Effects of Phenotypic Redundancy in Structure Optimization

Christian Igel and Peter Stagge

Institut für Neuroinformatik, Ruhr-Universität Bochum

44780 Bochum, Germany

tel: +49 234 32 25558 fax: +49 234 32 14209

Christian.Igel@neuroinformatik.ruhr-uni-bochum.de

April 8, 2003

Abstract

Concepts from graph theory and molecular evolution are proposed for analyzing the redundancy in the genotype-phenotype mapping in structure optimization stemming from graph isomorphism. Evolutionary topology optimization of neural networks serves as an example. By means of analytical and random walk methods it is shown that rare and frequent structures influence the search process: operators that are unbiased in genotype space may have a remarkable bias in phenotype space. In particular, if the desired structures are rare, the probability that an evolutionary algorithm evolves them may decrease. This is verified experimentally by comparing evolutionary structure optimization algorithms with and without search operators that take the redundancy of phenotypes into account. Further, it is shown how different encodings and restrictions on the search space lead to qualitatively different distributions of rare and frequent structures.

Keywords: graph representations, redundancy, neural networks, genotype-phenotype mapping

1 Introduction

Structure can be defined as the arrangement of parts of a whole to one another [1]. Therefore, it is obvious to represent candidate solutions for structure optimization tasks as graphs. Graph representations are hard to tackle analytically and hence there is only little theory on structure optimization. However, to set up an efficient optimization process it is essential to consider the search space and to take its characteristics into account. They determine—at least partly—the computational cost of the search and the final outcome.

Typically, one has to deal with two spaces: First, there is the true search space or genotype space. Each genotype can be mapped to a solution, e.g., a graph, a VLSI design or the configuration of a protein. Another genotype possibly represents an equivalent solution, an isomorphic graph, a VLSI design yielding the identical function, or a protein with the same 3D conformation. As a functionality is to be optimized, all solutions producing the same function are said to belong to the same phenotype or phenotypic class [2]. These classes constitute the second space, the phenotype space. For structures, it seems to be typical that there are often several, possibly very many, solutions (i.e., isomorphic graphs) leading to the same functionality. This results in a redundant genotype-phenotype map, which for example has been studied in the case of the mapping from RNA sequences to 2D RNA structures [3].

For a given phenotype space there is in general no canonical representation; instead many different genotype spaces are possible, and each one has its own characteristics. In a redundant genotype-phenotype map it is possible that some phenotypes are represented by a lot of genotypes while others are encoded only once. This distribution of redundancies is an inherent property of the representation, and we will see that it varies considerably between different graph encodings. The distribution of the redundancies strongly influences the search process and depends on the representation chosen in the evolutionary algorithm. Therefore, an appropriate encoding can greatly benefit the search performance.

In this article, we deal with the above mentioned redundancy in the search space of graphs. We extend previous work [4, 5] and present new theoretical results concerning structure optimization of neural networks. We show that different straightforward graph representations yield rather diverse characteristics. Further, we demonstrate the practical relevance of our considerations.

The investigation is directed towards general graph representations. Neural networks are used as a vehicle to demonstrate the basic ideas. Therefore, we start with a concise section on neural networks. As isomorphic graphs lead to redundancy in graph search spaces, some definitions and concepts from graph theory are needed and briefly introduced. With these means we analyze several encodings analytically as well as using random walk methods. Finally, the concepts are applied successfully to two neural network structure optimization tasks.

2 Structure optimization of neural networks

2.1 Neural networks

In this investigation, we concentrate on feedforward networks (see [6] for an excellent introduction to feedforward NNs). However, all the concepts can be extended to recurrent networks, cf. [4].

Consider a neural network $\mathcal{N}$ with d external inputs and m outputs and let $\mathcal{N}$ consist of n neurons $v_1, \ldots, v_n$. Additionally, there is an extra unit v_0 with constant output $y_0 = 1$ for the realization of bias (threshold) parameters. The topology of the network can be described by a directed graph $G_{\mathcal{N}} = (V_{\mathcal{N}}, E_{\mathcal{N}})$

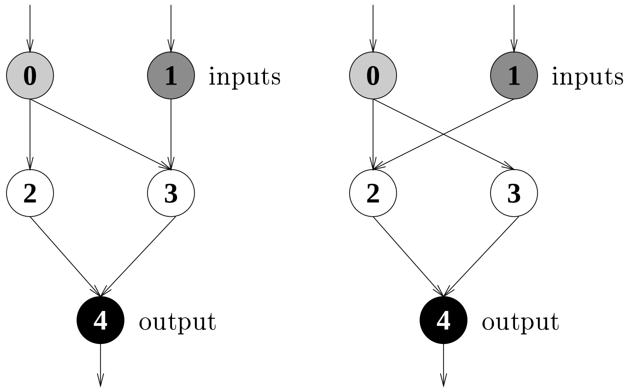


Figure 1: Two isomorphic 2-2-1 network structures; each input neuron and the output neuron have different colors.

with vertices $V_{\mathcal{N}} = \{v_0, \ldots, v_n\}$. There exists an edge $(v_j, v_i) \in E_{\mathcal{N}}$ iff the neuron v_i gets input from neuron v_j.

Without loss of generality, we assume that the neurons $v_1, \ldots, v_d$ are the d external input neurons and that $v_{n-m+1}, \ldots, v_n$ are the m output neurons. A graph representing a feedforward NN is regarded as *feasible* iff each hidden node lies on a path from an external input to an output unit, cycles and connections to the input nodes are not allowed.

The output y_i of each neuron v_i with $i > 0$ is given by

$$y_i = g_i\Big(\sum_{(v_j,v_i)\in E_{\mathcal{N}}} w_{ij} y_j\Big) \ , \tag{1}$$

where g_i is a so called activation function and w_{ij} is the strength (weight) of the connection between neuron v_j and v_i.

Each network topology $G_{\mathcal{N}}$ can realize different functions $\mathbb{R}^d \to \mathbb{R}^m$ depending on the weights. The weights are usually adjusted by a data-driven learning algorithm starting from an initial weight configuration, so that the NN approximates a target function described by sample data. The choice of the underlying network topology is crucial for the performance of the neural network. If there are too few degrees of freedom (weights), the final error of the NN is likely to be large. On the other hand, if there are too many degrees of freedom, the training time may become inappropriately long and the ability of the NN to generalize (i.e., to react to unseen data in the desired way) is likely to be poor (see e.g. [7] chapter 12).

2.2 Evolutionary optimization of neural networks

The problem of finding a suitable NN topology for a given task sets up a hard discrete optimization task, which is successfully tackled using evolutionary algorithms, see [8, 1, 9, 10, 11] for surveys of this field.

When evolving NNs, one has to encode the network topology. Two basic encoding schemes can be distinguished: In *direct encodings* each connection in the neural network is represented explicitly in the genotype, e.g., in a connection matrix [12]. Thus, one can get a good idea of the effects the search operators, which act on the genotypes, will have in the phenotype space. In *indirect encoding* schemes the genotype does not represent the NN topology directly, but it consists of—for example, grammar-based—information that describes a construction process for the NN. Another important distinctive feature of encoding schemes is whether only information about the topology or additionally information about the weights is stored in the genotype.

Any approach has to cope with several problems. There might be functionally equivalent NNs that are represented by numerous genotypes, others only by few or one, and some might not be coded for at all. Search operators can have the tendency to produce NNs with specific characteristics, e.g., NNs with many layers or sparsely connected NNs. The influence of the operators on the fitness—and even on the structure in some indirect encodings—is not obvious. Up to now there is neither a proven best representation–operator setting nor a constructive way to generate a good setting for a given problem except some general design heuristics [13].

3 Graph isomorphisms

3.1 Preliminaries

In this section, we summarize some classical graph theory needed to identify isomorphic neural networks. The notation and results used in this section are based on [14, 15].

Let $G = (V, E)$ be a directed graph, V and E denote the set of vertices and edges, respectively. A *coloring* π denotes an ordered partition of V into disjoint, non-empty color classes, i.e., for k different colors $\pi = (V_1, V_2, \ldots, V_k)$. The order of the color classes is significant, but the order of the vertices within each class is not.

Let γ be a permutation of the vertex set V of a graph G and let v^γ denote the image of the vertex $v \in V$ under γ. Similar, we define $W^\gamma = \{w^\gamma | w \in W\}$ with $W \subseteq V$ and G^γ as the graph in which vertices v^γ and w^γ are adjacent iff v and w are adjacent vertices in G. The image of a coloring π under γ is defined as $\pi^\gamma = (V_1^\gamma, V_2^\gamma, \ldots, V_k^\gamma)$. A permutation γ of a vertex set with coloring π is called *color-preserving* iff $\pi^\gamma = \pi$, i.e., each $v \in V$ has the same color as v^γ. Two colored graphs G_1 and G_2 are isomorphic iff there exists a color-preserving permutation γ that transforms one into the other, say $G_1 = G_2^\gamma$.

In order to handle isomorphic graphs, we use the concept of *canonical labeling maps*: Given a partition $\pi = (V_1, V_2, \ldots, V_k)$ of $\{0, 1, \ldots, n\}$, we firstly define $c(\pi)$ as the partition $(\{0, 1, \ldots, |V_1| - 1\}, \{|V_1|, \ldots, |V_1| + |V_2| - 1\}, \ldots, \{n + 1 - |V_k|, \ldots, n\})$, i.e., $c(\pi)$ is independent of π except it has the same number of classes with the same sizes in the same order. Then a canonical labeling map is a function $\mathcal{C}$ such that for any graph G and partition π of V we have

1. $\mathcal{C}(G,\pi) = G^{\delta}$ for some permutation δ such that $\pi^{\delta} = c(\pi)$, and

2. $\mathcal{C}(G^{\gamma},\pi^{\gamma}) = \mathcal{C}(G,\pi)$ for any permutation γ of V.

The usefulness of canonical labeling maps is shown by the following theorem [15].

Theorem 1 (McKay 1990) *Suppose the graphs G_1 and G_2 are colored using the same number of vertices of each color. Then $\mathcal{C}(G_1,\pi_1) = \mathcal{C}(G_2,\pi_2)$ iff G_1 and G_2 are isomorphic, i.e., $G_1^{\gamma} = G_2$ for some color-preserving permutation γ. (Here, π_1 and π_2 are the colorings, with the colors in the same order in each.)*

The general graph isomorphism problem and therefore the canonical labeling is a potential member of the problem class NPI which contains the so called NP-incomplete problems. NPI is defined as NP excluding P and all NP-complete problems [16]. So we cannot expect to have an algorithm that solves the general graph isomorphism problem in polynomial time. However, efficient heuristic algorithms exist that can handle the canonical labeling for graphs generated from NNs. The graphs representing NNs that can be regarded as tractable with today's learning algorithms have moderate size, so that the costs of computing their canonical representation can almost be neglected compared to the fitness evaluation (learning) time.

Fast, well-documented software that implements a canonical labeling map and in addition determines the size of the *automorphism group*, which is needed in the following sections, is freely available: the *nauty* package [15] developed by Brendan D. McKay. This software was first discussed in conjunction with NNs in [17] and is used in this investigation as well.

3.2 Counting adjacency matrices

Given a graph G we want to determine how many different adjacency matrices exist that code for graphs that are isomorphic to G. Neglecting the coloring, there are $|V|!$ permutations of the vertices. Now suppose that G is colored and that there are k different color classes. Then there exist $|V_1|! \cdot |V_2|! \cdot \ldots \cdot |V_k|!$ color-preserving permutations, but some of these permutations do lead to the same adjacency matrices. Two permutations give identical adjacency matrices iff there exists an *automorphism of* G that maps one of the permutations onto the other. Hence, we need to know the number of such automorphisms.

Definition 1 *An* automorphism *of a graph G is a color-preserving permutation of the vertices that map G onto itself. The* size of the automorphism group *of a graph G denoted by $a(G)$ is defined as the number of automorphisms of G.*

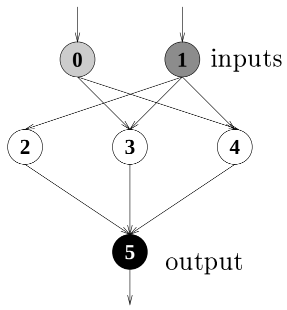
For example, let G be one of the graphs in Fig. 1. Then $a(G) = 1$ because the identity is the only color-preserving permutation that maps G onto itself. On the other hand if G is the graph shown in Fig. 2 a permutation of nodes 3 and 4 is an additional automorphism and $a(G) = 2$.

The following considerations lead to the number of different adjacency matrices. Suppose two isomorphic graphs G_1 and G_2 with $G_1^{\gamma_1} = G_2$ for a color-preserving permutation γ_1. Let γ_2 be an automorphism of G_1. Then the concatenation $\gamma_1 \circ \gamma_2$ of the two permutations leads to the same adjacency matrix as the permutation γ_1 and it holds $G_1^{\gamma_1} = G_1^{\gamma_1 \circ \gamma_2} = G_2$. There exist $a(G)$ possible choices for γ_2, i.e., for each color-preserving permutation there exist $a(G)$ permutations (including the identity mapping) that lead to the same adjacency matrix. Hence, the product of factorials derived above has to be normalized by $a(G)$ to account for the automorphisms:

Proposition 1 *The number of different adjacency matrices that code for graphs that are isomorphic to G is given by*

$$\frac{1}{a(G)} \prod_{i=1}^{k} |V_i|! \ . \tag{2}$$

For example, there are three different adjacency matrices that code for the networks that are isomorphic to the one shown in Fig. 2. The size of the automorphism group is two (the identity and the permutation that swaps the nodes 3 and 4), therefore we have $1 \cdot 1 \cdot 3! \cdot 1/2 = 3$.



$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 \end{pmatrix}$$

Figure 2: Network graph and the three adjacency matrices that code for all corresponding isomorphic networks (including identity).

3.3 Determining isomorphic neural network topologies

We want to use the tools presented in Sections 3.1 and 3.2 to determine isomorphic NN architectures. The topologies of two neural networks $\mathcal{N}_1$ and $\mathcal{N}_2$ are said to be isomorphic iff $V_{\mathcal{N}_1} = V_{\mathcal{N}_2}$ and there is a bijection $\gamma : V_{\mathcal{N}_1} \to V_{\mathcal{N}_2}$ such that

$$(u,v) \in E_{\mathcal{N}_1} \leftrightarrow (\gamma(u), \gamma(v)) \in E_{\mathcal{N}_2} \ . \tag{3}$$

Additionally, the input and output nodes of the one NN may only be mapped to the corresponding input and output nodes of the other. This can be ensured by a coloring. Therefore, we define for each network graph $G_{\mathcal{N}} = (V_{\mathcal{N}}, E_{\mathcal{N}})$ the partition

$$\pi_{\mathcal{N}} = (\{0\}, \{1\}, \dots, \{d\}, \{d, d+1, \dots, n-m\}, \\ \{n-m+1\}, \{n-m+2\}, \dots, \{n\}) \ . \tag{4}$$

Based on Theorem 1 it holds:

Corollary 1 *Two feedforward neural networks $\mathcal{N}_1$ and $\mathcal{N}_2$ are isomorphic iff the colored graphs $G_{\mathcal{N}_1}$ and $G_{\mathcal{N}_2}$ with coloring $\pi_{\mathcal{N}_1}$ and $\pi_{\mathcal{N}_2}$, respectively, are isomorphic.*

In particular, the networks are isomorphic iff

$$\mathcal{C}(G_{\mathcal{N}_1}, \pi_{\mathcal{N}_1}) = \mathcal{C}(G_{\mathcal{N}_2}, \pi_{\mathcal{N}_2}) \tag{5}$$

for some canonical labeling map $\mathcal{C}$.

The colorings fix the input and output nodes. Note that each input and output neuron gets a class of its own and that the order of the color-classes within a coloring is significant. If different types of neurons are encoded, the partitions can also be used to group these neurons. For example, when a *sigma-pi* network [18] is described, coloring can prevent from mapping a sigma unit to a pi unit and vice versa.

For a fixed function $\mathcal{C}$ and for each feedforward neural network $\mathcal{N}$ there exists exactly one isomorphic *canonical network* that is uniquely defined by the graph $\mathcal{C}(G_{\mathcal{N}}, \pi_{\mathcal{N}})$. This fact can be regarded as a way to solve the *competing conventions* problem, also called permutation problem [19, 11, 20, 10, 21]. That is, many genotypes (conventions) map to equivalent NNs, i.e., the corresponding network topologies are isomorphic. It was argued that this has a harmful effect on evolutionary optimization, especially on recombination. As pointed out in [4, 5], if only canonically labeled networks are stored in the population, there is exactly one representation of each structure and the competing conventions problem taken literally can be regarded as solved. This requires that each genotype that is created by the search operators is replaced by the genotype belonging to the canonical representative of the NN. Although this isomorphism-free representation ensures a single "convention," it does not necessarily lead to something like a "homologous" crossover between NNs. Further, it does not solve problems in the encoding like those to be discussed in the following section.

4 Frequent and rare phenotypes

4.1 Results from molecular evolution

This investigation is influenced by findings concerning genotype-phenotype relations in the domain of RNA molecular evolution, see [3] for a survey. The genotype is the object under variation storing the genetic information of an individual. The definition of the term phenotype is a bit delicate [2]. Here we define it as the class of which an individual is a member based on observable qualities [22], this means its *function* in technical frameworks.[1] In case of RNA molecular evolution, P. Schuster and co-workers identify the genotype with RNA sequences and their secondary structure with the corresponding phenotype. Let $\mathcal{G}$ and $\mathcal{S}$ denote the genotype space and phenotype (structure) space, respectively. The function $\Gamma : \mathcal{G} \to \mathcal{S}$ maps each genotype to its corresponding phenotype. We assume Γ to be surjective and do not consider environmental influences on the development. The redundancy of a phenotype can be defined based on the number of genotypes that are mapped onto it:

Definition 2 *The* redundancy $r(s)$ *of a phenotype* $s \in \mathcal{S}$ *is the cardinality of its pre-image in the genotype space given by*

$$r(s) = |\{g \in \mathcal{G} \mid \Gamma(g) = s\}| \ .$$

The average redundancy *of the encoding is defined as*

$$\bar{r} = \frac{1}{|\mathcal{S}|} \sum_{s \in \mathcal{S}} r(s) = \frac{|\mathcal{G}|}{|\mathcal{S}|} \ .$$

Now we can define frequent and rare phenotypes according to [23] and the fractions of frequent phenotypes and frequent genotypes[2] [5].

Definition 3 *A phenotype* $s \in \mathcal{S}$ *is* frequent *iff* $r(s) > \bar{r}$*, it is* rare *iff* $r(s) \leq \bar{r}$*. The* fraction of frequent phenotypes $h_{\mathcal{S}}$ *and the* fraction of frequent genotypes $h_{\mathcal{G}}$ *can be defined as*

$$\begin{aligned} h_{\mathcal{S}} &= \frac{|\{s \in \mathcal{S} \mid r(s) > \bar{r}\}|}{|\mathcal{S}|} \quad \textit{and} \\ h_{\mathcal{G}} &= \frac{|\{g \in \mathcal{G} \mid r(\Gamma(g)) > \bar{r}\}|}{|\mathcal{G}|} \ . \end{aligned}$$

Analysis of the sequence-structure map of RNA led to the following fundamental results (see [3] for details and references): (i) There are more sequences than structures. (ii) There are few frequent and many rare structures. (iii) The

[1] Phenotype is used as a synonym for *phenotypic class*, which would be the most appropriate term [2].

[2] The terms *frequent genotype* and *rare genotype* should be regarded as short forms for *genotype coding for a frequent phenotype* and *genotype coding for a rare phenotype*, respectively.

genotypes forming frequent structures are distributed (almost) randomly in the genotype space. All frequent structures are within close proximity from any genotype. (iv) The frequent structures form extended *neutral networks* that may percolate through large regions of the search space. A neutral network (neutral with respect to the phenotype) of a structure S_i is a graph, where the vertices are the genotypes that map to S_i and edges are drawn between all nearest neighbors in the genotype space. Herein, neighborhood is defined with respect to the mutation distribution.

4.2 Structure optimization

Schuster claims that "Understanding evolution will always be incomplete unless phenotypes are considered explicitly" [3]. However, in simulated evolution the genotype-phenotype and the phenotype-fitness mapping can in most cases be combined to a single function $(\Phi \circ \Gamma) : \mathcal{G} \to \mathbb{R}$ (regardless whether the functions involved are stochastic). Further, the choice of the phenotype level often appears—and is—arbitrary in evolutionary algorithms. Nevertheless, there are good reasons to carry over the genotype-phenotype concept from biology to EAs, because often this intermediate level is needed to gain insight into the search process. As an example, suppose an evolutionary algorithm with binary encoding is used for optimization of a real-valued function: it has proven to be beneficial to consider the effects of the variation operators in the real-valued parameter domain, which can be identified as the phenotype space [24].

What are sensible definitions of the genotypes and the phenotypes in structure optimization, e.g, in the evolution of NN topologies? We consider the architecture of the NNs and not the configuration of the weights. Thus, the genotype space is set up by the encoding of the network graphs. Identifying the phenotype with the functions a NN can represent and its learning behavior, it arises that isomorphic NNs have the same phenotype (assuming that the learning process is invariant under color-preserving permutations). Therefore, we refer to the equivalence class of graphs that correspond to NNs that are isomorphic to $\mathcal{N}$ as the phenotype or structure of $\mathcal{N}$. Throughout this article, we use the following nomenclature:

Definition 4 Topology *or* architecture *denotes the NN graph encoded in the genotype.*
Structure *denotes the phenotype (or phenotypic class) of the encoded NN, i.e., the equivalence class of isomorphic NNs.*

In the following, we investigate whether or not the results from molecular evolution carry over to structure optimization, in particular to topology optimization of artificial neural networks. The fact that there are more genotypes than structures holds trivially in cases where the genotype-phenotype mapping is not injective. The phenomenon of rare and frequent phenotypes is discussed in detail in this study.

Redundancy can be beneficial or problematic for an evolutionary process. For example, random drift on neutral networks [25, 26, 3] can be useful for leaving local optima and increasing the accessibility of phenotypes. In this study we concentrate on redundancy in structure optimization stemming from graph isomorphism. Further, we assume that the evolutionary search is based on standard mutation operators like addition or deletion of a connection or node. With respect to such operators, isomorphic graphs

1. do not form neutral networks, and, more fundamental,
2. have the same neighborhood in the structure space, i.e., replacing a genotype in a population with one that maps to the same structure does not change the generating / search distribution (cf. [27, 28] and references therein) in the phenotype space for the next generation.

Hence, important requirements for typical scenarios where redundancy can be beneficial (e.g., those discussed recently in [29, 30, 31]) are not fulfilled.

5 The neural network search space

In this section, we discuss the characteristics of the search space using the notions of genotype and structure as defined above. The introduced concepts of rare and frequent structures and genotypes are used to characterize several different encodings. For these encodings we specify the genotype and structure space, $\mathcal{G}$ and $\mathcal{S}$, under consideration and determine some dominating properties of each setting. We start with an analytical example before more complex cases are treated by random walk methods.

5.1 Analytical Example

In this case, we analyze NNs with d input nodes, l hidden nodes and a single output node. We restrict the hidden nodes to one layer. That means, the hidden nodes only get input from the input nodes. In the case without shortcut connections, the genotype space $\mathcal{G}$ using a direct binary encoding is a subset of the bit strings of length $d \cdot l$, where the bits specify the connections from the inputs to the hidden units. Each hidden neuron can be connected to the inputs in $T = 2^d$ different ways, encoded by d bits describing its connections to the inputs. The pattern without any connection to the input does not lead to a feasible NN and is excluded. Thus, for each of the l hidden units there are $T-1$ acceptable bitstrings. Hence, the cardinality of $\mathcal{G}$ is given by:

$$|\mathcal{G}| = (T-1)^l \quad . \tag{6}$$

The size of the structure space $\mathcal{S}$ is given by the following consideration: We can label a hidden unit with a tag $t \in \{1, \dots, T-1\}$. A NN structure is then defined by the tags of its l hidden nodes, irrespective of the order of the hidden neurons. Therefore, a multiset $(t_1, \dots, t_l)$ uniquely specifies a class of equivalent NNs. The number of these multisets is known from combinatorics to be:

$$|\mathcal{S}| = \binom{(T-1)+l-1}{l} \quad . \tag{7}$$

As an example, the average number of matrices per NN class for a 4-5-1 NN is $\bar{r} = 65.3057$.

The average redundancy does not characterize the encoding sufficiently. Additionally, we calculate the fraction of frequent structures and genotypes, $h_{\mathcal{S}}$ and $h_{\mathcal{G}}$, as a function of hidden nodes $l = 1, \ldots, 30$. This is done by enumerating the structure space $\mathcal{S}$. The result for a NN with four input nodes is shown in Fig. 3 and reveals the following properties: The fraction of frequent struc-

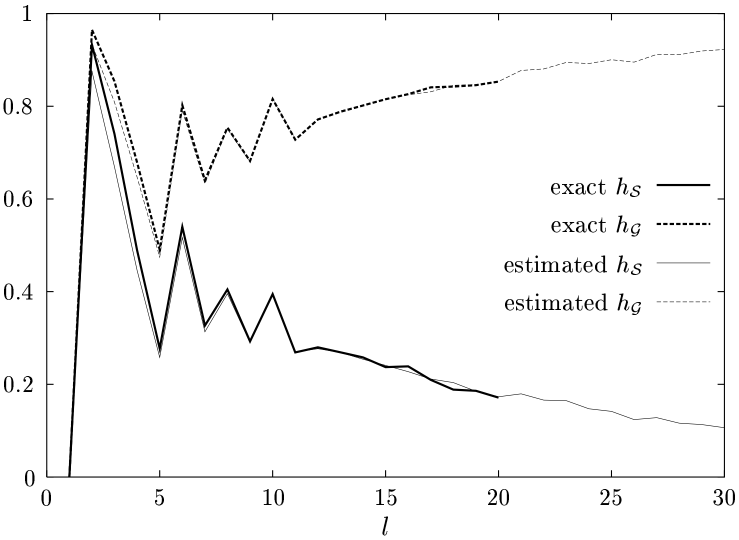


Figure 3: Fraction of *frequent* structures ($h_{\mathcal{S}}$, solid line) and of matrices coding for these structures ($h_{\mathcal{G}}$, dashed line) against the number of the hidden neurons. The bold lines show the results of the analytical example. The thin lines correspond to the random walk results, see Section 5.2.

tures decreases with increasing number of hidden nodes. This means that an ever-smaller proportion of all structures is represented above average. The proportion of genotypes which code for these structures is given by the fraction of frequent genotypes $h_{\mathcal{G}}$. We see that $h_{\mathcal{G}}$ increases with growing l. This tells us that a majority of genotypes codes for a minority of structures. For $l = 20$: 85.0% of the genotypes represent only 17.7% of the structures. Thus, if the search operators in the genotype space are unbiased they are heavily biased in the structure space.

Similar results have been found in the coding of RNA sequences to 2D RNA structures in biology as noted in Section 4.1. One may wonder whether this is a simple consequence of the definition of frequent and rare genotypes and structures. It is not. In particular, the following three scenarios can be envisioned:

- There could be a small number of structures with low redundancy and

many structures with an equally high redundancy. Thus, both the fractions of frequent structures and of frequent genotypes would be high.
Example: one single rare structure $s_0 \in \mathcal{S}$ with $r(s_0) = 1$ and for all $s \in \mathcal{S} \setminus s_0 : r(s) = r_s$. This leads to $h_{\mathcal{S}} = (|\mathcal{S}| - 1)/|\mathcal{S}|$ and $h_{\mathcal{G}} = ((|\mathcal{S}| - 1) \cdot r_s)/((|\mathcal{S}| - 1) \cdot r_s + 1)$, stating that both quantities approach 1 with growing $|\mathcal{S}|$.

- There could be very many structures with a small redundancy and exceptional few structures with a higher redundancy. This would lead to a small fraction of genotypes coding for these frequent structures. As there are only very few of them the fraction of frequent structures as well as the fraction of frequent genotypes would decrease.
  Example: one single frequent structure $s_0 \in \mathcal{S}$ with $r(s_0) = r_0$ and else only rare structures $s \in \mathcal{S} \setminus s_0$ with $r(s) = 1$. In this case, it holds $h_{\mathcal{S}} = 1/|\mathcal{S}|$ and $h_{\mathcal{G}} = r_0/(|\mathcal{S}| + r_0 - 1)$, stating that both quantities approach 0 with growing $|\mathcal{S}|$.

- Finally, we have quite a large number of structures with low or medium redundancy but also some structures with *very* high redundancies. These structures constitute the frequent structures and as their redundancy is very high the fraction of genotypes coding for them is also high. Therefore, the fraction of frequent genotypes increases whereas the fraction of frequent structures decreases. This is the case in the analytical example shown in Fig. 3.

This discussion shows that the observed behavior is not a mere consequence of the definition of rare and frequent structures. Different effects could be observed if most of the structures had the same or nearly the same redundancy, i.e., if there existed a dominating redundancy. For the exact example, a detailed look at the different redundancies and the number of corresponding structures reveals that there is no such dominating redundancy, see Fig. 4. Instead, a broad range of different values is present and leads to the increasing fraction of frequent genotypes coding for a decreasing fraction of frequent structures.

In the following section, different encoding schemes are discussed allowing for networks with more than just one hidden layer.

5.2 Random Walk Analysis

For a broader class of structures, e.g., those with an unrestricted number of hidden layers, the afore used analysis becomes more difficult and enumeration takes too much time. Therefore, we use (generalized) random walks in the genotype space and collect information about the encoding by calculating the redundancies of the genotypes found. In each case the number of input nodes is fixed to four, there is one output node and the number of hidden neurons l is fixed to values $l \in \{1, \ldots, 20\}$. In these experiments we accept only feasible feedforward NNs (all hidden nodes lie on a path from the input to the output,

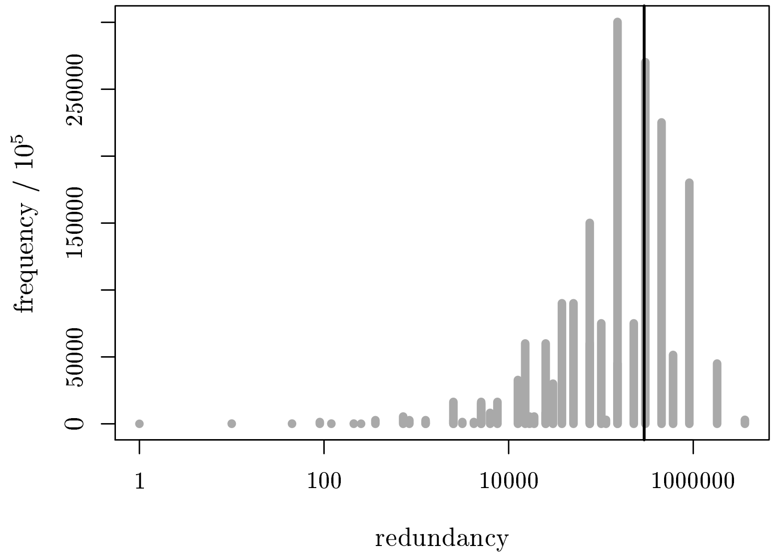


Figure 4: Number of structures with a given redundancy for a 4-10-1 NN with one hidden layer. Note the logarithmic scale of the x axis. The black vertical line marks the average redundancy $\bar{r} = 2.94 \cdot 10^5$.

no cycles and no connections to the input nodes are permitted). We use the *bitflip* operator as standard operator during the random walks.

5.2.1 Search Spaces under Consideration

We investigate the following search spaces:

1a. In order to compare the random walk method with the analytical result from the previous section, we restrict the topology to networks with one hidden layer.[3] The corresponding genotype space is a bit array with $4 \cdot l$ bits. The number of isomorphic topologies is given by equation (2), which reduces in this simple case to $\frac{1}{a(G)} \cdot l!$.

1b. In this case, the encoding is still layer oriented, but networks with two

[3]A brief consideration [4] shows that neural networks with one hidden layer are very rare if one allows connections in between the hidden neurons. Networks with one hidden layer are sufficient—from the theoretical and practical point of view—to solve a large class of problems and are often chosen as standard architectures. Therefore, this class of NNs makes up an important phenotype space.

hidden layers are also included. The number of hidden layers is encoded explicitly in the genotype. An additional operator may increase or decrease the number of layers allowed for the network. The number of isomorphic architectures is again given by equation (2), which in this case reads $\frac{1}{a(G)} \cdot l_1! \cdot l_2!$, where l_1, l_2 are the sizes of the two layers and each layer gets a color of its own.

2. In the second set of experiments there is no restriction on the number of layers of the NN. In a first approach, we use a lower-triangular matrix to describe the NNs. In this way, we ensure feedforward NNs, i.e., acyclic graphs. The search space $\mathcal{G}$ consists of the lower-triangular matrices coding for feasible networks. For a given architecture all those topologies are counted as isomorphic that belong to the same structure and are also described by a lower-triangular matrix. To calculate the number of isomorphic topologies consistent with this encoding an enumeration scheme is used.

3. In the final setup there is again no layer restriction. But this time we use the complete quadratic connection matrix to describe a NN. Obviously there are matrices resulting in cyclic graphs; they are not feasible and therefore dismissed. In this case, the number of isomorphic topologies is again given by equation (2), here $\frac{1}{a(G)} \cdot l!$.

The two final encodings include the same structures, namely all feasible feedforward NN structures. The difference is that for the most NN structures there a fewer network topologies in the triangular scheme than in the full matrix scheme. An example is shown in Fig. 5: the two architectures are isomorphic, but the right one is not included in encoding scheme 2 whereas it is a valid genotype in scheme 3.

5.2.2 Random Walk Results

For each encoding scheme we analyze NNs with four inputs, one output, and l hidden neurons. Random walks over 500,000 genotypes are performed for each value of l. The collected data are used to find the fraction of frequent structures and genotypes $h_{\mathcal{S}}$ and $h_{\mathcal{G}}$. In the case where the topologies are restricted to one hidden layer the result is shown together with the analytical result in Fig. 3; the random walk results are very close to the exact values. In the case where two layers are permitted the random walks reveal the same characteristics of the search space: the fraction of frequent genotypes increases whereas the fraction of frequent structures decreases with growing l.

For the NN without layer restriction we have two different encoding schemes: the triangular matrices and the full connection matrix. As it is computationally quite expensive to calculate the number of isomorphic topologies in the case of the triangular matrices, we pursued random walks only for $l \in \{1, \dots, 15\}$. The result is shown in Fig. 6. For increasing l, the same tendency is observed that was already evident for the other schemes. The result for the encoding via a full

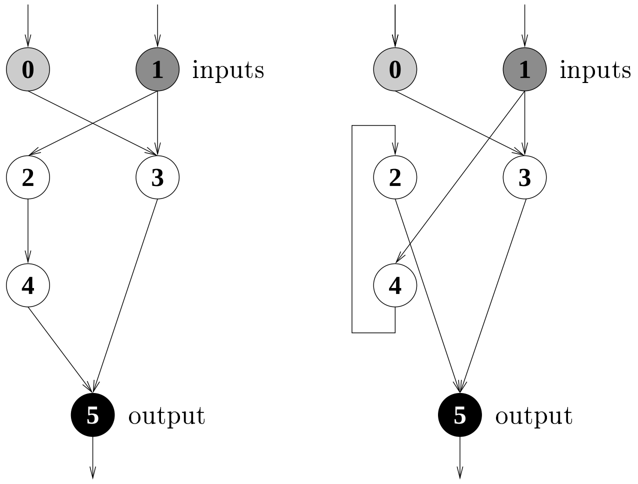


Figure 5: Two isomorphic network topologies. The right one is a color-preserving permutation of the left graph by changing the position of neuron 2 and neuron 4. The connection matrix of the right topology is not a triangular matrix.

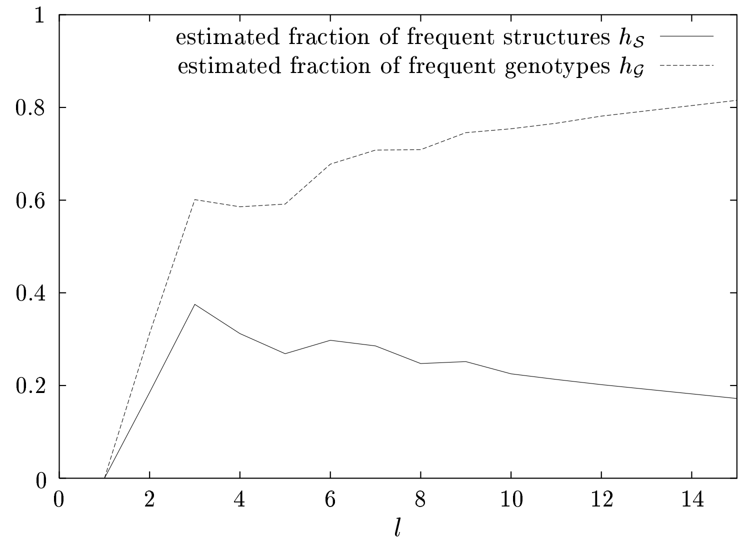


Figure 6: Fraction of frequent genotypes and structures for NN coded by lower-triangular matrices.

connection matrix shows a completely different behavior, as can be seen in Fig.

redundancy	number of structures
$6.81 \cdot 10^9$	1
$2.72 \cdot 10^{10}$	1
$8.17 \cdot 10^{10}$	11
$1.63 \cdot 10^{11}$	6
$6.54 \cdot 10^{11}$	55
$1.31 \cdot 10^{12}$	499926

Table 1: Number of structures for each redundancy observed during a random walk for a NN with four inputs, 1 output, and 15 hidden neurons. The full connection matrix encoding scheme was used.

7. Nearly all genotypes and structures are frequent. Analyzing the exact data

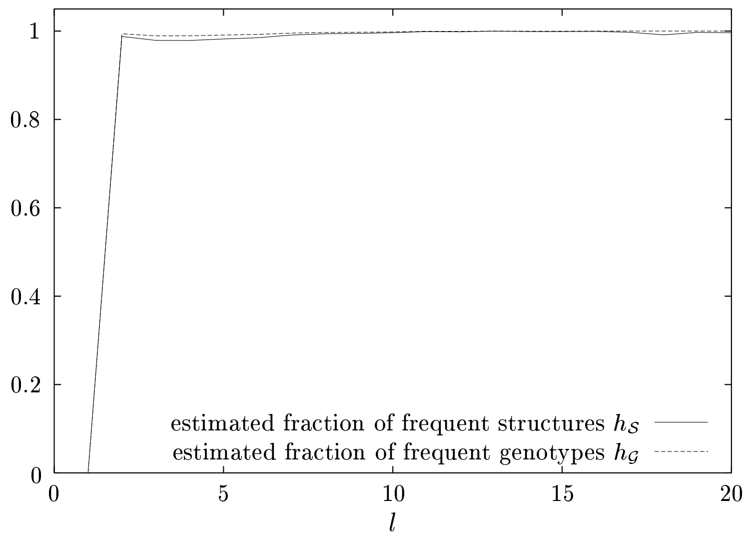


Figure 7: Fraction of frequent genotypes and structures for NN coded by a full connection matrix.

collected during the random walk reveals the reason, see Tab. 1 for $l = 15$: The situation corresponds to the first point of the discussion on possible outcomes of redundancy statistics in Section 5.1. Nearly all encountered structures have maximal redundancy leading to a dominating redundancy and a high fraction of frequent structures *and* genotypes.

The results of this section reveal that the choice of the encoding for a structure optimization task may have drastic side effects as it can introduce a strong bias in the structure space. For example, the fully connected NN with one hidden layer is a structure that is represented only once irrespective of the number of hidden neurons. Different coding schemes will probably result in different characteristics as is evident in our examples. The encoding, together with the search operators, determine the success of an evolutionary setup, thus it is important to know the implications of a specific setting. Thus, we propose the following

Postulate 1 *The search bias stemming from rare and frequent structures should be removed, unless there is* a priori *knowledge that it improves the search process.*

6 Experimental study

6.1 Fair Operators

In order to remove the bias induced by different redundancies of topologies, the transition probability $\Pr(g_i \to g'_i)$ that a variation operator generates an offspring g'_i from a chosen parent g_i could explicitly consider the redundancy $r(\Gamma(g'_i))$. In [5] we suggested

$$\Pr(g_i \to g'_i) \propto \frac{1}{r(\Gamma(g'_i))} \quad . \tag{8}$$

Such an operator could increase the performance of the search process because there may be well-performing rare structures that are not likely to be produced by an operator that does not take the redundancies into account and therefore tends to explore mostly frequent structures.

In this study, we use such *fair* operators, which are realized in the following straightforward manner. For each offspring genotype g_i

1. generate an intermediate offspring $\tilde{g}_i$, using the standard operators,
2. calculate the redundancy $r(\Gamma(\tilde{g}_i))$,
3. with probability $1/r(\Gamma(\tilde{g}_i))$ the genotype $\tilde{g}_i$ becomes the offspring g'_i, otherwise goto step 1.

Herein, the term *standard operators* refers to the variation operators used in an arbitrary EA for structure optimization. This method is well suited for our experiments, where $r(\Gamma(\tilde{g}_i))$ is small (e.g., $r(\Gamma(\tilde{g}_i)) \leq 120$ for all $\tilde{g}_i \in \mathcal{G}$ in Section 6.4). For real-world optimization tasks, however, more sophisticated ways to deal with the isomorphism problem could be chosen. In particular, the employed encoding could ensure an (almost) equal distribution of the redundancies.

In the following experiments we want to evaluate the bias induced by isomorphic topologies by comparing evolutionary optimization using standard and fair operators.

6.2 General setup

We show the effects of isomorphic topologies by means of two different evolutionary algorithms (EAs) for structure optimization. These algorithms are intended to be instructive rather than as efficient as possible. In particular, we preferred to keep the elemental mutation operators simple in order to make the search process more accessible. We describe settings common to both EAs first, before we explain their dissimilarities in Sections 6.3.1 and 6.4.1.

Both EAs are basically EP-style (evolutionary programming) systems using direct encodings (cf. [32]), where the population size is set to 20 and EP-style tournament selection [33] with 5 opponents is applied to determine the parents for the next generation.

The EAs are employed to solve two different classification tasks. The overall error of a network $\mathcal{N}$ is computed as

$$\mathcal{E}(\mathcal{N}) = \nu \cdot \mathcal{E}_{\text{class}}(\mathcal{N}) + |E_{\mathcal{N}}| + \mathcal{E}_{\text{mse}}(\mathcal{N}) \ . \tag{9}$$

Herein $\mathcal{E}_{\text{class}}$ is the classification error, i.e., the percentage of wrongly classified patterns, and $\mathcal{E}_{\text{mse}}$ is the mean squared error. The weighting factor ν is set to 10^6 in our experiments, so normally $\mathcal{E}_{\text{class}}(\mathcal{N}) \gg |E_{\mathcal{N}}| \gg \mathcal{E}_{\text{mse}}(\mathcal{N})$. The activation functions of the hidden neurons are sigmoidal, the activation functions of the output neurons are linear.

Each individual created is trained once using the complete set of input and output patterns, i.e., generalization need not be considered here. The training is stopped either after a predefined number of t_{max} cycles or when the *training progress* as defined in [34] measured after a *training strip* of length 5 dropped below 0.01. An improved version of the Rprop algorithm is used for training [35, 36]. The algorithm is outlined in Appendix A. Every experiment is conducted twice, one time with "Lamarckian" and one time with "Darwinian" interaction of gradient-based and evolutionary optimization (see [10, 37] and references therein). In the Lamarckian [38] scheme, the value of each network weight is stored in the genotype together with the corresponding connection. After the gradient-based learning the weights are updated in the genotype (i.e., acquired characteristics are inherited). In the case of Darwinian inheritance, the network weights are randomly initialized before each training process (i.e., the learning guides the evolution by the so-called Baldwin effect [38, 10]).

6.3 Variable-length representation

6.3.1 Evolutionary Algorithm

In the first set of experiments, we used a variable-length graph representation (similar to [32]) to encode feasible network structures together with their corresponding weights.

The population is initialized with NN topologies having a random number of 2 to four hidden nodes and a random number of connections. It is ensured that each hidden neuron gets at least one input and has one output. A maximum of three hidden layers is allowed, i.e., the maximum length of a (directed) path from an external input to an output is four.

Each reproduced individual is mutated by applying $1+x$ times an elemental operator, where x is the realization of a Poisson distributed random number, with mean $\lambda = 1$. This procedure inspired by [39] allows for correlated mutations and ensures detection of the optimal structure with probability one [40], see also [27].

There are two classes of elemental operators. The first class contains node changing operators:

addNode A new node is added, which gets two inputs and one output connection obeying the layer restriction.

deleteNode A hidden node, say v_a, and all connections to or from v_a are deleted. If v_a was the only input to a hidden node, this node is connected with one of the former inputs to v_a. If after the deletion a hidden node has no output connection, this node is connected with one of the former outputs of v_a.

Second, there are connection changing operators:

addConnection A forward connection is added obeying the layer restriction.

deleteConnection A connection that is not necessary for the network to be valid is deleted.

Node changing operators are only applied during the first 25 generations [9], with a probability of 0.40. If an elemental operator could not be applied successfully, another operator is chosen.

6.3.2 The TC problem

As a test problem, we use a TC classification task inspired by [18] as described in [9], see also [41]. The goal is to distinguish between five pixel 'T' and 'C' patterns on a 4×4 pixel grid. Thus, there are 16 external inputs and one output. The complete set of 40 possible patterns is used for training, the maximum learning time $t_{\max}$ and the maximum number of generations are set to 100.

6.3.3 Results

We performed 100 independent trials using the algorithm outlined in the previous section and 100 trials starting from the same initial populations using the corresponding fair mutations as described in Section 6.1.

The trajectory of the median of the best individuals' fitness is shown in Figure 9. After a few generations, the results using the fair mutations are significantly better (Wilcoxon rank sum test at generation 100, $p < 0.001$ for both Darwinian and Lamarckian inheritance).[4]

[4] We prefer the median because of its higher robustness (e.g., against the choice of ν). The

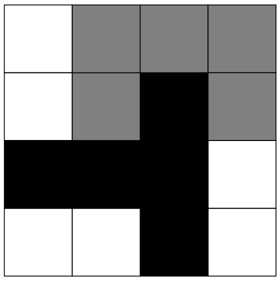


Figure 8: 'Receptive fields' of the input neurons in the TC problem. Two training patterns are shown, a 'T' and a 'C', which are to be distinguished.

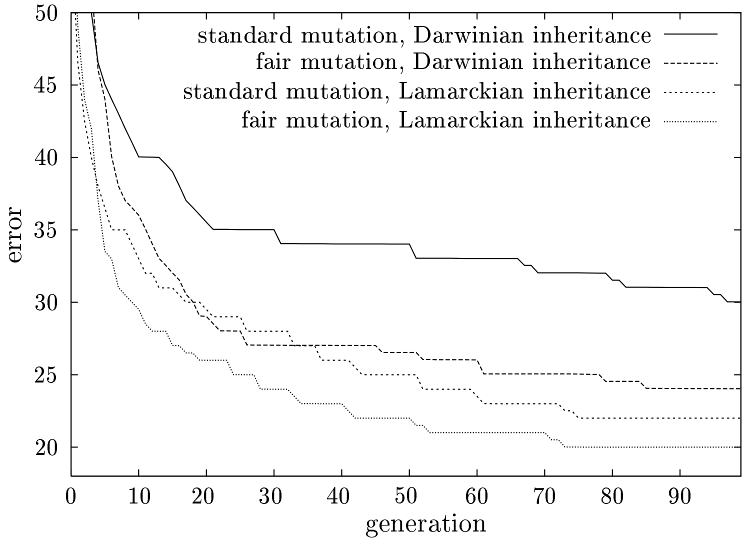


Figure 9: Median of best fitness values at each generation for the variable-length encoding. The results for Darwinian and Lamarckian inheritance using standard and fair mutation are shown. 100 trials have been performed for each setup.

There is a simple reason for this observation. If we neglect the size of the automorphism group in equation (2), we see that the redundancy scales exponentially with the number of hidden nodes. Therefore, intermediate offspring with fewer nodes have a higher probability to be accepted if the fair mutation is used. Hence, there is a tendency towards smaller structures. After NNs that

results are not normally distributed, so the nonparametric Wilcoxon test is used. However, all statistical results also hold when Student's t-test is applied. Where we deal with crisp binary success results as in Sec. 6.4.2 Pearson's χ^2-test is used.

	Lamarckian		Darwinian	
operator	standard	fair	standard	fair
addConnection	179055	178852	179345	176963
deleteConnection	178072	179166	178851	178550
addNode	20678	8250	20632	7325
deleteNode	20851	37374	20717	41304

Table 2: Operator frequencies.

solve the classification problem ($\mathcal{E}_{\text{class}} = 0$) have been evolved, the task reduces to finding networks which are as small as possible. Therefore, it is advantageous to apply *deleteNode* more often. This indeed happened in our experiments as shown in Table 2: the node deletion operator is applied about two times as often when the redundancy of the offspring is taken into account because intermediate offspring that were created involving *addNode* are discarded more often.

This experiment is an example where considering the redundancy of the generated structures increases the performance. But it could be argued that this is partially due to a "side-effect," which could have been achieved by dynamically changing the probability of the elemental operators [27].

6.4 Fixed length representation

6.4.1 Evolutionary Algorithm

In the second set of experiments, we used an encoding where the 'side-effect' described in Section 6.3.3 does not occur.

The number of hidden neurons is fixed to 5, only one hidden layer is allowed. The genotype can be regarded as a bit vector of length $\ell =$ (number of input neurons+1) $\cdot$ (number of hidden neurons+1). The bits indicate whether (i) a hidden neuron is connected to an input, (ii) a hidden or the output neuron utilizes a bias parameter, and (iii) a shortcut connection exists. Additionally, for each connection the corresponding weight value is stored. A *bitflip* operator that changes the status of each connection from present to not present and vice versa with probability $p_{\text{flip}} = 1/\ell$ is employed. The connections from the hidden neurons to the output are added and removed automatically: a hidden neuron is connected with the external output iff it has at least one connection to an external input. The population is initialized with topologies having a random number of 15 to 35 connections.

This encoding realizes a search without a bias in $\mathcal{G}$; in absence of selection setting $p_{\text{flip}} = 0.5$ yields an uniform random search, where each connection matrix represented in $\mathcal{G}$ has the same probability to be generated. As the number of hidden neurons is the same for all networks under consideration, the numerator in equation (2) is the same for all individuals. In this encoding there are genotypes that do not correspond to feasible networks because isolated hidden neurons are allowed.

6.4.2 Test problem

We have designed a second classification task, which is described by the Boolean function $\{0,1\}^3 \to \{0,1\}$

$$f(x_1, x_2, x_3) = x_1 x_2 x_3 \vee \overline{x}_1 \overline{x}_2 \overline{x}_3 \ . \tag{10}$$

Two hidden neurons (corresponding to two decision boundaries) getting input from all input nodes are sufficient to solve this problem, see Fig. 10 for an example topology. Therefore, the task is regarded as being solved iff the overall error drops below 9, i.e., all input patterns are classified correctly and there are only 8 connections. For this task, we set $t_{\max} = 50$ and the maximum number of generations to 200.

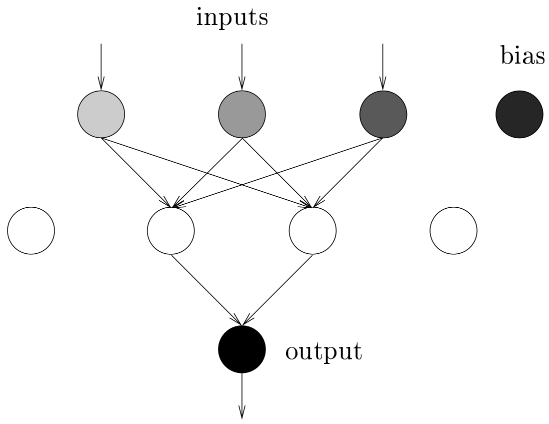


Figure 10: Network sufficient to solve the second classification task.

This problem has been chosen because we know in advance that the desired structure is not very frequent. If all hidden neurons get different inputs, then the redundancy of the structure is $5! = 120$, but the target architecture shown in Fig. 10 is very regular and its redundancy is only 5.

6.4.3 Results

We performed 100 independent trials using the described algorithm and 100 trials starting from the same initial populations using the fair *bitflip* with respect to redundancy. The success rate versus the generation is shown in Fig. 11. As expected from our theoretical considerations, the fair *bitflip* outperformed the standard operator. The differences are statistically significant (e.g., χ^2-test at generation 200, $p < 0.005$ for Lamarckian and $p < 0.001$ for Darwinian inheritance).

It is interesting that in contrast to the results in Sec. 6.3 the Darwinian inheritance scheme performs better in this task: Storing and updating the weights allows faster optimization (i.e., higher success rates in the beginning), but leads to premature "convergence" to not desired local optima.

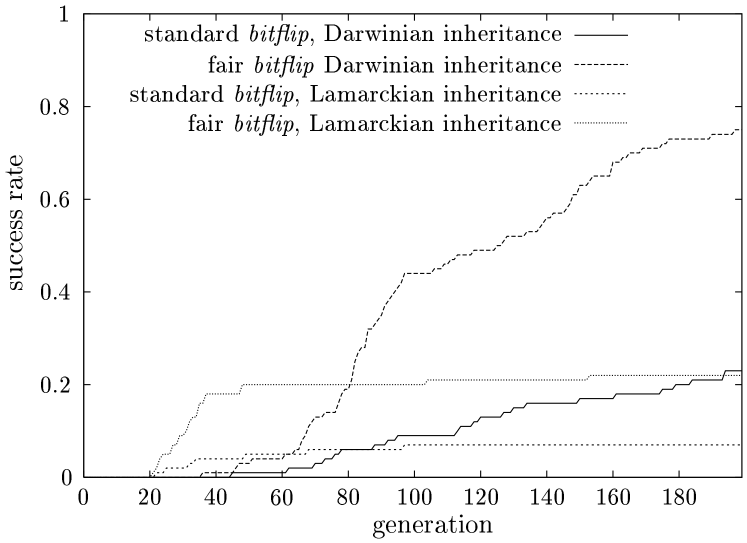


Figure 11: Success rates of the trials using fixed length encoding with the standard *bitflip* and the fair *bitflip* operator for the two inheritance schemes. 100 trials have been performed for each setup.

7 Conclusion

In this study, we proposed and applied concepts for describing the search space in *structure optimization using graph representation*.

In particular, we analyzed the role of redundancy in the genotype-phenotype mapping *stemming from isomorphic graphs* using the notion of rare and frequent structures as known from molecular evolution. Unequal distribution of redundancy implies that unbiased search in the genotype space leads to a biased search in the phenotype space. In cases where one cannot decide whether this bias is beneficial or harmful for the optimization process, we suggest to reduce or remove it because otherwise the sought structures may be rare ones that are unlikely to be detected.

As an example, we considered topology optimization of feedforward NNs. Analytical and random walk analysis showed that the effect of rare and frequent structures may become very prominent. Different distributions of redundancies can be observed, depending on the employed encoding and restrictions on the search space. For example, allowing only triangular matrices for encoding the adjacencies has a remarkable effect on the fractions of rare and frequent structures.

The question whether such effects actually matter in optimization processes is positively answered: Two different EAs employed for evolving NNs showed improved performance if the variation operators are modified in such a way that the bias in the phenotype space is removed. The modification of the operators is based on explicit calculation of the redundancy, which is suitable for smaller graphs; for larger systems the redundancy problem has to be tackled by an appropriate choice of representation and operators.

The methodical concepts described in this paper can be transferred to other evolutionary tasks and may prove beneficial whenever noninjective genotype-phenotype mappings are applied.

Acknowledgments

C. Igel would like to thank C. M. Friedrich for fruitful discussions. We thank the John von Neumann Institute for Computing (NIC) for granting access to their parallel computer system that made the experiments possible.

A The improved Rprop learning algorithm

The Rprop algorithm proposed by Riedmiller and Braun is one of the best performing first-order learning methods for neural networks with arbitrary topology in terms of convergence speed, accuracy and robustness with respect to its parameters [42, 35, 7]. We employ an improved version of the algorithm called iRprop$^+$, which was introduced in [36]. Figure 12 describes iRprop$^+$ in pseudocode. Herein w_{ij} denotes the weight in a neural network from neuron j to neuron i and $\mathcal{E}$ is an arbitrary error measure that is differentiable with respect to the weights. Bias parameters are regarded as being weights from an extra input; superscripts indicate the learning epoch (iteration). The weight update-values (step-sizes) $\Delta_{ij}^{(0)}$ have been initialized to 0.0125. The update-values are bounded by $\Delta_{\min} = 0$ and $\Delta_{\max} = 50$. The increase factor $\eta^+ = 1.1$ and the decrease factor $\eta^- = 0.5$ have been chosen.

For a comparison of iRprop$^+$ with other Rprop variants the reader is referred to [36].

for each w_{ij} **do**

if $\frac{\partial \mathcal{E}}{\partial w_{ij}}^{(t-1)} \cdot \frac{\partial \mathcal{E}}{\partial w_{ij}}^{(t)} > 0$ **then**

$\Delta_{ij}^{(t)} := \min\left(\Delta_{ij}^{(t-1)} \cdot \eta^{+}, \Delta_{\max}\right)$

$\Delta w_{ij}^{(t)} := -\operatorname{sign}\left(\frac{\partial \mathcal{E}}{\partial w_{ij}}^{(t)}\right) \cdot \Delta_{ij}^{(t)}$

$w_{ij}^{(t+1)} := w_{ij}^{(t)} + \Delta w_{ij}^{(t)}$

elseif $\frac{\partial \mathcal{E}}{\partial w_{ij}}^{(t-1)} \cdot \frac{\partial \mathcal{E}}{\partial w_{ij}}^{(t)} < 0$ **then**

$\Delta_{ij}^{(t)} := \max\left(\Delta_{ij}^{(t-1)} \cdot \eta^{-}, \Delta_{\min}\right)$

if $\mathcal{E}^{(t)} > \mathcal{E}^{(t-1)}$ **then** $w_{ij}^{(t+1)} := w_{ij}^{(t)} - \Delta w_{ij}^{(t-1)}$

$\frac{\partial \mathcal{E}}{\partial w_{ij}}^{(t)} := 0$

elseif $\frac{\partial \mathcal{E}}{\partial w_{ij}}^{(t-1)} \cdot \frac{\partial \mathcal{E}}{\partial w_{ij}}^{(t)} = 0$ **then**

$\Delta w_{ij}^{(t)} := -\operatorname{sign}\left(\frac{\partial \mathcal{E}}{\partial w_{ij}}^{(t)}\right) \cdot \Delta_{ij}^{(t)}$

$w_{ij}^{(t+1)} := w_{ij}^{(t)} + \Delta w_{ij}^{(t)}$

fi

od

Figure 12: The iRprop^{+} algorithm with improved weight-backtracking scheme. The algorithm differs only in one line from the one proposed in [35].

References

[1] B. Sendhoff, *Evolution of Structures – Optimization of Artificial Neural Structures for Information Processing*, vol. ISBN 3-8265-4155-3, Shaker Verlag, 1998.

[2] M. Mahner and M. Kary, "What exactly are genomes, genotypes and phenotypes? And what about phenomes?," *Journal of Theoretical Biology*, vol. 186, pp. 55–63, 1997.

[3] P. Schuster, "Molecular insights into evolution of phenotypes," in *Evolutionary Dynamics—Exploring the Interplay of Accident, Selection, Neutrality and Function*, J. P. Crutchfield and Peter Schuster, Eds., Santa Fe Institute Series in the Science of Complexity. Oxford Univ. Press, 2001.

[4] P. Stagge and C. Igel, "Structure optimization and isomorphisms," in *Theoretical Aspects of Evolutionary Computing*, L. Kallel, B. Naudts, and A. Rogers, Eds., Natural Computing, pp. 409–422. Springer-Verlag, 2001.

[5] P. Stagge and C. Igel, "Neural network structures and isomorphisms: Random walk characteristics of the search space," in *IEEE Symposium on Combinations of Evolutionary Computation and Neural Networks, ECNN 2000*, X. Yao and D. B. Fogel, Eds. 2000, pp. 82–90, IEEE Press.

[6] C. M. Bishop, *Neural Networks for Pattern Recognition*, Oxford University Press, Oxford, 1995.

[7] R. D. Reed and R. J. Marks II, *Neural Smithing*, MIT Press, 1999.

[8] X. Yao, "Evolving artificial neural networks," *Proc. of the IEEE*, vol. 87, no. 9, pp. 1423–1447, 1999.

[9] H. Braun, *Neuronale Netze: Optimierung durch Lernen und Evolution*, Springer-Verlag, 1997.

[10] D. Whitley, "Genetic algorithms and neural networks," in *Genetic Algorithms in Engineering and Computer Science*, J. Periaux and G. Winter, Eds., chapter 11. John Wiley & Sons, 1995.

[11] J. D. Schaffer, D. Whitley, and L. J. Eshelman, "Combinations of genetic algorithms and neural networks: A survey of the state of the art," in *COGANN-92: International Workshop on Combinations of Genetic Algorithms and Neural Networks*, L. Darrell Whitly and J. David Schaffer, Eds., 1992, pp. 1–37.

[12] G. F. Miller and P. M. Todd, "Designing neural networks using genetic algorithms," in *Proc. 3rd Int. Conference on Genetic Algorithms and their Applications*, J.D. Schaffer, Ed. 1989, pp. 379–384, Morgan Kauffman.

[13] M. Emmerich, M. Grötzner, and M. Schütz, "Design of evolutionary algorithms: A case study for chemical process networks," *Evolutionary Computation*, 2001, to appear.

[14] B. D. McKay, "Practical graph isomorphism," *Congressus Numerantium*, vol. 30, no. 30, pp. 47–87, 1981.

[15] B. D. McKay, "*nauty* user's guide (version 1.5)," Tech. Rep. TR-CS-90-02, Australian National University, Computer Science Department, 1990.

[16] M. R. Garey and D. S. Johnson, *Computers and Intractability, A Guide to the Theory of NP-Completeness*, W. H. Freeman and Company, New York, 1979.

[17] C. M. Friedrich and C. Moraga, "An evolutionary method to find good building-blocks for architectures of artificial neural networks," in *Sixth International Conference on Information Processing and Management of Uncertainty in Knowledge Based Systems, IPMU '96*, Granada, Spain, 1996, pp. 951–956.

[18] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error backpropagation," in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, D. E. Rumelhart, J. L. McClelland, and the PDP Research Group, Eds., vol. 1, pp. 318–362. MIT Press, 1986.

[19] R. K. Belew, J. McInerney, and N. N. Schraudolph, "Evolving networks: Using genetic algorithm with connectionist learning," Tech. Rep. CS90-174 (revised), Computer Science & Engg. Dept., Univ. of California at San Diego, USA, 1991.

[20] P. J. B. Hancock, "Genetic algorithms and permutation problems: A comparison of recombination operators for neural net structure specification," in *COGANN-92: International Workshop on Combinations of Genetic Algorithms and Neural Networks*, L. Darrell Whitley and J. D. Schaffer, Eds., Baltimore, 1992, pp. 108–122, IEEE Press.

[21] X. Yao and Y. Liu, "Evolving artificial neural networks for medical applications," in *Proc. of the Australian-Korea Joint Workshop on Evolutionary Computation*, Taejon, Korea, 1995, pp. 1–16.

[22] R. C. Lewontin, "Genotype and phenotype," in *Keywords in Evolutionary Biology*, Evelyn Fox Keller and Elisabeth A Lloyd, Eds., pp. 137–144. Havard Univ. Press, 1992.

[23] W. Grüner, R. Giegerich, D. Strothmann, C. Reidys, J. Weber, I.L. Hofacker, P.F. Stadler, and P. Schuster, "Analysis of RNA sequence structure maps by exhaustive enumeration II. Structures of neutral networks and shape space covering," *Monatshefte für Chemie (Chemical Monthly)*, vol. 127, pp. 375–389, 1996.

[24] R. A. Caruna and J. D. Schaffer, "Representation and hidden bias: Gray vs. binary coding for genetic algorithms," in *Proc. of the 5th Int. Conference on Machine Learning*, J. Laird, Ed. 1988, pp. 153–161, Morgan Kauffmann.

[25] M. Kimura, "Evolutionary rate at the molecular level," *Nature*, vol. 217, pp. 624–626, 1968.

[26] M. Conrad, "The geometry of evolution," *Biosystems*, vol. 24, pp. 61–81, 1990.

[27] C. Igel and M. Kreutz, "Operator adaptation in evolutionary computation and its application to structure optimization of neural networks," Tech. Rep. IRINI 2001-03, Institut für Neuroinformatik, Ruhr-Universität Bochum, 44780 Bochum, Germany, 2001.

[28] Marc Toussaint, "Self-adaptive exploration in evolutionary search," Tech. Rep. IRINI 2001-05, Institut für Neuroinformatik, Ruhr-Universität Bochum, 44780 Bochum, Germany, 2001.

[29] M. A. Shackleton, R. Shipman, and M. Ebner, "An investigation of redundant genotype-phenotype mappings and their role in evolutionary search," in *Proceedings of the International Congress on Evolutionary Computation, CEC 2000.* 2000, pp. 493–500, IEEE Service Center.

[30] R. Shipman, "Genetic redundancy: Desireable or problematic for evolutionary adaptation," in *4th International Conference on Artificial Neural Networks and Genetic Algorithms, ICANNGA'99*, 1999.

[31] K. Weicker and N. Weicker, "Burden and benefits of redundancy," in *Proceedings ot the Sixth Workshop on Foundations of Genetic Algorithms, FOGA 6*, W. Spears and W. Martin, Eds. 2001, Morgan Kaufmann.

[32] P. J. Angeline, G. M. Saunders, and J. B. Pollack, "An evolutionary algorithm that constructs recurrent neural networks," *IEEE Transactions on Neural Networks*, vol. 5, no. 1, pp. 54–65, 1994.

[33] D. B. Fogel, *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence*, IEEE Press, 1995.

[34] L. Prechelt, "Early stopping – but when?," in *Neural Networks: Tricks of the Trade*, G. B. Orr and K.-R. Müller, Eds., vol. 1524 of *LNCS*, chapter 2, pp. 57–69. Springer-Verlag, 1999.

[35] M. Riedmiller and H. Braun, "A direct adaptive method for faster backpropagation learning: The RPROP algorithm," in *Proceedings of the IEEE International Conference on Neural Networks.* 1993, pp. 586–591, IEEE Press.

[36] C. Igel and M. Hüsken, "Improving the Rprop learning algorithm," in *Proceedings of the Second International ICSC Symposium on Neural Computation, NC 2000*, H. Bothe and R. Rojas, Eds. 2000, pp. 115–121, ICSC Academic Press.

[37] T. Sasaki and M. Tokoro, "Comparison between Lamarckian and Darwinian evolution on a model using neural networks and genetic algorithms," *Knowledge and Information Systems*, vol. 2, pp. 201–222, 2000.

[38] P. J. Bowler, "Lamarckism," in *Keywords in Evolutionary Biology*, Evelyn Fox Keller and Elisabeth A Lloyd, Eds., pp. 188–193. Havard Univ. Press, 1992.

[39] K. Chellapilla, "Evolving computer programs without subtree crossover," *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 3, pp. 209–216, 1998.

[40] G. Rudolph, *Convergence Properties of Evolutionary Algorithms*, Kovač, Hamburg, 1997.

[41] J. R. McDonnell and D. E. Waagen, "Neural network structure design by evolutionary programming," in *Proceedings of the Second Annual Conference on Evolutionary Programming, EP '93*, D. B. Fogel and W. Atmar, Eds., San Diego, CA, USA, 1993, pp. 79–89, Evolutionary Programming Society.

[42] M. Riedmiller, "Advanced supervised learning in multi-layer perceptrons – from backpropagation to adaptive learning algorithms," *Computer Standards and Interfaces*, vol. 16, no. 5, pp. 265–278, 1994.