Trait-based Diversity Measurement in Genetic Algorithms using Artificial Neural Networks

Kent Munthe Caspersen
Department of Computer Science
Aalborg University, Denmark

Elias Khazen Obeid
Department of Computer Science
Aalborg University, Denmark

Martin Bjeldbak Madsen
Department of Computer Science
Aalborg University, Denmark

Abstract

Controlling diversity in a genetic algorithm's population is crucial for finding the global optimum. When diversity is overlooked, premature convergence is the consequence, which possibly only leads to local optima. We propose a trait-based diversity measurement for genetic algorithms using artificial neural networks, which we call Neural Network Trait Diversity (NNTD).

Experiments are conducted to compare NNTD to the genotypic diversity measure of Hamming distance, and a phenotypic fitness-based diversity measure. We argue that both of these measures have weaknesses that NNTD overcomes. Our experiments show that NNTD consistently mirrors the intuition of trait diversity in populations, better than Hamming distance and fitness-based diversity measurement methods, which seem less predictable. Interestingly, the experiments also show that there might be a connection between the diversities returned by Hamming distance and NNTD among neural networks.

1. Introduction

For many problems in computer science, using a greedy strategy to solve the given problem will not always lead to an optimal result. A greedy strategy chooses the next step in solving the problem by considering how promising each step is at that given moment. Consider a mountain climber, who is randomly placed in the midst of peaks, and wants to reach the highest of these peaks. A greedy strategy would be to move in any direction he thinks is the best, as long as he climbs upwards and never downwards. Using this strategy, he is definitely able to climb up some peak, but it is not certain that this peak is the highest. Unfortunately, his strategy is to never back down, so now he is stuck at what we call a local optimum. What he wanted was the global optimum, the highest peak.

Genetic algorithms (GAs) can be used to search for a global optimum in many types of problems, such as optimization problems. A GA manages a number of individuals, which together constitute a population. Each of the individuals represents a candidate solution to the same problem. Neural networks, among other data structures, can be chosen to represent individuals. In this article, we use neural networks as individuals, and the terms *neural network* and *individual* are used interchangeably. A neural network takes a number of inputs, does some internal calculations, and yields a number of outputs. The neural network as a whole can be interpreted as a solution to the problem. For instance, for the particular problem of adding two integers, the neural network can take two integers s and t as input, and output a single value, which ideally should equal $s+t$.

Most often, the individuals in a GA do not adequately solve the problem in question. However, different individuals may have good solutions to different aspects of the problem. Individuals maintained by the GA procreate to form offspring that combine the best traits from both its parents. Intuitively, the combination of these traits constitutes a better solution to the given problem. Each individual has a fitness value, which indicates how adequately it solves the problem in question.

To identify a constructive combination of traits, a diverse set of individuals is required. If diversity is not maintained, only a local optimum of the population's available traits will be explored, and a global optimum

might never be found [4, 15].

We believe it is essential that a diversity measure reflects the difference in traits among individuals. Different combinations of traits induce different behaviour. Since a *trait* is a rather vague term, we introduce a clear definition of traits among neural networks: "*two neural networks have different traits, if they produce different outputs for some input*". We now argue why we believe that neither fitness-based nor genotypic diversity measures catch a diversity among traits, or *trait diversity*.

Genotypic diversity is concerned with the representation of individuals, while phenotypic diversity is concerned with their behaviour and is often measured based on their fitness. Consider two individuals who try to find the highest peak on an elevation map, where the fitness of each individual is proportionate to the height of the peak they return. The two individuals might have completely different strategies, causing them to get stuck on two different peaks. If both peaks happen to have the same height, the two individuals will have the same fitness, and hence a fitness-based diversity measure will return a low diversity, even though the two individuals have different traits.

Two individuals of different genotypes can have the same traits. An example of such two neural networks is shown in fig. 1. No matter what input they receive, their output will always be the same. They are genotypically diverse, because their genetic structure is different, but they are not trait diverse, because they produce the same output. The genetic structure could be represented as bit string as shown in fig. 1. We develop a new method for measuring phenotypic diversity in GAs using neural networks as individuals. We claim that our method better reflects different traits among the individuals.

This paper is organized as follows. Section 2 introduces the concepts used in our diversity measure, and can be skipped if the reader knows about genetic algorithms and neural networks. Section 3 describes NNTD, which is experimentally evaluated and compared to other measures in section 4. In section 5 we present or conclusion, followed by section 6 in which we present our thoughts for future work.

2. Preliminaries

In this section we present the concepts used in this paper. We begin with an introduction of artificial neural networks, followed by an introduction of genetic algorithms, a description of different replacement rules, and existing diversity measures.

2.1. Artificial Neural Network

An artificial neural network is a finite directed graph that, from the outside, can be seen as a black box, which given the values $x_1, x_2, \ldots, x_n$, outputs the values $y_1, y_2, \ldots, y_m$. With the right internal structure, a neural network can be used for a variety of applications, e.g. recognition of handwriting [19], where the intensities of different pixels in an image are used as input, and a set of output values are produced. The highest output value represents the predicted letter of the network.

We now describe the structure and inner workings of neural networks to understand how these output values are calculated based on input values.

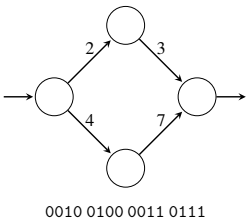
Neurons. Nodes in the graph of a neural network are called neurons. Three types of neurons exist: input, hidden, and output. Each input neuron receives one of the input values $x_1, x_2, \ldots, x_n$ and forwards the same value to each neuron it has an edge pointing to. Hidden and output neurons take a number of values as input from edges exiting other neurons, applies a weight to each value, sums them, and then applies a function to produce a single output value. The function applied is called the transfer or activation function, and is the same for all hidden and output neurons in the network. The output value of neuron i is recursively expressed by

$$y_i = \begin{cases} x_i & \text{if } i \text{ is an input neuron} \\ \sigma\left(\sum_{j=1}^{n} w_{ji} y_j\right) & \text{otherwise} \end{cases}$$

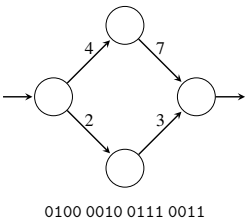
where x_i is the ith input, n is the amount of neurons, y_i is the output value of neuron i, w_{ji} is the weight of the edge from neuron j to i (0 if no connection exists), and σ is the activation function, typically defined to be the sigmoid function

$$\sigma(t) = \frac{1}{1+e^{-t}}$$

In a feed-forward network, neurons are placed in one or more layers in an acyclic directed graph, where each neuron in layer i is connected to every neuron in layer $i+1$. The value(s) output by the last layer, or the output layer, becomes the output of the neural network. Figure 2 illustrates the generic graph structure of a feed-forward neural network with a single hidden layer. If a neuron is given the value 0 on all of its inputs, most activation functions will only allow that neuron to output 0 as well. This is not desirable for all applications, and is often overcome by giving each neuron a bias value. The bias value of a neuron is added to the sum of inputs it receives, before finally applying the activation function. Thus, if a neuron receives only 0's as input, it will output



(a) An artificial neural network with connections and weights.



(b) An artificial neural network equivalent to fig. 1a.

Figure 1: Networks with same phenotype, but different genotypes. Each weight is represented by four bits.

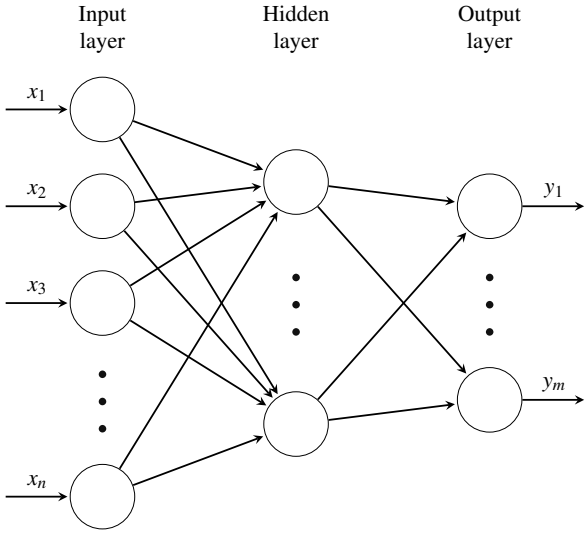


Figure 2: Structure of a neural network.

the value obtained by applying the activation function to its bias value.

2.2. Genetic algorithms

Genetic algorithms are optimization algorithms, which imitate the process of natural selection in the search of a global optimum.

Individuals and chromosomes. GAs maintain a list of *individuals*, which together form a *population*. Each individual represents a possible solution to the optimization problem in question and has a fitness value, denoting how adequately the individual can solve the optimization problem. Individuals in GAs can take on any form of data structure, as long as they wholly represent a possible solution to the problem. An individual is encoded by its *chromosome*, which is typically represented by a bit string. A substring of this bit string is called a *gene*. The GA manipulates the bit strings to form new individuals. Therefore, using a GA requires a way of encoding/decoding between an individual's chromosome and a solution to the given problem. We refer to the individuals' chromosomes or bit strings synonymously.

The population used by a GA typically has a fixed number of individuals, each initialized with a random chromosome when the GA is run. That is, the bit string representing the chromosome is initialized with random bits. As the GA iterates, new individuals are made by combining and modifying chromosomes from existing individuals of the population. In steady state GAs, some of the new individuals will replace older individuals according to a replacement rule. In contrast, a generational GA will choose only from offspring when forming the next generation [14, 16, 18]. We will focus on steady state GAs, using the term *generation* G_n to denote the content of the population after $n-1$ iterations. A replacement rule then determines which individuals of the nth generation G_n, and their offspring β_n are selected to form the next generation G_{n+1}.

Neural networks as individuals. We assume that every neural network in a population will have the same architecture. That is, the number of neurons, the size of each layer, and how neurons are connected is the same.

Thus, neural networks differ only in their weights between neurons and the bias of each neuron. Each individual is therefore represented only in terms of the weights and biases.

Crossover and mutation. In natural evolution, a pair of individuals come together to produce one or more children, having genes from both parents. In GAs, the process of procreation is done by performing a *crossover* of the two parent individuals' chromosomes. Parts of each parent's bit strings are used to create the child individual.

Mutations can also occur randomly at any point in time upon creating a child individual. If genes are encoded as bit strings, then a mutation arbitrarily toggles one or more bits. This ensures that any gene can be formed.

Fitness functions. A *fitness function* must be defined to calculate the desirability of each individual. This function is used to define the most fit individuals in a population. By giving more fit individuals a greater chance of reproducing, the intuition is that more fit individuals will be created, having the best traits from each of their parents.

2.3. Crowding

Methods to overcome decreasing diversity in a population include inserting random immigrants (randomly initialized individuals) [3], using complex population structures to lower the gene flow, and the use of special selection procedures [5, 15], known as crowding. How a particular crowding method works is commonly described by a replacement rule.

Greedy. Common descriptions of GAs that do not try to account for premature convergence, construct generation G_i from the n most fit individuals from $G_{i-1} \cup \beta_{i-1}$, where n is a fixed population size. We will refer to this as the *Greedy replacement rule* [8].

Ancestor Elitism. The Ancestor Elitism replacement rule is similar to $(\mu + \lambda)$ and generation gap algorithms [9], but we developed it independently of these algorithms.

Generation G_{i+1} is made from generation G_i as follows: every individual from G_i is put into G_{i+1}. Any offspring individual in β_i who only has a single parent (made from mutation) and is more fit than its parent, will replace that parent in G_{i+1}. Any individual in β_i who has two parents (made from crossover) and is more fit than both of its parents, will replace both parents in G_{i+1} by itself and a random immigrant. Finally, the 50 % least fit individuals in G_{i+1} are replaced by random immigrants.

Single Parent Elitism. Single Parent Elitism is similar to Restricted Tournament Selection [9], but we developed it independently of that method.

G_{i+1} is made from G_i as follows: every individual from G_i is put into G_{i+1}. Any individual in β_i who only has a single parent (made from mutation) and is more fit than its parent, will replace that parent in G_{i+1}. Any individual in β_i who has two parents (made from crossover), and is more fit than a randomly chosen parent, will replace its parent in G_{i+1} by itself.

Mass Extinction Explore Exploit. Inspired by the changes between exploration and exploitation phases used by Ursem [15], we introduce the following replacement rule, which we will refer to as the *Mass Extinction Explore Exploit* (MEEE) replacement rule. The MEEE replacement rule is equivalent to Single Parent Elitism, except that it additionally performs a mass extinction when diversity drops below a threshold. In any generation G_i, if $d(G_i) < \alpha$, we replace the 50 % least fit individuals from G_i by random immigrants, where $d(G_i)$ denotes the diversity of a generation G_1 and α is a threshold ratio.

2.4. Diversity measures

It is often argued that the weakness of GAs is the fall in diversity over generations, often resulting in premature convergence [2, 6, 21]. Therefore, it is important to measure diversity and be able to act accordingly to maintain a desired diversity.

Genotypic diversity measures. To measure genotypic diversity, methods to compute the distance between any two individuals' encoded bit strings are often used as diversity measures. The diversity between a set of bit strings can then be expressed as the average distance between any two bit strings. Summation can also be used instead of averaging, which is merely a convenient optimization.

Genotypic diversity can be measured using methods such as *Hamming distance* and *Levenshtein distance*. The Hamming distance between two bit strings A and B of equal length is the number of indices i, such that $A[i] \neq B[i]$. The Levenshtein between these two bit strings is the number of bits that must be inserted, deleted, or substituted to change A into B. To compare these two measures, consider the following example

$$01010101 \quad (1)$$
$$10101010 \quad (2)$$

where the difference is 8 when using Hamming distance, because all 8 bits differ, and 2 when using Levenshtein distance, because transforming (1) into (2) can be done by deleting the first bit and prepending a 0, totalling 2 operations [20].

If we define F to be a set of neural networks, the complexity of calculating h_{ij}, where h_{ij} is the Hamming distance between all pairs of neural networks $f_i, f_j \in F$, is $\mathrm{O}(l)$, linear in the length of an encoded network's bit string l. The complexity of computing Hamming distance for all $f \in F$ is thus $\mathrm{O}\left(|F|^2 \cdot l\right)$.

The complexity of calculating v_{ij}, where v_{ij} is the Levenshtein distance between all pairs of neural networks $f_i, f_j \in F$ is $\mathrm{O}\left(l^2\right)$ [7]. Therefore, computing the Levenshtein distance between all individuals in F yields the complexity $\mathrm{O}\left(|F|^2 \cdot l^2\right)$.

Phenotypic diversity measures. Phenotypic diversity measures include computing the standard deviation of fitness values, the average number of unique fitness values in a population, and entropy-based methods [2, 20].

One advantage of fitness-based diversity measures is that no extra computations are associated with calculating the diversity, because the fitness values have already been calculated by the GA to assess how fit each individual is [10].

Taking the precomputation of fitness values into account gives these diversity measures an advantage when it comes to complexity. To calculate the number of distinct individuals, we can use a hash table. If we assume it unlikely to have a clash between hash values and choose to ignore this, we can achieve a complexity of $\mathrm{O}(|F|)$, linear in the size of the population.

Other measurements. Some diversity measures exist that are neither genotypic nor phenotypic. For instance the *Ancestral ID* method, which assigns a unique ID to each individual in the initial population. Every mutated individual receives a new unique ID while every child gets the ID of one of its parents. The diversity is then based on the uniqueness of IDs in a population [20].

3. Neural Network Trait Diversity

In the following, we propose a method for measuring trait diversity, which we call *Neural Network Trait Diversity* (NNTD). NNTD aims to reflect the diversity of different traits among individuals.

NNTD is based on Simpsons Diversity Index (SDI), which is a diversity measure used in ecology to quantify the biodiversity of a habitat [13]. SDI is the probability that two randomly chosen samples belong to different species [17].

3.1. Algorithm

NNTD assigns each individual a species, based on a random input. Afterwards, SDI is calculated. This process is repeated for a number of random inputs, and NNTD is defined as the average SDI. To calculate SDI, we must define a method for distributing the individuals of a population into a number of species.

Let $F = \{f_1, f_2, \ldots, f_a\}$ be the set of neural networks contained in a population, all with the same architecture of n input and m output neurons, and let r be any random input for the neural networks. r must then be an n-tuple of values. If $b_m b_{m-1} \ldots b_1$ is the binary representation of a number i, we define the species $S_i(r)$ to contain any individual $f \in F$, that given r as input satisfies

$$\forall j,h \in \{1,2,\ldots,m\} (b_j \rightarrow (o_j \geq o_h)) \tag{3}$$

where $o_k \in O$ is the value of the kth output neuron of neural network f. As an example, assume that we have a neural network with six output neurons. If the third output neuron has the highest output when given r as input, the neural network will belong to species $S_4(r)$, because the `1` in `000100` (the binary representation of 4) match the indices of the output neurons with the highest value. If multiple output neurons have the highest value, say the first and third output neuron, the neural network will belong to species $S_5(r)$ since `000101` in binary is 5 in decimal. If all output neurons have the same value, the neural network belongs to species $S_{63}(r)$ since `111111` in binary is 63 in decimal. Once each individual is assigned a single species, we calculate SDI given r, denoted D_r, by inserting into the formula for SDI [17]

$$D_r = 1 - \frac{\sum_{q \in Q_r} (|q|(|q|-1))}{|F|(|F|-1)} \tag{4}$$

where $Q_r = \{S_1(r), S_2(r), \ldots, S_{2^m-1}(r)\}$ is the set of species for random input r. The NNTD D (the actual diversity) is calculated as the average SDI given a number of random inputs $R = \{r_1, r_2, \cdots, r_p\}$

$$D = \frac{\sum_{x=1}^{|R|} D_x}{|R|} \tag{5}$$

One disadvantage of NNTD is that it relies on random inputs, which means that fewer random inputs implies less statistical significance. Also, the choice of random inputs might affect the calculated NNTD. We suggest that each random input is chosen according to the probability of a neural network receiving that input in the intended application. Due to the nature of NNTD, it is not suitable for continuous problems. Consider for instance two neural networks f_1 and f_2, used for approximating a function g, where f_1 and f_2 both have only a single output neuron. To tell how different f_1 and f_2 behave, it is necessary to distinguish between different values of each neural network's output neuron. Since NNTD defines species based on the output neuron with the highest value, and we in this case only have a single output neuron in f_1 and f_2, the two neural networks will always belong to the same species $S_1(r)$ for all $r \in R$, yielding a diversity of 0.

3.2. Complexity

Computations associated with NNTD are split into two halves. The first half distributes neural networks

into a number of species for each random input corresponding to eq. (3). These species are then used in the second half, where SDI is calculated for each random input, corresponding to eq. (5).

The first part of the computation is carried out as follows. For each random input $r \in R$, every neural network $f \in F$ must calculate an output based on random input r. Each output neuron $o \in O$ belonging to f is now considered once to calculate the particular species f belongs to. The complexity of the first part is thus $\mathrm{O}(|R| \cdot |F| \cdot (t + |O|))$, where t is the time required to calculate the output of a neural network given any input. When calculating the output of a neural network, every neuron in the network must be considered. Therefore, t is an upper bound of $|O|$. This allows us to reduce the complexity to $\mathrm{O}(|R| \cdot |F| \cdot t)$ for the first half of NNTD.

In the second part, we note that SDI considers every species once for every random input, and hence this part has complexity $\mathrm{O}(|R| \cdot |Q|)$, where Q is the largest set of non-empty species for all random inputs R. Since each neural network is put in only one species, $|F|$ is an upper bound of $|Q|$, yielding the complexity $\mathrm{O}(|R| \cdot |F|)$ for the second half.

Combining both halves results in a total complexity of $\mathrm{O}(|R| \cdot |F| \cdot t + |R| \cdot |F|)$, which is $\mathrm{O}(|R| \cdot |F| \cdot t)$.

3.3. Complexity comparison

Out of all diversity measures, the complexity of the fitness-based measure is difficult to compete with.

Comparing the complexities of Hamming distance and NNTD allows some interesting reductions and assumptions. Upon comparison, we can remove the common factor F in both of two complexities $\mathrm{O}(|R| \cdot |F| \cdot t)$ of NNTD and $\mathrm{O}\left(|F|^2 \cdot l\right)$ of Hamming distance resulting in $\mathrm{O}(|R| \cdot t)$ and $\mathrm{O}(|F| \cdot l)$, respectively.

Additionally, we can make an assumption about how the length of a bit string l is related to the value t, which is the number of steps required to calculate the output of a neural network given an arbitrary input. Let E denote the set of edges in a neural network. Each weight must be considered only once when the output of a neural network is calculated. Applying the activation function one time can be considered a constant amount of work. In the worst case, every neuron will only have a single input connection, hence the activation function will be calculated at most $|E|$ times. Therefore, we have that $t = \theta(|E|)$. If we assume that $l = \theta(|E|)$, we also get that $t = \theta(l)$. This assumption is reasonable, since it is fulfilled if the weight of each edge is separately encoded in the bit string. With this assumption, we can simplify the complexity comparison of NNTD and Hamming distance to a comparison of the complexities $\mathrm{O}(|R|)$ and $\mathrm{O}(|F|)$, respectively.

In this manner, we can conclude that the complexities of NNTD and Hamming distance are asymptotically the same, whenever $|R| = c \cdot |F|$ for some constant c. That is, whenever the number of random inputs chosen for NNTD is a constant times the population size.

4. Experiments

In this section, we evaluate our diversity measurement (NNTD) by using it to measure diversity of a population in two different environments.

In the first environment, which we refer to as the static environment, the GA does not iterate. For the three maximization problems introduced in section 4.3, we create an initial population using a constraint c and a significance α. We choose c such that we have an intuition about how the trait diversity of individuals are dependent on α. By experimenting, we see if it is likely that NNTD catches this dependency.

In the second environment, which we refer to as the dynamic environment, the GA runs for a number of iterations on each of the three problems. For each problem, we perform an experiment using each of the four replacement rules introduced in section 2.3. During all of the experiments, we measure the diversity using NNTD as well as each of the diversity measures described in section 2.4, except Levenshtein because it is too computationally expensive.

By these experiments we can conclude whether the diversities returned by NNTD better match the expected trait differences of the problems under study, compared to the other diversity measures.

4.1. Encoding an individual

For each neural network, any weight and bias is encoded with a fixed number of bits, q and w, respectively. The bit string is constructed in an ordered manner, such that the first q bits represent the weight of the connection between the first input neuron and the first hidden neuron, the next q bits represent the weight of the connection between the first input neuron and the second hidden neuron, and so forth. Figure 1 shows an example of two neural networks and the bit string that encodes each of them. If biases are used by the GA, these are encoded right after the weights, and ordered such that the first w bits encode the bias of the first neuron, the next w bits encode the bias of the second neuron, and so forth.

We limit the weights between neurons and biases of any neuron to lie in the range $[-5,5]$. This is because we use the sigmoid activation function, for which the

Parameter		Specification
Number of runs		100
Generations per run	snake	2000
	XOR	2000
	leaf	500
Population size		100
Selection method		rank-based

Table 1: GA parameters used throughout experimenting.

value $f(x)$ changes only a little when $5 \leq x \leq -5$. If a weight or bias is encoded by the u bits $b_u b_{u-1} \cdots b_1$, we decode its real value w using the formula

$$w = \begin{cases} \left(\sum_{i=1}^{u-1} b_i 2^{i-1}\right)(1-2b_u)\frac{5}{2^u-1} & \text{if } u \geq 2 \\ 5b_1 & \text{if } u = 1 \end{cases}$$

For the case where $u \geq 2$, the first $u-1$ bits represent an increasing sequence of powers of two. The last bit b_u negates the entire value if set (by the factor $1-2b_u$). The value is finally normalized to the range $[-5,5]$ (by the factor $\frac{5}{2^u-1}$). The number of bits u used to encode a weight or bias is dependent on the problem in question.

4.2. Parameter settings

For any GA, we use a population size of 100 individuals. For each iteration, 100 offspring individuals are created. Half of the offspring is cloned from a random parent. The parents are chosen by using a rank-based selection, and then mutated. Rank-based selection orders individuals according to their fitness, and assigns probability of selection to each individual equal to their order, such that the most fit individual has the best chance of getting selected. The other half of the offspring is made by performing crossover between two random parents, also using rank-based selection. These settings are presented in table 1.

Three crossover methods are used: one-point, two-point, and uniform crossover. When a new individual is chosen for creation using crossover, there is an equal chance for any of the crossover methods to be used. Each offspring created by crossover has a 10 % chance to be mutated. When mutating an individual, each bit in its bit string has a 5 % chance to be assigned a random boolean value. For specific implementation details, we refer to the source code [11].

4.3. Problems under study

We hereby explain the three discrete problems we perform our experiments on. The first problem of approximating the XOR function is interesting, since it is the simplest boolean function that is not linearly separable. This fact has made it quite popular in neural network research communities [8]. Next is the classification of leaves from a dataset in the UCI Machine Learning Repository, which is interesting because it is radically different from the XOR problem. XOR is a simple and well defined function, whereas classifying leaves is more complex and depends on observations in nature, which may contain noise. Finally, the Snake game differs in that it is an agent decision problem, where each decision changes the information available to the agent.

XOR. We use a neural network to approximate the XOR operator between two 8-bit strings. To evaluate the fitness of an individual solving this problem, we calculate the XOR of 1000 random two 8-bit strings. For each instance, we determine how many bits the individual calculates correctly. The fitness is defined as the average number of bits correctly calculated for each pair of strings. For the 8-bit XOR problem, any fitness value will thus be a real number in the range $[0,8]$. The XOR between two bits of random values has equal probability of yielding the value 0 or 1. Therefore, randomly guessing a solution to the XOR between two 8-bit strings is expected to yield a fitness of 4 in the average case.

The network has 16 input neurons, 16 hidden neurons, 8 output neurons, 3 bits per weight, and 1 bit per bias for each hidden and output neuron. We represent the two 8-bit strings as being side by side. So, any output neuron i represents the XOR between the two input neurons i and $i+8$.

We have used as few neurons and bits to represent weights and biases as possible, while still being able to guarantee that the maximum fitness value of 8 is achievable. The same random seed is always used for generating the 1000 problem instances.

Leaf classification. This problem requires classifying instances from the Leaf data set from the UCI Learning Repository [1, 12]. The individuals are given 16 properties about a leaf and have to decide between 40 types of leaves. Fitness is evaluated based on how many instances out of the entire data set the neural network correctly classifies. The implementation consists of 16 input neurons, 10 hidden and 40 output neurons. The output neuron with the highest value decides the classification. Each weight is encoded by 9 bits and neurons have no bias.

Snake. Snake is a game found on old Nokia cell phones, where you control a snake around a grid to pick up pieces of food. Every time a piece of food is collected, both the length of the snake and your score increases by 1. You lose if the snake head hits its body or one of the

edges of the grid. At all times, the grid contains only a single piece of food. The game becomes harder as the length of the snake increases and, as the snake is constantly moving, the challenge in not trapping oneself gets tougher.

We use a neural network to play a game of Snake in a 10×10 grid with an initial snake length of 5 units. We have defined the fitness of a neural network to be

$$\rho + \frac{\rho}{s/1000}$$

where ρ is the amount of collected food, and s is the total number of steps the snake has been alive. The game is constrained such that the snake can only change its direction 90° per step. The neural network has 6 input neurons, each receiving information about the game's state, as shown by

1. $\{-1,0,1\}$ food is left of, verti. aligned, or right of
2. $\{-1,0,1\}$ food is above, hori. aligned, or below
3. $\{0,1\}$ death upon moving up
4. $\{0,1\}$ death upon moving down
5. $\{0,1\}$ death upon moving right
6. $\{0,1\}$ death upon moving left

where the first two inputs are relative to the snake's head, giving information of the whereabouts of the food in the grid. The neural network has 4 output neurons, one for each direction. The neuron with the highest value determines which direction the snake moves. The neural network uses 5 hidden neurons, 9 bits per weight and neurons have no bias. For every game of Snake, we always use the same random seed to decide the positions where pieces of food will spawn.

4.4. Static experiments

We perform two different static experiments, which differ in the way they constrain the initially generated population.

Initial similarity. Here, we introduce the variable *initial similarity*, which is a real value in the range $[0,1]$. When making an initial population, an initial similarity of α means that α of the individuals in the population will have the exact same genotype, and $(1-\alpha)$ of the individuals are completely random. An initial similarity of 1 means that all genotypes are the same, and hence the traits of all individuals are the same as well. In this case, we expect the lowest diversity possible. As initial similarity increases, more genotypes will be identical, and hence more individuals will have similar traits. We therefore expect the diversity to decrease as initial similarity is increased. An initial similarity of 0 means that all genotypes are random, and hence we expect the most diverse traits among individuals to be found here.

Initial mutation. Here, we introduce the variable *initial mutation*, which is also a real value in the range $[0,1]$. When making an initial population, an initial mutation of α affects the population in the following way. A random genotype is created and given to every individual in the population, such that all individuals have an identical genotype. Now, the bit string of each individual is mutated. Each bit will with probability α be set to a random boolean value. We expect to see increasingly different traits, as a result of mutating bit strings.

Results of initial similarity. The results of various initial similarity values shown in fig. 3 play along with our intuition. For each of the problems, each diversity measure's output gradually falls upon increasing the amount of similar individuals in the population. Interestingly enough, at an initial similarity of 0, we expect all diversity measures to output maximum diversity. Here, NNTD outputs the maximum possible diversity of 1, whereas Hamming distance outputs a diversity that is only half of its maximum. One could assume that you could just double Hamming distance's diversity, but it is actually possible for the Hamming distance between two individuals to output 1, if every pair of bits are different. It seems that NNTD captures the chaotic nature of a completely random population better than the other two measures, with a larger fall in diversity over time.

Results of initial mutation. By changing the initial mutation in a population, with the results shown in fig. 4, we notice a larger difference between diversity measures. At an initial mutation rate of a mere 1 %, NNTD takes a great leap compared to the other two measures. For Hamming distance, changing only a few bits of a genotype will only cause a small change in diversity. This is because Hamming distance measures diversity based on genotypes. Consider for instance changing just a single bit of a chromosome. This bit could be a sign bit that causes a significant change to the neural network's traits, or it could cause no change at all. Hamming distance neglects this fact and produces the same diversity measure regardless of whether the bit caused a change of traits for the individual or not.

For the fitness-based diversity measure, it must be noted that the fitness values are dependent on the particular problem in question and how one chooses to define the fitness function. Consider for instance the problem of making an AI for the game Snake. We have experienced that if we define the fitness only in terms of how many pieces of food a snake collects, then about 98 % of randomly initialized individuals get a fitness value of 0. This is the reason why we chose a fitness function that also takes into account the number of steps a snake has

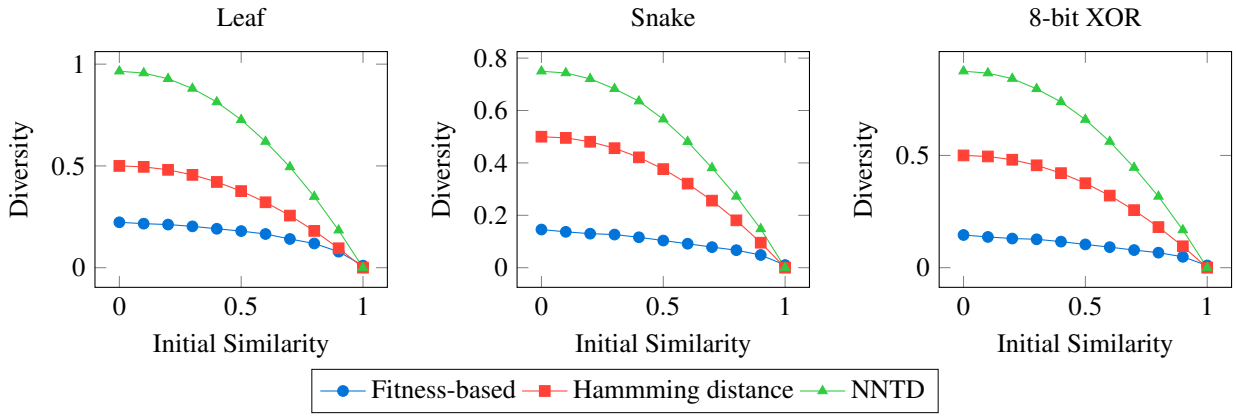


Figure 3: Average diversity for each diversity measure on each data set over intervals of initial similarity. Each point represents the average of 100 runs.

been alive. Despite yielding more diverse fitness values, it also notably increased the fitness values obtained after just 100 iterations. This does not mean that a fitness function cannot reflect differences in traits, but it shows that how one defines the fitness function is crucial, if one wishes to catch the different traits among individuals.

4.5. Dynamic experiments

Our main goal with the experiments we perform is to see how diversity progresses throughout the many iterations of a GA. To measure diversity, we use a fitness-based measurement, Hamming distance, and our own measure, NNTD. For each experiment we use the four replacement rules: the Greedy, Ancestor Elitism, Single Parent Elitism, and MEEE, described in section 2.3.

For the snake and XOR problems, we measured the average diversity of 100 runs over 2000 iterations, meaning each run initializes an entirely new population, and each population goes through 2000 iterations of procreation in the search for optimal solutions. After each iteration, diversity of the population is measured with each diversity measure. We then take the average diversity for each generation of the 100 runs. For the leaf data set, we ran the same experiment only through 500 generations. This is due to the fact that we saw no improvement in the population beyond this number of iterations.

Results. The results are presented in figures 5 through 7. The experiments performed on the leaf data set are presented in fig. 5, the XOR experiments are presented in fig. 6, and the Snake game results are presented in fig. 7. Each of these three figures contain four plots. The first three of these plots illustrate each diversity measure, with the fourth plot illustrating the average fitness for the four replacement rules throughout the iterations.

Results of Leaf An interesting observation in fig. 5 is the irregular spike in diversity for the fitness-based measure around generation 50. Why this happens is hard to say, due to the stochastic nature of GAs. NNTD and Hamming distance measures look very similar for this data set.

Results of XOR The results of each diversity measure presented in fig. 6 are very similar to those of the Leaf data set. Once again, we see the irregular spike in the plot presenting the fitness-based measure. The results returned by NNTD and Hamming distance also look very similar for this problem.

Results of Snake The results shown in fig. 7, again show similar slopes between the NNTD and Hamming distance measures. The fitness-based measure is again very irregular.

The big picture If we compare experiments run on each data set illustrated in figures 5 through 7, we see that the fitness-based diversity measure stands out from the other two in each of the experiments. For example, the fitness-based plots have some irregular slopes, which are not easily explained. Because of this, combined with the fact that multiple individuals can have the same fitness yet different traits, we believe that fitness-based diversity does not accurately represent the actual diversity. Fitness-based diversity seems much more dependent on the problem, as its results are not regular. In each figure presenting the dynamic experiments, the plot presenting the fitness-based measurement is notably different from the two other plots, presenting NNTD and Hamming distance.

The experiments also show that there might be a

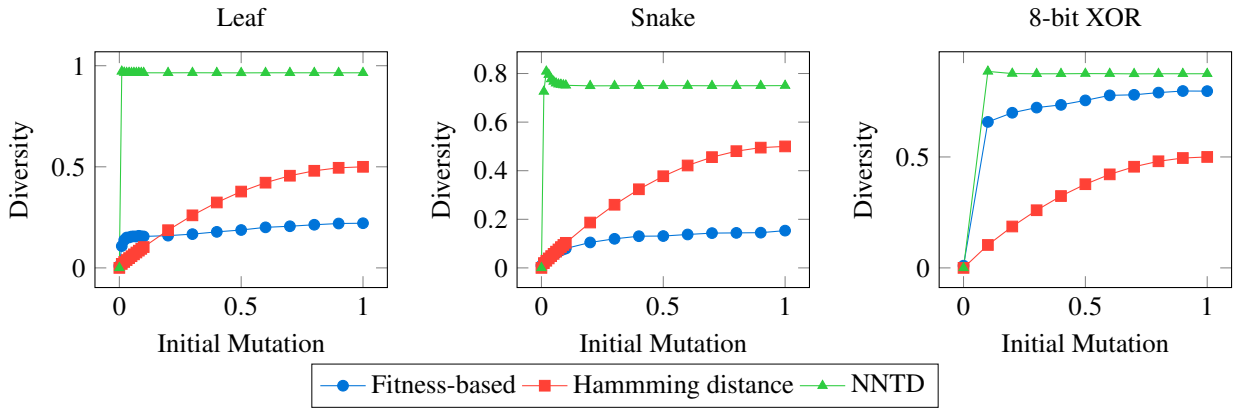


Figure 4: Average diversity for each diversity measure on each data set over intervals of initial mutation. Each point represents the average of 100 runs.

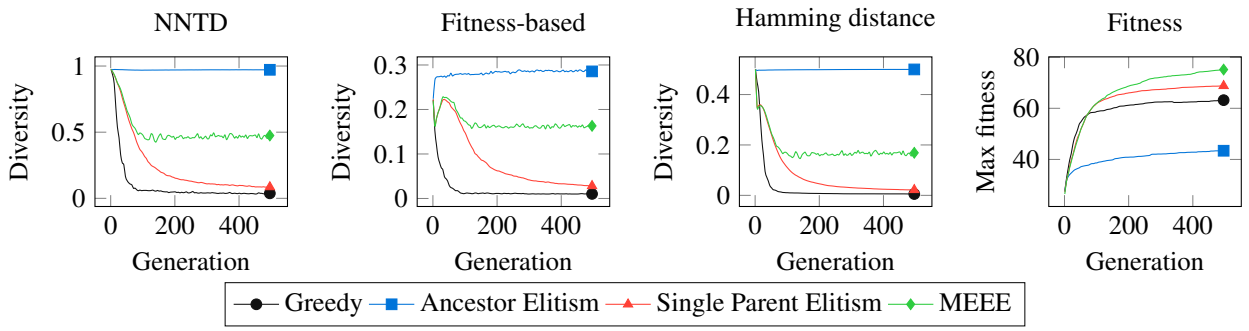


Figure 5: Average diversity over 100 runs for each replacement rule over 500 generations of the leaf data set.

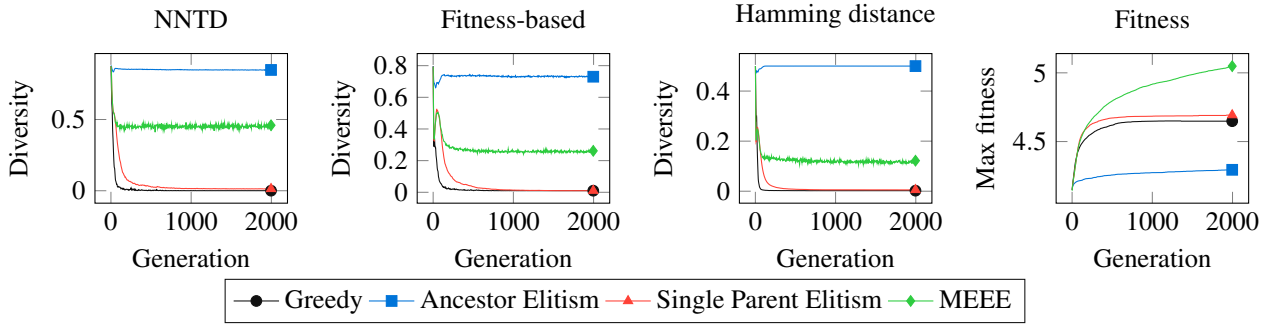


Figure 6: Average diversity over 100 runs for each replacement rule over 2000 generations for the XOR logical operator.

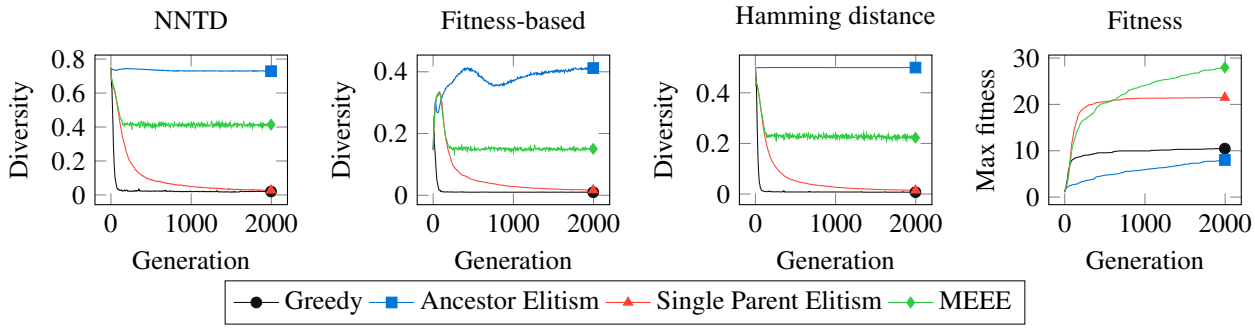


Figure 7: Average diversity over 100 runs for each replacement rule over 2000 generations for the Snake game.

correspondence between the phenotypic NNTD diversity measure and the genotypic Hamming distance diversity measure. If we take a closer look at the plots illustrating NNTD and Hamming distance, we can see that their slopes are similar.

When it comes to the fitness of the individuals, presented in the fourth plot of figures 5 through 7, it is clear to see that fitness is at its best when using the MEEE replacement rule. This rule does not have the most diverse population, but as we see, this is not necessarily a bad thing. This indicates that it is not the best solution to aim for a highly diverse population. One must find a balanced solution and take advantage of the fact that it is possible to switch between maintaining diversity and not doing so, which is in correspondance with the results presented in [4].

5. Conclusion

We have shown that fitness-based diversity measurements do not always catch the difference in traits, since two different individuals can have the same fitness, yet have different traits. This is also the case for genotypic diversity measures, where we have shown how two individuals of different genotypes can have the exact same traits.

We have performed and compared a number of experiments to test our own diversity measure, which we call Neural Network Trait Diversity (NNTD). For the diversity measures Hamming distance, fitness-based, and NNTD, we performed experiments using each of the four replacement rules: the Greedy, Ancestor Elitism, Single Parent Elitism, and Mass Extinction Explore Exploit (MEEE) replacement rule. These experiments indicate that fitness-based diversity measures are unpredictable.

When it comes to NNTD, which is a phenotypic measure, and Hamming distance, which is a genotypic measure, we observe something interesting. For all the dynamic experiments, we see that the slopes of each replacement rule seem similar. In real world applications, like those investigated in the dynamic experiments, there might be some connection between Hamming distance and NNTD.

From experiments performed in a static environment, we saw that even a slight mutation among chromosomes of identical individuals caused a major peak in NNTD, but only a small change in Hamming distance. We have shown that even a small change in an individual's genotype can cause a significant change in its traits. It is also important to be aware of the fact that the Hamming distance measure never exceeded a diversity of 0.5 during our experiments, possibly due to the fact that two random bits have the probability 0.5 of being identical. A Hamming distance above 0.5 is possible, but neither during static nor dynamic experiments, did this happen.

Based on our experiments and observations, we can conclude that it is reasonable to believe that we have succeeded in creating a diversity measure that better reflects different traits among individuals of a population.

One disadvantage of the species classification approach of NNTD is that it defines species based on the output neurons yielding the highest output value. So, neural networks with only a single output neuron will always belong to the same species.

The complexity of the fitness-based diversity measure is less than that of NNTD and Hamming distance, yet also yielding the most stochastic results. Comparing the complexity of NNTD and Hamming distance depends on the population size and how many random inputs one chooses to use for NNTD. If the number of random inputs is just a constant times the population size, the two diversity measures asymptotically have the same complexity.

6. Future work

The number and domain of random inputs used for NNTD will surely affect the reliability of the diversity returned. It is beyond the scope of this article, but indeed crucial to determine how the amount of random inputs can be chosen to produce a reliable result. Despite the amount of random inputs used, it is also interesting to investigate how each random input should be chosen. We chose to assign a random value v to an input neuron x based on the probability that x receives v in a real application. However, other methods can also be used for determining these random values. The impact of other methods can be investigated in the future.

We have shown how NNTD can be used to measure the diversity among neural networks for discrete problems. In the future, we would find it interesting to investigate how the concepts of NNTD can be applied to continuous problems as well. An idea is to modify eq. (3) such that each output neuron o_i is split into a set of new output neurons O_i, and a set of ranges Z_i, such that no ranges in Z_i overlap. The value of each output neuron O_i is then defined to be 1 if the value of o_i is in the range Z_i and 0 otherwise. Experiments must be performed to reason about how these ranges should be chosen, as well as the number of ranges used.

References

[1] K. Bache and M. Lichman. UCI machine learning repository, 2013.

[2] E. Burke, S. Gustafson, and G. Kendall. Diversity in genetic programming: an analysis of measures and correlation with fitness. *Evolutionary Computation, IEEE Transactions on*, 8(1):47–62, Feb 2004.

[3] H. G. Cobb. Genetic algorithms for tracking changing environments. In *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 523–530. Morgan Kaufmann, 1993.

[4] P. J. Darwen and X. Yao. Does extra genetic diversity maintain escalation in a co-evolutionary arms race. *International Journal of Knowledge-Based Intelligent Engineering Systems*, pages 191–200, 2000.

[5] K. A. De Jong. *An Analysis of the Behavior of a Class of Genetic Adaptive Systems.* PhD thesis, University of Michigan, Ann Arbor, MI, USA, 1975. AAI7609381.

[6] P. A. Diaz-Gomez and D. F. Hougen. Empirical study: Initial population diversity and genetic algorithm performance. In *Artificial Intelligence and Pattern Recognition*, pages 334–341, 2007.

[7] A. T. Freeman, S. L. Condon, and C. M. Ackerman. Cross linguistic name matching in english and arabic: A "one to many mapping" extension of the levenshtein edit distance algorithm. In *Proceedings of the Main Conference on Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics*, HLT-NAACL '06, pages 471–478, Stroudsburg, PA, USA, 2006. Association for Computational Linguistics.

[8] P. Koehn. Combining Genetic Algorithms and Neural Networks: The Encoding Problem. Master's thesis, The University of Tennessee, Knoxville, December 1994.

[9] S. Luke. *Essentials of Metaheuristics*. Lulu, second edition, 2013. Available for free at http://cs.gmu.edu/∼sean/book/metaheuristics/.

[10] T. H. Nguyen and X. H. Nguyen. A brief overview of population diversity measures in genetic programming. In T. L. Pham, H. K. Le, and X. H. Nguyen, editors, *Proceedings of the Third Asian-Pacific workshop on Genetic Programming*, pages 128–139, Military Technical Academy, Hanoi, Vietnam, 2006.

[11] E. K. Obeid, K. M. Capsersen, and M. B. Madsen. P6 project. `https://bitbucket.org/Obeyed/p6-project`, April 2014. Accessed: 2014-04-29.

[12] P. Silva, A. Marçal, and R. Silva. Evaluation of features for leaf discrimination. In M. Kamel and A. Campilho, editors, *Image Analysis and Recognition*, volume 7950 of *Lecture Notes in Computer Science*, pages 197–204. Springer Berlin Heidelberg, 2013.

[13] E. H. Simpson. Measurement of diversity. *Nature*, 1949.

[14] G. Syswerda. Uniform crossover in genetic algorithms. In *Proceedings of the 3rd International Conference on Genetic Algorithms*, pages 2–9, San Francisco, CA, USA, 1989. Morgan Kaufmann Publishers Inc.

[15] R. K. Ursem. Diversity-guided evolutionary algorithms. In *Parallel Problem Solving from Nature—PPSN VII*, pages 462–471. Springer, 2002.

[16] F. Vavak and T. Fogarty. Comparison of steady state and generational genetic algorithms for use in nonstationary environments. In *Proceedings of IEEE International Conference on Evolutionary Computation*, pages 192–195, May 1996.

[17] V. Venturi, K. Kedzierska, S. J. Turner, P. C. Doherty, and M. P. Davenport. Methods for comparing the diversity of samples of the t cell receptor repertoire. *Journal of Immunological Methods*, 321(1–2):182 – 195, 2007.

[18] D. Whitley. The GENITOR Algorithm and Selection Pressure: Why Rank-based Allocation of Reproductive Trials is Best. In *Proceedings of the Third International Conference on Genetic Algorithms*, pages 116–121, San Francisco, CA, USA, 1989. Morgan Kaufmann Publishers Inc.

[19] B. Widrow, D. E. Rumelhart, and M. A. Lehr. Neural networks: Applications in industry, business and science. *Commun. ACM*, 37(3):93–105, Mar. 1994.

[20] K. Zhu. A diversity-controlling adaptive genetic algorithm for the vehicle routing problem with time windows. In *Tools with Artificial Intelligence, 2003. Proceedings. 15th IEEE International Conference on*, pages 176–183, Nov 2003.

[21] E. Zitzler, K. Deb, and L. Thiele. Comparison of multiobjective evolutionary algorithms: Empirical results. *Evolutionary Computation*, 8:173–195, 2000.