Measuring Diversity in Steady State Genetic Algorithms using Artificial Neural Networks

Todo list

Abstract

This paper presents SDINN, a phenotypic diversity measurement for neural network representations of genetic programs. 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.

Many studies have been conducted in this field of research, and we compare our method against other results, which aimed to solve the same problem.

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

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

1. Introduction

Many problems have non-linear search spaces with multiple local maxima. Genetic algorithms can be used to lead a search for global maxima. A genetic algorithm manages a number of individuals, which consitute a population. Each of these individuals is a possible optimal solution to the problem. Individuals possibly offering a better solution to this problem are formed by two individuals in the population combining their traits by breeding, or procreating. The intuition is that by combining two individuals with sought-after traits, their offspring will have a better solution to the problem. How well an individual solves a problem is expressed by a fitness value associated with each individual.

Ideally, offspring should contain the best traits from both parents, hence a diverse set of individuals is crucial to identify the combination of traits that work best. Consider a function for which the population must find an optimal solution. 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 [17].

We develop a new method for measuring phenotypic diversity in steady state genetic algorithms using neural networks as individuals. We claim that our method better reflects different traits among the individuals of a population. Furthermore, we use our method to explore how different replacement rules affect the diversity of a population.

2. Preliminaries

In the following sections we assume that the reader is familiar with neural networks and genetic algorithms (GAs). Otherwise, see appendix A for explanations of these.

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, 21].

We begin by presenting crowding methods, which use special selection procedures to overcome a decreasing diversity. Next, we introduce different methods of measuring diversity, and finish this section by explaining our own proposal for measuring diversity.

2.1. 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 [17]. 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}.

2.1.1. Greedy replacement rule. 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 [7]. We will refer to this as the *Greedy replacement rule*.

2.1.2. Ancestor Elitism replacement rule. Ancestor Elitism is similar to $(\mu + \lambda)$ and Generation Gap algorithms, but it was developed independently of these algorithms [10, 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.

2.1.3. Single Parent Elitism replacement rule. Single Parent Elitism is similar to Restricted Tournament Selection, but was developed indepentdently of this method [10, 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.

2.1.4. Mass Extinction Explore Exploit replacement rule. Inspired by the changes between exploration and exploitation phases used by Ursem [17], 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.

2.2. 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, 21].

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

2.2.1. 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 [20], totalling 2 operations. Because computing the Levenshtein distance requires $\mathrm{O}\left(n^2\right)$ time, 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.

2.2.2. 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, 20]).

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 [11].

2.2.3. 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 [20].

3. Simpsons Diversity Index NN

We believe it is essential that a diversity measure reflects the difference in traits among individuals. To the best of our knowledge, no current genotypic or phenotypic measures reflect this. Recall, that we are only con-

cerned with neural networks as individuals. 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*". By this definition, we denote the difference in traits among neural networks.

Fitness-based diversity measures do not catch this trait diversity. Consider two artificial intelligent (AI) players for the cell phone game "Snake". Their fitness can be calculated based on how much food they collect before they die. One AI player may have traits that makes it good at avoiding death by not hitting any walls or its own body. Sometimes, by chance, it hits a piece of food. The other AI player may have traits that makes it good at searching for food. The two AI players can have the same fitness value, because they collect the same amount of food before they die, yet still have completely different traits. Genotypic diversity measures do not catch the trait diversity either. This is because two individuals of different genotypes can yield the same output on any input. An example of this are the two neural networks 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 are not trait diverse.

In the following, we propose a method for measuring trait diversity, which we call *Simpsons Diversity Index NN* (SDINN). SDINN aims to reflect the diversity of different traits among individuals.

3.1. Algorithm

SDINN is based on Simpsons Diversity Index (SDI), which is a diversity measure used in ecology to quantifiy the biodiversity of a habitat [15]. 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_j for $0 \leq j \leq n$ denote the set of neural networks contained in a population, which all have the same architecture of a input and b output neurons. SDINN is calculated with respect to a number of random inputs $\{R_1, R_2, \ldots, R_m\}$, where each R_i for $1 \leq i \leq m$ is an a-tuple of real values chosen randomly. For each sample R_i, a diversity 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.

If $b_n b_{n-1} \ldots b_1$ is the binary representation of a number i, we define the species S_i to contain any neural network satisfying:

$$\forall o_j \forall o_k \in O((b_j = 1) \rightarrow (o_j \geq o_k))$$

$$\forall o_j \forall o_k \in O((b_j \wedge 1) \rightarrow (o_j \geq o_k))$$

where O is the set of all output neurons. In a population where each individual has b output neurons, the total number of species is $2^b - 1$. 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.

The diversity is calculated m times with SDINN as

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

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.

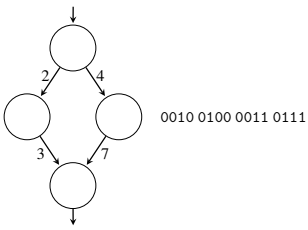
The SDINN D (that is, the actual diversity) is calculated as the average of all m D_i values

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

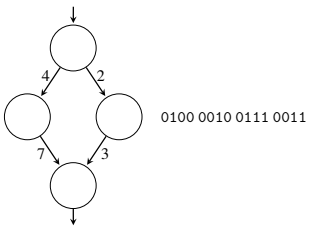
The definition of species implies that a single network can actually be part of all species for each input. This happens if the value of all output neurons is the same. One disadvantage of SDINN is that it relies on random inputs, which means that fewer random inputs implies less statistical significance.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

4. Experiments

We perform our experiments on two different discrete problems, chosen based on their different characteristics. The first problem is a classification problem over the Leaf dataset. The second problem is making an AI player for the game Snake, which was found on many of the first cell phones.



(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.

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

Table 1: GA parameters used throughout experimenting.

4.1. Parameter settings

We use a population size of 100. For each iteration, 100 new offspring individuals are created. Half of the offspring is cloned from a random parent, found using rank-based selection, and then mutated. The other half is made by performing crossover between two random parents, also using rank-based selection. 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 individual 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, see [12].

4.2. Problems

The discrete problem is the small leaf data set [1, 14]. Fitness is evaluated based on how many instances out of the entire data set the neural network correctly classifies. It consists of 16 inputs and 1 output.

The continuous problem used is the Snake game known from Nokia cell phones. This game is very similar to the artificial ant problem [8, p. 147–155]. A snake has to traverse a square 10×10 grid and search for a randomly positioned piece of food using a sensing function that's able to see the squares and their contents directly next to its head and above and below its head. Once the snake traverses into the square with food, the food disappears and the snake's length grows by one square. A new piece of food then spawns at a new random position. The snake has four actions to move around the grid, each of which takes one time unit: move up, down, left, and right. The game ends if the amount of time units exeeds 1000, or if the snake collides into a wall. Its fitness function is based upon the amount of food eaten.

4.3. Diversity measurements

To test our SDINN method, we introduce two variables, both of which only have an effect during the very first generation of a population: initial similarity and initial mutation of a population.

Initial similarity describes how equal the individuals initially are. It can take on a range between 0 and 1, where 0 means that all individuals are random, and 1 denotes that 100 % of all individuals have the exact same genotype. With a value of 0.5, one individual is cloned so that half of the population has the same genetic makeup and the other half is initialized to be completely random. A population with the value of 1 for initial similarity should have the lowest diversity, and a completely randomly initialize population (initial similarity of 0) should have the highest diversity, since every individual is randomly chosen.

Initial mutation requires an initial similarity set to 1, e.g. the initial population consists of a single, cloned individual. This is due to the fact that mutating individuals already randomly initialized will not yield a different diversity of individuals. The initial mutation rate simply signifies the mutation percentage of all bits in every individual of the population. A population consisting of the same cloned individual with an initial mutation rate of 1 should have the highest diversity, since every bit in every, initially similar individual, will have a 100 % chance to be selected for mutation, resulting in a randomly initialized population.

Experiments on SDINN using both of these two variables are shown in fig. 2. Each point represents the average diversity measure of 100 runs at a variable intervals of 0.05. As we can see by the chart, diversity increases as expected for both variables. Diversity is maximum at a value of 7.2 for a completely random population. To reach this maximum, it requires an initial mutation rate of a mere 2 % and an initial similarity of 0 %.

Figure 2: Diversity measurements with SDINN, given increased ranges of initial similarity and initial mutation rates. Each point is the average diversity over 100 runs.

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] J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA, 1992.

[9] 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.

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

[11] 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.

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

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

[14] 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.

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

[16] 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.

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

[18] 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.

[19] 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.

[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.

A. Artificial Neural Networks and Genetic Algorithms

A.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.

A.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, usually 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 3 illustrates the generic graph structure of a feedforward neural network with a single hidden layer.

A.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 [13].

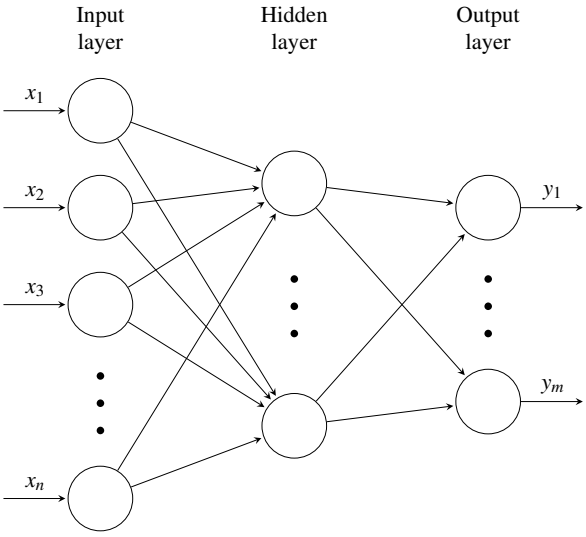


Figure 3: Structure of a neural network.

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 [9]. 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 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.

A.2. Genetic algorithms

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

A.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 [16, 18, 19]. 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.

A.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.

A.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.

A.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.

A.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.