A Runtime Analysis of Bias-invariant Neuroevolution and Dynamic Fitness Evaluation

Paul Fischer
DTU Compute
Technical University of Denmark
Kongens Lyngby, Denmark

John Alasdair Warwicker
Institute of Operations Research
Karlsruhe Institute of Technology
Karlsruhe, Germany

Carsten Witt
DTU Compute
Technical University of Denmark
Kongens Lyngby, Denmark

ABSTRACT

In the field of neuroevolution (NE), evolutionary algorithms are used to update the weights, biases and topologies of artificial neural networks (ANNs). A recent theoretical work presented the first runtime analysis of NE in a simple setting, considering a single neuron and intuitive benchmark function classes. However, this work was limited by the unrealistic settings with regard to activation functions and fitness measurements.

In this paper, we extend upon this first work by overcoming the two shortcomings. Firstly, we consider a more realistic setting in which the NE also evolves a third parameter, termed the bend, allowing the previous benchmark function classes to be solved efficiently even in the fixed bias case. This setting mimics rectified linear unit activation functions, which are common in real-world applications of ANNs. Secondly, we consider a dynamic fitness function evaluation paradigm where the weights and biases are updated after each new sample. Experimental results in both cases support the presented theoretical results.

CCS CONCEPTS

• **Theory of Computation** → Theory of randomized search heuristics.

KEYWORDS

Neuroevolution, Theory, Runtime analysis, Dynamic fitness

ACM Reference Format:
Paul Fischer, John Alasdair Warwicker, and Carsten Witt. 2024. A Runtime Analysis of Bias-invariant Neuroevolution and Dynamic Fitness Evaluation. In *Genetic and Evolutionary Computation Conference (GECCO '24), July 14–18, 2024, Melbourne, VIC, Australia.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3638529.3654044

1 INTRODUCTION

Neuroevolution (NE) refers to the application of evolutionary computation techniques to artificial neural networks (ANNs), and is a common approach when backpropagation or network weight adjustments are inefficient or unavailable. Further, NE techniques can also adapt network topologies, avoiding a time-consuming parameter-setting process. Recent advances in deep neural networks have brought NE back to the forefront of research; see recent surveys [10, 13, 16].

The field of evolutionary algorithms (EAs), a subset of evolutionary computation, considers (typically heuristic) optimisation techniques inspired by evolutionary paradigms that are applied across a variety of problems. The theoretical analysis of EAs has grown from analysing simple algorithms in basic settings (e.g., [7]) to considering more complex realistic paradigms, such as the widely studied NSGA-II algorithm [6, 17]. In fact, the theoretical analysis of EAs has allowed the design of algorithms that showcase improvements over the state-of-the-art [3].

Fischer et al. [8] recently presented the first theoretical analysis of NE. They presented an optimisation setting in which the evolution of the weights and bias terms in the ANN correspond to seeking a hyperplane within a unit sphere to classify its edge points as either positive or negative. Further, they considered simple ANN topologies, where even single neurons with a binary activation function can be efficient on the benchmark instances they presented. A two-layer topology further allowed the analysis of multiple neurons. [8] also showed that the harmonic mutation operator introduced by [4] led to exponentially smaller runtime bounds compared to local mutation operators.

In this work, we extend upon the introductory analysis of NE by [8] by analysing more realistic ANN settings. As a first contribution, we consider an updated solution representation, which mimics real-world ANN paradigms using rectified linear unit (ReLU) activation functions. ReLU functions are common in real-world ANN studies, and lead to ANN frameworks that can be represented by piecewise linear functions [1]. This allows the representation of ReLU-based ANNs as mixed-integer linear programs, meaning robustness can be analysed and adversarial examples can be quickly generated [9, 15]. This new solution representation allows the construction of *bended* hyperplanes, meaning globally optimal solutions can be found on the presented benchmark functions that are invariant to the setting of the bias terms. This also allows the identification of a Pareto front of optimal solutions; however, for ease of analysis, we remain in the single objective case.

As a second contribution, we analyse a setting where the fitness of the solution is calculated in a more realistic way. Whereas previous research calculated the fitness as a fraction of two volumes corresponding to two infinite sets, we consider in this work an online setting which mimics the weights and biases of an ANN being re-trained after receiving new inputs.

The rest of the paper is structured as follows. In Section 2, we introduce the considered ANN topology, as well as the proposed advancements and the benchmark functions. In Section 3, we present

GECCO '24, July 14–18, 2024, Melbourne, VIC, Australia

ACM ISBN 979-8-4007-0494-9/24/07.
https://doi.org/10.1145/3638529.3654044

runtime results for the new NE algorithm, while Section 4 provides the theoretical analyses for the online fitness setting. We present experimental results in Section 5 and we conclude with a discussion and ideas for future analyses. Additional experimental data and figures can be found in the supplementary material.

2 PRELIMINARIES

We define $[N]$ as the set $\{1, \dots, N\}$.

2.1 ANN Topology

Fischer et al. [8] presented the first runtime analysis of NE approaches. They presented the (1+1) Neuroevolution algorithm ((1+1) NA) and a number of variants.

They considered so-called perceptron neurons, with input weights $w_1, \dots, w_D$ and a threshold parameter t. For a given input $x = (x_1, \dots, x_D) \in \mathbb{R}^D$, the neuron outputs 1 if $\sum_{i=1}^{D} w_i x_i \geq t$, and 0 otherwise. Geometrically, this means that the given point x would lie above the hyperplane with normal vector $(w_1, \dots, w_D)$ and bias t. This framework lends itself well to the solution of binary classification problems, where for a given input, the ANN must decide whether it belongs to a given class (output 1) or not (output 0). The typical search space for the classification problems considered by [8] were point sets in the unit hypersphere.

This model extends to ANN topologies with two layers. The idea behind the (1+1) NA is to optimise the weights and biases of a neural network with one hidden layer, where each neuron in the hidden layer is considered as a perceptron-based neuron with binary (threshold) activation, while the final layer is the output of the OR function.

Typically, $D = 2$ was used. In their setting, [8] consider the representation (w_1, w_2, t) as a tuple (φ,b), where φ corresponds to the angle of the unit normal vector for the given hyperplane, and b is the bias. Geometrically, this topology outputs the union of a number of D-dimensional hyperplanes.

In this work, we consider a new topology which we describe for the case of two dimensions. In this model with three hidden layers, each of the presented neurons uses a rectified linear unit (ReLU) activation function (i.e., they output $\max\{0, s\}$ where $s = \sum_{i=1}^{k} w_i x_i$ for k inputs from the previous layer). We fix the weights between layers 1 and 2, and between 2 and 3. Let $\text{out}(\cdot)$ denote the output from the neurons in the first layer. Then, the final output is calculated as $z := \max\{0, \min\{\text{out}(n_{11}), \text{out}(n_{12})\}\}$. This new topology in shown in Figure 1. The use of ReLU activation functions within NNs incorporates non-linearity, while producing outputs that are known to be piecewise linear [1]. Hence, for this topology, we consider the decision boundary no longer as a straight line, but rather V-shaped. For simplicity, we consider the outputs from such topologies as a singular neuron (possibly within a standard ANN topology), which we term 'V-neurons'.

We describe each of the V-neurons by three parameters, namely the angle of the unit normal vector φ, the bias b, and a *bending* parameter $\theta \in [0, \pi]$ (hereafter referred to as the *bend*, for brevity). For a given two-dimensional V-neuron with normal vector φ and bias b, let H denote the decision boundary (i.e., a hyperplane). Let P be the point on H that is closest to the origin, and emanate the normal vector from this point. Then, the area that is positively

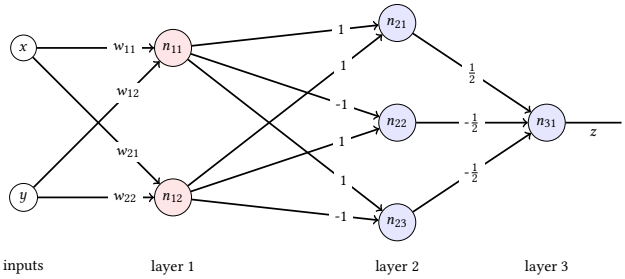

Figure 1: A network with ReLUs which computes a V-shaped area of positive classifications with angle $\theta \in [0, \pi/2]$. The red neurons compute hyperplanes, the part with the blue ones computes the minimum of the outputs of the red ones. That is, the final output is positive iff both red neurons compute something positive (the "continuous version" of AND). For angles $\theta \in [\pi/2, \pi]$, the blue part has to compute the maximum (i.e., OR) which is achieved by replacing the two weights $-1/2$ by $+1/2$.

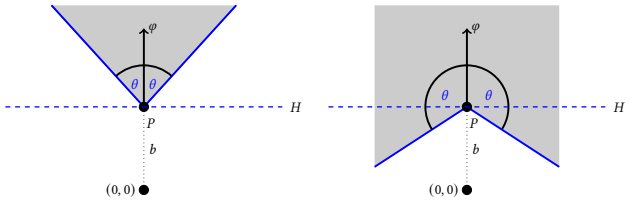

Figure 2: The grey area shown corresponds to the (bottom part of) the "wedge" where points are classified 1; (left) for $\theta \leq \pi/2$, (right) for $\theta > \pi/2$.

classified consists of all points at an angle at most θ to the right or left of the normal vector φ.

Formally, for a given point X, let $\beta \in [0, \pi]$ be the angle between φ and the vector $(X - P)$. Then, X is positively classified if $\beta \leq \theta$. Figure 2 gives two examples for the case $\theta \leq \pi/2$ and $\theta > \pi/2$. As a generalisation to dimensions greater than two, the area of positive classification would be a multi-dimensional cone.

2.2 Algorithms

As with [8] and other related works, we consider only the adaptation of the parameters of the network (and not its topology, as with classical neuroevolution frameworks, e.g., NEAT [14]). Hence, for each of the parameters (namely φ_i, b_i and θ_i for $i \in [N]$), we continue the use of the state space $\{0, \dots, r\}^N$, where N is proportional to the number of neurons and r is the resolution of the discretisation of the domain (see e.g., [4]). Since N and D are assumed constant in this paper, our theoretical runtime results will only depend on r. Clearly, the bigger r is, the more accurately an optimal solution can be represented, but this usually also requires a larger runtime.

Regarding search operators, we seek a mutation operator that changes the given parameter by some term chosen from a given distribution (i.e., by some term $\pm\ell/r$). [8] showed that the harmonic mutation operator (see e.g., [11]) leads to efficient algorithms, and

Algorithm 1 (1+1) NA [8]

$t \leftarrow 0$; select x_0 uniformly at random from $\{0, \dots, r\}^{2N}$
while termination criteria not satisfied **do**
  Let $y = (\varphi_1, b_1, \dots, \varphi_N, b_N) \leftarrow x_t$;
  For all $i \in [N]$, mutate φ_i and b_i with probability $1/(2N)$, independently of each other and other indices;
  Mutation chooses $\sigma \in \{-1, 1\}$ u.a.r. and $\ell \sim \text{Harm}(r)$ and adds $\sigma\ell$ to the selected component; the result is taken modulo r for angle and modulo $r+1$ for bias;
  For $i \in [N]$, set polar angle $2\pi\varphi_i/r$ and bias $2b_i/r - 1$ for neuron i to evaluate $f(y)$;
  if $f(y) \geq f(x_t)$ **then** $x_{t+1} \leftarrow y$;
  else $x_{t+1} = x_t$;
  $t \leftarrow t + 1$;

Algorithm 2 Bias-Invariant (1+1) NA (BINA)

$t \leftarrow 0$; select x_0 uniformly at random from $\{0, \dots, r\}^{3N}$
while termination criteria not satisfied **do**
  Let $y = (\varphi_1, b_1, \theta_1 \dots, \varphi_N, b_N, \theta_N) \leftarrow x_t$;
  For all $i \in [N]$, mutate φ_i, b_i and θ_i with probability $1/(3N)$, independently of each other and other indices;
  Mutation chooses $\sigma \in \{-1, 1\}$ u.a.r. and $\ell \sim \text{Harm}(r)$ and adds $\sigma\ell$ to the selected component; the result is taken modulo r for angle and modulo $r+1$ for bias and bend;
  For $i \in [N]$, set polar angle $2\pi\varphi_i/r$, bias $2b_i/r - 1$ and bend $\pi\theta_i/r$ for neuron i to evaluate $f(y)$;
  if $f(y) \geq f(x_t)$ **then** $x_{t+1} \leftarrow y$;
  else $x_{t+1} = x_t$;
  $t \leftarrow t + 1$;

hence we continue with this choice. We write $X \sim \text{Harm}(r)$ to denote that the random variable X follows the harmonic distribution with parameter r, defined by $\text{Prob}[X = j] = 1/(jH_r)$ for $j \in [r]$. Here H_r denotes the rth harmonic number.

We firstly present the (1+1) Neuroevolution Algorithm ((1+1) NA) in Algorithm 1.

Alongside harmonic mutation, [8] also presented the following variants:

- The local (1+1) NA fixes $\ell = 1$ (known as unit mutation [4]).
- The (1+1) NA without bias fixes $b_i = r/2$ for $i \in [N]$.

Regarding the local (1+1) NA, [8] showed that theoretically and empirically it is less efficient than the (1+1) NA with harmonic mutation at solving the presented benchmark functions. Therefore, we typically only consider the harmonic mutation operator in this work. Further, for the BINA approach we consider, we do not believe that local mutations are likely to be effective, due to the presence of larger areas of local optima associated with the extra parameter. A general analysis of advantageous situations for the local (1+1) NA would be of interest; we leave this for future work.

We present the algorithm for the new paradigm using V-neurons, which we term the *bias-invariant neuroevolution algorithm* (BINA), in Algorithm 2. We note that for $\theta_i = r/2$ (for $i \in [N]$), the bend becomes $\pi/2$ and BINA reduces to the (1+1) NA.

2.3 Benchmark Functions

Previous work [8] also considered a number of benchmark problems, designed to mimic possible structures that can appear in realistic settings. Analysing algorithms on these benchmark functions will provide insights into their real-world performance, as well as laying the foundations for building up a series of analysis tools.

The feasible region of the benchmark functions are points on the edge of the unit hypersphere. The problems are defined for an arbitrary number of dimensions D, but $D = 2$ is typically considered in the analysis:

(1) Half $:= \{x \in \mathbb{R}^D \mid ||x||_2 = 1 \text{ and } x_D \geq 0\}$.
(2) Quarter $:= \{x \in \mathbb{R}^D \mid ||x||_2 = 1 \text{ and } (x_{D-1}, x_D) \geq (0, 0)\}$.
(3) TwoQuarters $:= \{x \in \mathbb{R}^D \mid ||x||_2 = 1 \text{ and } x_{D-1}x_D \geq 0\}$.

For the problem Half, the optimal setting (when $D = 2$) consists of a single neuron with angle and bias $(\pi/2, 0)$ as presented in Figure 3. However, we note that BINA can identify an optimal solution for any value of the bias by optimising for $\varphi = \pi/2$ and setting θ as necessary for the given bias.

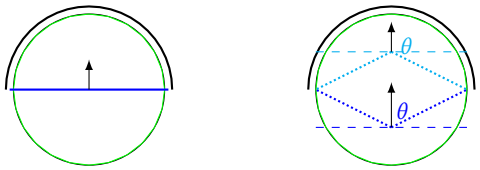

Figure 3: Illustration of Half where the black arc represents the target area, and the thin green arcs on the surface represent correctly classified points. The left side shows the optimal solution of the (1+1) NA. The right side shows two examples of optimal solutions for BINA: (1) The upper dotted line shows $\varphi = \pi/2$, $B \approx 0.5$, $\theta \approx 2\pi/3$; (2) The lower dotted line shows $\varphi = \pi/2$, $B \approx -0.5$, $\theta \approx \pi/3$.

As a general function class, we also consider

$$\text{Fraction}_c := \{x \in \mathbb{R}^D \mid ||x||_2 = 1 \text{ and } \psi_{D-1} \in [0, c\pi];\ c \in \mathbb{R}^+\},$$

where ψ_{D-1} is the polar spherical angle between x and the unit hypersphere in the first $D-1$ dimensions. The Fraction_c function class comprises an arc of constant length which must be positively classified; setting $c = 1/2$ results in Half, while setting $c = 1/4$ results in Quarter. We consider any positive constant $0 < c \leq 1/2$. In the no-bias setting for BINA, the optimal solution has $\varphi = c\pi$ and $\theta = c\pi$, while the (1+1) NA can only find the global optimal solution (in the no-bias setting) when $c = 1/2$.

Alongside these function classes, we note that BINA lends itself well to population-based algorithms and for multi-objective optimisation, where a Pareto front of non-dominated solutions is sought. We leave this analysis for future work, and note that this can be translated to the single objective case by imposing some limit on θ, i.e., preferring linear hyperplanes. This would lead to the same optimal solutions as the (1+1) NA in the case of instances of Fraction.

2.4 Discussion on Fitness Function

[8] presented the following fitness function to calculate the proportion of correctly classified points in the unit hypersphere. For a binary classification with labels in $\{0, 1\}$, we define:

- $S_D := \{x \in \mathbb{R}^D \mid ||x||_2 = 1\}$ as inputs to the ANN;
- C_D as the union of half-spaces above (or on) the hyperplanes spanned by the N neurons (either perceptron or V-neurons);
- $L_D \subset S_D$ as the set of points classified as 1;
- $\overline{A} = \mathbb{R}^D \setminus A$ for a given set A;
- $\mathrm{vol}(\cdot)$ as the (hyper)volume.

Then, the fitness (*true fitness)* of a solution x is given by

$$f(x) = \frac{\mathrm{vol}(((C_D \cap L_D) \cup (\overline{C_D} \cap \overline{L_D})) \cap S_D)}{\mathrm{vol}(S_D)}.$$

The sets L_D and $\overline{L_D}$ assume that all possible classifications are known; that is, we assume that we have access to the correct classification for any possible input. We refer to this as the *standard* setting for fitness evaluations throughout.

However, from the perspective of applied machine learning, this setting is idealised since this corresponds to training sets of infinitely large size in a batch-learning setting.

Hence, in this work, we also consider a fitness function where the fitness is computed online, based on a finite-size sample from the (possibly) infinite training set. For simplicity, we assume that in each step of the neuroevolutionary algorithm (NA), one point from the training set is sampled uniformly at random and the fitness function is updated accordingly. This models more closely a practical setting where a neural network is re-trained after receiving exactly one input from the training set. In future work, we aim also to consider a system where sets of points (i.e., more than one) are revealed in between steps of the NA.

Formally, we consider the fitness as a time-dependent function f_t, and again consider the unit hypersphere S_D. We have that L_D is the set of points to be classified positively, and define $C_D^{(t)}$ as the union of half-spaces above or on the hyperplanes spanned by the N neurons at time t of the algorithm.

At each time $t \geq 0$, immediately before the algorithm evaluates the mutated search point y in relation to the current search point x_t, a point $s_t \in S_D$ is sampled uniformly at random. Let $\Sigma_t := s_0 \cup \cdots \cup s_t$. Then, f_t is defined as follows:

$$f_t = \frac{|\Sigma_t \cap L_D \cap C_D^{(t)}| + |\Sigma_t \cap \overline{L}_D \cap \overline{C}_D^{(t)}|}{t+1}.$$

We refer to this throughout the paper as the *online* version of the underlying problem (e.g., Half, Quarter, etc.).

In contrast to the previous fitness evaluation, a perfect fitness of 1 can be achieved for usually more than one setting of the hyperplanes(s) maintained by the NA. Note also that f_t can change (both increase and decrease) even if the current search point of the algorithm does not change in an iteration. Therefore, we assume that the current search points x_t of the algorithm are re-evaluated in each fitness comparison of the algorithm before a comparison with the offspring y is made.

3 ANALYSIS OF THE BIAS-INVARIANT NA

In this section, we present runtime analyses of the bias-invariant (1+1) NA (BINA) across the benchmark functions defined in Section 2.3 in the standard setting. As with previous work, we consider only the case $D = 2$, and the use of harmonic mutation. Throughout this section, we utilise the following Lemma, which gives the probability of reaching a sufficiently large area of the search space.

Lemma 3.1 ([8]). *Let X denote the random outcome of the harmonic mutation operator with parameter r, and let $a < b$ be two positive integers. Then $P(a < X \leq b) \geq (\ln(b/a) - 1/a)/H_r$.*

In particular, when $b - a = \Omega(r)$, there is a probability $\Omega(1/\log r)$ of hitting the interval $(a, b]$; that is, an expected time of $O(\log r)$.

3.1 No Bias

We first consider the analysis of BINA in the setting where there is no bias (i.e., it is fixed at 0 and not mutated). In this case, there is only one globally optimal solution that BINA can find. Clearly, if we also fix the bending parameter $\theta = \pi/2$ then BINA reduces to the (1+1) NA without bias and the results from [8] hold.

We present results for BINA on the generalised function class $\textsc{Fraction}_c$. The results presented in this section will hold for Half and Quarter in particular.

The following Lemma gives an insight into the quality of solutions found by BINA in the no-bias case.

Lemma 3.2. *Let $x_t = (\varphi_t, \theta_t)$ be the current search point of BINA with $N = 1$ and without bias on $\textsc{Fraction}_c$ (for some $c > 0$), and assume $f(x_t) > 0$ and $\pi\theta_t/r \leq \pi/2$. Let $d_\varphi^{(t)} = |2\pi\varphi_t/r - c\pi|$ be the (absolute) difference of the angle from its optimal value, and let $d_\theta^{(t)} = |\pi\theta_t/r - c\pi|$ be the (absolute) difference of the bend from its optimal value. Then, it holds for the current fitness that*

$$f(x_t) = \frac{1}{2\pi}\left(2\pi - 2\max\{d_\varphi^{(t)}, d_\theta^{(t)}\}\right).$$

Proof. We are assuming that $f(x) > 0$; that is, at least some of the points are being (correctly) positively classified. We exploit the following.

In the two-dimensional unit hypersphere (in the no-bias case), a displacement of the angle φ_t by δ will move the decision boundary a distance of δ around the circumference of the circle on both sides (in the same direction). Hence, for $d_\theta^{(t)} = 0$, an area of size $d_\varphi^{(t)}$ will be misclassified as positive, and an area of size $d_\varphi^{(t)}$ will be misclassified as negative. Somewhat similarly, a displacement of the bend θ_t by δ will move the decision boundary a distance of δ around the circumference of the circle on both sides (in opposite directions). That is, for $d_\varphi^{(t)} = 0$, an area of size $2d_\theta^{(t)}$ will be either positively or negatively classified, depending on the sign (of $\pi\theta_t/r - c\pi$). When both values are non-zero, the areas of misclassified points can be calculated by the interplay of the two terms. We use this to calculate the length of the misclassified area in the remainder of the proof.

We distinguish between two cases:

(1) $d_\varphi^{(t)} \leq d_\theta^{(t)}$;
(2) $d_\varphi^{(t)} > d_\theta^{(t)}$.

In the first case, the bend is more *displaced* than the angle. We consider again two subcases:

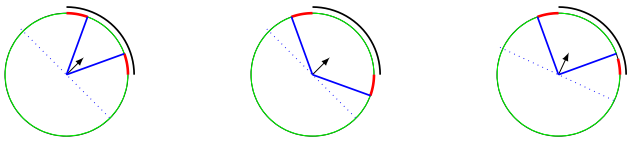

Figure 4: Illustration of the subcases described by Lemma 3.2 for Quarter (i.e., Fraction$_{1/4}$), where the black arc represents the target area. The thin green arcs on the surface represented correctly classified points and the thick red arcs on the surface represent incorrectly classified points. (a) $d_\varphi^{(t)} \le d_\theta^{(t)}$ and $\pi\theta/r \le c\pi$; (b) $d_\varphi^{(t)} \le d_\theta^{(t)}$ and $\pi\theta/r \ge c\pi$; (c) $d_\varphi^{(t)} > d_\theta^{(t)}$.

(a) $\pi\theta_t/r \le c\pi$
(b) $\pi\theta_t/r > c\pi$.

In subcase (1a), the offset of the bend is more than the offset of the angle. Since we are assuming $\pi\theta_t/r \le \pi/2$, this means that although the angle is offset, the bend is small enough such that the line is correctly classifying all points above it positively (see e.g., Figure 4a). However, some positive points are also being incorrectly classified.

The total length of the misclassified section is given by the sum of misclassified lengths on either side of the (positively classified) decision boundary. On one side, this distance will be $d_\theta^{(t)} + d_\varphi^{(t)}$, while on the other side, this distance will be $d_\theta^{(t)} - d_\varphi^{(t)}$. Hence, the total displacement will be $2d_\theta^{(t)}$.

In subcase (1b), we have that the total positive area is being correctly classified, but some points either side of this area being incorrectly classified as positive (see e.g., Figure 4b). Nevertheless, the sum of the lengths of the incorrectly classified areas is also given by

$$(d_\theta^{(t)} + d_\varphi^{(t)}) + (d_\theta^{(t)} - d_\varphi^{(t)}) = 2d_\theta^{(t)}.$$

Note that if $d_\theta^{(t)} = d_\varphi^{(t)}$, then one side of the decision boundary will exactly intersect with the boundary between positive and negative points. In this case, the sums above still hold; i.e., the total displacement is still $2d_\theta^{(t)}$.

In case (2), the offset of the angle is more than the offset of the bend. That is, some positive points are being incorrectly classified as negative, and some negative points are being incorrectly classified as positive (see e.g., Figure 4c). Again, we consider the summed length of the two displacements. In this case, the sum amounts to

$$(d_\varphi^{(t)} + d_\theta^{(t)}) + (d_\varphi^{(t)} - d_\theta^{(t)}) = 2d_\varphi^{(t)}.$$

Hence, combining the two arguments, the total displacement is given by $\max\{2d_\varphi^{(t)}, 2d_\theta^{(t)}\}$. The fitness is calculated as the proportion of correctly classified points. Since the total length of the circumference is 2π, the length of correctly classified points is given by $2\pi - \max\{2d_\varphi^{(t)}, 2d_\theta^{(t)}\}$. Dividing this by 2π will give the proportion of correctly classified points, i.e., the fitness, which is given in the theorem statement. □

We now use Lemma 3.2 to prove the efficiency of BINA on the Fraction$_c$ function class. First, however, we present the following result regarding the fixed-angle case.

Theorem 3.3. *The expected optimisation time of BINA with $N = 1$, without bias (i.e., $b = 0$) and fixed angle $\varphi = c\pi$ on Fraction$_c$ is $O(\log^2 r)$.*

The proof of Theorem 3.3 works similarly to that of [8, Theorem 1], i.e., a standard multiplicative drift argument is used (see e.g., [5]). Theorem 3.4 shows that BINA attains the same runtime bound when the angle is also variable.

Theorem 3.4. *The expected optimisation time of BINA with $N = 1$ and without bias on Fraction$_c$ is $O(\log^2 r)$.*

Proof. To prove the result, we consider two cases at a given time $t \ge 0$:

(1) $f(x_t) > 0$ and $\pi\theta_t/r \le \pi/2$;
(2) $f(x_t) = 0$ or $\pi\theta_t/r > \pi/2$.

We firstly consider case 1, in which the conditions of Lemma 3.2 hold. That is, the fitness depends on the maximum of the displacement of the angle and the displacement of the bend. Define $f(x_t)$ as in Lemma 3.2, and consider the fitness distance $g(x_t) = 1 - f(x_t)$.

Let $X_t := d_\theta^{(t)}$ and let $Y_t := d_\varphi^{(t)}$. Clearly, the fitness distance can be reduced by reducing the maximum of the two displacements, i.e., $\max\{X_t, Y_t\}$. That is, if $X_t > Y_t$, then we improve by reducing X_t until the opposite holds. Likewise, if $X_t < Y_t$, we improve by reducing Y_t until the opposite holds. That is, BINA makes progress by either reducing the displacement of the angle or the displacement of the bend. In either case, both terms must be reduced to zero, which happens simultaneously.

Although the reduction of the angle term and the bend term are not independent, we note that only one of the two processes (i.e., displacement reductions) is *active* at a given time. For example, if the bend is being reduced, the angle displacement can be ignored until the bend displacement has been sufficiently reduced to be the lower term (note that the angle displacement could increase in this period, so long as it remains the lower term).

We now consider the term $Z_t = 2X_t + Y_t$. We consider Z_t as a potential function and analyse the drift on the term Z_t by the sum of the drifts of its inherent summands. Clearly, once Z_t reaches 0, the globally optimal solution has been found.

For the drift on the term Y_t, we note that this term can either increase (to some value less than X_t), decrease towards 0, or change its sign (i.e., a jump of size at least $Y_t + 1$ and at most $X_t + Y_t$) if the bend parameter is selected with probability $1/2$. Then, we obtain

$$\begin{aligned}
\mathrm{E}[Y_t - Y_{t+1} \mid Y_t] &= \frac{1}{2}\left(-\sum_{j=1}^{X_t - Y_t} \frac{j}{jH_r} + \sum_{j=1}^{Y_t} \frac{j}{jH_r} + \sum_{j=Y_t+1}^{X_t+Y_t} \frac{2Y_t - j}{jH_r}\right) \\
&= \frac{1}{2}\left(\frac{2Y_t - X_t}{H_r} - \frac{X_t}{H_r} + \sum_{j=1}^{X_t} \frac{2Y_t}{(Y_t + j)H_r}\right) \\
&\ge \frac{2Y_t - X_t}{2H_r}.
\end{aligned}$$

For the drift of X_t, we consider any decrease in X_t, noting a possible change in sign. Hence, we have

$$\mathrm{E}[X_t - X_{t+1} \mid X_t] = \frac{1}{2}\left(\sum_{j=1}^{X_t} \frac{j}{jH_r} + \sum_{j=X_t+1}^{2X_t} \frac{2X_t - j}{jH_r}\right) \ge \frac{X_t}{2H_r},$$

Hence, summing these terms gives

$$\mathrm{E}[Z_t - Z_{t+1} \mid Z_t] \geq \frac{X_t + 2Y_t}{2H_r} \geq \frac{X_t + \frac{1}{2}Y_t}{2H_r} = \frac{Z_t}{4H_r} = \Omega(Z_t),$$

which gives the stated runtime bound via a multiplicative drift argument (see e.g., [5]).

We finally consider case 2. In this case, in order to reach the situation described in case 1, it suffices to jump (i.e., mutate the angle and bend simultaneously) to a position where

- $2\pi\varphi_t/r \in [c\pi/4, 3c\pi/4]$; i.e., $\varphi \in [cr/8, 3cr/8]$ (of size $\Omega(r)$);
- $\pi\theta/r \in [0, \pi/(2c)]$; i.e., $\theta \in [0, r/(2c)]$ (of size $\Omega(r)$).

Hence, by Lemma 3.1, the expected time to reach case 1 from case 2 is $O(\log^2 r)$, which combined with the runtime from case 1 gives the theorem statement. □

Theorem 3.4 shows that BINA without bias can match the runtime bound of the (1+1) NA on Half and maintains the same performance for every instance of $\textsc{Fraction}_c$. The (1+1) NA, however, cannot solve any instance of $\textsc{Fraction}_c$ to optimality when $0 < c < 1/2$. However, by an extension of [8, Theorem 1], it can find a local optimum quickly in time $O(\log^2 r)$ in expectation. For Quarter, it requires variable bias and expected time $O(\log^3 r)$ to find the optimal; we expect similar performance for a general instance of $\textsc{Fraction}_c$ with $0 < c < 1/2$.

3.2 Further Discussion

We firstly discuss the general case where the bias is fixed (i.e., $b \in (-1, 1)$) but not necessarily fixed to 0. In this case, it can be shown that for a general instance of $\textsc{Fraction}_c$, the optimal solution of BINA attains $\varphi = c\pi$ and $\theta = \arctan\left(\frac{\sin(c\pi)}{-b+\cos(c\pi)}\right)$. In this setting, the optimisation process will mirror that in the no-bias case. That is, BINA will simultaneously reduce the distance of the bend and the angle to their globally optimal positions and a similar drift argument will hold. Hence, we posit that the expected runtime in the fixed-bias case will also be $O(\log^2 r)$ and omit a formal proof.

Secondly, we briefly discuss the variable bias case. As above, we posit that BINA can find the globally optimal solution in time $O(\log^2 r)$ if the bias does not mutate in any iteration. If the bias term is mutated, the distances of the bend and angle from their optimal settings will change. However, any mutation of the bias which results in a lower fitness will be rejected, and hence any accepted mutation of the bias term will increase the fitness, bringing at least one of the bend or angle closer to its optimal position for the given bias, while not affecting the other too much (i.e., it would reduce the term $\max\{X_t, Y_t\}$). Again, we posit a similar upper bound on the expected runtime of $O(\log^2 r)$ and omit a formal proof.

Finally, we also discuss the case of multiple neurons. Clearly, BINA can fall into the same locally optimal positions as the (1+1) NA on the example function TwoQuarters, whereby one hyperplane impedes the progress of the other [8]. Therefore, we assume a similar situation whereby an efficient runtime can be proved with constant probability, noting such a result is likely for any fixed or variable bias setting. We further mention one interesting point. For a bias value of $b < -1$, BINA can find solutions to the TwoQuarters problem with fitness arbitrarily close to 1 (with $N = 1$) for $b \to -\infty$. In Figure 5, we illustrate the setting with $b = -4$.

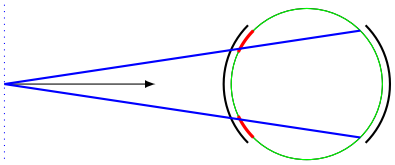

Figure 5: Near-optimal solution of BINA to TwoQuarters with one neuron ($b = -4$).

4 ONLINE FITNESS FUNCTION CALCULATION

In this section, we analyse the (1+1) NA in the online setting as described in Section 2.4 for the first time. That is, we are considering a time-dependent fitness function where exactly one new input from the training set, drawn uniformly at random, is added at each time step. Throughout, we assume $D = 2$ and $N = 1$.

We start with a general observation on the size of local optima in this setting. Note that at time $t - 1$, only t points drawn uniformly from the unit circle have been revealed. Consider these points in a clockwise direction. The expected distance between a point and its successor in this direction is well known to be $2\pi/t$ and the maximum distance of two subsequent points has an expected value of $O(\log t/t)$. This follows from a straightforward reduction to the stochastic experiment of uniformly choosing $t - 1$ points from the unit interval and modelling the circle as the unit interval wrapped around at its limits, which represent the extra point. Further, the distance D of any two subsequent points of the circle satisfies the distribution $\mathrm{Prob}[D \geq a] = (1 - a/(2\pi))^t$ (i. e., a scaled Beta distribution with parameters 1 and t), so the maximum distance is $O(\log t/t)$ with probability $1 - O(1/t)$; see David and Nagaraja [2, Chapter 6.4].

As a consequence, if the two intersections of the hyperplane formed by the (1+1) NA with the unit circle lie roughly in the middle between two of the sample points, then a local step of size $1/r$ may not change the set of points lying on one side of a hyperplane and are fitness neutral. Only random walks may lead to further progress in this situation. Hence, it makes sense to analyse the algorithm only from time $t \geq cr$ onward, where c is a sufficiently large constant. This idea will be used in the proof of the following theorem. Note that it bounds an "expected time with high probability", which means that the expected time is bounded conditional on an event that occurs with high probability.

Theorem 4.1. *The expected optimisation time of the local (1+1) NA without bias on the online version of Half is bounded by $O(r \log r)$ with probability $1 - O(1/r)$.*

Proof. Consider the algorithm at time $4\pi r \ln r$ and the sampled points in clockwise order. By the argumentation above, the probability of a distance between two adjacent points exceeding $1/r$ is at most $(1 - 1/(2\pi r))^{4\pi r \ln r} \leq 1/r^2$. By a union bound over all r distances, with high probability of at least $1 - 1/r$, the maximum distance between two adjacent points is bounded from above by $1/r$. In the following, we assume such a maximum distance to hold.

The rest of the proof picks up the argumentation for the analysis of the local (1+1) NA without bias on Half from [8, Theorem 3.1], where the aim is to reach the optimal angle of $\pi/2$. Let ϕ_t denote the

current representation of angle in the (1+1) NA at time $t \geq 4\pi r \ln r$, i. e., the actual angle is $2\pi\phi_t/r$, and let $\xi_t = \min\{|r/4 - \phi_t|, 5r/4 - 1 - \phi_t\}$, i. e., the smallest distance of ϕ_t from its optimum $r/4$ in the representation with wrap-around, where 0 is a neighbor of $r-1$. If $\xi_t = i > 0$ (corresponding, e. g., to an angle $2\pi i/r + \pi/2$), then incrementing or decrementing ϕ_t by 1 (modulo r) will improve the fitness. This holds since two samples have a maximum distance of at most $1/r$ by assumption, so moving the angle by an amount of $2\pi/r$ closer to its optimum value will move at least one incorrectly classified point to the other side of the hyperplane.

As argued in [8], whether increasing or decreasing (or both) improves the fitness depends on whether $\phi_t < 3r/4$. The local (1+1) NA chooses the improving direction for the angle with probability at least $1/2$ and reduces ξ_t by 1 with probability at least $1/2$. Altogether, the probability of improving is at least $1/4$. Since at most $r/2$ improvements are sufficient, the expected number of steps spent after time $4\pi r \ln r$ until reaching the optimum angle is at most $(r/2) \cdot 4 = O(r)$. Together with the time $4\pi r \ln r$ that we spend at the beginning, the expected time is $O(r \log r)$ in this case. □

As shown in [8, Theorem 3.1], the runtime bound for the (1+1) NA without bias on Half becomes $O(\log^2 r)$ if the Harmonic mutation operator is used. This analysis is based on multiplicative drift analysis [5]. It exploits heavily that if the error of the angle denoted by ξ_t above satisfies $\xi_t = i > 0$, then all steps decreasing i will be accepted and all steps increasing i rejected. In particular, the expected improvement is bounded from below by $\sum_{j=1}^{i} j \cdot \frac{1}{jH_r} = \Omega(i/\log r)$ then, where $1/(jH_r)$ is the probability of a step of size j. Now, if only time $\Theta(\log^2 r)$ has elapsed, then there will be adjacent points on the circle that have an expected gap of $\Omega(r/\log^2 r)$ in the chosen representation on $\{0, \dots, r\}$, and there must exist at least one gap of this size. Considering the above sum, this can invalidate the terms for $j \leq r/\ln^2 r$. Hence, after the distance to the optimum has become $O(r/\log^2 r)$ in the representation ($O(1/\log^2 r)$ in the actual angle), we have no guarantee for further fitness improvements by large jumps and the algorithm may have to rely on a random walk on a plateau. However, if the distance is still larger than $cr/\ln^2 r$ for a sufficiently large constant, then we can still exploit the ability of the Harmonic mutation performing large jumps similar to the original setting. This gives rise to the following theorem.

Theorem 4.2. *The expected time until the (1+1) NA with harmonic mutation and without bias on the online version of Half achieves a solution of additive error $O(1/\log r)$ is bounded by $O(\log^2 r)$ with probability $1 - O(1/r)$.*

Proof. Similarly to the proof of Theorem 4.1, we start our consideration only after $c \ln^2 r$ steps (for some sufficiently large constant $c > 0$) have elapsed. Then using a union bound and the bound on a single gap from above, the largest gap between two adjacent points is at most $d^* := 2 \ln r/\ln^2 r = O(1/\log r)$ with probability $1 - O(1/r)$, which we assume to happen. In the representation space, the gap is at most $(r/(2\pi))d^*$.

As in the proof of Theorem 4.1, we consider the quantity ξ_t. Given the current search point with distance $\xi_t = i > (2r/\pi)d^*$ in the representation space (i. e., 4 times the maximum gap) between adjacent samples), all step sizes $j \in (r/(2\pi))d^*, ..., i - (r/(2\pi))d^*$ can increase the x_t-value. The aim is to analyze the time until $\xi_t \leq (r/\pi)d^*$. This corresponds to an approximation error, i. e., an additive deviation of the angle from its optimum value, that is bounded by $O(1/\log r)$. Plugging in the probability for a step of size j, we obtain a drift of

$$\mathrm{E}[\xi_t - \xi_{t+1} \mid \xi_t] \geq \sum_{j=rd^*/(2\pi)}^{i-rd^*/(2\pi)} j \cdot \frac{1}{H_r} = \frac{(i - rd^*/\pi)}{H_r} \geq \frac{i}{2H_r}$$

by our assumption on i. Applying the multiplicative drift theorem [5] on the ξ-process with $\xi_0 \leq r$, drift parameter $\delta = 1/(2H_r)$ and a minimum state of $(r/\pi)d^*$ gives an expected time of $O(\log^2 r)$ to reach the target. □

We discuss how the (1+1) NA with Harmonic mutation could achieve smaller approximation errors. The challenge is that a gap of size $\Theta(1/\log r)$ between two points corresponds to a step of size $\Theta(r/\log r)$ in the representation space. In a worst-case situation, a step of that size might be the only possible strict improvement and it has a probability only $\Theta((\log r)/(rH_r)) = \Theta(1/r)$, so the expected time for it to happen would be $\Theta(r)$. This would be exponentially larger than the time bound of Theorem 4.2 for the $O(1/\log r)$-approximation. Random walk arguments would also result in exponential (in $\log r$) time bounds like $\Theta((r/\log r)^2)$ to bridge the distance. Hence, we do not see any immediate room for improvement of Theorem 4.1.

Summing up, the online fitness calculation does not substantially change the results from the earlier paper [8], which implicitly considered an infinite point set by calculating the proportion of correctly classified points on the unit hypersphere. We think that the analyses of the (1+1) NA on Quarter, where mutations changing the bias were crucial, do not change substantially either. Again, in the polylogarithmic time span $O(\log^3 r)$ considered in [8, Theorem 3.4], an error of $O(1/\log^2 r)$ must be taken into account, which corresponds to the expected maximum distance of two points on the hypersphere. Such results are likely generalisable to instances of Fraction$_c$ for $0 < c < 1/2$.

As a final point, we expect respectively similar results to hold for BINA in the online setting. We note that for non-zero fixed biases, different requirements on the number of points required to have a sufficiently small gap between adjacent points with high probability will be required. However, this is required only for a formal proof; experimental results presented in the next section show its efficient performance.

5 EXPERIMENTAL RESULTS

We now present experimental results for the algorithms presented in this paper. Firstly, we present an analysis of BINA in the standard fitness setting and compare it to the (1+1) NA across a number of different settings for the bias terms. Secondly, we present a series of simulations analysing the performance of the (1+1) NA and BINA in the online fitness setting. Harmonic mutation is used in both cases.

5.1 Standard Fitness Setting

We firstly present a comparison of the two algorithms in the standard fitness setting for two settings for the bias: no bias and variable bias. Figure 6 presents results for the Half and Quarter benchmark

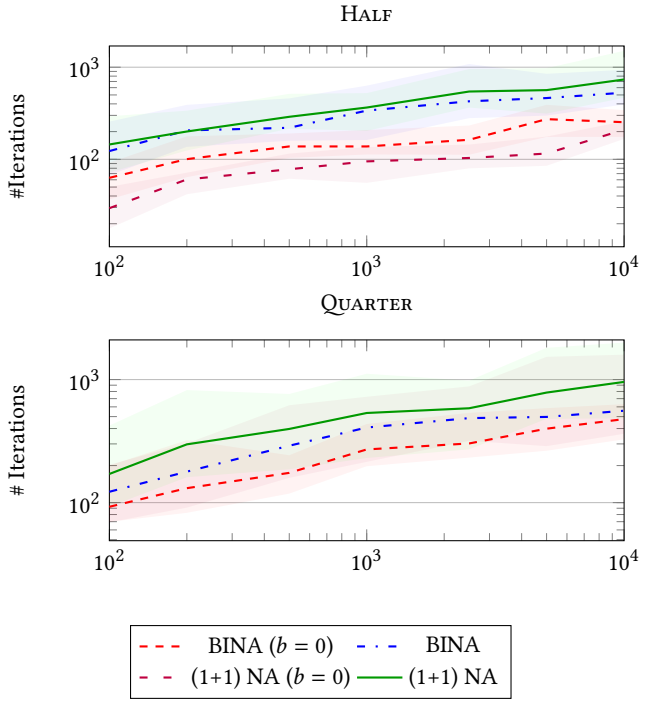

Figure 6: Average number of iterations to solve the benchmark problems at different resolutions (x-axis).

(1+1) NA (Harmonic Mutation)

Points	Res.	Mean	Stdev.	# Correct	True Fitness
200	400	1848	4424	199.25	0.9888
400	800	3447	4657	399.41	0.9950
600	1200	6830	7013	599.30	0.9964
800	1600	8030	8614	799.27	0.9969
1000	2000	9421	8221	999.33	0.9979
1200	2400	11466	11934	1199.29	0.9982
1400	2800	14085	14141	1399.34	0.9985
1600	3200	15193	13111	1599.31	0.9986
1800	3600	17282	18460	1799.36	0.9989
2000	4000	19402	19238	1999.33	0.9990

BINA (No Bias, Harmonic Mutation)

Points	Res.	Mean	Stdev.	# Correct	True Fitness
200	400	693	826	198.58	0.9853
400	800	1218	1856	398.44	0.9924
600	1200	1221	632	598.56	0.9952
800	1600	1825	1574	798.46	0.9960
1000	2000	1868	1109	998.53	0.9970
1200	2400	2327	1661	1198.56	0.9976
1400	2800	2345	998	1398.50	0.9980
1600	3200	2728	1267	1598.41	0.9980
1800	3600	3009	1154	1798.57	0.9984
2000	4000	3462	1688	1998.55	0.9986

Table 1: Summarised results for the (1+1) NA and BINA on Quarter.

functions (i.e., special instances of the general Fraction function). Each result shows the median and quartiles taken from 100 runs.

On Half, the (1+1) NA in the no-bias setting is the fastest as it is only evolving one parameter. However, it cannot solve Quarter to global optimality so we consider it as an idealised setting. The (1+1) NA with variable bias is slower than each setting for BINA across both functions. BINA in the variable bias setting attains a similar performance to both fixed settings, even being faster for larger resolutions on Quarter. This suggests that due to its generality and comparable efficiency, the variable bias setting may be most favourable. However, a fixed-bias setting is also suitable, which we see in the next section.

5.2 Online Fitness Setting

We now consider the online fitness setting, which was implemented as follows. The algorithms run in rounds, whereby at each round, a new point is given uniformly at random according to the function (up to a total of k points). The algorithm runs until all but one points are correctly classified (up to a maximum of $10r \log r$ steps, but possibly none if the new point is already correctly classified). This is due to the algorithms having difficulty (in practice) separating two areas with alternating classifications on instances of Fraction with the setting $r = 2k$, i.e., the borders between the positive and negatively classified inputs. Note that the resolution is chosen to be double the number of data points to allow for finer adjustments in the decision boundary at larger problem sizes.

In Table 1, we present the average results for (1+1) NA and the fixed-bias BINA on Quarter from 100 runs, varying the number of points from 200 to 2000. We also present the mean and standard deviation of the runtime alongside the number of correctly classified points and the true fitness. Overall, we see how much more effective BINA is than the (1+1) NA. This is supported by the theoretical and experimental results presented thus far.

6 CONCLUSIONS

In this work, we have extended upon the first framework for the runtime analysis of neuroevolutionary algorithms in two ways. Firstly, we considered more realistic topologies making use of rectified linear unit activation functions. This resulted in an algorithm that performs much more efficiently on the studied benchmarks. The second advancement was to consider an online fitness setting, which more closely resembles real-world neural network training.

While we have made advancements in the theoretical analysis of neuroevolution, there are still clear pathways for future research. From the perspective of runtime analysis, self-adapting mutation operators could lead to speedups of the presented algorithms. Secondly, this work only considers adapting the weights and biases, whereas traditional neuroevolutionary algorithms also update the topology. A more general framework combining this strand of research with those analysing updating topologies (e.g., [12]) would be beneficial.

REFERENCES

[1] Raman Arora, Amitabh Basu, Poorya Mianjy, and Anirbit Mukherjee. 2016. Understanding deep neural networks with rectified linear units. *arXiv preprint*

arXiv:1611.01491 (2016).
[2] H. A. David and H. N. Nagaraja. 2003. *Order Statistics*. Vol. 3rd. Wiley.
[3] Benjamin Doerr, Carola Doerr, and Franziska Ebel. 2015. From black-box complexity to designing new genetic algorithms. *Theoretical Computer Science* 567 (2015), 87–104.
[4] Benjamin Doerr, Carola Doerr, and Timo Kötzing. 2018. Static and self-adjusting mutation strengths for multi-valued decision variables. *Algorithmica* 80 (2018), 1732–1768.
[5] Benjamin Doerr, Daniel Johannsen, and Carola Winzen. 2010. Multiplicative drift analysis. In *Proceedings of the 12th annual conference on Genetic and evolutionary computation*. 1449–1456.
[6] Benjamin Doerr and Zhongdi Qu. 2023. Runtime analysis for the NSGA-II: Provable speed-ups from crossover. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 37. 12399–12407.
[7] Stefan Droste, Thomas Jansen, and Ingo Wegener. 2002. On the analysis of the (1+ 1) evolutionary algorithm. *Theoretical Computer Science* 276, 1-2 (2002), 51–81.
[8] Paul Fischer, Emil Lundt Larsen, and Carsten Witt. 2023. First Steps towards a Runtime Analysis of Neuroevolution. In *Proceedings of the Workshop on Foundations of Genetic Algorithms (FOGA '23)*.
[9] Matteo Fischetti and Jason Jo. 2018. Deep neural networks and mixed integer linear optimization. *Constraints* 23, 3 (2018), 296–309.
[10] Edgar Galván and Peter Mooney. 2021. Neuroevolution in deep neural networks: Current trends and future challenges. *IEEE Transactions on Artificial Intelligence* 2, 6 (2021), 476–493.
[11] George T Hall, Pietro S Oliveto, and Dirk Sudholt. 2020. Fast perturbative algorithm configurators. In *Parallel Problem Solving from Nature–PPSN XVI: 16th International Conference, PPSN 2020, Leiden, The Netherlands, September 5-9, 2020, Proceedings, Part I 16*. Springer, 19–32.
[12] Zeqiong Lv, Chao Qian, and Yanan Sun. 2024. A First Step Towards Runtime Analysis of Evolutionary Neural Architecture Search. arXiv:2401.11712 [cs.NE]
[13] Kenneth O Stanley, Jeff Clune, Joel Lehman, and Risto Miikkulainen. 2019. Designing neural networks through neuroevolution. *Nature Machine Intelligence* 1, 1 (2019), 24–35.
[14] Kenneth O Stanley and Risto Miikkulainen. 2002. Evolving neural networks through augmenting topologies. *Evolutionary computation* 10, 2 (2002), 99–127.
[15] Vincent Tjeng, Kai Xiao, and Russ Tedrake. 2017. Evaluating robustness of neural networks with mixed integer programming. *arXiv preprint arXiv:1711.07356* (2017).
[16] Hamit Taner Ünal and Fatih Başçiftçi. 2022. Evolutionary design of neural network architectures: a review of three decades of research. *Artificial Intelligence Review* (2022), 1–80.
[17] Weijie Zheng and Benjamin Doerr. 2023. Runtime Analysis for the NSGA-II: Proving, Quantifying, and Explaining the Inefficiency For Many Objectives. *IEEE Transactions on Evolutionary Computation* (2023).