmental Encoding . . . . .	10
2.3.2 Encoding Natural Patterns . . . . .	11
2.3.3 Temporal Unfolding and Local Interactions Approximation . . . . .	12
2.4 NeuroEvolution of Augmenting Topologies . . . . .	14
2.4.1 Competing Conventions . . . . .	15
2.4.2 Protecting Innovation with Speciation . . . . .	17
2.4.3 Initial Population and Topological Innovation . . . . .	18
2.4.4 Genetic Encoding and Historical Markings . . . . .	18
2.4.5 Mutation and Crossover in NEAT . . . . .	18
2.4.6 Speciating and Shared Fitness in NEAT . . . . .	22
2.5 Hypercube-based NeuroEvolution of Augmenting Topologies . . . . .	23
2.5.1 Mapping Spatial Patterns to Connectivity Patterns . . . . .	23
2.5.2 Types of Substrate Configuration . . . . .	25

2.5.2.1	State-Space Sandwich Substrate . . . . .	25
2.5.2.2	6-Dimensional Hypercube Substrate . . . . .	26
2.5.3	Algorithm Description . . . . .	27
<b>3</b>	<b>Algorithms Proposed and Tested</b>	<b>28</b>
3.1	Hypercube-based NeuroEvolution of Augmenting Topologies for Hybrid Artificial Neural Network Systems . . . . .	28
3.2	Sandwich Substrate . . . . .	30
3.2.1	Encoding Additional Inputs and Outputs of each Neural Module as a Vector . . . . .	30
3.2.2	Encoding Additional Inputs and Outputs of each Neural Module as a Matrix . . . . .	32
3.3	6-Dimensional Hypercube Substrate . . . . .	32
<b>4</b>	<b>Algorithm Implementation and Testing</b>	<b>36</b>
4.1	HyperNEAT for HANNS Algorithm Implementation . . . . .	36
4.1.1	Proposed Library Extension . . . . .	37
4.2	Experiments Background . . . . .	38
4.3	Experiment 1 - Implementing XOR Logic Function Using HANNS . . . . .	39
4.3.1	Experiment 1 - A) Simple HANNS XOR Problem . . . . .	40
4.3.2	Experiment 1 - B) Medium HANNS XOR Problem . . . . .	43
4.3.3	Experiment 1 - C) Large HANNS XOR Problem . . . . .	46
4.4	Experiment 2 - Motivation-driven Reinforcement Learning HANNS Description . . . . .	49
4.4.1	HyperNEAT Parameters Setting Justification . . . . .	50
4.4.2	Experiment 2 - A) Smaller Gridworld . . . . .	52
4.4.3	Experiment 2 - B) Bigger Gridworld . . . . .	56
4.4.4	Reinforcement Learning Experiments Best Found Genomes and Discussion . . . . .	61
4.5	Conducted Experiments Discussion & Conclusion . . . . .	63
<b>5</b>	<b>Thesis Conclusion and Contributions</b>	<b>64</b>
	<b>Bibliography</b>	<b>68</b>



# List of Abbreviations

<b>DNA</b>	Deoxyribonucleic acid
<b>MIMO</b>	Multiple Input Multiple Output
<b>AHNI</b>	Another HyperNEAT Implementation
<b>MNN</b>	Modular Neural Networks
<b>HANNS</b>	Hybrid Artificial Neural Network Systems
<b>TWEANNs</b>	Topology and Weight Evolving Artificial Neural Networks
<b>CPPN</b>	Compositional Pattern Producing Network
<b>NEAT</b>	Neuro-evolution of Augmented Topologies
<b>HyperNEAT</b>	Hypercube Neuro-evolution of Augmented Topologies
<b>EA</b>	Evolutionary Algorithm
<b>RL</b>	Reinforcement Learning
<b>ANN</b>	Artificial Neural Network
<b>AI</b>	Artificial Intelligence
<b>ASM</b>	Action Selection Mechanism
<b>CTU</b>	Czech Technical University in Prague
<b>ROS</b>	Robotic Operating System

# List of Figures

2.1	HANNS subsystem. . . . .	5
2.2	HANNS system. . . . .	6
2.3	CPPN-generated Regularities . . . . .	12
2.4	Function Produces a Phenotype . . . . .	13
2.5	CPPN example . . . . .	14
2.6	Competing Conventions Problem . . . . .	16
2.7	Mapping of Genotype to Phenotype in NEAT . . . . .	19
2.8	Mutation in NEAT . . . . .	20
2.9	Crossover in NEAT . . . . .	21
2.11	Connectivity Patterns Produced by Connective CPPNs . . . . .	24
2.10	Hypercube-based Geometric Connectivity Pattern Interpretation. . . . .	24
2.12	Substrate Configurations . . . . .	25
2.13	Organization of the HyperNEAT substrate in the robot movement example. . . . .	26
2.14	CPPN for Six-Dimensional Hypercube. . . . .	27
3.1	HANNS Implementing Reinforcement Learning Agent Architecture . . . . .	29
3.2	HyperNEAT for the HANNS Using Sandwich Substrate, Vector Encoding . . . . .	31
3.3	HyperNEAT for the HANNS Using Sandwich Substrate, Matrix Encoding . . . . .	33
3.4	HyperNEAT for the HANNS Using 6-Dimensional Substrate . . . . .	35
4.1	Evolutionary cycle of the HyperNEAT algorithm. . . . .	37
4.2	Exclusive-OR HANNS . . . . .	39
4.3	Exclusive-OR HANNS input/output spreading in the substrate . . . . .	40
4.4	Exclusive-OR Simple HANNS Maximal Fitness . . . . .	42
4.5	Exclusive-OR Simple HANNS Mean Fitness . . . . .	42
4.6	Exclusive-OR Simple HANNS Distinct Species Count . . . . .	43
4.7	Exclusive-OR Medium HANNS HyperNEAT Maximal Fitness . . . . .	44
4.8	Exclusive-OR Medium HANNS Basic EA Maximal Fitness . . . . .	44

4.9	Exclusive-OR Medium HANNS HyperNEAT Mean Fitness . . . . .	45
4.10	Exclusive-OR Medium HANNS Basic EA Mean Fitness . . . . .	45
4.11	Exclusive-OR Big HANNS HyperNEAT Maximal Fitness . . . . .	47
4.12	Exclusive-OR Big HANNS Basic EA Maximal Fitness . . . . .	47
4.13	Exclusive-OR Big HANNS HyperNEAT Mean Fitness . . . . .	48
4.14	Exclusive-OR Big HANNS Basic EA Mean Fitness . . . . .	48
4.15	Hand-wired Motivation-driven Reinforcement Learning HANNS . . . . .	51
4.16	HyperNEAT Setup for Reinforcement Learning . . . . .	52
4.17	Gridworld $5 \times 5$ for Reinforcement Learning Experiment . . . . .	53
4.18	$5 \times 5$ Grid World - Maximal Fitness Comparision . . . . .	54
4.19	$5 \times 5$ Grid World - Mean Fitness Comparision . . . . .	54
4.20	$5 \times 5$ Grid World - HyperNEAT Species count . . . . .	55
4.21	$5 \times 5$ Grid World - HyperNEAT Maximal and Minimal Species Count . .	55
4.22	Gridworld $10 \times 10$ for Reinforcement Learning Experiment . . . . .	56
4.23	HyperNEAT QLambda $10 \times 10$ Grid World - Maximal Fitness . . . . .	57
4.24	Basic Evolutionary Algorithm QLambda $10 \times 10$ Grid World - Maximal Fitness . . . . .	57
4.25	HyperNEAT QLambda $10 \times 10$ Grid World - Mean Fitness . . . . .	58
4.26	Basic Evolutionary Algorithm QLambda $10 \times 10$ Grid World - Mean Fitness	58
4.27	HyperNEAT QLambda $10 \times 10$ Grid World - Species Count . . . . .	59
4.28	HyperNEAT QLambda $10 \times 10$ Grid World - Species Size . . . . .	60

# List of Tables

4.1	Best found genomes in Exclusive-OR Experiment . . . . .	41
4.2	HyperNEAT parameters in Exclusive-OR Experiment . . . . .	49
4.3	HyperNEAT parameters in Reinforcement Learning Experiment . . . . .	60
4.4	Best genomes found in the Grid World experiments . . . . .	61

# Chapter 1

## Introduction

Training large Artificial Neural Network for solving great variety of tasks, or single difficult task can be computationally too complex. Even if we succeed in training such network, the whole system is then viewed as a black box and we do not know which part of the network is responsible for solving concrete subtask of the complex problem. These are the key reasons for the recent upturn in the research of the Modular Hybrid Neural Systems and study of their capabilities, when solving concrete problems.

Besides the present real-world problems, which these systems tackle, Hybrid Neural Network Systems play great role in the present research of the General Artificial Intelligence capable of solving arbitrary task solvable by biological brain. Scientist believe, that biological brain is modular in the different spatial scales and that different parts of brain specialize on slightly different tasks (McGarry, 1999), which makes Modular Hybrid Neural Systems promising in the development of brain-like structures.

When designing Hybrid Neural Systems, just as Artificial Neural Networks, the learning algorithms play significant role in the final performance of the structure. The optimization of the network topologies can bring up the usability of the network. Therefore it makes sense to experiment with different optimization techniques and develop new approaches. Recently, the biologically inspired indirect encoding neuro-evolutionary approaches have been studied and utilized for the learning of the Artificial Neural Networks. The HyperNEAT (Hypercube-based Neuro-evolution of Augmented Topologies) algorithm builds on the top of these approaches and many experiments showed, that it is capable of solving certain learning tasks with enhanced performance compared to the other techniques (Stanley, 2009).

In this thesis, I study the design of the HyperNEAT algorithm and HANNS (Hybrid Artificial Neural Network Systems). With the strong theoretical background of these

state-of-the-art algorithms, I develop the extension of the HyperNEAT algorithm focused on the learning of the topology of arbitrary HANNS. Respective algorithm is then tested on conducted experiment and compared with other algorithms for learning the topology of the HANNS. The results are then summarized in the conclusion chapter of the thesis.

## 1.1 Thesis Outline

Here is the brief description of this thesis outline:

**Chapter 1** The first chapter provides introduction to the thesis goals and aims of the Diploma Thesis.

**Chapter 2** The second chapter describes theoretical foundation required for understanding the Hybrid Artificial Neural Network Systems and HyperNEAT algorithm in detail. It also familiarizes reader with the present state-of-the-art for these structures and algorithms.

**Chapter 3** The third chapter introduces the algorithms, which were chosen and implemented.

**Chapter 4** In the fourth chapter, the conducted experiments for evaluation and comparison of the implemented algorithms are described.

**Chapter 5** The Fifth chapter contains conclusion and compares the predicted and actual goal.

# Chapter 2

## Theoretical Foundation

### 2.1 Hybrid Artificial Neural Network Systems

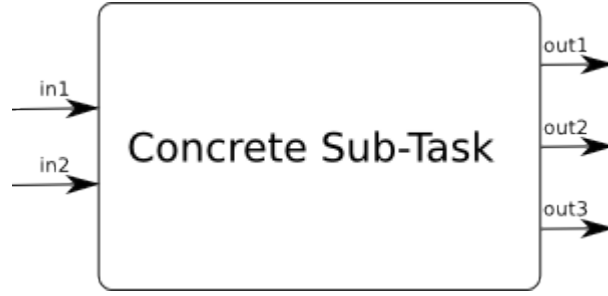
Because we aim to design learning algorithm for Hybrid Artificial Neural Network Systems, we will define them in this section. The capabilities of the Hybrid Artificial Neural Network Systems have been studied extensively in the recent years. Other scientists refer to them as Modular Hybrid Neural Systems (Mcgarry, 1999) or as Modular Neural Networks (Auda G., 1999). They have been studied so vastly, because they can be utilized in many different areas, where they provide significant advantages compared to the ANNs. We will define the HANNS and then discuss the motivation for the study of these modular architectures.

#### 2.1.1 Definition of the Hybrid Artificial Neural Network Systems

Hybrid Artificial Neural Network System is a Neural Network, that consists of several modules, each module carrying out one sub-task of the global task solved by the HANNS. All modules are functionally integrated. A module can be a sub-structure or a learning sub-procedure of the whole network. The network's global task can be any neural network application, e.g., mapping, function approximation, clustering or associative memory application (Auda G., 1999).

The example of the module is depicted in the figure above. In general it can have arbitrary number of inputs and outputs. The whole system has several of such modules, which are interconnected. In additions there are some external inputs and outputs





**Figure 2.1:** Example of the module of the hybrid modular system with 3 outputs and 2 inputs.

defined.

Formally, HANNS is given by set of local inputs and local outputs to the nodes :

$$In \in \{in_1, \dots, in_n\}, Out \in \{out_1, \dots, out_n\} \quad (2.1)$$

.

Each output is either external, or connected to some local input. This connection is defined by the Binary Adjacency Matrix :

$$A : a_{ij} = 1 \Leftrightarrow (in_i \rightarrow in_j) \quad (2.2)$$

That is, whenever the value in matrix is 1, the corresponding input connects to the corresponding output, and otherwise they are not connected. In addition, each node is defined by concrete mapping:

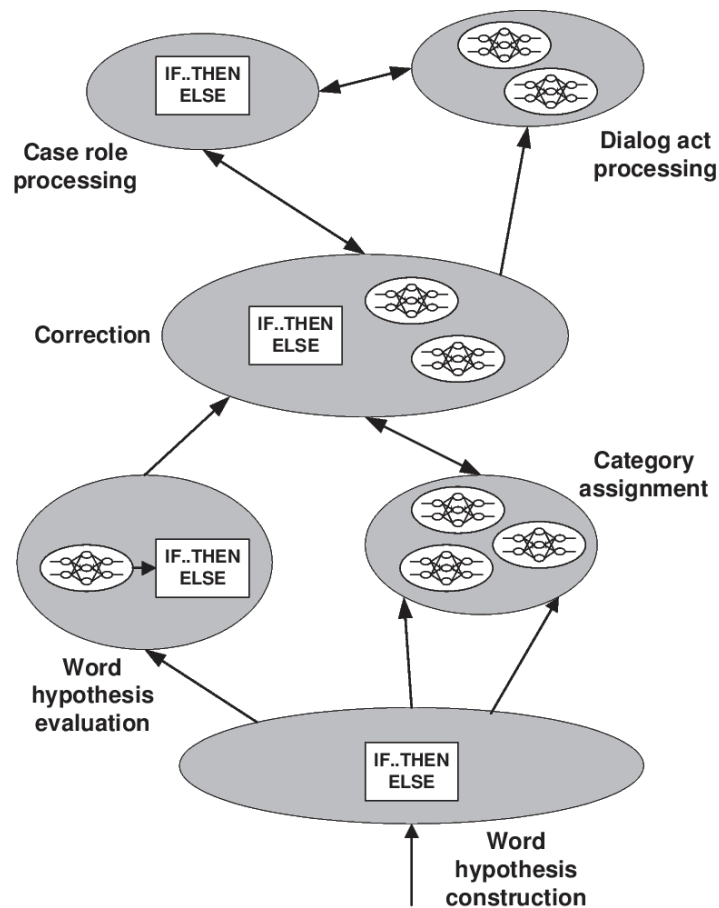
$$M (in_1, \dots, in_n) \rightarrow (out_1, \dots, out_m) \quad (2.3)$$

which provides unique transformation of the node inputs to outputs. To define HANNS correctly, we must also define which of the inputs and outputs of the system are external and connect them to the system of nodes.

In the following part, we will describe the motivation for the study of the Modular Architectures.

## 2.1.2 Motivation for Modular Architectures

Modular Architectures exhibit certain features, which are useful when designing complex system for solving, decision making, learning tasks, etc.



**Figure 2.2:** Example of the hybrid modular system. Concretely the SCREEN (*Sym-bolic Connectionist for Robust Enterprise for Natural language*) hybrid system is depicted. It has been developed to learn the acoustics, syntax, semantics and pragmatics of spoken language. Due to the complexity of this task SCREEN required an interleaved architecture for learning spoken language analysis (S. Wermter, 1997).

They exhibit **modularity** attribute. To understand, how modularity can be useful, let us take as an example human brain. It appears that biological brain is also modular in different spatial scales. On the smallest scale, synapses are clustered on dendrites (S. Johannes and Munte, 1996). On the largest scale, the brain is composed of several anatomically and functionally distinct areas (Hrycej, 1992). Between these two levels, columnar structures appear 28 with intracolumnar connections (Murre, 1992). The common belief that the biological brain is modular makes the study of modular architectures key for the development of artificial brain-like structure capable of solving problems with similar complexity. It has been and still is one of the great goals of Artificial Intelligence to study and develop such structures.

Another key feature of these systems is **Functional specialization**. Different parts of brain specialize on slightly different tasks. Local and global stimulus perceptions are handled by parts of the left and right brain-hemispheres, respectively. Therefore modular architecture, compared to the ANN has certain advantages when developing systems with functionality of the biological brain (Auda G., 1999).

**Competition/Cooperation** among the modules. In biological brains, the visual cortex modules cooperate between each other in certain connectivity patterns to create overall visions of objects. These observations suggest forms of communication among modules in order to take the final system's decision. There are other examples in nature which suggest, that designing a cooperation scheme can give better performance than direct summation of all modules capabilities (Auda G., 1999).

**Scalability** is feature highly appreciated in the computer systems. In brain, scalability provides constant processing time in spite of a range of sizes of its modules. It is suggested that this is a result of the brain's highly modular structure (Auda G., 1999).

**Extendability** - modular architectures are easily extendable compared to ANNs. If we want to modify functionality of ANN we need to retrain the whole network. In Modular Architectures adding another module, or changing single module can modify the behavior of the system as well.

**Learning in Stages** is another interesting feature of Modular Architecture. Because the modules are autonomous, they can be trained separately and generally it is easier to train single module of Modular System then the whole system at once. This enables the training of the modules simultaneously, which speeds up the phase in the multi-threaded environments.

**Mixing learning methods** - this feature enables us to model systems, where unsupervised learning modules recognize patterns from the noisy and unreadable raw input

data from some environment. These modules extract the important features from environment and then recognized patterns can be processed by supervised learning modules. This way, supervised learning modules work with cleaner and processed data and this can improve their performance significantly. This kind of architecture can be utilized when designing autonomous agents interacting with artificial environment (Vitku and Nahodil, 2014).

**Task Decomposition** enables Modular Systems to solve problem, which is too complex to be solvable at once. A similar approach is used for solving the traveling salesman problem using a MNN mathematical model (Foo and Szu, 1989).

**Loose coupling** of the systems is a another property of the HANNS. It is desirable in certain applications. For example if we use ANN in classification task, the network will first learn the the easy (nonoverlapping, odd, separable) categories. It will devote most of its hidden units for them. Due to the high coupling of the neurons there might be insufficient amount of the neurons for defining the boundary for the difficult(overlapping) categories. For modular structures, dividing the classification task to separate neural modules decreases the coupling effect (Auda G., 1999).

**Smaller amount of training samples** is needed to train the Modular System to be general, compared to the standard ANN. This can be explained by the fact, that the most famous theoretical worst-case bound on the number of training samples - VC dimension (Vapnik and Chervonenkis, 1971) decreases by splitting the network into the modules. For ANN it is roughly, a linear function to the number of weights. This can be explained by the fact, that decomposing the objective task over smaller, sparsely-connected, and less complex modules decreases the connections-per-node ratio substantially (Auda G., 1999).

**Readability** is another advantage of modular architecture. Because the system is divided into modules, we are able to understand more precisely which part of the system performs concrete sub-task. In contrast to the ANNs, where it can be extremely difficult to track which group of neurons is responsible for which sub-task.

HANNS also provide improve the speed of learning by eliminating the **crosstalk problem**. Spatial crosstalk occurs when the outputs of the network provide conflicting error information to a single hidden unit in a single iteration. It occurs mainly in ANN's with fully connected layers. Temporal crosstalk occurs when the network receives conflicting training information over time (iterations), e.g., when the network is forced to learn several dissimilar functions simultaneously. HANNS prevent the crosstalk by the decomposition of subtasks into the modules (Auda G., 1999).

Because of these and many other reasons it is important to study the capabilities of the HANNS.

We have defined and discussed the importance of studying the HANNS. In the following section we will explain methods of design of ANN and categorize approach, which will be developed later in the following chapter.

## 2.2 Methods of Design of Artificial Neural Networks

Determining the correct (optimal) topology of the ANN manually is often infeasible when designing ANN for solving difficult problems. Throughout the years of the research in Artificial Intelligence, scientists have developed many different approaches and algorithms for optimization of the network structures. Present methods of design of ANN can be divided in the following groups :

- **Learning Algorithms:** In this case, the network has predefined topology (e.g. feed-forward network) and a local learning rule, which modifies connection weights between particular neurons. Here can be mentioned supervised learning in feed-forward network by means of back-propagation algorithm, or unsupervised Hebbian or competitive learning (Vitku and Nahodil, 2014).
- **Topology Optimization** Compared to the previous case, the topology of ANN can be optimized by globally operating optimization algorithm. The topology is often partially predefined (e.g. to the feed-forward networks) and the Evolutionary Algorithm (EA) is used to find correct weights (Vitku and Nahodil, 2014).
- **Neural Engineering** This represents the Top-Down approach in designing neural systems. Instead of starting from an individual neuron, it works over populations of neurons. Each population has purpose of solving particular part of the problem (Vitku and Nahodil, 2014). Here, the qualitative tools are often used to compute particular connections between neurons and/or between populations of neurons. These methods are often used in largerscale neural models (H. de Garis and Ruiting, 2010).

In following sections we will explain in detail all of the components of the HyperNEAT algorithm. It is neuro-evolutionary optimization algorithm, used for development and optimization of the topology of ANN. We will utilize this algorithm, to design similar approach for optimization of connection weights between engineered sub-systems. In order to explain, how HyperNEAT algorithm works, we will describe all of its features in details, starting with Compositional Pattern Producing Networks.

## 2.3 Compositional Pattern Producing Networks

CPPN is type of ANN, which indirectly encodes typically order of magnitude larger pattern. In the following subsections, we will discuss the advantages of the indirect encoding. We will explain how CPPNs utilize this approach and what is the main difference between CPPN encoding, and the way these complex patterns are encoded in nature.

### 2.3.1 Developmental Encoding

In nature, the DNA encodes complex structures on an enormous scale. The genes in DNA represent astronomically complex structures with trillions of interconnecting parts, such as the human brain. Despite this fact, DNA does not contain such amount of genes, only 30 thousand genes encode the entire human body (Zigmond and Squire., 1999).

Inspired by the DNA, the researchers are attempting to achieve the same efficiency in the representation of sophisticated patterns by studying and implementing developmental encodings. These encodings map the genotype to the phenotype through a process of growth from a small starting point to a mature form. A major challenge in this effort is to find the right level of abstraction of biological development to capture its essential properties without introducing unnecessary inefficiencies (Stanley, 2007).

Compositional Pattern Producing Networks represent one of these attempts of the computer scientists in the field of Artificial Intelligence to utilize the idea of the developmental encoding of the order of magnitude larger patterns. Unlike other abstractions, like cellular growth simulations, Compositional Pattern Producing Networks map to the phenotype without local interaction. This means, that each individual component of the phenotype is determined independently of every other component. CPPNs also break the tradition of temporal unfolding in the developmental encoding research. They achieve the struc-

tural relationships which result from a process of development without simulating the process itself. This enables them to avoid the temporal unfolding and local interactions (Stanley, 2007).

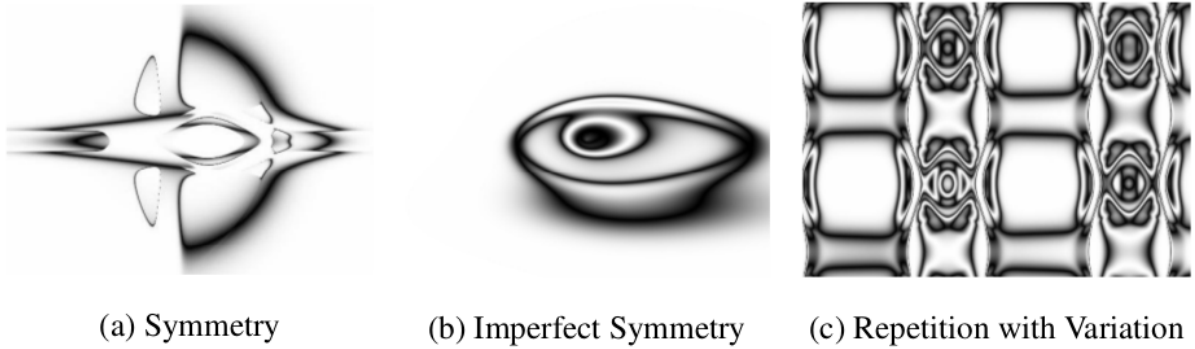
### 2.3.2 Encoding Natural Patterns

CPPN is structurally similar to ANN, but generally operates with larger set of activation functions. Inspired by natural encoding, it is designed to be capable of encoding common general properties of pattern in nature. These patterns are following:

- **Repetition** : Multiple instances of the same substructure is a hallmark of biological organisms. On different scales, the grouping of cells throughout the body and connection of neurons in the brain, exhibit the repetition of the same motifs in a single organism. Repetition in the phenotype is also called self-similarity (Bentley and Kumar, 1999).
- **Repetition with Variation** : Frequently, motifs are repeated yet not entirely identical. Each vertebrae in the spine is similar, yet they each have slightly different proportions and morphologies (Zigmond and Squire., 1999). Similarly, human fingers repeat a regular pattern, yet no two fingers on the same hand are identical. Repetition with variation is abundant throughout all of natural life.
- **Symmetry** : Often repetition occurs through symmetry, as when the left and right sides of a body are identical mirror images in classic bilateral symmetry.
- **Imperfect Symmetry** : While an overall symmetric theme is observable in many biological structures, they are nevertheless generally not perfectly symmetric. Such imperfect symmetry is a common feature of repetition with variation. The human body, while overall symmetric, is not equivalent on both sides; some organs appear only on one side and one hand is usually dominant over the other.
- **Elaborated Regularity** : Over many generations, regularities are often elaborated and exploited further. For example, the bilaterally symmetric fins of early fish eventually became the arms and hands of mammals, displaying some of the same regularities (Stanley and Miikkulainen, 2004).
- **Preservation of Regularity** : Over generations, established regularities are often strictly preserved. Bilateral symmetry does not easily produce three-way symmetry,

and four-limbed animals rarely produce offspring with a different number of limbs, even as the limb design itself is elaborated (Stanley, 2007).

It has been shown through interactive evolution, that CPPN can encode these patterns seen in nature. In this process, the CPPNs were evolved to match certain patterns from the nature to determine, if they are capable of encoding such complex patterns.



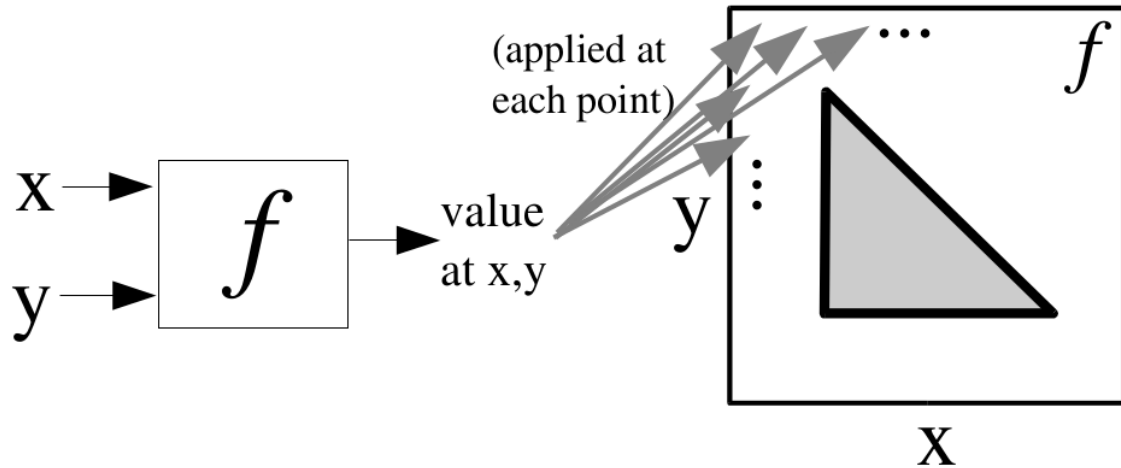
**Figure 2.3: CPPN-generated Regularities.** Spatial patterns exhibiting (a) bilateral symmetry, (b) imperfect symmetry, and (c) repetition with variation (notice the nexus of each repeated motif) are depicted. These patterns demonstrate that CPPNs effectively encode fundamental regularities of several different types (Stanley, 2009).

### 2.3.3 Temporal Unfolding and Local Interactions Approximation

As we mentioned earlier, the CPPN does not operate with temporal unfolding and local interactions, unlike other developmental encodings do. In fact it tries to approximate these features by function, which is in fact composition of set of functions. It conceives the phenotype as a distribution of points in a multidimensional Cartesian space. Viewed this way, a phenotype can be described as a function of  $n$  dimensions, where  $n$  is the number of dimensions in the physical world. For each coordinate, the presence or absence of a point, or its level of expression, is an output of the function that describes the phenotype. Although the original pattern has been created through temporal progression and local interaction it is possible, to represent it through a functional description. Cybenko (Cybenko, 1989) showed that any function can be approximated by a neural network

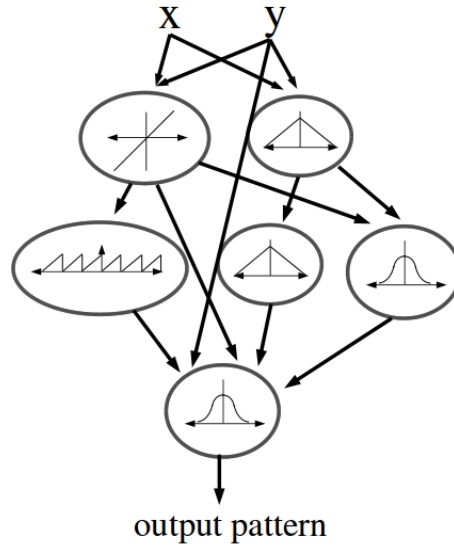


with single hidden layer. The more neurons in the network, the more accurate the approximation can be. Thus, any morphology, when viewed as a distribution of particles in space, is possible to represent as a function without the notion of time (Stanley, 2007).



**Figure 2.4:** *A Function Produces a Phenotype.* The function  $f$  takes arguments  $x$  and  $y$ , which are coordinates in a two-dimensional space. When all the coordinates are drawn with an intensity corresponding to the output of  $f$  at that coordinate, the result is a pattern, which can be conceived as a phenotype whose genotype is  $f$ . In this example,  $f$  produces a triangular phenotype (Stanley, 2007).

CPPN is capable of approximating function, which approximates the morphological pattern commonly seen in nature. CPPN outputs these patterns by operating with various activation functions. For example, it achieves bilateral symmetry using the gaussian function. To achieve repetition of the same pattern it utilizes periodic functions like sine. Linear functions can be employed to produce linear or fractal-like patterns.



**Figure 2.5:** *The graph determines which functions connect to which. The connections are weighted such that the output of a function is multiplied by the weight of its outgoing connection. If multiple connections feed into the same function then the downstream function takes the sum of their weighted outputs. Note that the topology is unconstrained and can represent any possible relationships (including recurrent) (Stanley, 2006).*

Although it has been shown that CPPN are capable of encoding various patterns commonly seen in nature, we still need an algorithm which would enable us to evolve the CPPN to match the certain patterns, that we want to encode. In the next section, we will describe an algorithm for the NeuroEvolution of Augmenting Topologies ( NEAT ). This algorithm can be modified to evolve not strictly ANNs , but also CPPNs , whose neurons typically contain much more various activation functions than ANN.

## 2.4 NeuroEvolution of Augmenting Topologies

NEAT is an algorithm for evolving ANN using the direct encoding technique. In nature the encoding of the sophisticated structures such as human brain is done through indirect encoding in the DNA. Indirect encoding has many advantages over direct encoding. It is capable of encoding large structures using the order of magnitude lower amount of information. NEAT provides background for HyperNEAT algorithm, which is separated

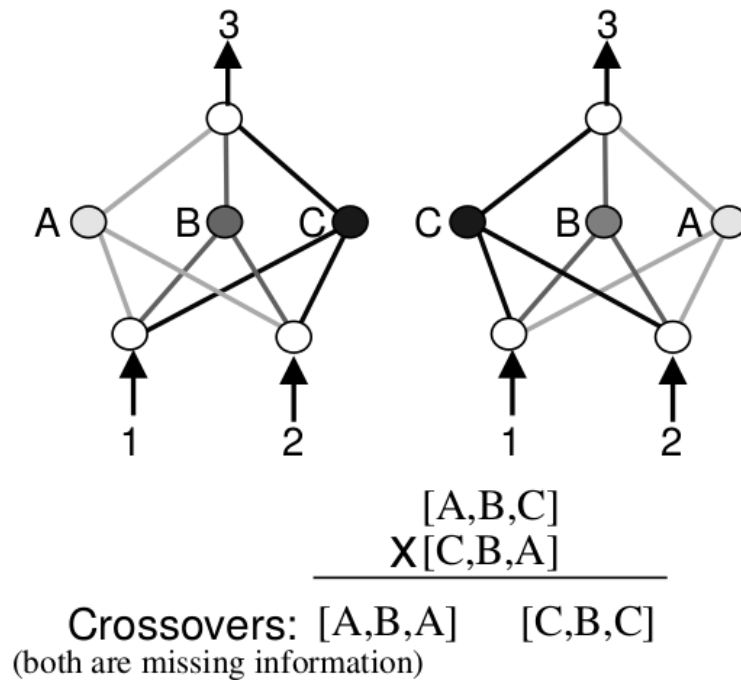
algorithm build on top of the NEAT algorithm. HyperNEAT in contrast uses indirect encoding technique similar to what we see in nature. To understand HyperNEAT algorithm we must first understand, how NEAT works. In addition, it has been shown, that in certain tasks NEAT outperforms other algorithms for evolving ANN structures which use indirect encoding. For example, in the pole balancing task it outperforms Cellular Encoding (Gruau, 1993) technique 25 times (Stanley, 2002). NEAT evolves topology and weights of the ANN incrementally. This presents several technical challenges:

1. Finding a meaningful way for two disparate topologies to cross over.
2. Protecting the topological innovation, which needs few generations to get optimized. Preventing it from disappearing from the population prematurely.
3. Finding the structure with minimal size throughout the evolution. Ideally without the need of special fitness function which would measure complexity.

NEAT addresses all of these problems efficiently. In the following parts it will be shown how algorithm handles these challenges.

### 2.4.1 Competing Conventions

One of the main problems for NeuroEvolution is the Competing Conventions Problem, also known as the Permutations Problem (Radcliffe, 1993). Competing conventions means having more than one way to express a solution to a weight optimization problem with a neural network. When genomes representing the same solution do not have the same encoding, crossover is likely to produce damaged offspring.



**Figure 2.6:** *The two networks compute the same exact function even though their hidden units appear in a different order and are represented by different chromosomes, making them incompatible for crossover. The figure shows that the two single-point recombinations are both missing one of the 3 main components of each solution. The depicted networks are only 2 of the 6 possible permutations of hidden unit orderings (Stanley, 2002).*

Figure 2.6 depicts the problem for a simple 3-hidden-unit network. The three hidden neurons A, B, and C, can represent the same general solution in  $3! = 6$  different permutations. When one of these permutations crosses over with another, critical information is likely to be lost. For example, crossing  $[A, B, C]$  and  $[C, B, A]$  can result in  $[C, B, C]$ , a representation that has lost one third of the information that both of the parents had. In general, for  $n$  hidden units, there are  $n!$  functionally equivalent solutions (Stanley, 2002). In the problem of Topology and Weight Evolving Artificial Neural Networks (TWEANNs) the problem of competing conventions is even greater, because two networks with completely different topologies can represent equivalent functionality (Stanley, 2002).

NEAT algorithm tries to solve this problem by finding the way to match up the genotype representing different structures. The genomes can vary in length and the same structures can appear at different positions in the ANN. Nature faces same problem

when matching up genes for crossover. Nature's solution utilizes homology: two genes are homologous if they are alleles of the same trait (Stanley, 2002). In the NEAT, the main idea is, that historical origin of two genes is direct evidence of their homology if these genes share the same origin. NEAT performs artificial synapsis based on historical markings. This allows the algorithm to add new structure and still track the origin of the gene. Two genes are therefore compared based on the historical markings, which measures their (dis)similarity (Stanley, 2002).

### 2.4.2 Protecting Innovation with Speciation

By introducing mutation to the ANN new structures are developed in the network. It is quite probable, that these structures will not have optimized parameters as soon as they appear, they can reduce the fitness of the individual. It is unlikely that a new node or connection just happens to express a useful function as soon as it is introduced. Thus, it is necessary to somehow protect networks with structural innovations so they have a chance to make use of their new structure. In nature, different structures tend to be in different species that compete in different niches. Thus, innovation is implicitly protected within a niche. Similarly, if networks with innovative structures could be isolated into their own species, they would have a chance to optimize their structures before having to compete with the population at large (Stanley, 2002).

Speciating, also known as niching, has been studied in Genetic Algorithms, where a function has multiple optima, but is not usually applied to neuroevolution. The competing conventions problem makes measuring compatibility particularly problematic because networks that compute the same function can appear very different. NEAT solution to the competing conventions problem allows the population to be easily speciated. NEAT uses explicit fitness sharing (Goldberg and Richardson, 1987), which forces individuals with similar genomes to share their fitness payoff. Fitness sharing divides the population into different fitness peaks, which consist of ANNs sharing the fitness. Each peak is limited by size, so there is no threat of any one species taking over the whole population. Explicit fitness sharing is well-suited for NEAT, because similarity can easily be measured based on the historical information in the genes (Stanley, 2002).

### 2.4.3 Initial Population and Topological Innovation

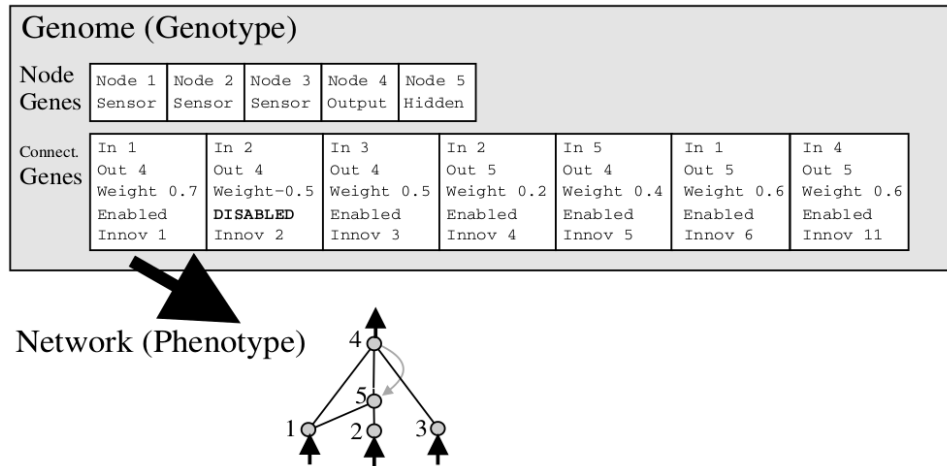
In general, it is desirable to evolve minimal solutions; that way, the number of parameters that have to be searched is reduced. If we start with the set of networks with random topologies, this does not lead to finding the minimal solutions, because the networks can incorporate many unnecessary parts from the beginning. The way to overcome this problem would be to penalize large networks by evaluating them with worse fitness. NEAT uses alternative solution. It starts out with a minimal population of networks, and in the next iterations extends these simple structures. The structures therefore grow only if it benefits the solution, resulting in the desired minimal solution to be found.

### 2.4.4 Genetic Encoding and Historical Markings

NEAT's genetic encoding scheme is designed to allow corresponding genes to be easily lined up when two genomes cross-over during mating. Genomes are linear representations of network connectivity. Each genome includes a list of connection genes, each of which refers to two node genes being connected. Node genes provide a list of inputs, hidden nodes, and outputs that can be connected. Each connection gene specifies the in-node, the out-node, the weight of the connection, whether or not the connection gene is expressed (an enable bit), and an innovation number, which allows finding corresponding genes (Stanley, 2002).

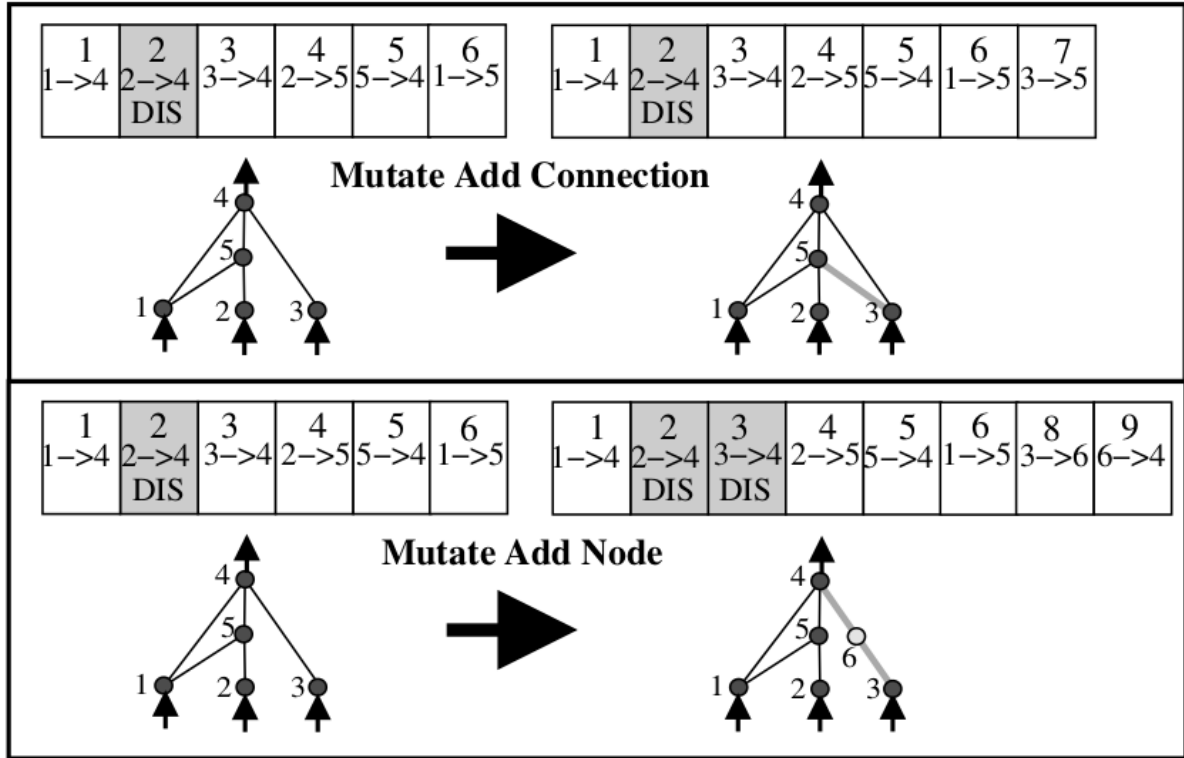
### 2.4.5 Mutation and Crossover in NEAT

Mutation in NEAT can change both connection weights and network structures. Connection weights mutate as in any NeuroEvolution system, with each connection either perturbed or not at each generation. Structural mutations occur in two ways. Each mutation expands the size of the genome by adding gene(s). In the add connection mutation, a single new connection gene with a random weight is added connecting two previously unconnected nodes. In the add node mutation, an existing connection is split and the new node placed where the old connection used to be. The old connection is disabled and two new connections are added to the genome. The new connection leading into the new node receives a weight of 1, and the new connection leading out receives the same weight as the old connection. This method of adding nodes minimizes the initial effect of the mutation (Stanley, 2002). Through mutation, the genomes in NEAT will



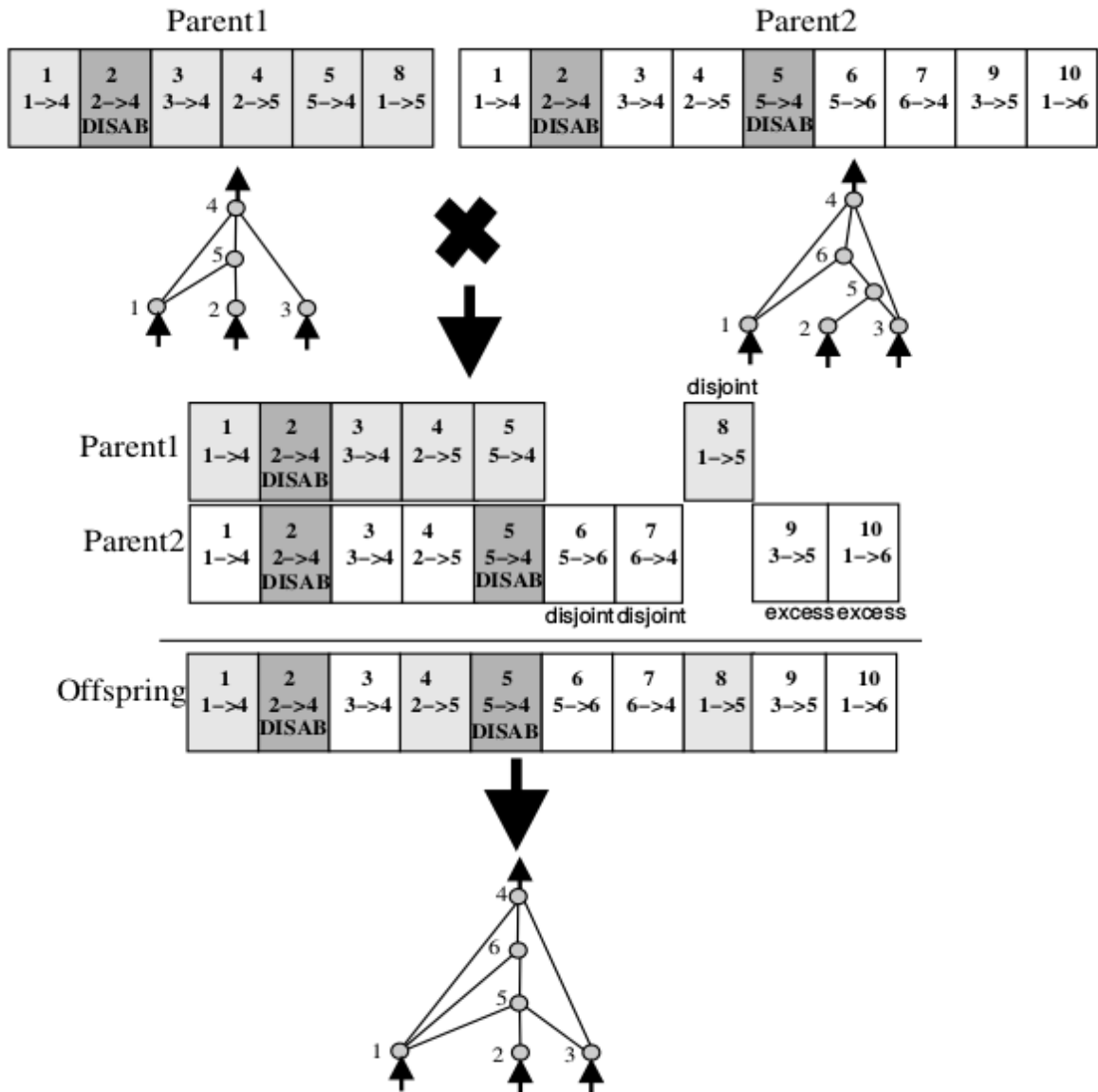
**Figure 2.7:** *A genotype to phenotype mapping example. A genotype is depicted that produces the shown phenotype. There are 3 input nodes, one hidden, and one output node, and seven connection definitions, one of which is recurrent. The second gene is disabled, so the connection that it specifies (between nodes 2 and 4) is not expressed in the phenotype (Stanley, 2002).*

gradually get larger. As we mentioned earlier, NEAT tracks the historical information about the genes using the historical markings. Tracking these historical origins requires very little computation. Whenever a new gene appears (through structural mutation), a global innovation number is incremented and assigned to that gene. When performing the crossover, the offspring inherits the same innovation numbers on each gene as the parents. The historical markings give NEAT a powerful new capability (Stanley, 2002). The system now knows exactly which genes match up with which. When crossing over, the genes in both genomes with the same innovation numbers are lined up. These genes are called matching genes. Genes that do not match are either disjoint or excess, depending on whether they occur within or outside the range of the other parent's innovation numbers (Stanley, 2002). They represent structure that is not present in the other genome. In composing the offspring, genes are randomly chosen from either parent at matching genes, whereas all excess or disjoint genes are always included from the more fit parent. This way, historical markings allow NEAT to perform crossover using linear genomes without the need for expensive topological analysis (Stanley, 2002).



**Figure 2.8:** *The two types of structural mutation in NEAT. Both types, adding a connection and adding a node, are illustrated with the connection genes of a network shown above their phenotypes. The top number in each genome is the innovation number of that gene. The innovation numbers are historical markers that identify the original historical ancestor of each gene. New genes are assigned new increasingly higher numbers. In adding a connection, a single new connection gene is added to the end of the genome and given the next available innovation number. In adding a new node, the connection gene being split is disabled, and two new connection genes are added to the end the genome. The new node is between the two new connections. A new node gene (not depicted) representing this new node is added to the genome as well (Stanley, 2002).*





**Figure 2.9:** Matching up genomes for different network topologies using innovation numbers. Although Parent 1 and Parent 2 look different, their innovation numbers (shown at the top of each gene) tell us which genes match up with which. Even without any topological analysis, a new structure that combines the overlapping parts of the two parents as well as their different parts can be created. Matching genes are inherited randomly, whereas disjoint genes (those that do not match in the middle) and excess genes (those that do not match in the end) are inherited from the more fit parent. In this case, equal fitnesses are assumed, so the disjoint and excess genes are also inherited randomly. The disabled genes may become enabled again in future generations: there's a preset chance that an inherited gene is disabled if it is disabled in either parent (Stanley, 2002).

### 2.4.6 Speciating and Shared Fitness in NEAT

As we mentioned earlier NEAT also incorporates speciation of the population. The individuals compete in their own niches. This way, topological innovations are protected in a new niche where they have time to optimize their structure through competition within the niche. The idea is to divide the population into species such that similar topologies are in the same species. The division of the population into the niches is solved via historical markings. The compatibility distance  $\delta$  of different structures in NEAT is defined as linear combination of the number of excess  $E$ , disjoint  $D$  genes, and weight differences of matching genes  $W$ .

$$\delta = \frac{c_1 E}{N} + \frac{c_2 D}{N} + c_3 \bar{W} \quad (2.4)$$

The coefficients  $c_1$ ,  $c_2$ , and  $c_3$  allow us to adjust the importance of the three factors, and the factor  $N$ , the number of genes in the larger genome, normalizes for genome size ( $N$  can be set to 1 if both genomes are small, i.e., consist of fewer than 20 genes) (Stanley, 2002). The genomes with small compatibility distance belong into the same niche while the ones with big distance have different niches. As the reproduction mechanism for NEAT, explicit fitness sharing is used (Goldberg and Richardson, 1987), where organisms in the same species must share the fitness of their niche. The adjusted fitness  $f'_i$  for organism  $i$  is calculated according to its distance  $\delta$  from every other organism  $j$  in the population:

$$f'_i = \frac{f_i}{\sum_{j=1}^n sh(\delta(i, j))} \quad (2.5)$$

The sharing function  $sh$  is set to 0 when distance  $\delta(i, j)$  is above the threshold  $\delta_t$ ; otherwise,  $sh(\delta(i, j))$  is set to 1 (Spears, 1995). Thus,  $\sum_{j=1}^n sh(\delta(i, j))$  reduces to the number of organisms in the same species as organism  $i$ . Every species is assigned a potentially different number of offspring in proportion to the sum of adjusted fitnesses  $f'_i$  of its member organisms. Species then reproduce by first eliminating the lowest performing members from the population. The entire population is then replaced by the offspring of the remaining organisms in each species. NEAT further provides search towards minimal-dimensional spaces by starting out with a uniform population of networks with zero hidden nodes (i.e., all inputs connect directly to outputs) (Stanley, 2002).

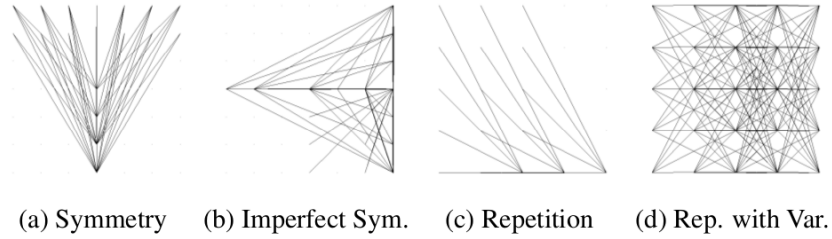
## 2.5 Hybercube-based NeuroEvolution of Augmenting Topologies

It has been shown in the previous sections, how CPPN is able to encode relatively sophisticated spatial pattern by order of magnitude smaller amount of information. To utilize this approach, when evolving large-scale ANN, the CPPN must be capable of encoding connectivity pattern. Fortunately it has been shown that this is indeed possible.

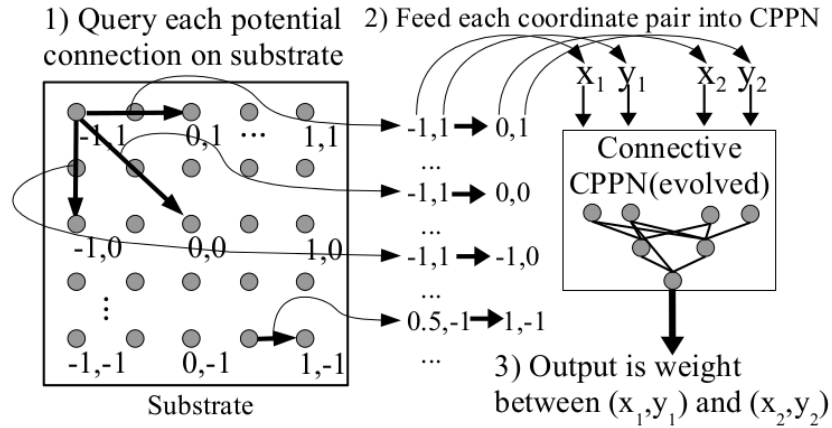
### 2.5.1 Mapping Spatial Patterns to Connectivity Patterns

It turns out that there is an effective mapping between spatial and connectivity patterns that can elegantly exploit geometry. The main idea is to input into the CPPN the coordinates of the two points that define a connection rather than inputting only the position of a single point. The output is then interpreted as the weight of the concrete neural connections rather than intensity of a point. The CPPN for example takes on input pair of points in the four-dimensional space  $(x_1, y_1, x_2, y_2)$  and returns value of the weight  $w$ . This way it can be queried for the connections between any two points in this space. By the convention, if the weight  $w$  is bellow certain value  $w_{min}$  it is thought of as if it would be zero. The magnitude of weight above this threshold are scaled between zero and certain maximum magnitude  $w_{max}$ . Therefore, the pattern produced by the CPPN can represent any network topology (Stanley, 2009).

The connectivity pattern produced by a CPPN in this way is called the substrate. Spatial patterns with symmetries and regularities correspond to connectivity patterns with the same properties (Stanley, 2009). In the next part we will discuss the possibilities of different substrates for HyperNEAT, and how we need to modify the CPPN to be able to query for the connections between the points in the concrete Cartesian space and produce substrate.



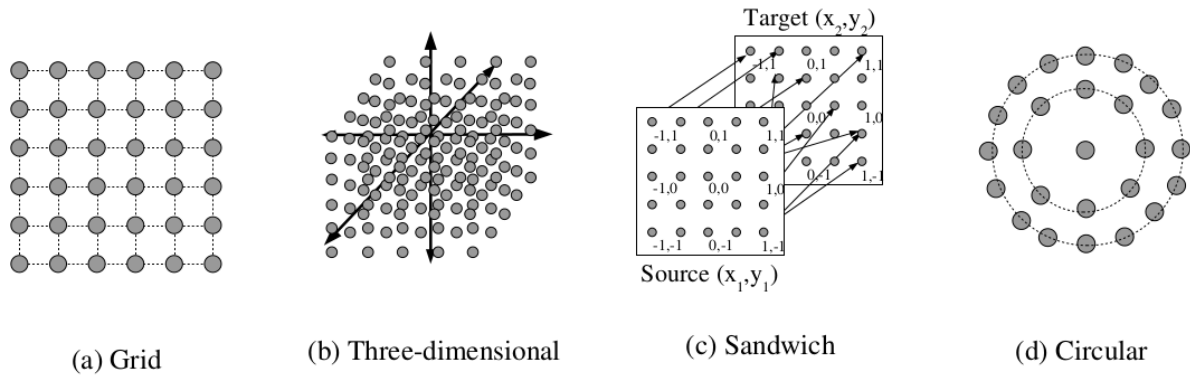
**Figure 2.11:** *Various Connectivity Patterns Produced by Connective CPPNs. These patterns, produced through interactive evolution, exhibit several important connectivity motifs: (a) bilateral symmetry, (b) imperfect symmetry, (c) repetition, and (d) repetition with variation (Stanley, 2009).*



**Figure 2.10:** *Hypercube-based Geometric Connectivity Pattern Interpretation. A grid of nodes, called the substrate, is assigned coordinates such that the center node is at the origin. (1) Every potential connection in the substrate is queried to determine its presence and weight; the dark directed lines shown in the substrate represent a sample of connections that are queried. (2) For each query, the CPPN takes as input the positions of the two endpoints and (3) outputs the weight of the connection between them. After all connections are determined, a pattern of connections and connection-weights results that is a function of the geometry of the substrate. In this way, connective CPPN produces regular patterns of connections in space (Stanley, 2009).*

## 2.5.2 Types of Substrate Configuration

So far we have mentioned the substrate produced by CPPN with four inputs. This four-dimensional hypercube is often called as state-space sandwich (Churchland, 1986) and we will use this terminology as well. The sandwich is a restricted three-dimensional structure in which one layer can send connections only in one direction to one other layer. This substrate allows us to represent ANN with two layers: input and output layer. In general we have more substrate possibilities and all of them are useful, but each for different task.

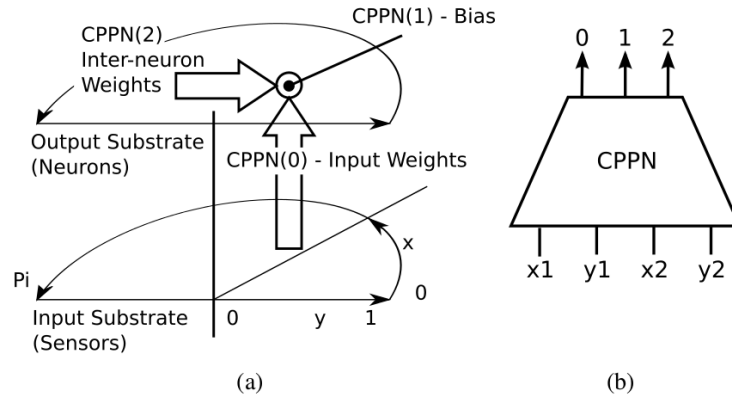


**Figure 2.12:** *Various Substrate Configurations.* This figure shows (b) a three-dimensional configuration of nodes centered at  $(0, 0, 0)$ , (c) a “state-space sandwich” configuration in which a source sheet of neurons connects directly to a target sheet, and (d) a circular configuration. Different configurations are likely suited to problems with different geometric properties.. (Stanley, 2009).

Although other types of substrates are useful as well, we will mainly focus on sandwich and 6-dimensional hypercube substrates. We will show, that with this subset of possible substrates, we are capable of encoding arbitrary ANN including recurrent connections.

### 2.5.2.1 State-Space Sandwich Substrate

Sandwich is one of the typical substrate configurations used in HyperNEAT. It is a single two-dimensional sheet of neurons that connects to another two-dimensional sheet. It can be expressed by the single CPPN with 4-dimensional input  $(x_1, y_1, x_2, y_2)$ , where  $(x_2, y_2)$  is interpreted as a location on the target sheet rather than as being on the same plane as the source coordinate  $(x_1, y_1)$ . In general, we can design CPPN with multiple outputs instead of the single output. The additional outputs provide the possibility of encoding recurrent



**Figure 2.13:** *Organization of the HyperNEAT substrate. There were two distinct substrates used (a) and the CPPN has three outputs. CPPN(0) output is a weight between input substrate (sensor) and a neuron in the upper substrate. Second, CPPN(1) output is used as bias for neurons in the upper substrate. For bias calculation 3rd and 4th CPPN inputs are set to 0. Last CPPN(2) output represents connection weights among neurons in the upper substrate (Drchal et al., 2009).*

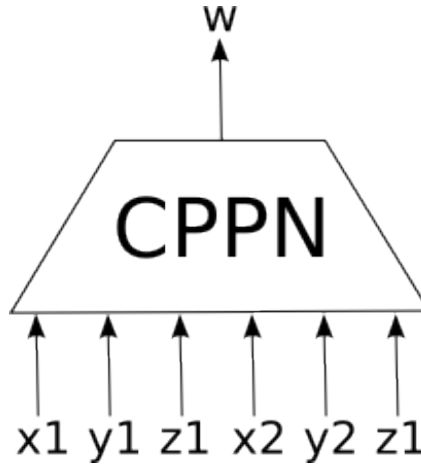
connections and bias the neurons. I provide a figure, which depicts the CPPN and the substrate for HyperNEAT which evolves the ANN for the controlling of the robots in the simulated environment. This experiment has been adopted from another scientific article which also explores possibilities of HyperNEAT. In this experiment, the additional outputs of the CPPN enable encoding of the connections among the neurons in the upper layer of the sandwich substrate and bias the neurons.

The advantage of encoding the inputs and outputs in both x-coordinate and y-coordinate is, that we can adjust the resolution of the substrate, so that it suits better for modules with various number of inputs/outputs. This approach is more flexible. However this also means, that there are more options of spreading the inputs and outputs of the modules in the space. For the concrete task, we might have to enumerate different options of spreading the inputs and outputs and choose the best option experimentally, which can be difficult task.

### 2.5.2.2 6-Dimensional Hypercube Substrate

6-Dimensional hypercube( $x_1, y_1, z_1, x_2, y_2, z_2$ ) allows us to query the CPPN for the weight of the connection between two arbitrary neurons in the 3-dimensional space. This formalism is interesting because the topologies of biological brains, including the human

brain, theoretically exist within its search space (Stanley, 2009). It allows us to develop ANN with multiple hidden layers.



**Figure 2.14:** *CPPN for the Six-Dimensional Hypercube substrate. Six inputs representing the position of two neurons in the three-dimensional space. Single output - the weight of the connection*

### 2.5.3 Algorithm Description

We have described all the parts of the framework needed for the HyperNEAT to evolve large-scale Artificial Neural Networks. We have described CPPNs which allow us to encode potentially huge patterns inside relatively small structure. We have shown how these CPPN can encode not only spatial, but also connectivity patterns. Therefore they can encode also ANNs. We have also described in detail the NEAT algorithm which is able to evolve CPPNs. HyperNEAT utilizes NEAT for evolving CPPNs and then uses them to query for connections in concrete ANN, which structure we aim to evolve. It evaluates the CPPN depending on the performance of the ANN which connections were determined by querying the CPPN. This way HyperNEAT can evolve large-scale ANNs while using limited amount of computational resources.

# Chapter 3

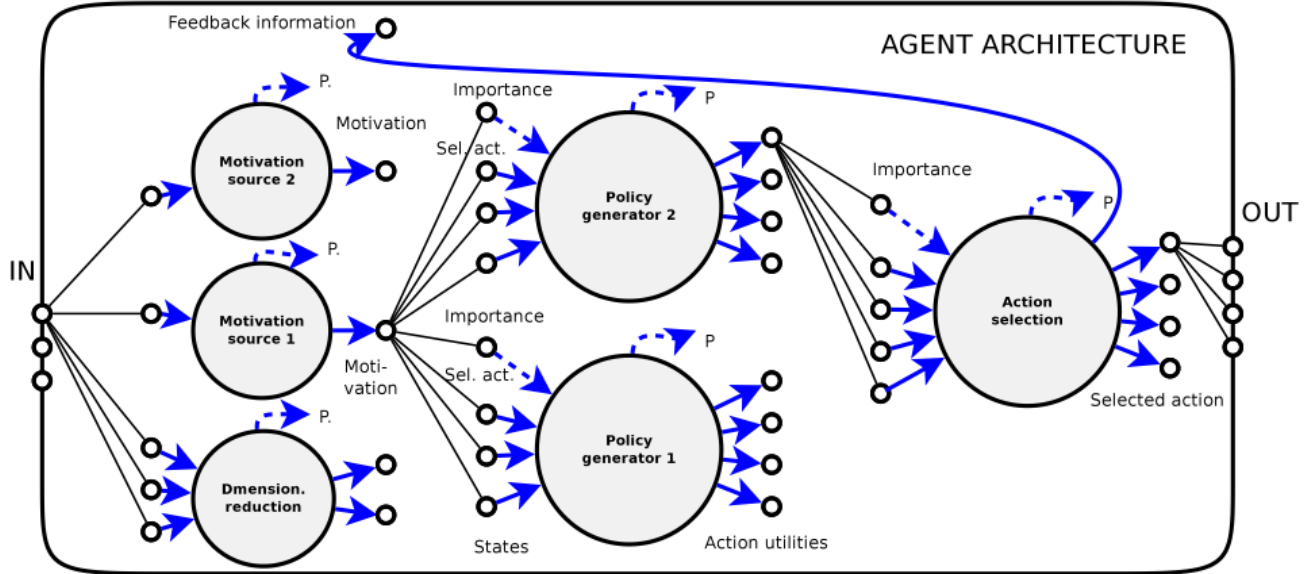
## Algorithms Proposed and Tested

In this chapter, the extension of the HyperNEAT for the HANNS will be explained in detail. The different ways of utilizing the HyperNEAT for HANNS will be discussed and each approach will be evaluated and compared with the other approaches. The solution for different substrates will be included.

### 3.1 Hybercube-based NeuroEvolution of Augmenting Topologies for Hybrid Artificial Neural Network Systems

The HANNS unlike standard ANN is built up from the Neural Modules. These modules have concrete internal structure, which is encapsulated inside of the Neural Module. In the concrete HANNS, they are approached as if they were black boxes, which provide mapping between  $n$  inputs and  $m$  outputs. Example of the single Neural Module is depicted on the figure 3.1. As can be seen on the figure, the internal structure of the Neural Module can be sophisticated, in fact it can even incorporate autonomous ANN with lots of neural connections solving concrete task. In the feed-forward HANNS, there are multiple Neural Modules in each layer. They accept inputs from the previous layer of the HANNS, and provide outputs to the following layer. The whole system accepts external inputs and provides external outputs for the concrete task. The structure looks intuitively similar to the ANN, but Neural Modules can typically operate on completely different principles. In addition they receive arbitrary number of inputs and provide





**Figure 3.1:** An example of Hybrid Artificial Neural Network System, which uses feed-forward topology. particular Neural Modules are placed in layers and are fully connected between layers. Each Module as given number of inputs, outputs (potentially configuration inputs too) and one prosperity output (see further in the text).

arbitrary number of outputs to the next layer of the HANNS.

HANNS can be evolved using the evolutionary algorithm with direct encoding (Vitku and Nahodil, 2014). This algorithm will be later compared with the extension of the HyperNEAT algorithm. we will name the EA used in the publication as "Basic EA", to distinguish it verbally from other evolutionary algorithms.

However in case of the large scale HANNS it would not be feasible to use the direct encoding technique. In addition, if the pattern encoded by the network exhibits geometrical properties, which the HANNS provides (symmetry, imperfect symmetry, etc.), it could be suitable to use this method instead of an evolutionary algorithm with the direct encoding. In general, the HANNS can contain various types of the Neural Modules in each layer which can complicate the task of finding the right topology significantly.

In the next sections, we will use the term **interlayer** for the connections between the outputs of the Neural Modules from one layer to the inputs of the Neural Modules of the next layer of the HANNS.

## 3.2 Sandwich Substrate

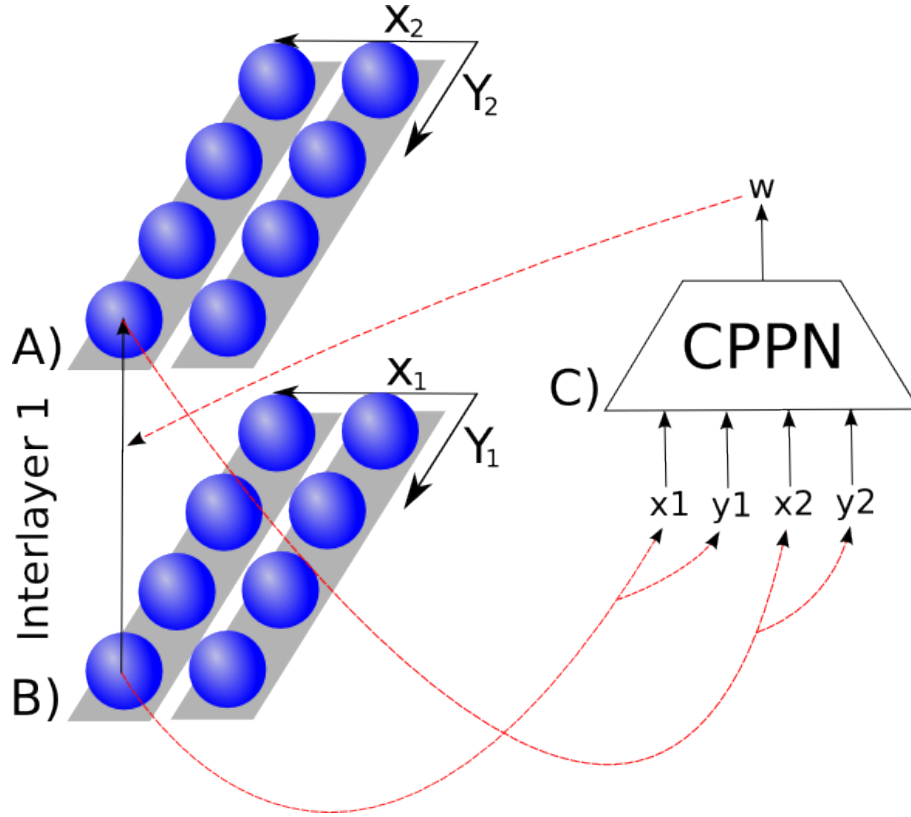
Firstly, let us focus on the HANNS without hidden layer. By this we mean a structure, which has given number of inputs connected to the input layer of the Neural Modules. This layer is then connected to the output layer of other Neural Modules by given connections. Because this kind of HANNS does not have hidden layer, it in fact uses single interlayer of connections between Neural Modules. The output layer of HANNS then provides external outputs of the whole system.

If we were given a HANNS restricted in such way, that each subsystem has exactly one input and one output, we would be able to use the HyperNEAT with sandwich substrate and would not have to develop different method. However, we want the algorithm to evolve HANNS without restricting the type of Neural Modules they incorporate. In this section we will discuss the ways, how to evolve these HANNS using HyperNEAT.

### 3.2.1 Encoding Additional Inputs and Outputs of each Neural Module as a Vector

In this case, each of the modules of the layer has organized inputs/outputs in separate vector. We have to use CPPN with 4 inputs  $(x_1, y_1, x_2, y_2)$ . The  $x_1$  coordinate stands for the index of the Neural Module in the input layer.  $y_1$  is given by the index of the output of the neuron  $x_1$ . similarly  $x_2$  represents the index of the Neural Module in the output layer and  $y_2$  index of the concrete input of this neuron. This way we organize the multiple inputs/outputs of each subsystem in the y-coordinate as a vector. In the two-dimensional space, x-dimension is divided equally among the Neural Modules. The y-dimension is sampled for each neural module individually depending on the number of inputs or outputs of the neuron and depending on whether it is input or output layer of the HANNS. If we want to have only feed-forward connections in this HANNS, we can use single-output CPPN. If we want to incorporate recurrent connections, we can achieve this by using multiple-output CPPN. With multiple output CPPN we can assign different outputs for feed-forward connections and recurrent connection. This principal was mentioned earlier in the robot motion example (Drchal et al., 2009).

The advantage of this approach is, that the mapping of the inputs and outputs of the Neural Modules is straightforward. We simply generate vector of inputs and outputs for each subsystem and do not have to consider how to spread these inputs and outputs in the space. Trade-off for this is, that if the number of inputs or outputs in the concrete layer



**Figure 3.2:** *HyperNEAT for the HANNS using sandwich substrate. Additional inputs and outputs are encoded as a vector by y-coordinate of the layer. A) - output layer of the HANNS. B) - input layer of the HANNS. C) - CPPN used to query for the weight of the connection between subsystems from input layer and subsystems from output layer. Grey rectangles - individual Neural Modules of the HANNS. Blue circles - the inputs and outputs of these subsystems. In this case we have Neural Modules with 4 inputs and 4 outputs. Assuming that the size of the substrate space is normalized, that is  $x \in \langle -1, 1 \rangle$ . and  $y \in \langle -1, 1 \rangle$ , the input into the CPPN for the queried connection would be  $x_1 = -1, y_1 = -1, x_2 = 1, y_2 = -\frac{1}{3}$ . This holds true, if we sample the space in a way, that the substrate space is distributed evenly among the inputs/outputs of the neural module. For example, in the input layer we have two subsystems. Their position differs in the x-dimension, so first one receives  $x = -1$  and second  $x = 1$ . Each of them have four outputs. So if we sample them equidistantly along the y-dimension, we get values  $y \in \{-1, -\frac{1}{3}, \frac{1}{3}, 1\}$  for the outputs of the subsystems.*

of the HANNS vary significantly, the resolution for the subsystem with lesser number of inputs/outputs will be too large, while the resolution for the subsystem with bigger number of inputs/outputs will be too small. However, if we have Neural Modules with similar number of inputs/outputs, this can be feasible solution.

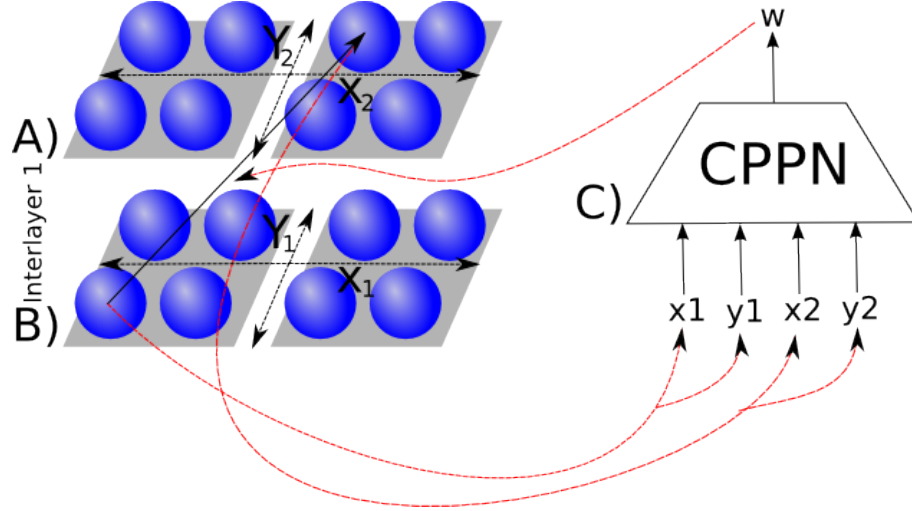
### 3.2.2 Encoding Additional Inputs and Outputs of each Neural Module as a Matrix

Another option of organizing the multiple outputs and inputs of the Neural Modules in the HANNS is to spread the inputs and outputs of each Neural Module into the two-dimensional space. Consequently, we do not get single vector for each neural module, but instead we get two-dimensional matrix, because we use both x-coordinate and y-coordinate. This means, that the inputs and outputs of the single neural module will not differ only in the y-coordinate but also in the x-coordinate, depending on how we spread the inputs and outputs in the two-dimensional space. For the feed-forward network, we need single output of the CPPN. In case we want to add recurrent connections, we can increase the number of outputs of the CPPN.

The advantage of encoding the inputs and outputs in both x-coordinate and y-coordinate is, that we can adjust the resolution of the substrate, so that it suits better for modules with various number of inputs/outputs. This approach is more flexible. However this also means, that there are more options of spreading the inputs and outputs of the modules in the space. For the concrete task, we might have to enumerate different options of spreading the inputs and outputs and choose the best option experimentally, which can be difficult task.

## 3.3 6-Dimensional Hypercube Substrate

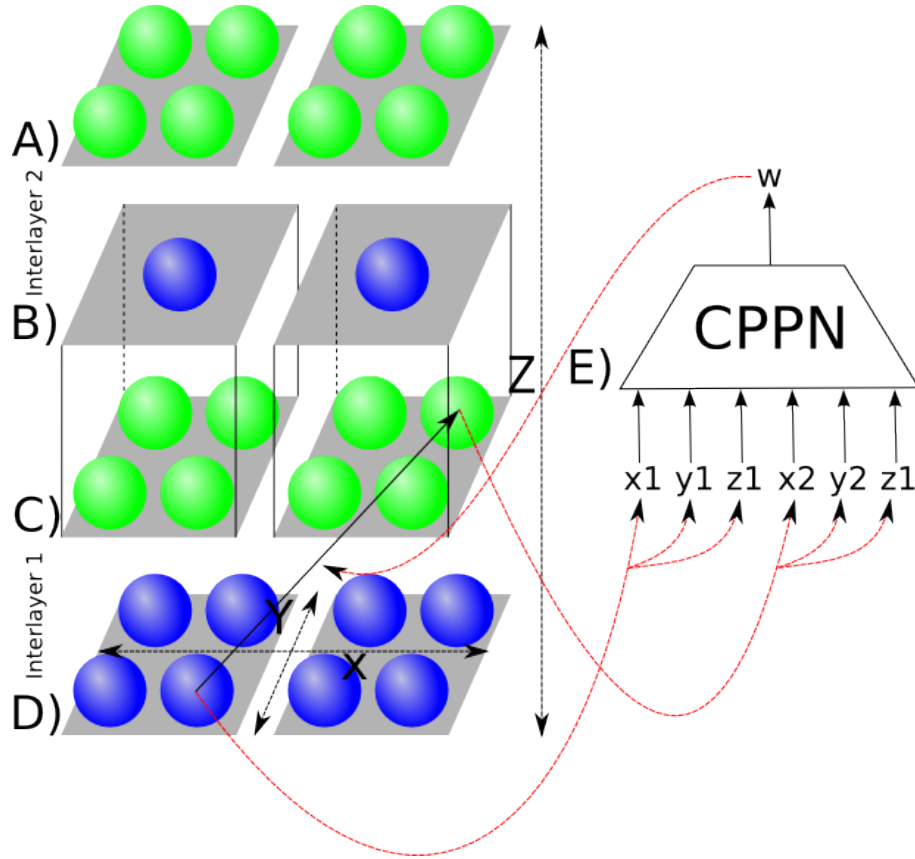
For the HANNS, which contain only input and output layer, the HyperNEAT with sandwich substrate is sufficient and we do not need to develop another approach using different substrate. However, in general we want to be able to evolve HANNS with arbitrary number of layers. To achieve this we must introduce more expressive substrate than sandwich. When we want to apply HyperNEAT on ANN with hidden layer, we can use 6-dimensional hypercube substrate. Corresponding CPPN receives 6 inputs ( for each point we feed the



**Figure 3.3:** *HyperNEAT for the HANNS using sandwich substrate. Additional inputs and outputs are encoded as a matrix by  $x$ -coordinate and  $y$ -coordinate of the layer. A) - output layer of the HyperNEAT. B) - input layer of the HANNS. C) - CPPN used to query for the weight of the connection between subsystems from input layer and subsystems from output layer. Grey rectangles - individual Neural Modules of the HANNS. Blue circles - the inputs and outputs of these subsystems. In this case we have Neural Modules with 4 inputs and 4 outputs. Assuming that the size of the substrate space is normalized, that is  $x \in \langle -1, 1 \rangle$  and  $y \in \langle -1, 1 \rangle$ , the input into the CPPN for the queried connection would be  $x_1 = -1, y_1 = -1, x_2 = \frac{1}{3}, y_2 = 1$ . The coordinates of the outputs of the first neural module from the left in the input layer would be following :  $\{(-1, -1), (-1, 1), (-\frac{1}{3}, -1), (-\frac{1}{3}, 1)\}$*

CPPN with the coordinates in the 3-dimensional space). One output is sufficient both for the feed-forward and recurrent networks, because of the z-coordinate. This coordinate is different for the neurons belonging to the different layers of the network. To apply this substrate to evolution of HANNS we must handle the obstacle of the multiple inputs and outputs of the Neural Modules. This can be done in the similar way as mentioned in the previous subsection where we dealt with the sandwich substrate for HANNS. The difference is, that for all hidden layers, we have to spread both inputs and outputs of the Neural Modules, instead of spreading only inputs or outputs. That means, that each subsystem will receive certain part of the 2-dimensional space, and spread its inputs evenly in this space. Then it will spread the outputs in the same space. Both inputs and outputs will be therefore encoded in the 2-dimensional matrix, or as a vector depending on the chosen spreading. CPPN will receive 6 inputs. Input  $(x_1, y_1, z_1)$  will correspond to the concrete output of the subsystem from the layer  $z_1$ . Input  $(x_2, y_2, z_2)$  will correspond to the concrete input of the subsystem from the layer  $z_2$ . For the 6-dimensional hypercube, the z-dimension will be sampled evenly for each interlayer of the HANNS.

We have successfully designed a way, to encode any given HANNS in a 6-dimensional hypercube. If we have only 2-layer network(with single interlayer), we can use sandwich substrate method, which is simpler. However if we need to evolve more sophisticated HANNS we can still do that using the 6-dimensional hypercube substrate.



**Figure 3.4:** *HyperNEAT for the HANNS using 6-dimensional hypercube substrate. A) - output layer of the HANNS . B) - outputs of the hidden layer of the HANNS. C) - inputs of the hidden layer of the HANNS. D) - input layer of the HANNS. E) - CPPN used to query for the weights of the connections between inputs and outputs of the Neural Modules. Blue circles - outputs of the Neural Modules. Green circles - inputs of the Neural Modules. Each cube represents single neural module from the hidden layer. As we can see the subsystems in the hidden layer have 4 inputs and single output. Subsystems in the output layer have 4 inputs each and subsystems in the input layer have 4 outputs each. Assuming that the size of the substrate space is normalized, that is  $x \in \langle -1, 1 \rangle$ ,  $y \in \langle -1, 1 \rangle$  and  $z \in \langle -1, 1 \rangle$ , the input into the CPPN for the queried connection would be  $x_1 = -\frac{1}{3}$ ,  $y_1 = -1$ ,  $z_1 = -1$ ,  $x_2 = 1$ ,  $y_2 = -1$ ,  $z_2 = 0$ . Note : although from the picture, it seems that the  $z$ -coordinate of B), and C) differs, it is not true. They are equal. The  $z$ -coordinate is always equal for the inputs and outputs of Neural Modules from the same layer.*

# Chapter 4

## Algorithm Implementation and Testing

In this chapter I justify the choice of the concrete HyperNEAT algorithm implementation. This implementation is further extended, to operate on HANNs as well as ANNs. In the later sections I describe experiments which have been conducted to observe the performance of the implemented algorithm.

### 4.1 HyperNEAT for HANNs Algorithm Implementation

From the various implementation of standard HyperNEAT algorithm, which can be found on the HyperNEAT users webpage <sup>1</sup>, I have chosen Java library called "Another HyperNEAT Implementation (AHNI)" (Coleman, 2010) as the basis for implementation of the HyperNEAT for HANNs. AHNI provides efficient implementation of the algorithm with the possibility of the parallel processing of the evaluation function for the concrete ANN. Because of its object oriented architecture, it can be extended for conducting experiments on custom Artificial Neural Networks. With some additional coding it can serve as the basis for the experiments on hybrid artificial neural networks as well. The parameters of the evolution and evolutionary operators can be modified and different results can be observed. The whole evolution is finished, once the network with desired performance

---

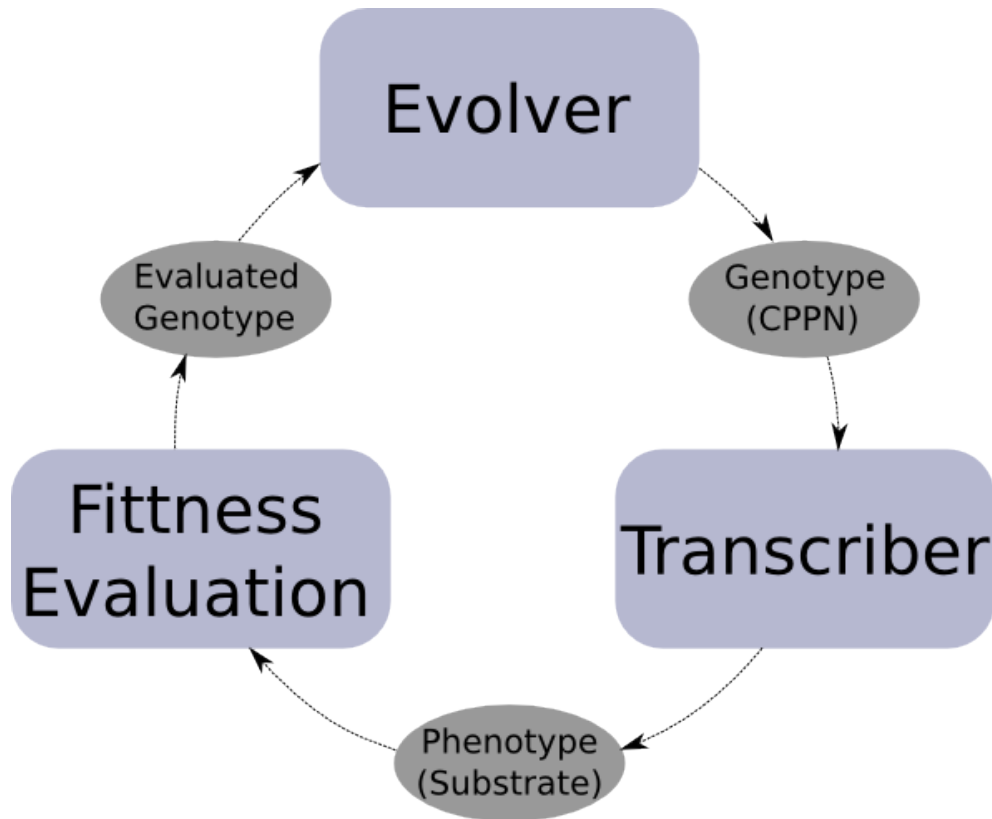
<sup>1</sup>HyperNEAT Users Webpage: <http://eplex.cs.ucf.edu/hyperNEATpage/>



has been developed or after given number of generations.

### 4.1.1 Proposed Library Extension

AHNI performs several evolution runs until it achieves desired precision of the solution on the given training set, or until the given number of iterations expires. Each iteration starts with the given population. In this case the genotype stands for the set of the compositional pattern producing networks. These are then transcribed to the phenotype corresponding to the set of substrates with assigned weights. These substrates are then evaluated using the fitness function either sequentially or simultaneously using the parallel fitness evaluation function. After that the evolutionary operators are performed on the evaluated individuals and they are pushed to the next iteration of the algorithm. The main loop of the implementation is depicted on the figure bellow.



**Figure 4.1:** *Evolutionary cycle of the HyperNEAT algorithm*

In order to evaluate the hybrid artificial neural networks using the given framework, custom substrate is generated which contains information about the weights of the connections between the modules of the network. For the HANNNS with sandwich substrate,

the use of the CPPN with 4 inputs  $(x_1, y_1, x_2, y_2)$  is sufficient. For the arbitrary feed-forward or recurrent HANNS, CPPN with 6 inputs  $(x_1, y_1, z_1, x_2, y_2, z_2)$ , where  $z$  stands for the index of the layer can be used. The framework provides the possibility of evolving both of these CPPNs, so we just have to design and implement the transcriber from the genotype to our custom substrate. For every experiment, custom evaluation function has to be coded, which will evaluate the performance of each of the substrates on the given problem. The properties ".prop" file used in this program allows us to modify the parameters of the evolution, CPPNs and substrates. This allows us to conduct various experiments in order to properly determine the performance of the extended HyperNEAT algorithm.

## 4.2 Experiments Background

All discussed experiments have been conducted using the extension of the AHNI java implementation (Coleman, 2010). Corresponding hybrid modular systems have been implemented in the NengoROS framework, they have been adopted from other experiments conducted by Jaroslav Vitku (Vitku, 2015). The nodes of the system are implemented as autonomous processes in the robotic operation system (Quigley et al., 2009). They exchange data by sending and receiving ROS messages. Simulator NengoROS decodes these ROS messages into the vectors of float values, which are then used for interconnecting the Neural Modules. When evaluating the concrete genome in the genotype, the information about the topology of the network is pushed to the simulator, which modifies the topology of the system correspondingly. Afterwards it launches simulation for the selected number of steps. When the simulation is finished, the performance is measured either by separate node of the system (evaluator) or by combining the Prosperity outputs of the nodes. The Prosperity output is a subjective heuristics which tries to define how well is particular Neural Module used in the current system. In the following part, the concrete experiments will be described, and evolutionary results will be depicted.

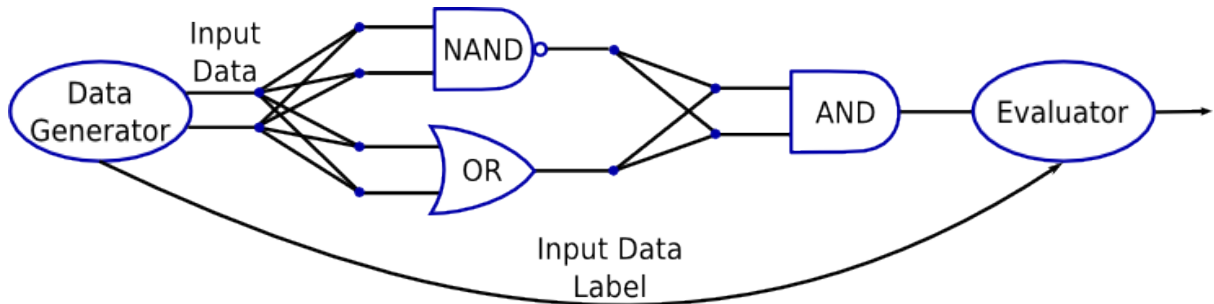
### 4.3 Experiment 1 - Implementing XOR Logic Function Using HANNS

The first set of experiments tested the optimization of the weights of the connections in the HANNS, which is capable of simulating the Exclusive-OR logic function. This kind of experiment has been very popular especially when designing classifiers in AI, because the input dataset is lineary inseparable so it can not be solved by ANN without hidden layer, without mapping the input data into the higher dimension.

The HANNS consist of five types of the MIMO subsystems. Three of them logical-AND, OR and NAND logic gates, one input data generator and one evaluator - mean square error calculator. The input data generator sends the inputs into the input layer of logic gates and the expected output to the evaluator. The output layer of the logic gates sends the outputs to the evaluator and the results are compared. The processing of a single input until the evaluator corresponds to the single time step of the simulator. The simulator runs in several time steps and the fitness value of the concrete setup is calculated by the formula:

$$fitness = \frac{\sum_{i=1}^N output = input\_label}{N} \quad (4.1)$$

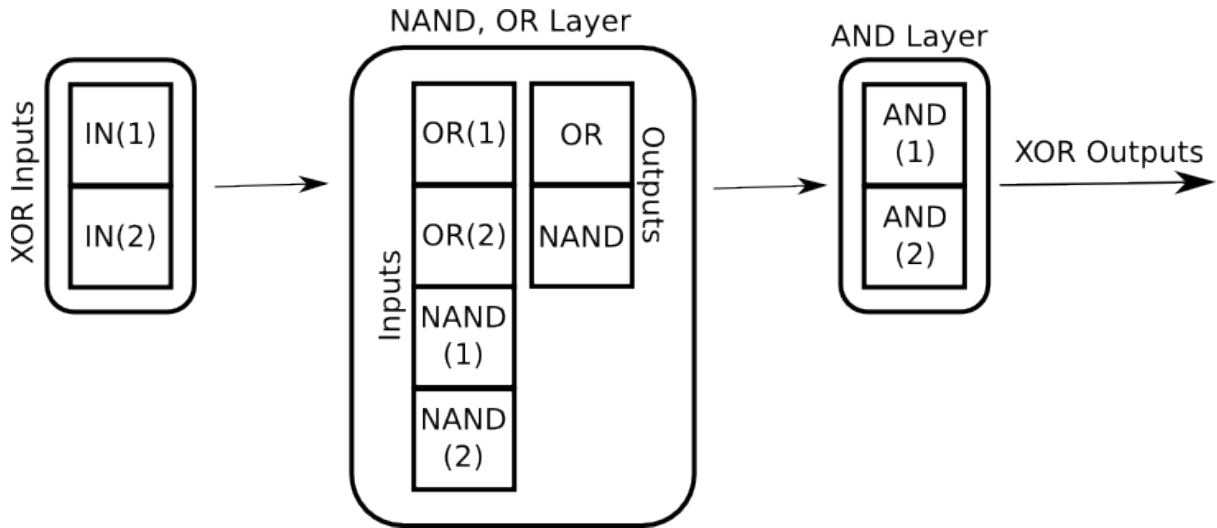
,where N=number of time steps of the simulator.



**Figure 4.2:** The structure of the HANNS simulating Exclusive-OR function. Whenever the sum of the weights leading to the single input of the concrete logic gate is greater or equal to 0.5, the corresponding input is evaluated as if it was equal to 1, otherwise the 0 input is taken.

I have conducted experiments for evolving the Exclusive-OR HANNS using simple vector input/output dispersion setting mentioned in the previous chapter, because its performance proved to be more efficient than the matrix substrate setting. The 6-dimensional

feed-forward hypercube substrate has been used, because the HANNS consists of 2 interlayers, so the sandwich substrate would be insufficient in this case. The results of the algorithm were evaluated and compared with the performance of the evolutionary algorithm named Basic EA implemented by my thesis supervisor (Vitku and Nahodil, 2014).



**Figure 4.3:** *The figure depicts the spreading of the inputs and outputs of the modules in the substrate. The inputs and outputs are simply ordered in a row forming a vector. This is the simplest setting but it proved to be more efficient than spreading the inputs and outputs of subsystems in concrete layer of the HANNS in a matrix.*

Three experiments from simplest to the more complicated HANNS consisting of more Neural Modules have been conducted. In the following subsections, the results from each of the experiments will be depicted and evaluated.

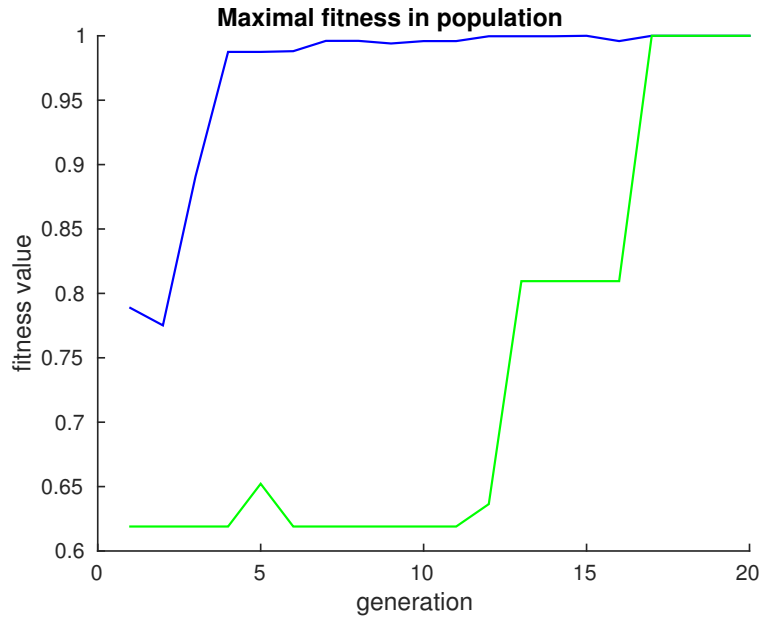
### 4.3.1 Experiment 1 - A) Simple HANNS XOR Problem

In the simplest version of the experiment, the structure of the HANNS corresponds to the one depicted in the Fig 4.2. There are single NAND and OR Neural Modules in the second layer of the HANNS and single NAND Neural Module in the third layer of the HANNS. The task for the EA is to find connection weights in order to approximate the XOR function (note that the communication between Neural Modules is continuous, discretization happens only inside the Modules). The results from the simple version of the experiment are presented below:

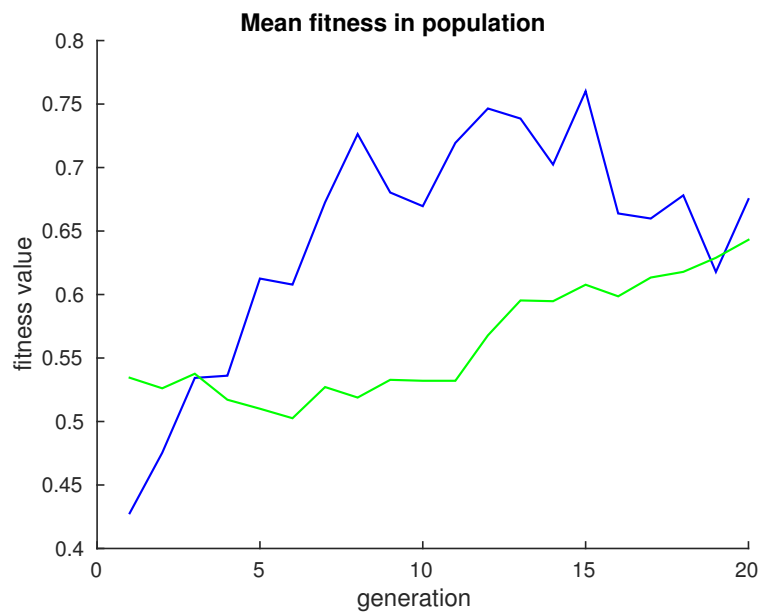
Weights Setting Legend	IN(1) $\rightarrow$ OR(1)	IN(1) $\rightarrow$ OR(2)
	IN(2) $\rightarrow$ OR(1)	IN(2) $\rightarrow$ OR(2)
	IN(1) $\rightarrow$ NAND(1)	IN(1) $\rightarrow$ NAND(2)
	IN(2) $\rightarrow$ NAND(1)	IN(2) $\rightarrow$ NAND(2)
	OR(1) $\rightarrow$ AND(1)	OR(1) $\rightarrow$ AND(2)
	NAND(1) $\rightarrow$ AND(1)	NAND(1) $\rightarrow$ AND(2)
Weights Setting Legend	IN(1) $\rightarrow$ OR(1)	IN(1) $\rightarrow$ OR(2)
	1.0	1.0
	1.0	1.0
	1.0	1.0
	1.0	1.0
	1.0	1.0
Weights Setting Legend	1.0	0.0
	0.27	0.03
	0.34	0.09
	0.24	0.75
	0.68	0.08
	0.92	1.0
Weights Setting Legend	1.0	1.0
	0.8	0.15
	1.0	1.0
	1.0	0.41
	0.43	0.19
	0.79	0.38

**Table 4.1:** *Best genomes found by the algorithms in the Exclusive-OR Experiment. The mapping of genomes to topology of the HANNS is shown in the Weight Setting Legend.*

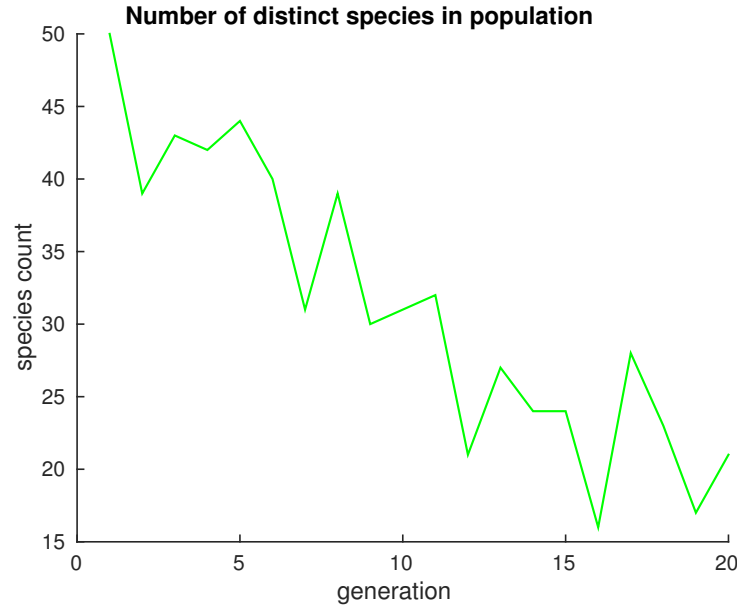
In this relatively simple task, the Basic Evolutionary Algorithm converged faster to the globally optimal weights setting. The reason for this can be, that the HyperNEAT algorithm performs better on tasks exhibiting some geometrical properties, which this task does not and every module has different functionality. The HyperNEAT algorithm might perform better on tasks, where the connections to the nodes are symmetrical, or repeat with small variations.



**Figure 4.4:** Comparison of the maximal fitnesses found by the Basic EA and HyperNEAT. Both algorithms had population size set to 50 individuals. Green - HyperNEAT, Blue - Basic EA



**Figure 4.5:** Comparison of the mean fitnesses found by the Basic EA and HyperNEAT. Both algorithms had population size set to 50 individuals. Green - HyperNEAT, Blue - Basic EA



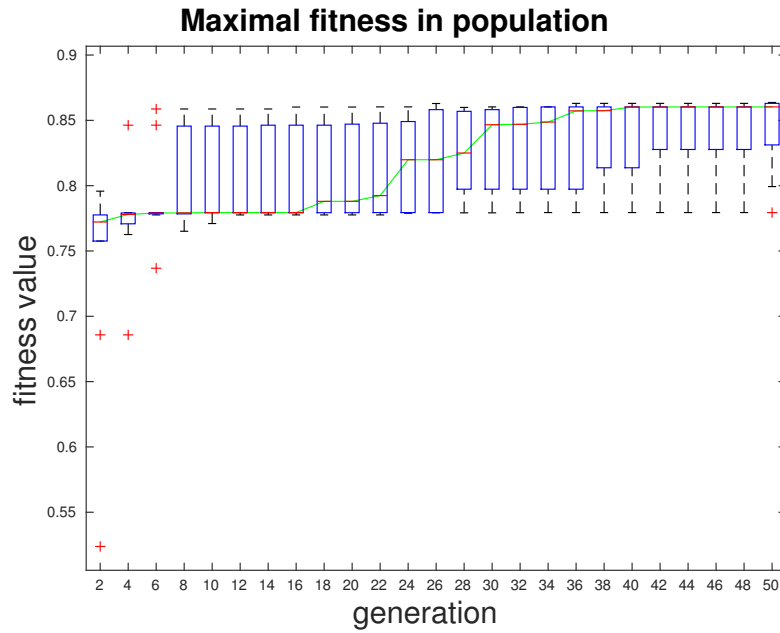
**Figure 4.6:** *Figure depicts the number of distinct species in the CPPN population throughout the evolution. Speciation target has been set to 5.*

Another reason for HyperNEAT to converge slower to the global optima could be, that the HyperNEAT algorithm started with the population of distinct individuals which evolved in the separate niches. It took few generations for algorithm to pick out fewer, more promising species and focus on their evolution. The graph depicts, how the number of distinct species has been reduced drastically throughout the evolution. The examples of the best found individuals from each algorithm are depicted bellow. From the found champion genomes it is obvious, that this particular problem can be solved by different topologies of the HANNS.

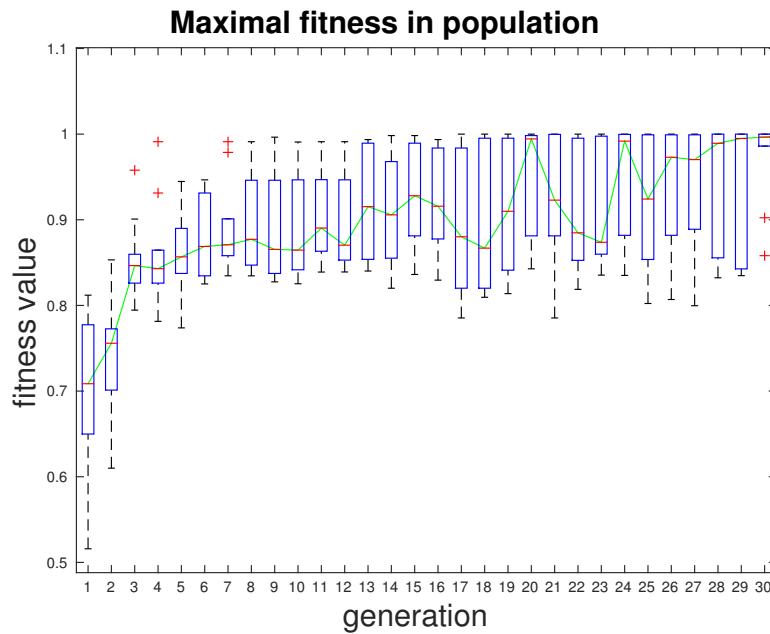
### 4.3.2 Experiment 1 - B) Medium HANNS XOR Problem

The evolved HANNS in this case had 6 Logic Neural Modules in the 2nd layer ( $2 \times OR$ ,  $2 \times NAND$ ,  $2 \times NAND$ ) and 6 Logic Neural Modules in the 3rd layer ( $2 \times OR$ ,  $2 \times NAND$ ,  $2 \times NAND$ ). Due to the extended number of Neural Modules, the HANNS consists of 3 interlayers. Other components of the HANNS (data generator and evaluator Neural Modules) remained the same as in the simple HANNS XOR experiment.

In the terms of the maximal fitness, the Basic EA performed better than the HyperNEAT algorithm. The results have been measured for 200 generations, but I depicted less generations on the figures. For HyperNEAT, after 50 generations, the algorithm got

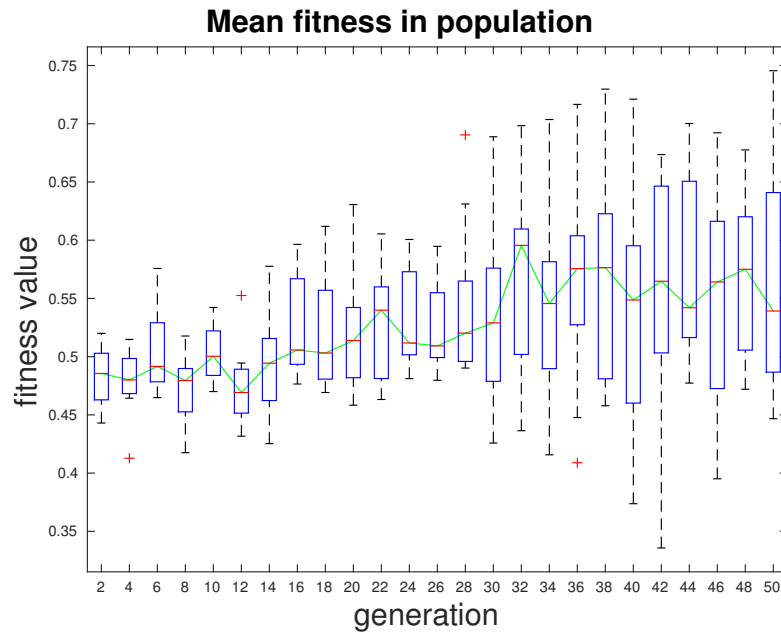


**Figure 4.7:** *The maximal fitness found by the HyperNEAT algorithm. 50 generations of the evolution are depicted. The results have been measured in 10 separate runs of the experiment.*

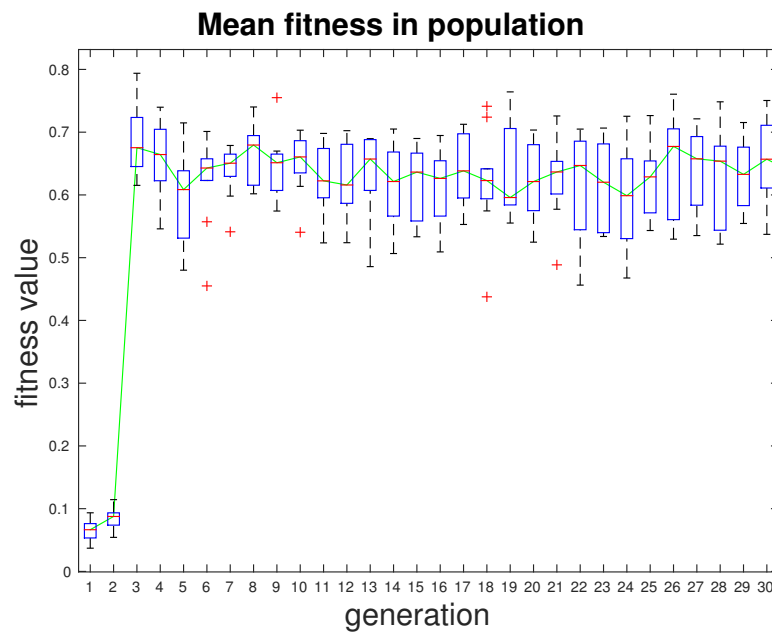


**Figure 4.8:** *The maximal fitness found by the Basic EA. 30 generations of the evolution are depicted. The results have been measured in 10 separate runs of the experiment.*





**Figure 4.9:** *The mean fitness found by the HyperNEAT algorithm. 50 generations of the evolution are depicted. The results have been measured in 10 separate runs of the experiment.*



**Figure 4.10:** *The mean fitness found by the Basic EA. 30 generations of the evolution are depicted. The results have been measured in 10 separate runs of the experiment.*

either stuck in suboptimal solution or slowly converged to the globally optimal solution. Because the results of the generations of 51 to 200 were very similar I decided to omit them in the graph and depict the interesting part of the evolution. For the Basic EA, there were 200 generations in the experiment, but after the 30 generations, the algorithm converged to the optimal solution.

The mean fitness of the population in case of the HyperNEAT was lower approximately by the value  $\in (0.1, 0.2)$  compared to the Basic EA. This is due to the fact, that HyperNEAT allowed less promising individuals from distinct species to evolve themselves. These individuals are kept in the population even though their fitness is lower. In the next subsection, the last and the most difficult XOR experiment in terms of the Neural Modules count will be described.

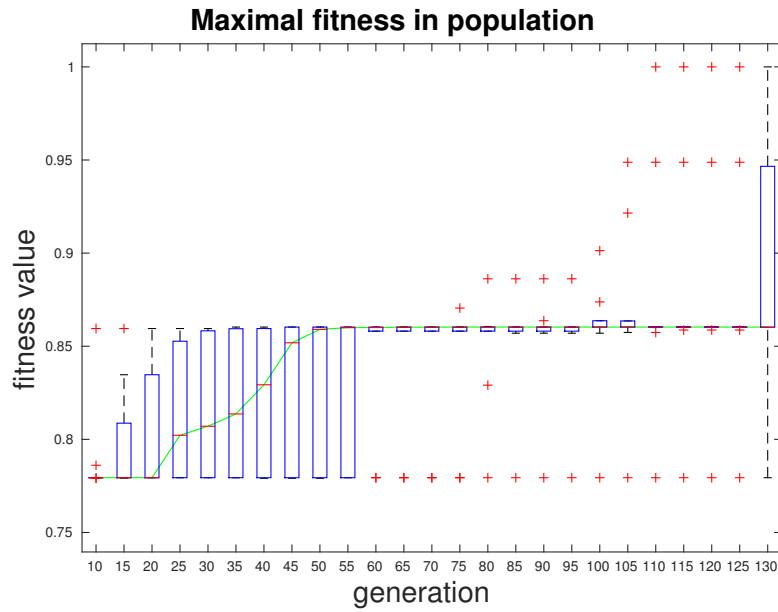
### 4.3.3 Experiment 1 - C) Large HANNS XOR Problem

The evolved HANNS in this case had 9 Logic Neural Modules in the 2nd layer ( $3 \times OR$ ,  $3 \times NAND$ ,  $3 \times NAND$ ) and 6 Logic Neural Modules in the 3rd layer ( $2 \times OR$ ,  $2 \times NAND$ ,  $2 \times NAND$ ). The system consists of 3 interlayers and again single data generator and single evaluator.

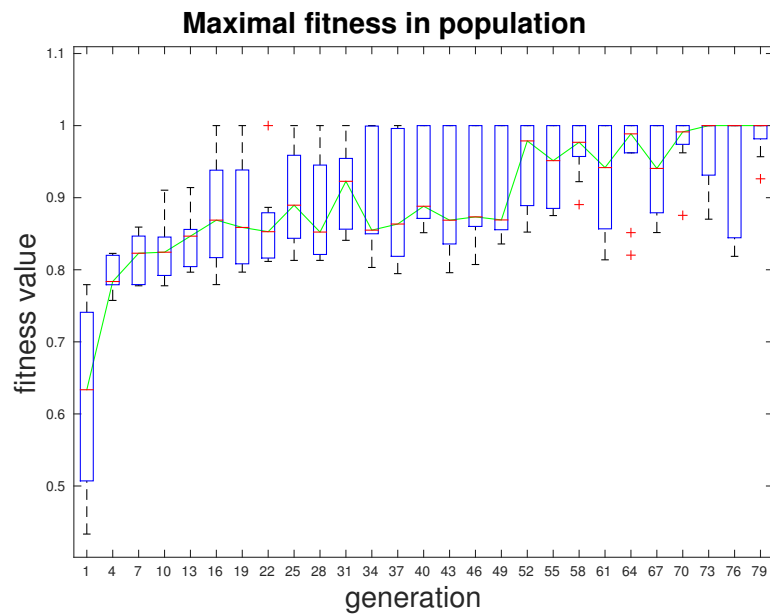
In the case of bigger HANNS to be evolved, the Basic EA also outperformed the HyperNEAT algorithm. The results have been extracted for 200 generations, but after the depicted number of generations, the algorithms converged to the optimal solution or got stuck in some suboptimal solution. The HyperNEAT algorithm did not always converge to the global optimum.

The mean fitness of the HyperNEAT was lower by the value  $\in (0.1, 0.2)$  compared to the Basic EA. This is not surprising due to the speciating of the population in the HyperNEAT algorithm.

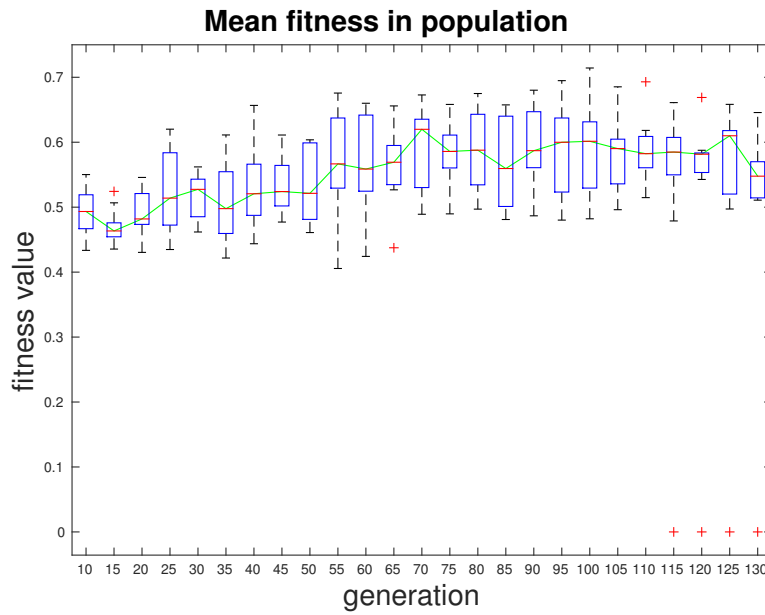
The reason for Basic EA to perform better than HyperNEAT in this case was, that HyperNEAT is more suitable for tasks with larger networks, which exhibit geometrical properties. This was not the case, for the XOR problem, the network did not exhibit geometrical properties and therefore the HyperNEAT could not benefit from the CPPN repeating patterns in the space.



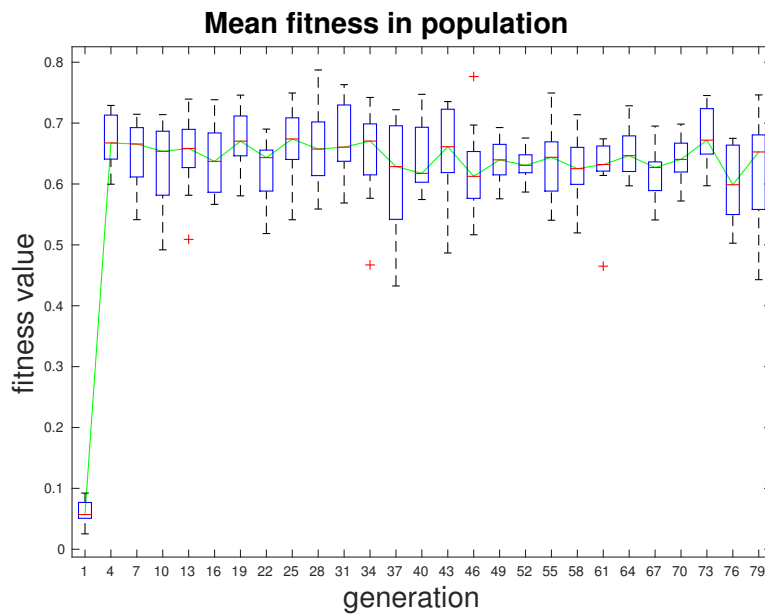
**Figure 4.11:** *The maximal fitness found by the HyperNEAT algorithm. 130 generations of the evolution are depicted. The results have been measured in 10 separate runs of the experiment.*



**Figure 4.12:** *The maximal fitness found by the Basic EA. 79 generations of the evolution are depicted. The results have been measured in 10 separate runs of the experiment.*



**Figure 4.13:** *The mean fitness found by the HyperNEAT algorithm. 130 generations of the evolution are depicted. The results have been measured in 10 separate runs of the experiment.*



**Figure 4.14:** *The mean fitness found by the Basic EA. 79 generations of the evolution are depicted. The results have been measured in 10 separate runs of the experiment.*

HyperNEAT parameters			
Population size	50	Elitism	0.05
Survival rate(parents %)	0.3	Crossover/Mutation Ration	0.5

**Table 4.2:** *The chosen parameters for the HyperNEAT algorithm in the Exclusive-OR Experiment. The detailed information about all parameters can be found in the corresponding experiments properties files. Elitism stands for the percentage of the champions surviving to the next generation.*

In the following part, the 2nd experiment will be explained and the results will be evaluated and compared to the Basic EA.

## 4.4 Experiment 2 - Motivation-driven Reinforcement Learning HANNS Description

Motivation-driven Reinforcement Learning HANNS is system designed on Faculty of Cybernetics of CTU in Prague (Vitku and Nahodil, 2014). It uses two neural modules, which are interconnected. First one - **Reinforcement Learning Module**, implements modified version of the Q-Learning algorithm. This discrete algorithm learns desired strategy only by means of interaction with the environment based on actions produced rewards/punishments received (Vitku and Nahodil, 2014). It learns the strategy on-line by iteratively updating its Q-matrix, which maps state-action pairs to concrete utility value. The RL module is connected to the second module - **Motivation Source Module**. This module serves as a motivation source and represents the physiological state of the artificial being. Depending on the physiological state, the agent explores the environment randomly, or commits to the previously learned action. Both of these modules have prosperity values, which determine how well they perform in the given simulation (agent moves in the grid-world with obstacles and sources of reward).

If the agent behaves efficiently enough, the mean motivation produced by the physiology module is low. The mean motivation value can be expressed by the Mean State Distance to optimal conditions (SF), which is defined as follows (Vitku and Nahodil, 2014):

$$SF_t = \frac{\sum_i d_i}{i} \quad \forall i = 0, 1, \dots, t \quad (4.2)$$

where  $d_i$  stands for the distance of the state variable  $V_i$  from the optimal conditions of  $V = 1$ .  $SF_t$  is computed online for each simulation step. Since the Prosperity is indirectly proportional, its value is computed as:

$$P_t = 1 - SF_t \quad (4.3)$$

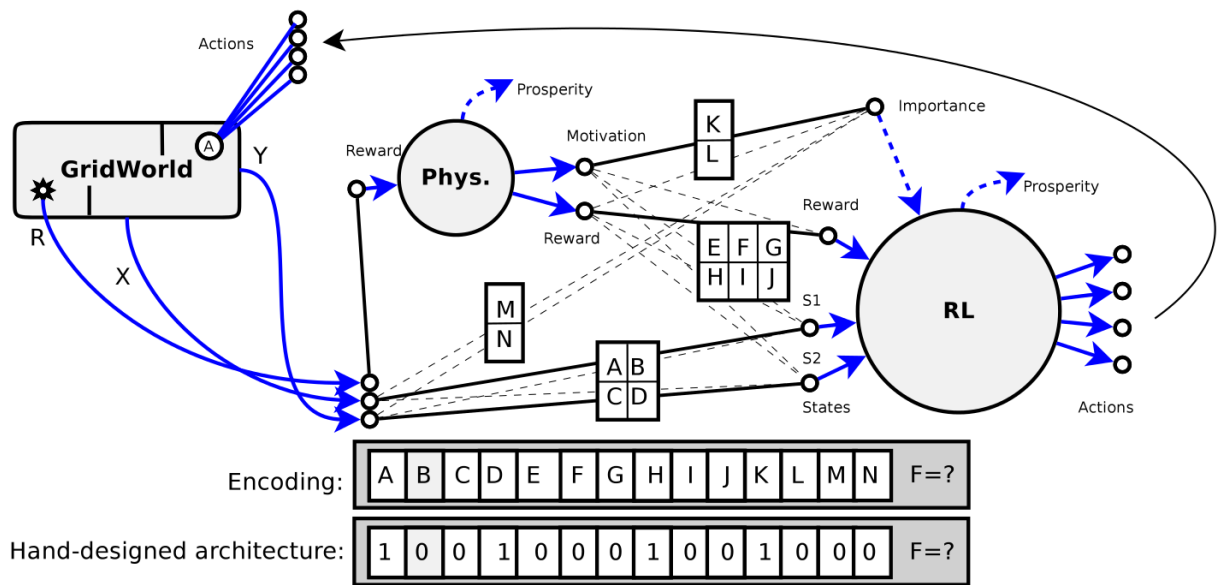
By optimization of the topology of this HANNS, higher physiology module prosperity values can be obtained. The fitness value in the experiments is defined by this prosperity value of the physiology module.

In the following part we will evaluate designed algorithms for optimizing the topology of the HANNS (by measuring of the prosperities of the physiological module in the simulator of the artificial environment) and compare them with the simple Evolutionary (EA) designed by Jaroslav Vitku.

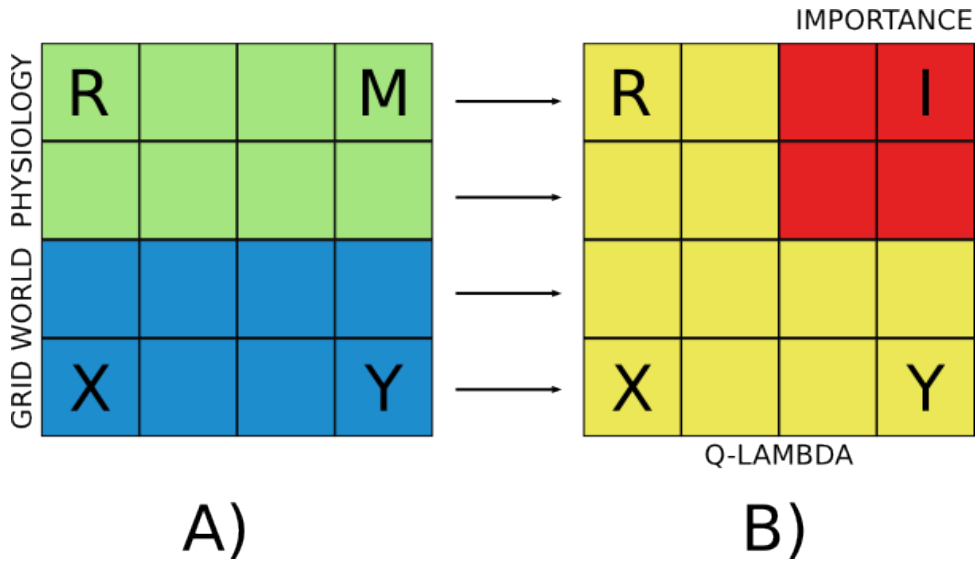
#### 4.4.1 HyperNEAT Parameters Setting Justification

I have successfully tested the ability of the modified HyperNEAT algorithm to learn the weights of Motivation-driven Reinforcement Learning HANNS on two concrete experiments. The evaluation of single individual was computationally expensive, because the simulation had to run for several thousand steps to test the performance of the modules correctly. Therefore I have chosen the evolutionary parameters of the algorithm by designing simple 'trick' experiment (although it is not generally correct), which measured the distance of the evaluated vector of the weights from the intuitive hand-wired weights setting. After several runs of the experiment, the best setup of the substrate and HyperNEAT parameters has have determined.

Although the complete HANNS has more than two layers, the optimization of weights between two layers of the HANNS were tested and other parts of the system have been hand-wired. Sandwich substrate was therefore chosen for this concrete task. As discussed in the preceding chapter, there are different ways of spreading the inputs and outputs of the particular modules into the 2-dimensional space. In the input layer, each module received sub-matrix of the 2-dimensional space with the same size. In the output layer, the reward and importance occupied the same sub-matrix, although they belong to the separate modules. This kind of setup enabled more efficient evolution of the weights of the modular connections.



**Figure 4.15:** Example of mapping the genotype (vector of binary/real values) to the phenotype (working agent architecture). The Physiological Module is wired to the reward source in the map, this determines the main goal of the agent (by the module’s Prosperity value). Outputs of Q-Lambda Module are directly wired to agents actuators. The genotype of the hand-designed architecture is depicted in the bottom and its connections are highlighted in the schematics. Variables representing the state of environment (X, Y coordinates) are connected to data inputs of Q-Lambda node. Reinforcement is connected to the Physiological Module, which produces motivation and reward for learning in the Q-Lambda Module (Vitku and Nahodil, 2014).



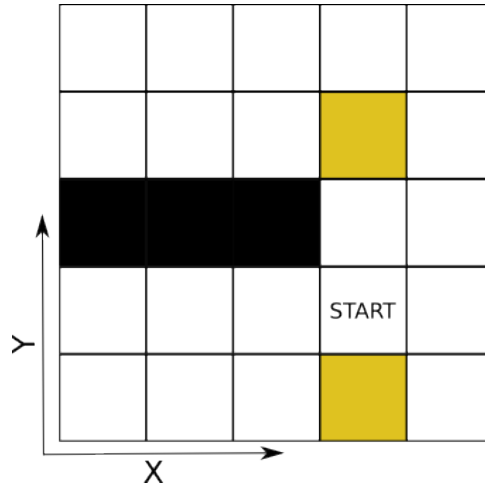
**Figure 4.16:** *The setup of the substrate in HyperNEAT for evolution of the Reinforcement Learning modular system. The dimensions of the sandwich were  $4 \times 4 \rightarrow 4 \times 4$ , A) - input layer of the sandwich, Green color - the sub-matrix of Physiology Module is depicted, Blue color - the sub-matrix of the output of the Grid World is depicted. B) the output layer of the sandwich substrate. Yellow color - the input space of the Q-Lambda RL module, Red color - The input space of the Importance module.*

In the following part I will evaluate the performance of the designed algorithm on two experiments with different instances of the gridworld and compare the results with the Basic EA (Vitku and Nahodil, 2014).

#### 4.4.2 Experiment 2 - A) Smaller Gridworld

In the first reinforcement learning experiment, the smaller gridworld has been chosen. The gridworld has 25 fields ( $5 \times 5$ ), two sources of reward and three obstacles. The number of steps of the agent in the environment has been set to 7000 for single evaluation run.



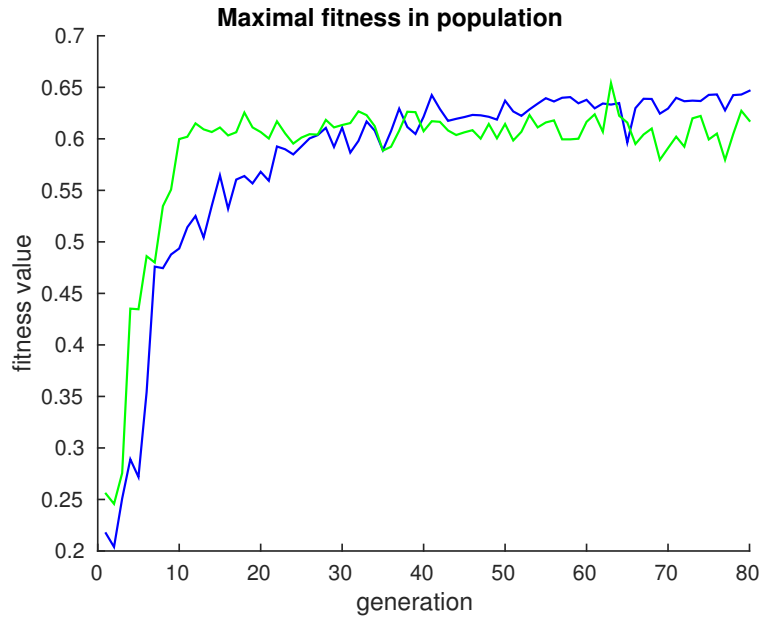


**Figure 4.17:** *Gridworld  $5 \times 5$  for Reinforcement Learning Experiment. Yellow - sources of reward. Black - obstacles, Start - initial position of the agent in the virtual environment*

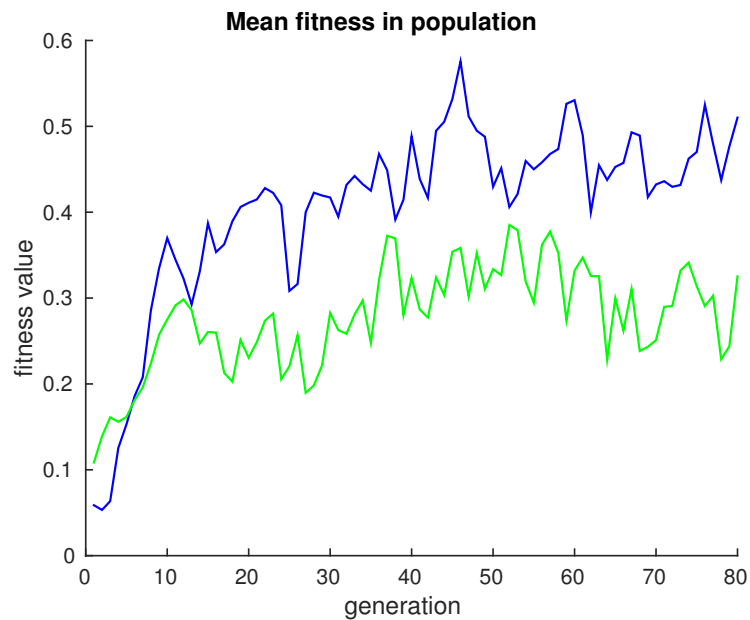
As can be seen in the graphs, both algorithms performed comparably in terms of the fitness of the champion in each generation of the evolution. Compared to the Basic EA, the HyperNEAT converged slightly faster to the solution with better fitness than the topology setup by hand. HyperNEAT algorithm had worse mean fitness in the population. This fact is due to the speciating the population into the niches, which improves the chance of different topologies to optimize their internal parameters.

From the graphs depicting the speciation of HyperNEAT run it can be observed, that the specified desired speciation has been achieved approximately 2 generations before the 10th generation, which evolved individuals with better performance than the hand-wired solution. I interpret this as correct speciation target setup, because it created space for evolution of fewer species, rather than creating distinct species, which would reduce the speed of perfecting other more promising species.

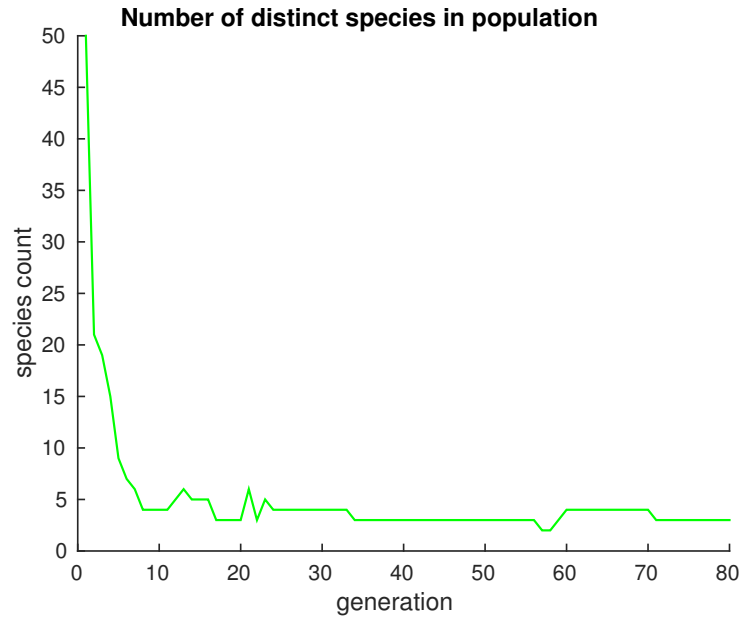
In this experiment, the topology by the HyperNEAT performed better than the hand-wired solution. The reason for this could be the fact, that the Y-output of the gridworld was connected to the importance value, which causes the agent to choose the learned action rather than exploring the world randomly. The learned action caused the agent to visit the state with the reward more often, which resulted in higher prosperity values of the Neural Modules and higher fitness of the individual. In the next part, performance of the algorithm in the bigger gridworld will be evaluated.



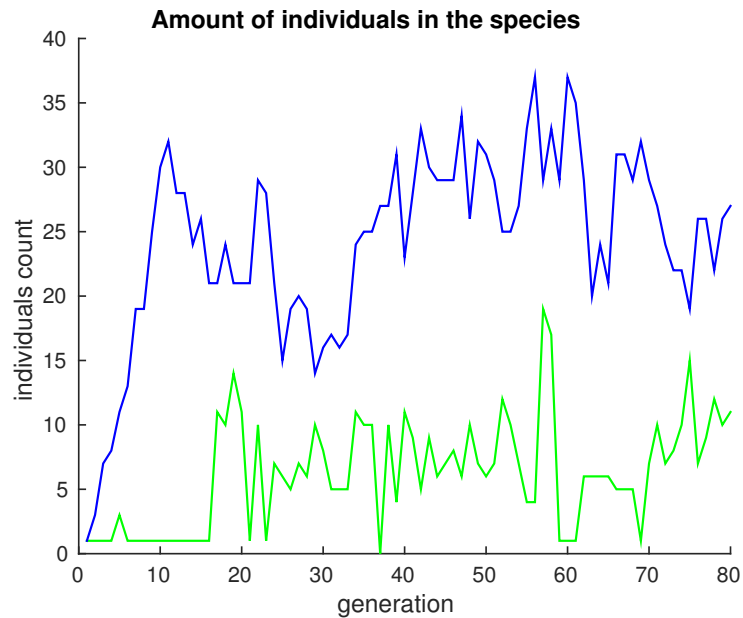
**Figure 4.18:** *The maximal fitness in the 80 generations of the Basic Evolutionary Algorithm (blue color) and HyperNEAT (green color) algorithm is depicted. The size of the population has been set to 50 individuals.*



**Figure 4.19:** *The mean fitness in the 80 generations of the Basic Evolutionary Algorithm (blue color) and HyperNEAT algorithm (green color) is depicted. The size of the population has been set to 50 individuals.*



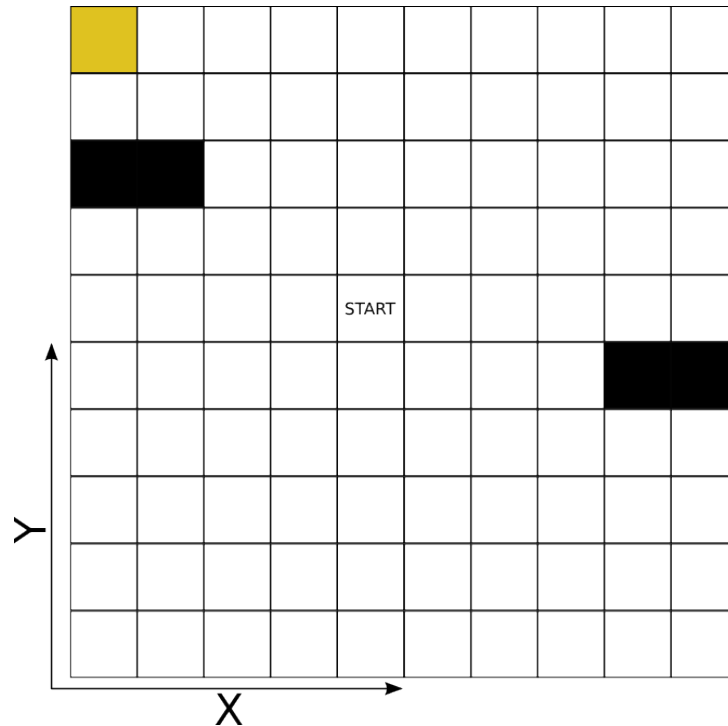
**Figure 4.20:** Figure depicts the number of distinct species in the CPPN population throughout the evolution. Speciation target has been set to 3.



**Figure 4.21:** The maximal and minimal size of the species in HyperNEAT is depicted. Blue- maximal specie size, Green -minimal specie size. The size of the population has been set to 50 individuals and speciation target has been set to 3.

### 4.4.3 Experiment 2 - B) Bigger Gridworld

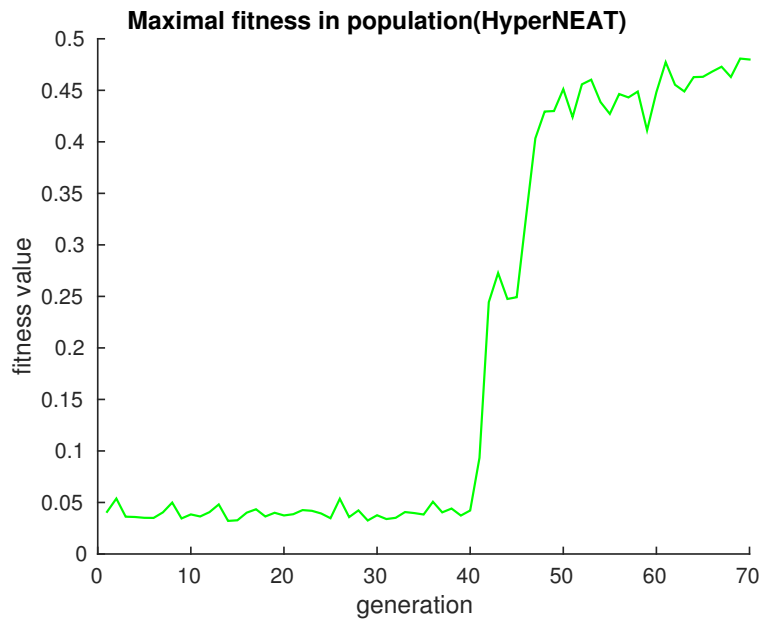
In the second reinforcement learning experiment, the gridworld with bigger size of 100 fields ( $10 \times 10$ ), single source of reward and four obstacles has been chosen. The number of steps of the agent in the environment has been set to 15000 for single evaluation run, to assure more precise evaluation function values.



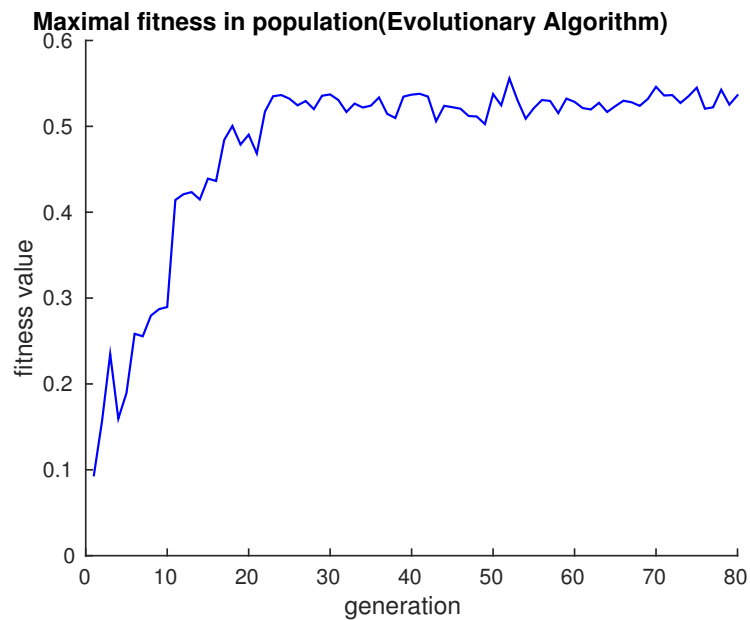
**Figure 4.22:** *Gridworld  $10 \times 10$  for Reinforcement Learning Experiment. Yellow - sources of reward, Black - obstacles, Start - initial position of the agent in the virtual environment.*

This gridworld differs from the smaller version in the size, but also in the number of reward sources. Instead of two, there is just single reward source which causes the found fitnesses to be smaller compared to the first smaller gridworld.

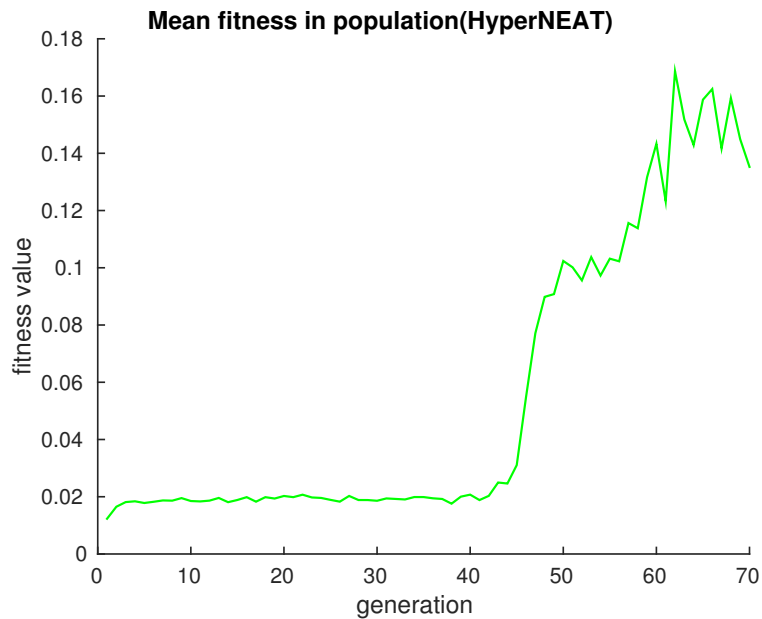
The Basic EA converged much faster to the desired value (close to the value generated by hand-wired setting). The HyperNEAT algorithm has been stuck in the local optima with low fitness value for 4000 evaluations but once it perturbed to solution with higher fitness it converged relatively quickly to the value close to the hand-wired setup of the algorithm.



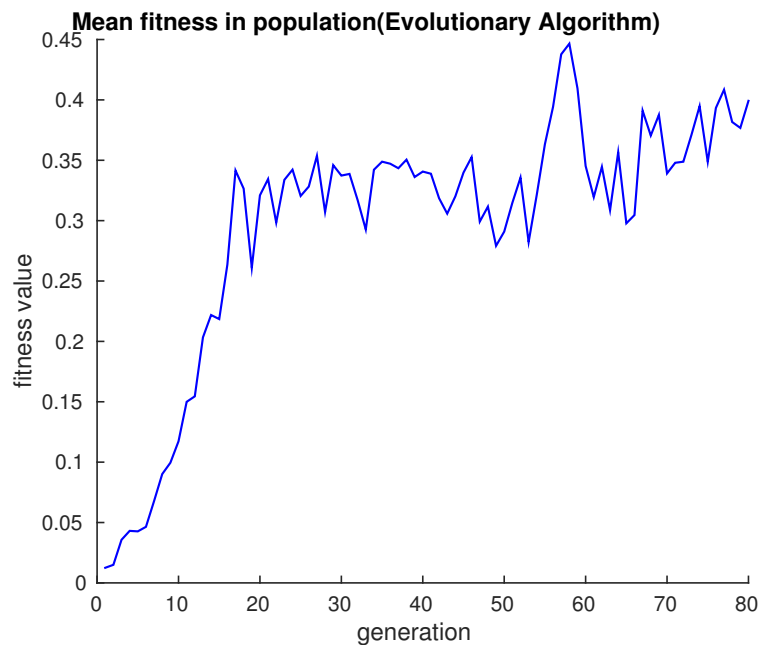
**Figure 4.23:** *The maximal fitness in 70 generations of the HyperNEAT algorithm is depicted. The size of the population has been set to 100 individuals.*



**Figure 4.24:** *The maximal fitness in 70 generations of the Basic Evolutionary Algorithm is depicted. The size of the population has been set to 50 individuals.*



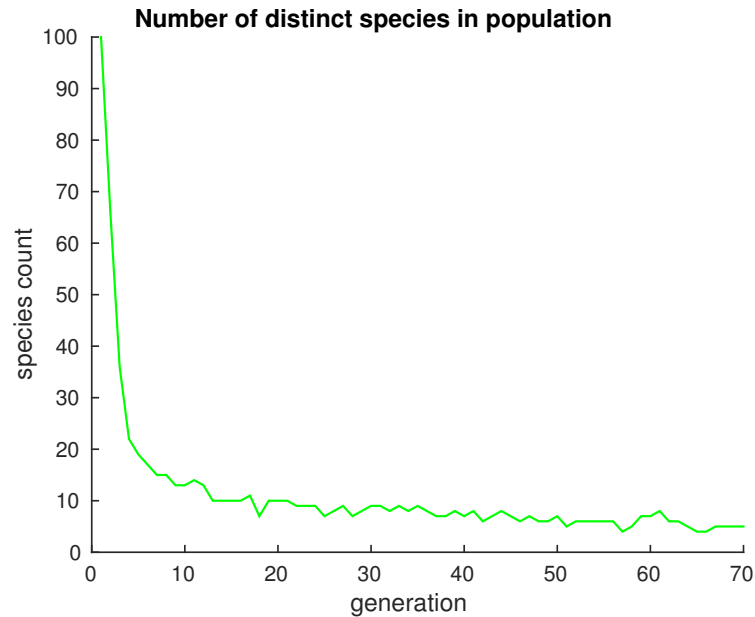
**Figure 4.25:** *The mean fitness in 70 generations of the HyperNEAT algorithm is depicted. The size of the population has been set to 100 individuals.*



**Figure 4.26:** *The mean fitness in 70 generations of the Basic Evolutionary Algorithm is depicted. The size of the population has been set to 50 individuals.*

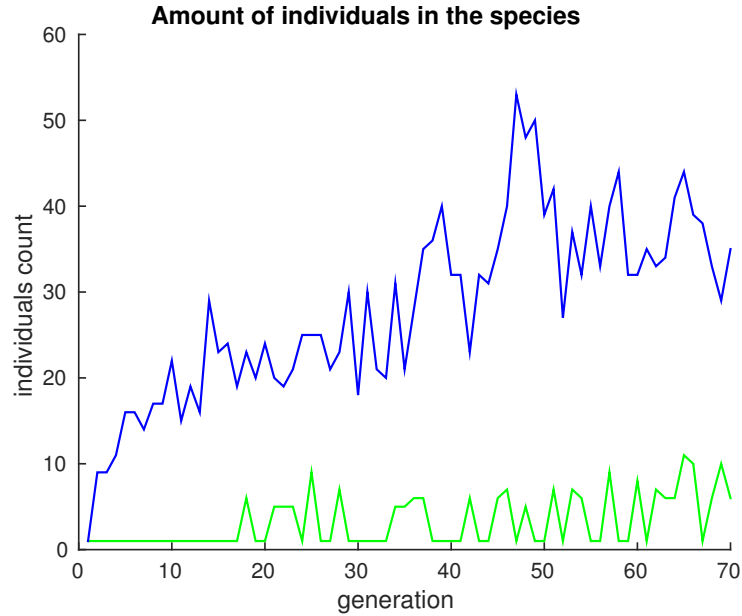
When comparing the mean fitnesses of the populations in both algorithms we can observe, that Basic EA has much higher mean fitness than HyperNEAT in this experiment.

The reason for this is, that HyperNEAT algorithm explores distinct solutions and does not disqualify the solutions with low fitness immediately, if they differ from the other individuals in genotype. This is due to the speciating, which is one of the basic features of the HyperNEAT algorithm.



**Figure 4.27:** *The species count throughout the evolution in HyperNEAT is depicted. The size of the population has been set to 100 individuals and speciation target has been set to 7.*

The algorithm converged to the desired species count after 10 generations. The reason, why desired species count has been set to 7 was, that with the higher number of species, algorithm evolved very large mutated individuals instead of focusing on perfection of the concrete specie. On the other hand, the reason for the algorithm being stuck in local optima for so long could be also insufficient exploration of the variety of the genomes. As we mentioned earlier, the parameters have been set using the 'trick' experiment with the hand-wired configuration. The evaluation function in that case is more precise and differs greatly from the evaluation in the simulated environment, that means that experiments are also different and the parameters setting in this case is not entirely precise. At this point it is therefore difficult to determine, whether the reason for the algorithm to converge relatively slowly compared to the Basic Evolutionary Algorithm could be in the selected speciation target.



**Figure 4.28:** *The maximal and minimal size of the species in HyperNEAT is depicted. Blue- maximal specie size, Green -minimal specie size. The size of the population has been set to 100 individuals and speciation target has been set to 7.*

Gridworld $5 \times 5$			
Population size	50	Elitism	0.05
Survival rate(parents %)	0.3	Crossover/Mutation Ration	0.8
Gridworld $10 \times 10$			
Population size	100	Elitism	0.05
Survival rate(parents %)	0.3	Crossover/Mutation Ration	0.8

**Table 4.3:** *The chosen parameters for the HyperNEAT algorithm in the Reinforcement Learning experiment. Elitism stands for the percentage of champions surviving to the next generation.*



#### 4.4.4 Reinforcement Learning Experiments Best Found Genomes and Discussion

See Fig. 4.15	A	B	C	D		
	E	F	G	H	I	J
	K	L	M	N		
Hand-wired, F = 0.55(Small),0.46(Big)	1	0	0	1		
	0	0	0	1	0	0
	1	0	0	0		
HyperNEAT Small Grid World, F = 0.65	1	0	0	1		
	0	0	0	1	0	0
	1	0.26	0	0.98		
EA Small Grid World, F = 0.65	0	1	0.94	0		
	0	0	0	0.948	1	1
	0.97	0	0.9	0		
HyperNEAT Bigger Grid World, F = 0.48	1	0	0	1		
	0	0	0	1	0	0
	0.93	0	0	0.68		
EA Bigger Grid World, F = 0.56	1	0	0.49	1		
	0	0	0	1	0	1
	0.7	0.92	0.02	0		

**Table 4.4:** *The Best genomes found in the Grid World Experiments.*

**HyperNEAT Algorithm, Small Grid World :** The motivation output of the physiology module, and the X output of the gridworld are connected correctly to the next layer of the HANNS, the same way as in the case of hand designed connection. The reward is connected to the reward input of Q-Lambda module as expected, but in addition it is connected to the importance input. This does not modify the behavior of the agent, because once it receives reward, the importance is set to 0 by default. The Y-output of the gridworld is connected to the Y-input of the Q-Lambda module, which is correct and in addition it is connected to the importance input, which causes the agent to execute the learned strategy more often, when he traverses the positions with higher Y-coordinate. The algorithm has successfully found the valid topology. Moreover, the

resulting automatically optimized system has better fitness than the manually designed one.

**Basic EA, Small Grid World** : The reward output of the physiology module is connected not just to the reward input, but also to the X and Y inputs of the Q-Lambda module. This causes the agent to get distracted once it reaches the state with reward. In the reward state, the importance equals 0 and the agent moves randomly in the environment, so the behavior of the agent remains the same. The motivation output of the physiology module is connected similarly to the hand wired version. The X and Y spatial outputs of the gridworld are connected to the Y and X inputs of the Q-Lambda module ( $X \rightarrow Y, Y \rightarrow X$ ). This causes the agent to have the reversed axis in his representation of the surrounding environment, but does not affect his behavior in terms of moving between the states in the gridworld. X output of the gridworld is connected to the importance input, which causes the agent to execute the learned strategy when he is situated in the position of the gridworld with higher X-coordinate. The algorithm has successfully found the valid topology. The resulting system has better fitness than the manually designed one, similarly as in the case of HyperNEAT algorithm.

**HyperNEAT Algorithm, Bigger Grid World** : The output of the physiology modules are connected according to the hand designed version of the reinforcement learning system. The outputs of the Grid world are connected to the inputs of the Q-Learning module as expected, the agent receives correct information about his current position in the artificial environment. In addition, the Y output of the system is connected to the importance of the system, which results in committing to the learned behavior, when situated in the higher Y-coordinate in the simulated environment. Similar pattern has been observed in the topology found by HyperNEAT in the smaller grid world. The algorithm has successfully found the valid topology and the efficiency of the connections is comparable to the hand wired topology.

**Basic EA, Bigger Grid World** : The reward output of the physiology module is connected to the reward input of the Q-Lambda module, in addition it is connected to the importance input and Y input of the Q-Lambda module. The connection to the importance input does not affect the learned behavior, because once the agent receives reward, the importance is set to 0 by default. The connection to the Y input causes the distraction of the agent in this round. Agent believes to be situated in the different state, or even outside of the gridworld. However, because he received reward, his motivation for movement towards the reward has been reduced to 0 and therefore he performs random walk in any case. Motivation output of the physiology module is setup correctly. X output

of the gridworld is connected purely to the X input of the Q-Lambda module which is desired. Y output of the gridworld is connected to the Y input of the Q-Lambda module but also to the X input of the Q-Lambda module. However the weight of this connection is  $\frac{1}{2}$ . It appears, that it does not distract the agent significantly, because the modules still exhibit promising prosperity values resulting in better fitness of the individual than the hand wired configuration.

When comparing the performance of the algorithms it has been observed, that the Basic EA algorithm outperforms the HyperNEAT algorithm by producing higher prosperity values of the modules in the larger gridworld. On the other hand the topologies evolved by HyperNEAT intuitively appear to be more correct, because they are much closer to the hand wired configuration of this HANNS. In case of the smaller gridworld, both algorithms found solutions with similar fitnesses, but the HyperNEAT solution is again closer to the hand wired configuration. The HyperNEAT algorithm also converged faster to the best found solution than the Basic EA. While bearing in mind that the HyperNEAT's benefits would manifest more in bigger topologies with more connections, this is still a good result (compared to the Basic EA).

## 4.5 Conducted Experiments Discussion & Conclusion

I have successfully conducted several experiments on the use of HyperNEAT for evolving HANNS. Different substrate settings have been evaluated and the effect of input and output spreading on the evolution has been observed. In the XOR experiment, it turned out that the simplest is the most efficient solution. Automatic ordering of the neural inputs and outputs resulted in the smoother evolution with faster results. On the other hand, in case of the Reinforcement Learning Experiment, spreading the inputs and outputs within the sub-matrix of the substrate happened to be more efficient. The CPPN networks generated separated patterns in the separated parts of the space, which resulted in better flexibility and independence of the weights. To summarize the findings, it is not always clear, which way of spreading of the inputs and outputs is better, the correct dispersion has to be determined depending on the concrete problem to be solved. In order to deepen our knowledge about the efficient settings in HyperNEAT for HANNS, more experiments should be conducted in the future.

## Chapter 5

# Thesis Conclusion and Contributions

In the final chapter of the thesis, I will summarize the thesis conclusions and the contribution to the development of the hybrid artificial neural network systems. I have successfully fulfilled all the goals stated in the thesis assignment:

- I have studied the principles of optimization of ANN based on the HyperNEAT algorithm in depth theoretically, but also had to examine the concrete chosen implementation in detail in order to extend it. By going through several articles on this topic mainly from the author of the HyperNEAT algorithm (Stanley, 2009), but also from others, who introduced novel thoughts into this domain, the strong theoretical background has been created. This part is described comprehensively in the 2nd Chapter of this thesis.
- The modification of this algorithm has been designed for the purpose of learning the topology of arbitrary HANNS. Since this is very general task, several possibilities of algorithm extensions have been examined and described in detail. For the simpler HANNS, simpler modifications have been designed to achieve efficiency. For complex HANNS, more general methodology has been found. To assure better insight into this problem, all the key ideas have been discussed with my thesis supervisor Jaroslav Vitku.
- The algorithm has been successfully implemented to serve for the purpose of evolving of the topology of HANNS. The implementation extended open source implementation of HyperNEAT algorithm (Coleman, 2010) chosen due to its extensibility and efficiency. The extended implementation became part of the framework for development of HANNS called NengoROS (Vitku, 2015). Within this framework, it

can be tested on the capabilities of interconnecting modules for hybrid modular systems.

- The algorithm has been tested on several experiments selected and designed by my thesis supervisor. The tasks in this experiments have been chosen from the simpler to the more difficult ones to determine the performance of the algorithm in various environments and problems. All the conducted experiments were compared to the to the Basic EA (standard evolutionary algorithm with direct representation of weights) for learning the topology of the HANNS. All the experiments are available on the attached DVD-ROM together with the extended HyperNEAT implementation incorporated into the NengoROS framework. In addition, the NengoROS project together with this implementation can be pulled from the NengoROS repository (Vitku, 2015), where you can find the manual for installing the environment for concrete experiments on the linux-based operating system.

The main contributions of the thesis are in the development of novel methods for learning the connections of arbitrary HANNS using the HyperNEAT algorithm. All of the methods have been evaluated and results proved, that the algorithm is capable of optimizing the topology of the system, even when the topology does not exhibit the geometrical properties, which increases the performance of the HyperNEAT algorithm. By implementing this algorithm into the Framework of Hybrid Artificial Neural Network Systems I have created background for conducting more promising experiments. In recent years, the focus on the research and development of these systems has increased rapidly. Because these methods have been designed to work in general with arbitrary HANNS, they can be further tested and utilized on the various tasks.

# Bibliography

- Auda G., Kamel, M. (1999). Modular neural networks: a survey, *International Journal of Neural Systems* **9**: 129–151.
- Bennani, Y. (1995). A modular and hybrid connectionist system for speaker identification, *Neural Computation* **7**: 791–798.
- Bentley, P. J. and Kumar, S. (1999). *The ways to grow designs: A comparison of embryogenies for an evolutionary design problem.*, San Francisco: Kaufmann.
- Churchland, P. M. (1986). *Some reductive strategies in cognitive neurobiology.*
- Coleman, O. (2010). Java hyperneat implementation.
- Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function, *Mathematics of Control, Signals, and Systems* **2(4)**: 303–314.
- Drchal, J., Koutnik, J. and Snorek, M. (2009). Hyperneat controlled robots learn how to drive on roads in simulated environment, *2009 IEEE Congress on Evolutionary Computation* pp. 1087–1092.
- Foo, Y. and Szu, H. (1989). Solving large-scale optimization problems by divide-and-conquer neural networks, *Int. Joint Conf. Neural Networks* **1**: 507–511.
- Goldberg, D. E. and Richardson, J. (1987). Genetic algorithms with sharing for multimodal function optimization, *Proceedings of the Second International Conference on Genetic Algorithms* pp. 148–154.
- Gruau, F. (1993). Genetic synthesis of modular neural networks, *Proceedings of the Fifth International Conference on Genetic Algorithms* p. 318–325.
- H. de Garis, C. Shuo, B. G. and Ruiting, L. (2010). A world survey of artificial brain projects, part 1: Large-scale brain simulations, *Neurocomput.* **74**: 3–29.

- Hrycej, T. (1992). *Modular learning in neural networks: A modularized approach to classification*.
- Mcgarry (1999). Hybrid neural systems: from simple coupling to fully integrated neural networks, *Neural Computing Surveys* **2**: 62–93.
- Murre, J. (1992). *Learning and Categorization in Modular Neural Networks*, Harvester-Wheatsheaf.
- Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R. and Ng, A. (2009). Ros: an open-source robot operating system, *ICRA Workshop on Open Source Software* . <http://pub1.willowgarage.com/konolige/cs225B/docs/quigley-icra2009-ros.pdf>.
- Radcliffe, N. J. (1993). Genetic set recombination and its application to neural network topology optimisation. , 1(1):67–90., *Neural Computing and Applications* **1**: 67–90.
- S. Johannes, B. Wieringa, M. M. and Munte, T. (1996). Hierarchical visual stimuli: Electrophysiological evidence for separating left hemispheric global and local processing mechanisms in humans, *Neuroscience Lett.* **210(2)**: 111–114.
- S. Wermter, V. W. (1997). Screen: learning a flat syntactic and semantic spoken language analysis using artificial neural networks, *Journal of Artificial Intelligence Research* **6**: 35–85.
- Spears, W. (1995). Speciation using tag bits, *In Handbook of Evolutionary Computation* .
- Stanley (2002). Evolving neural networks through augmenting topologies, *Evolutionary Computation* **2**: 100–127.
- Stanley (2006). Exploiting regularity without development, *Proceedings of the 2006 AAAI Fall Symposium on Developmental Systems* .
- Stanley, David D’Ambrosio, J. G. (2009). A hypercube-based indirect encoding for evolving large-scale neural networks, *Artificial Life journal* **15**.
- Stanley, K. O. (2007). Compositional pattern producing networks: A novel abstraction of development, *Genetic Programming and Evolvable Machines Special Issue on Developmental Systems* **1**: 31.

Stanley, K. O. (n.d.). Dept. of eecs, computer science division.

**URL:** *webpage:* <http://eplex.cs.ucf.edu/hyperNEATpage/>

Stanley, K. O., R. J. and Miikkulainen, R. (2004). Exploiting morphological conventions for genetic reuse., *In Proceedings of the Genetic and Evolutionary Computation Conference* .

Vapnik, V. and Chervonenkis, A. (1971). On the uniform convergence of relative frequencies of events to their probabilities, *Theory of Probability and its Applications* **16**: 264–280.

Vitku, J. (2015). Nengoros project online.

**URL:** <https://nengoros.wordpress.com/>

Vitku, J. and Nahodil, P. (2014). Towards evolutionary design of complex systems inspired by nature, *Acta Polytechnica* **54**: 367–377.

Zigmond, Bloom, L. R. and Squire. (1999). *Fundamental Neuroscience*, London: Academic Press.