Measuring Diversity in Steady State Genetic Algorithms using Artificial Neural Networks

Todo list

Abstract

We propose a method we call Neural Network Trait Diversity(NNTD), which we have developed for measuring phenotypic diversity among neural networks in genetic algorithms. It is an extension of the Simpsons Diversity Index known from ecology.

Avoiding low diversity in a genetic algorithm's population is crucial for finding a global optimum. When diversity is overlooked, premature convergence is the consequence. This most likely leads to local optimum. In order to analyse methods aimed to maintain a high diversity, a diversity measure is needed. The diversity measure must reliably reflect the particular type of diversity one aims to keep high. Many existing diversity measures exist. Two popular measures are based on the distance between the encoding of individuals and the distinct fitness values among individuals, known as the Hamming distance and unique fitness, respectively.

We argue why both of these measures have weaknesses, which we try to overcome using NNTD.

Our results show that... (Succesful or failure?)

This leads to... (What are the implications of our answer?)

1. Introduction

For many problems in computer science, a greedy strategy will not always lead to the best result. A greedy strategy is one where each step taken is the step that looks most promising at the moment. If a mountain climber wants to climb to the highest peak in The Himalayas, a greedy strategy is to move in any direction he think is the best, as long as he climbs up, and never backs down. Using this strategy, he is definitely able to climb up somewhere, but once he reaches a mountain top, he might discover an even greater mountain he could not see before. 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 mountain of all.

Genetic algorithms (GA) can be used to search for a global optimum in many types of problems, e.g. optimization problems. A genetic algorithm manages a number of individuals, which constitute a population. Each of these individuals represents a solution to a given problem. Individuals are conceptually just a chromosome consisting of different genes. In a GA, individuals are represented internally as bit strings. There are many ways in which the bit string can be interpreted to form a solution to the problem in question. Often, neural networks are chosen as individuals and are decoded from the chromosome. A neural network takes a number of inputs, does some internal calculations, and yields a number of outputs, which can be interpreted as a solution to the problem. How the bit string is interpreted determines how these calculations are carried out. For the particular problem of adding two integers, the neural network can be given the two integers s and t as input, and output a single value, which ideally should be $s+t$. The goal of GAs is to find an optimal neural network to solve the addition as quickly as possible.

Two individuals in a population may solve a problem completely differently. Individuals in a GA procreate to form offspring. The solution an offspring individual proposes will typically combine traits from both its parents. Hopefully, the combination of these traits will constitute a better solution to the particular problem in question. A fitness value is given to each individual to indicate how well it solves the problem.

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 available traits in the population will be explored, and a global optimum might never be found [16].

We believe it is essential that a diversity measure reflects the difference in traits among individuals. By concentrating on neural networks as individuals, we develop a new diversity measure, that as we will see, is not affected by pitfalls in traditional diversity measures.

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 for some input produce different outputs*". We now argue why we think that neither fitness-based nor genotypic diversity measures catch this trait diversity.

Consider two individuals who try to find the highest peak on an elevation map, where the fitness of each individual is based on 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 be equivalent, meaning they yield the same output on any input. 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 very diverse (Hamming distance of 6), but not trait diverse at all. We develop a new method for measuring phenotypic diversity in genetic algorithms using neural networks as individuals. We claim that our method better reflects different traits among the individuals. Furthermore, we use our method to explore how different replacement rules affect the diversity of a population.

Section 2 introduces the concepts used in our diversity measure, and can be skipped if the reader is knowledgeable about genetic algorithms and neural networks. Section 3 describes our diversity measure, which is evaluated and compared with other measures in section 4. In section 5, we evaluate our solution.

2. Preliminaries

Here we present the needed background knowledge to understand the work explained in the following sections. We begin with an introduction to artificial neural networks, followed by an introduction to genetic algorithms (GAs).

In GAs, a high diversity among the individuals is important. It is often argued that the weakness of GAs is the fall in diversity over generations, which can result in premature convergence [2, 6, 20]. After presenting neural networks and GAs, we present methods used to overcome a decreasing diversity. Methods like crowding and use of special selection procedures are presented. Next, we introduce different methods of measuring diversity.

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 purposes, e.g. face recognition, where the intensities of different pixels in an image are used as input, and a single output value y_1 is produced, where $y_1 = 1$ if the image was of a face and 0 if not. We now describe the structure and inner workings of neural networks to understand how these output values are calculated based on input values.

2.1.1. Neurons. Nodes in the graph of a neural network are called neurons. Three classes of neurons exist: input, hidden, and output. Input neurons receive $x_1, x_2, \ldots, x_n$ and forward the same value. Hidden and output neurons take a number of values as input from edges exiting other neurons, applies a weight to each value, sums them, then applies a function to produce a single output value. The function applied is called the transfer function, and is the same for all hidden and output neurons in the network. This is recursively expressed by

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

where x_i is the input to input neuron i, 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), y_j is the output of neuron j, and θ is the transfer function, typically defined to be the sigmoid function

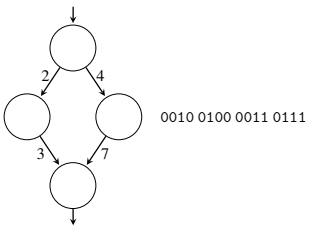
$$\theta(t) = \frac{1}{1+e^{-t}}$$

In a feedforward 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 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 feedforward neural network with a single hidden layer.

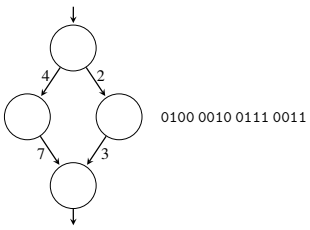
Write about bias neurons

2.1.2. Training a Neural Network. Any application of a neural network requires a suitable number of layers, neurons, and weights between neurons, to adequately solve a given problem. How many hidden neurons and layers to have is a highly debated subject, see [12].

The weights on edges connecting neurons are decided by a process called training. A well known training algorithm called backpropagation, requires that for each input to the neural network, the correct output is already known [8]. This kind of learning is called supervised learning. For the application of face recognition, this means that the pictures used for training, each has



(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. The binary representation assumes that each weight is represented by four bits.

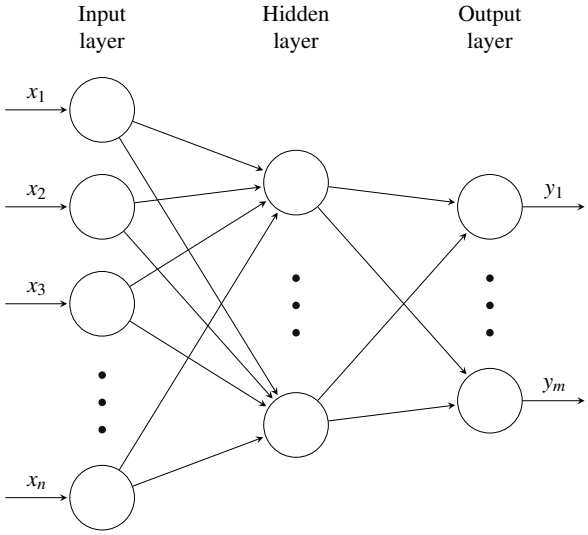


Figure 2: Structure of a neural network.

a predicate indicating whether or not it is a picture of a face.

Consider training an artificial intelligent (AI) player for a computer game. Given a state of the game, the desired action for the AI player to take might not be known beforehand, because any action may be either good or bad, depending on the actions that follow. Instead of backpropagation, a genetic algorithm (GA) can be used to construct a desired player. A GA requires that fitness can be measured for each individual. For the application of an AI player, the fitness value of a player indicates how well it performs according to some criteria. This can be measured by simulating the player in the game.

2.2. Genetic algorithms

Genetic algorithms are optimization algorithms which imitate the process of natural selection in search of global maxima.

2.2.1. Individuals and chromosomes. Genetic algorithms (GA) maintain a list of *individuals*, which together form a *population*. Each individual represents a possible solution to the optimization problem and has a fitness value, which denotes how adequately the individual can solve the optimization problem. An individual is encoded by its *chromosome*, which is typically represented by a bit string. Therefore, using a GA requires a way of decoding an individuals chromosome into a solution to the optimization problem.

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 some replacement rule. In contrast, a generational GA will choose a group of new individuals that will form a next generation, and then discard the old generation [15, 17, 18]. We will focus on steady state GAs, using the term *generation n* to denote the content of the population after $n-1$ iterations of producing offspring in a steady state manner.

2.2.2. Individuals. Individuals in GAs have a set of traits and behaviours which define each individual. They can take on any form of data structure, as long as they wholly represent a possible solution to the problem.

2.2.3. Neural networks as individuals. As we have discussed, neural networks are ideal for decision making and hence are appropriate as individuals in a GA. To support various GA operators, individuals are encoded as bit strings.

We represent each neuron and weight of any given neural network as a single bit string of a fixed length. Each weight is encoded with n bits. Each bit string is concatenated with the next, to form one large bit string containing the entire encoding for the network.

The bit string is constructed in an ordered manner, such that the first n bits represent the weight for the first connection between the first input neuron and the first hidden neuron, the next n bits represent the weight for the connection between the first input neuron to the second hidden neuron, and so forth. It is this large bit string that is manipulated by different GA operators. From now on, we will refer to the bit string encoding of a neural network as its *chromosome*.

If two chromosomes have different bit strings, we say that they have different genotypes. If the neural network they encode produces a different output for some input, we say they have different phenotypes.

2.2.4. Crossovers and mutations. In natural evolution, a pair of individuals come together to produce one or more new child individuals, with genes from both of the parent individuals. 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 of the bits. This ensures that the population can evolve if no progress would be made if the chromosomes did not allow it.

2.2.5. Fitness functions. A *fitness function* must be defined to calculate the desirability for each individual. This function is used to define the most fit individuals in a population. The higher the fitness level is of a given individual, the higher the chance it has to be chosen to reproduce with another individual. The intuition behind this is that choosing two fit individuals to crossover will create an even better individual with the best traits of each of its parents.

2.3. Crowding

Methods to overcome decreasing diversity, include inserting random immigrants (randomly initialized individuals) [3], using complex population structures to lower the gene flow, and the use of special selection procedures [16]. The latter is known as crowding [5]. How a particular crowding method works is commonly described by a replacement rule.

The replacement rule determines which individuals of the ith generation G_i, and their offspring O_i are selected to form the the next generation G_{i+1}.

2.3.1. Greedy. Common descriptions of GAs, that do not try to account for premature convergence, are often making the generation G_i from the n most fit individuals from $G_{i-1} \cup O_{i-1}$, where $|G_x| = n$ for all x. We will refer to this as the *Greedy replacement rule* [7].

2.3.2. Ancestor Elitism. Ancestor Elitism is similar to $(\mu + \lambda)$ and Generation Gap algorithms, but it was developed independently of these algorithms [9, p. 34, p. 50].

The next generation G_{i+1} is made from generation G_i as follows: every individual from G_i is put into G_{i+1}. Any individual in O_i who has only a single parent (made from mutation) and is more fit than its parent, will replace that parent in G_{i+1}. Any individual in O_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} is replaced by random immigrants.

2.3.3. Single Parent Elitism. Single Parent Elitism is similar to Restricted Tournament Selection, but was developed independently of this method [9, p. 132].

G_{i+1} is made from G_i as follows: every individual from G_i is put into G_{i+1}. Any individual in O_i who has only a single parent (made from mutation) and is more fit than its parent, will replace that parent in G_{i+1}. Any individual in O_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.

2.3.4. Mass Extinction Explore Exploit. Inspired by the changes between exploration and exploitation phases used by Ursem [16], we introduce the following replacement rule, which we will refer to as the *Mass Extinction Explore Exploit* (MEEE) replacement rule. The MEEE replacement rule performs a mass extinction when the diversity drops below a threshold. G_{i+1} is made from G_i as follows:

If $d(G_i) < \alpha$, replace the $\frac{n}{2}$ least fit individuals from G_i by random immigrants, where $n = |G_i|$. Use the *Single Parent Elitism* replacement rule on (G_i, O_i).

Where $d(G)$ denotes the diversity of a generation, and α is a threshold.

2.4. Diversity measures

It is often argued that the weakness of GAs is the fall in diversity over generations, which often results in premature convergence [2, 6, 20].

Here, we summarise key points regarding well-known diversity measures.

2.4.1. Genotypic diversity measures. The genotypic diversity of a set of individuals is determined by how different their genetic structures are. To measure this type of diversity, methods to compute the distance between

any two individuals' encoded bit strings are required.

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.

The *Hamming distance* between two bit strings A and B of equal length is the number of indexes i, such that $A[i] \neq B[i]$.

Another way to measure genotypic diversity, is using the *Levenshtein distance* between two bit strings A and B. It is then the number of bits that must be inserted, deleted or substituted to change A into B. For example, the Hamming distance and Levenshtein distances between between the following two strings

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

is 8 when using Hamming distance. The Levenshtein distance is 2, because transforming eq. (1) into eq. (2) is done by deleting the first bit and prepending a 0 [19], totalling 2 operations. Because computing the Levenshtein distance has a complexity of $O\left(n^2\right)$, where n is the length the encoded neural networks' bit strings, it is too computationally expensive for even the smallest populations.

As concluded by [4], constantly high genetic diversity does not guarantee better solutions, requiring other diversity measures to also be explored.

2.4.2. Phenotypic diversity measures. *Phenotypic diversity* is concerned with the individuals' behavioural differences, and can be calculated based on their fitness values. Such diversity measures include computing the standard deviation of fitness values, the average number of unique fitness values in a population, and entropy-based methods (see [2, 19]).

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

2.4.3. 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 [19].

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.

3.1. Algorithm

NNTD is based on Simpsons Diversity Index (SDI), which is a diversity measure used in ecology to quantifiy the biodiversity of a habitat [14]. We modify SDI to work with neural network. We will thus use the term individual and neural network interchangeably.

Let $F = \{f_1, f_2, \ldots, f_n\}$, where each f_q for $0 \leq q \leq n$ denote the set of neural networks contained in a population, which all have the same architecture of x input and y output neurons. NNTD is calculated with respect to a number of random inputs $R = \{r_1, r_2, \ldots, r_m\}$, where each r_i for $1 \leq i \leq m$ is an x-tuple of real values chosen randomly. For each sample r_i, a Simpson diversity index is calculated. The calculation depends on a set of species, so we distribute the neural networks into species based on which of their output neurons yields the highest value on input r_i.

Let $b_n b_{n-1} \ldots b_1$ be the binary representation of a number i, we define the species $S_i(r)$ to contain any neural network $f \in F$ that given $r \in R$ as input satisfies

$$\forall j,h \in \{1,2,\ldots,y\} \left(b_j \rightarrow (o_j \geq o_h)\right)$$

where o_l is the value of the lth output neuron of neural network f.

In a population where each individual has y output neurons, the total number of species is $2^y - 1$. As an example, assume that we have a neural network with six output neurons. If the third output neuron has the highest output, then that neural network is placed in species S_4, because the bits set to 1 correspond to `000100` (4 in binary). If another individual has multiple output neurons giving the highest output, say the first and third output neuron, then that neural network will be placed in species S_5 (`000101`, 5 in binary). If a third neural network has all of its output neurons give the same result, then that neural network is placed in S_{63}. Any species that is empty, does not count as a species in the diversity measurement.

Diversity is then defined by

$$D_i = 1 - \left(\frac{\sum_{j=1}^{k} |S_j(r_i)| \left(|S_j(r_i)| - 1\right)}{|F| \left(|F| - 1\right)} \right)$$

where $|S_j(r_i)|$ is the total number of individuals of the jth species, and k is the total number of non-empty

species, with respect to random input r_i. F denotes the total number of neural networks in the population.

The NNTD D (that is, the actual diversity) is then calculated as the average of all m D_i values by

$$D = \frac{\sum_{i=1}^{m} D_i}{m}$$

which has a complexity of $O(km)$, where k is the number of non-empty species and m is the amount random inputs ($|R|$).

The definition of species implies that a single network can actually be part of all species for each input. This happens if the value of all output neurons is the same. One disadvantage of NNTD is that it relies on random inputs, which means that fewer random inputs implies less statistical significance. Due to the nature of NNTD, it is not suitable for continuous problems. Consider for instance a neural network used for approximating any function σ, having only a single 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, all neural networks will belong to the same species, and hence we get a diversity of 0.

4. Experiments

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

In the first environment, which we will refer to as the static environment, the GA will not iterate. For three maximization problems we introduce in section 4.2, we create an initial population using a constrain c, and a significance α. We choose c such that we have an intuition about how the behavioural diversity of individuals will be dependent on α. By experiments, we will see if it is likely that NNTD catches this dependency.

In the second environment, which we will refer to as the dynamic environment, we will let the GA run for a number of iterations on each of the three problems. For each problem, we will run the GA using each of the four replacement rules introduced in section 2.3. During all of the tests, we will measure the diversity using each of the diversity measures described in **??**, as well as NNTD.

We hope to see that the diversities returned by NNTD better match the expected behavioural differences than any of the other diversity measures.

4.1. Parameter settings

We use a population size of 100. For each iteration, 100 new offspring individuals are created. Half of the offsprings are cloned from a random parent. These are found by using a rank-based selection, and then mutated. The other half is made by performing crossover between two random parents, also using rank-based selection. These settings are presented in table 1.

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

Table 1: GA parameters used throughout the experiments.

Three crossover methods are used: one-point crossover, two-point crossover, and uniform crossover. When a new individual is created 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].

4.2. Problems

We hereby explain the three discrete problems we perform our experiments on. In this section we use the terms neural network and individual interchangeably.

4.2.1. XOR. We use a neural network to approximate the XOR 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 then defined as the average number of bits correctly calculated. Fitness values will thus lie in the range [0–8], because the XOR between two random bits will be evenly distributed between 0 and 1. This means that a random guess of the solution to the 8-bit XOR problem, is expected to yield a fitness of 4.

The network has 16 input neurons, 16 hidden neurons, 8 output neurons, 2 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

verify that the maximum fitness value of 8 is achievable. The same random seed is always used for generating the 1000 problem instances.

4.2.2. Leaf classification. We use a neural network to classify the Leaf data set [1, 13]. The neural network is given 16 properties about an unknown leaf and has to decide which of 40 types of leaves it is. 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.

4.2.3. Snake. Snake is a game found on old Nokia cell phones, where you are moving 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

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

where f is the amount of collected food, and s is the total number of steps the snake is 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 a bit of information about the game state:

1. $\{-1, 0, 1\}$ whether food is to the left, vertically aligned, or to the right of the snake's head.
2. $\{-1, 0, 1\}$ whether food is above, horizontally aligned, or below the snake's head.
3. $\{0, 1\}$ if the snake dies, if its next move is up
4. $\{0, 1\}$ if the snake dies, if its next move is down
5. $\{0, 1\}$ if the snake dies, if its next move is right
6. $\{0, 1\}$ if the snake dies, if its next move is left

The neural network has 4 output neurons, one for each direction to choose. 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.

4.2.4. Criteria for selection. 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 [7]. A classification problem like Leaf 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. The Snake game differs in that it is an agent decision problem, where not just a single, but a sequence of decisions determines the outcome, where each decision changes the intermediate state.

4.3. Static experiments

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

4.3.1. Initial similarity. In this test, 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 behaviour of all individuals are the same as well. In this case, we will expect the lowest diversity possible. As initial similarity increases, more genotypes will be identical, and hence more individuals will behave the same. 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 will expect the most diverse behaviour among individuals to be found here.

4.3.2. Initial mutation. In this test, we introduce the variable *initial mutation*, which is 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, randomly chosen 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 an increase in the different of behaviour, as a result of the mutation of bit strings.

4.3.3. Results. Experiments run with initial similarity and initial mutation between 0 and 1 are shown in fig. 3 and fig. 4, respectively. The results of various initial similarity values shown in fig. 3 play along with our intuition. For each of the data sets, each diversity measure's diver-

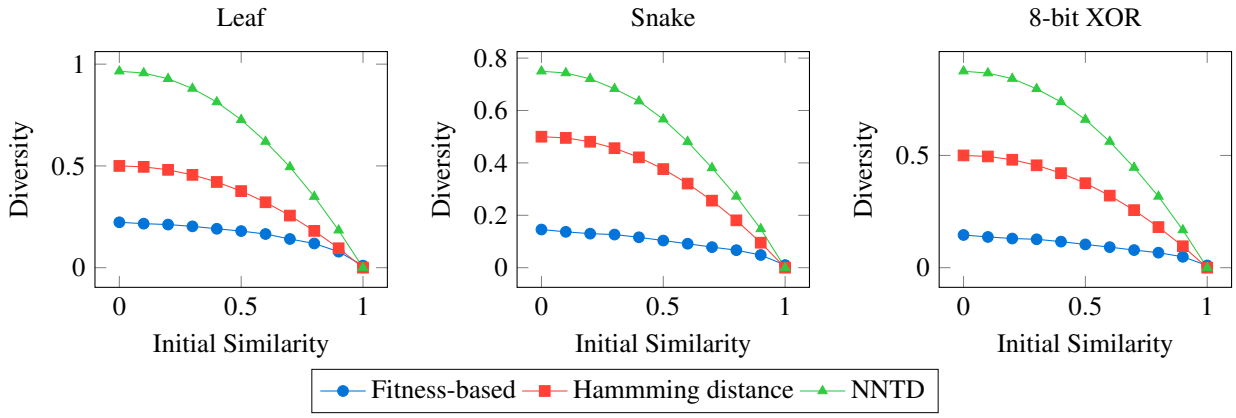


Figure 3: Average over 100 runs for each diversity measure on data sets over intervals of initial similarity. Each point represents the average of 100.

sity 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. 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. One could naively assume that you could just double Hamming distance's diversity, but it is actually possible for the Hamming distance measure to output 1 if each bit in two individuals is different.

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, it is obvious that changing only a few bits in the genotypes will only cause a small change in diversity. This is because Hamming distance measures diversity based on genotypes.

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 all random individuals get a fitness of 0. This is the reason why we chose a fitness function that also takes into account the number of steps a snake is alive. Despite yielding more diverse fitness values, it also increased the fitness values obtained after just 100 iterations notably. This does not mean that a fitness function cannot reflect behavioural differences, but it shows that how one defines the fitness function is crucial if one wishes to catch the behavioural differences of individuals.

4.4. Continuous Diversity

Our main goal with the experiments we perform is to see how diversity progresses throughout the many iterations a GA goes through. Furthermore, we must see if there are any significant differences between different diversity measures and replacement rules. To measure diversity, we use the fitness-based measurement, Hamming distance, and our own measure NNTD, after each iteration (we will use iteration and generation interchangeably). The four replacement rules are Greedy Replacement, Ancestor Elitism, Single Parent Elitism, and Explore-Exploit. We run the experiments four times, one with each of the replacement rules.

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 for each diversity measure. We then take the average diversity for each generation of the 100 runs. We made one exception, as we only ran 500 iterations on the Leaf dataset, because it is a smaller set consisting of only 340 instances.

4.4.1. Results. The results are presented through figures 5 to 7. The experiments performed on the Leaf data set is 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 subfigures. The first three of these subfigures illustrate each diversity measure, with the fourth subfigure illustrating the average fitness for the four replacement rules

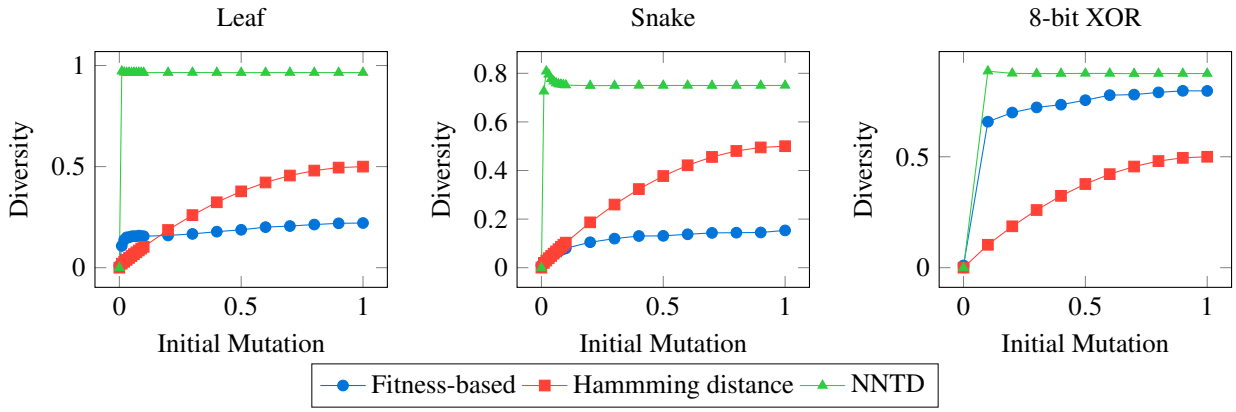


Figure 4: Diversity meaures for each interval of an initially mutated population. Each point represents the average of 100 runs.

throughout the iterations. We will not be discussing the fitness graph, it is there to illustrate how the populations improve with the different replacement rules.

Results of Leaf By comparing the first three graphs in fig. 5, we see that each replacement rule has somewhat the same curve fit for each diversity measure. Each diversity measures captures the intuition of the replacement rules, as described in section 2.3. An interesting observation 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 the replacement rules.

NNTD and Hamming distance measures look very similar for this data set. We can argue that NNTD better illustrates how diverse the traits of the individuals are, as with the Hamming distance, we do not see diversity ever rise over 0.5, even for the anecestor elitism replacement rule, which consistently introduces many random individuals.

Results of XOR The results of each diversity measure presented in fig. 6 are even more alike than in the Leaf data set above. It is interesting how NNTD and Hamming distance are a constant factor apart for each replacement rule. Also, for this data set, the fitness-based measurement has an overall higher diversity compared to the other two data sets.

Results of Snake The results shown in fig. 7, once again show a constant difference between the NNTD and Hamming distance measures. The fitness-based measure is again very irregular, this time for the ancestor elitism replacement rule. Here, the plot falls and rises before it stabilises. For this data set, the fitness-based measurements are closer to Hamming distance – another interesting observation.

The big picture If we compare experiments run on each data set illustrated in fig. 5 to fig. 7, we see that at least one diversity measure stands out from the other two in each of the experiments. For example, the fitness-based graphs have some irregular curves, which are not easily explained. Because of this, combined with the fact that multiple individuals can have the same fitness yet behave very differently, we argue that fitness-based diversity does not represent the actual diversity very accurately. This measure is simple and does not need any extra computations, but it is not suitable for every problem.

Another fact that is clear from the three experiments, is that NNTD and Hamming distance do not vary much between the different problems. Each replacement rule is always around the same interval for these two measures. Fitness-based diversity is much more dependent on the problem, as its results are not regular. Each graph illustrating the fitness-based measurement is different from the others.

The experiments explained above show that there might be a correspondence between the phenotypic (NNTD) diversity measure and the genotypic (Hamming distance) diversity measure. If we take a closer look at the graphs illustrating NNTD and Hamming distance, we can see that they are scaled with a constant compared to each other. The Hamming distance is always below the NNTD measurement.

When it comes to the fitness of the three experiments, it is clear to see that fitness is at its best when using the Explore-Exploit replacement rule. This rule does not have the most diverse population, but as we see, this is not necessarily a bad thing. This indicates

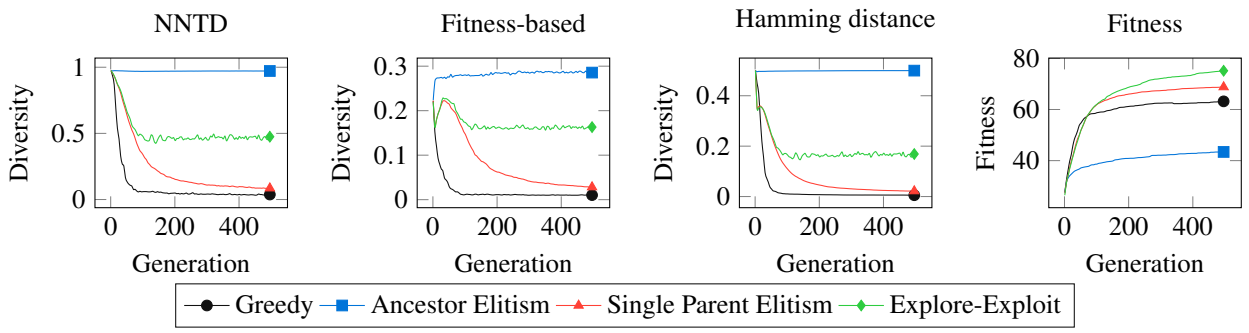


Figure 5: Average over 100 runs for each diversity measure on the leaf data set.

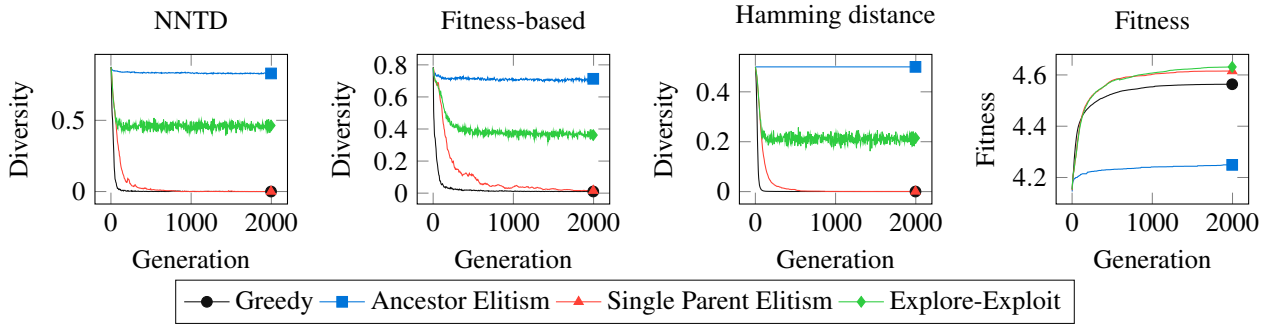


Figure 6: Average over 100 runs for each diversity measure on approximating an 8-bit XOR gate.

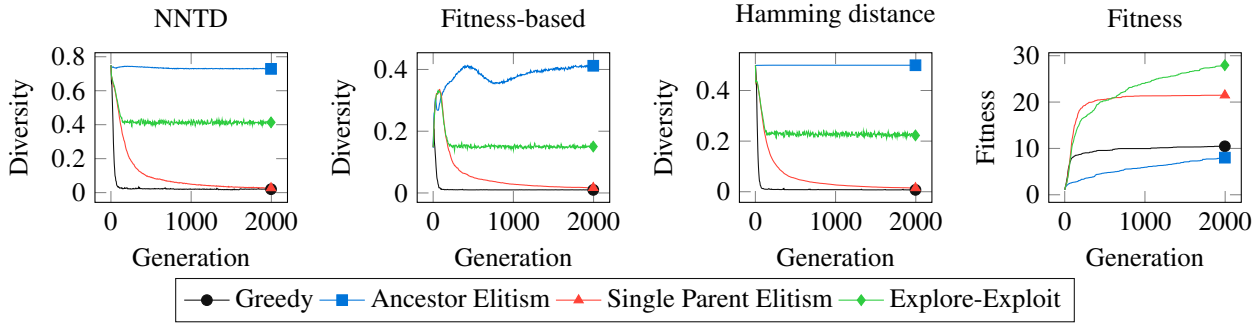


Figure 7: Average over 100 runs for each diversity measure on the snake data set.

that it is not the best solution to aim for a 100 % diverse population. One must find a balanced solution and take advantage of the fact that it is possible to switch between maintaining a high diversity and not doing so.

5. Conclusion

We have performed experiments to test our own diversity measure, which we call Neural Network Trait Diversity (NNTD). We experimented and compared the results of NNTD against two other widely used diversity measures, the Hamming distance and the fitness-based diversity measure. These experiments are discussed in section 4. We did the testing with four different replacement rules, the Greedy replacement rule, Ancestor Elitism, Single Parent Elitism, and Explore-Exploit. These replacement rules are explained in section 2.3.

We argue that fitness-based diversity measurements are much less useful for catching behavioural differences, since two different individuals can have the same fitness, and yet very different behaviour. Our experiments confirm that fitness-based diversity measures are unpredictable.

When it comes to NNTD, which is a phenotypic measure, and the Hamming distance, which is a genotypic measure, we see something interesting. It seems like these two measures follow each other in a manner. This only holds for tests performed in a dynamic environment though. In real world applications, like those investigated in the dynamic tests, there might be some constant factor between Hamming distance and NNTD.

From tests performed in a static environment, we saw that even a slight mutation among the chromosomes of identical individuals caused a major peak in NNTD, but only a small change in Hamming distance. We think it makes sense that even a small change in an individual's genotype can cause a significant change in its behaviour. Consider for instance a single bit that changes. This bit could be a sign bit that causes a significant change to the neural network's behaviour, or it could cause no change at all. The Hamming distance neglect this fact and focuses only on how the genotype is represented by a bit string.

It is also important to be aware of the fact that the Hamming distance measure never exceeded a diversity of 0.5 during our tests, which is obviously due to the fact that two random bits have the probability 0.5 of being identical. A hamming distance above 0.5 is indeed possible, but neither during static nor dynamic tests, 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.

Furthermore, it is clear to see that the replacement rule with the best outcome of fitness values, is the Explore-Exploit, which actively uses a diversity measure. Fore those type of replacement rules, it is important to have a diversity measure that can capture the concept of a population being diverse. We have found that NNTD seems to better catch different behaviour of the individuals.

6. Future work

Much research has already been done on determining how the tweaking of different parameters in a genetic algorithm (GA) affects the fitness values achieved, as well as undesirable convergence. Our diversity measure, Neural Network Trait Diversity (NNTD), is a new tool for investigating the impact on behavioural differences when tweaking the parameters of a GA. From our experiments, we saw that the Explore-Exploit replacement rule produced the most fit individuals for all test cases. Many other replacement rule exist that switch between exploration and exploitation phases based on changes in diversity. It is unknown whether these replacement rules will produce more fit individuals by using NNTD as a diversity measure.

NNTD is not limited to be used in GAs only. In fact, it can be used anywhere where one wants to measure the behavioural differences between two or more neural networks. The many possible applications of NNTD still have to be uncovered.

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

[8] Y. LeCun, L. Bottou, G. Orr, and K.-R. Müller. Efficient backprop. In G. Orr and K.-R. Müller, editors, *Neural Networks: Tricks of the Trade*, volume 1524 of *Lecture Notes in Computer Science*, pages 9–50. Springer Berlin Heidelberg, 1998.

[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] W. S. Sarle. Neural network FAQ. `ftp://ftp.sas.com/pub/neural/FAQ3.html#A_hu`, May 2012. Accessed: 2014-03-21.

[13] 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.

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

[15] 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.

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

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

[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] 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.

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