7-2021

Efficient white-box fairness testing through gradient search

Lingfeng ZHANG

Yueling ZHANG
Singapore Management University, ylzhang@smu.edu.sg

Min ZHANG

Published in ISSTA 2021: Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis, Virtual, July 11-17. pp. 103-114. DOI: 10.1145/3460319.3464820

Efficient White-Box Fairness Testing through Gradient Search

Lingfeng Zhang
East China Normal University
Shanghai, China
lanford217@gmail.com

Yueling Zhang*
Singapore Management University
Singapore, Singapore
ylzhang.ecnu@gmail.com

Min Zhang*
East China Normal University
Shanghai, China
mzhang@sei.ecnu.edu.cn

ABSTRACT

Deep learning (DL) systems are increasingly deployed for autonomous decision-making in a wide range of applications. Apart from the robustness and safety, fairness is also an important property that a well-designed DL system should have. To evaluate and improve individual fairness of a model, systematic test case generation for identifying individual discriminatory instances in the input space is essential. In this paper, we propose a framework EIDIG for efficiently discovering individual fairness violation. Our technique combines a global generation phase for rapidly generating a set of diverse discriminatory seeds with a local generation phase for generating as many individual discriminatory instances as possible around these seeds under the guidance of the gradient of the model output. In each phase, prior information at successive iterations is fully exploited to accelerate convergence of iterative optimization or reduce frequency of gradient calculation. Our experimental results show that, on average, our approach EIDIG generates 19.11% more individual discriminatory instances with a speedup of 121.49% when compared with the state-of-the-art method and mitigates individual discrimination by 80.03% with a limited accuracy loss after retraining.

CCS CONCEPTS

• **Software and its engineering → Software testing and debugging**; • **Computing methodologies → Artificial intelligence**.

KEYWORDS

software bias, fairness testing, neural networks, test case generation

ACM Reference Format:
Lingfeng Zhang, Yueling Zhang, and Min Zhang. 2021. Efficient White-Box Fairness Testing through Gradient Search. In *Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA '21), July 11–17, 2021, Virtual, Denmark.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3460319.3464820

*Corresponding authors.

1 INTRODUCTION

Recently, deep learning (DL) systems are becoming crucial enablers in many real-world applications involving decision-making, including self-driving [5, 21], traffic control [28], medical diagnosis [34], fraud detection [19], etc. Although deep neural networks (DNNs) have achieved an astonishing performance commensurate with the cognitive abilities of humans in widespread domains, assurance evidence for their trustworthiness is insufficient, especially when such models are designed to make life-changing decisions. It has been found that DNNs are vulnerable to adversarial examples, which are crafted to fool the model under test by slightly perturbing the original inputs [22, 31, 40, 47]. Apart from the robustness risk exposed by adversarial examples, fairness violation is also a notable threat to the full acceptance of DL.

For decision-making in data-driven applications, fairness is non-discrimination, which ensures that no group or individual is prejudiced or favored due to their inherent or acquired characteristics [37, 43]. We call such characteristics protected attributes (e.g., age, race, gender, etc.). Discrimination might limit the opportunity that a group or an individual is qualified and further exacerbate social inequity [14]. More and more real-world examples of algorithmic bias that demonstrate harmful societal effects have emerged, such as racial bias in the COMPAS recidivism prediction model [3] and gender bias in Amazon's recruiting model [11]. The bias in DNNs can be traced back to the training data. However, It is not effective enough for bias mitigation to simply remove the protected attributes before training, because these attributes could be derived by the combination of other features. Due to the difficulty in keeping fairness during design and engineering, systematical bias detection is a crucial step before deployment.

To detect and evaluate software bias, a growing number of studies have been targeted to group fairness and individual fairness respectively. The group fairness focuses on the statistical parity for different groups [24, 32], while the individual fairness emphasizes that any two individuals who differ only in their intrinsic or acquired traits should be treated in a similar fashion during the decision-making [16, 20]. Note that some discrimination behaviors are not captured in the context of the group fairness when the model discriminates against a group in one setting but favors them in another, while the individual fairness can identify these behaviors [20]. Therefore, we focus on the individual fairness in this work. To uncover individual fairness violation, Galhotra *et al.* [20] proposed Themis to generate test suites for discrimination measurement by randomly sampling from the input space. Udeshi *et al.* [50] presented AEQUITAS, which first discovers some discriminatory inputs through random sampling and then searches the neighborhood of them for identifying more discriminatory inputs by perturbing these seeds under the guidance of a global adaptive probability distribution. Agarwal *et al.* [2] introduced traditional

software testing techniques into fairness testing for machine learning models. Their approach Symbolic Generation (SG) first adopts local explanation to approximate the original model with a decision tree, and then performs symbolic execution to systematically generate test cases evidencing discrimination. Zhang *et al.* [51] proposed Adversarial Discrimination Finder (ADF) to generate individual discriminatory instances utilizing the gradient of the loss function as guidance. In the global generation phase, ADF iteratively perturbs the original inputs towards the decision boundary to rapidly identify a set of discriminatory seeds. In the local generation phase, ADF searches the neighborhood of these discriminatory seeds for more discriminatory instances according to an input-specific probability distribution. Despite their abilities to generate plentiful test cases violating the individual fairness for several real-world datasets, these approaches are far from efficient. We further discuss them in Section 5.

To this end, we propose a scalable and efficient approach called *Efficient Individual Discriminatory Instances Generator* (EIDIG) to systematically generate test cases that violate the individual fairness for differentiable models (e.g., DNNs). EIDIG inherits and improves the state-of-the-art approach ADF [51] in three aspects. Primarily, We utilize the gradient of the model output w.r.t the corresponding input instead of the gradient of the loss function to build a more direct and precise mapping between input perturbations and output variations. The absence of the backpropagation through the loss function significantly reduces the computation cost at each iteration for the two-phase generation. During global generation, we integrate the momentum term into the iterative search for identifying discriminatory seeds. The momentum term enables the memorization of the previous trend and helps to escape from local optima [15], which ensures a higher success rate of finding individual discriminatory instances. During local generation, we reduce frequency of gradient calculation by exploiting the prior knowledge of gradients. In our experiments, we find that attribute contributions are highly correlated at successive iterations due to minor perturbation preferred in the local phase. Consequently, gradients and attribute contributions are recalculated every few iterations to decrease time cost while maintaining effectiveness in discrimination detection.

We have implemented our framework EIDIG and comprehensively compared it with state-of-the-art approach ADF for 10 benchmarks on three real-world datasets. Our experimental results show that on average EIDIG-∞ generates 19.11% more individual discriminatory instances with a speedup of 121.49% when the maximum number of search attempts is fixed. After retraining, the individual discrimination in the model is reduced by 80.03% with a limited accuracy loss on average.

Overall, the main contributions of our paper are:

- We propose an efficient and effective approach EIDIG for generating test cases evidencing the individual discrimination for differentiable models through gradient search.
- We implement EIDIG with Python3.6 and Tensorflow2.0 [1]. All details about our implementation, experimental data, and trained models are available online[1].

[1]https://github.com/LingfengZhang98/EIDIG

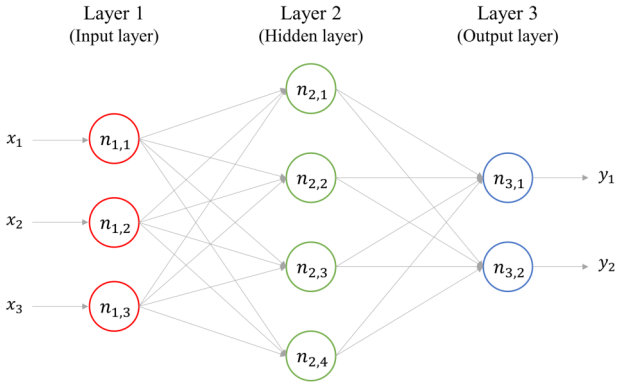


Figure 1: A shallow feedforward neural network.

- We evaluate EIDIG against the state-of-the-art method ADF [51] with 10 benchmarks on three real-world datasets, including four combinatorial benchmarks. It is shown that our approach significantly outperforms ADF.
- We leverage the generated individual discriminatory instances for data augmentation and then automatically retrain the original models to successfully mitigate bias.

The remainder of the paper is structured as follows. In Section 2, we briefly present the necessary background. In Section 3, we describe and justify EIDIG in detail. In Section 4, we discuss our experimental results. In Section 5, we review the evolution in the relevant research domains. Finally, we conclude our work in Section 6.

2 BACKGROUND

2.1 Deep Neural Networks

A feedforward neural network $\mathcal{N}$ contains one input layer for receiving the input data, multiple hidden layers for computing, and one output layer for formatting the outputs. We show a simple example in Fig. 1. Formally, we define $\mathcal{N}$ as a tuple (L, C, Φ), where $L = \{L_i | i \in \{1, ..., l\}\}$ denotes the set of layers, the i-th layer of which contains s_i neurons, $C \subseteq L \times L$ denotes connections between layers, and $\Phi = \{\phi_i | i \in \{2, ..., l\}\}$ denotes the set of activation functions (e.g., Rectified Linear Unit (ReLU)[39], sigmoid, hyperbolic tangent (tanh), etc.) used in each layer. The neuron j of the layer i is denoted by $n_{i,j}$, and its corresponding activation value is $v_{i,j}$. Except in the input layer, each neuron is connected to the neurons in the preceding layer with pretrained weights such that for $2 \leq i \leq l$ and $1 \leq j \leq s_i$, activation value of $n_{i,j}$ is given by

$$v_{i,j} = \phi_i(b_{i,j} + \sum_{k=1}^{s_{i-1}} w_{i-1,k,j} \cdot v_{i-1,k}) \tag{1}$$

where $w_{i-1,k,j}$ is the weight for the connection between $n_{i-1,k}$ and $n_{i,j}$, $b_{i,j}$ is the bias term for $n_{i,j}$, and ϕ_i is the activation function used in the layer i. Essentially, a DNN model is a composite function, whose gradients are easily computed by applying the chain rule [46]. The output vector $\mathcal{F}(x)$ of the model $\mathcal{N}$ is composed of the activation values from the output layer and each element of $\mathcal{F}(x)$ represents the prediction probability for the corresponding class.

We can compute the Jacobian matrix of $\mathcal{F}(x)$ w.r.t. a given input vector x by

$$J_{\mathcal{F}}(x) = \frac{\partial \mathcal{F}(x)}{\partial x} = \left[\frac{\partial \mathcal{F}_n(x)}{\partial x_m}\right]_{m \times n} \tag{2}$$

where the n-th column is the gradient vector of the n-th output element w.r.t. input vector x.

2.2 Gradient-Based Adversarial Attacks

Recently, DNNs have been found vulnerable to well-designed inputs called adversarial examples[47], which is imperceptible to humans but able to easily fool state-of-the-art models. Many adversarial attack methods have been proposed, which deliberately perturb the original inputs from the training set to generate adversarial examples. These methods are potentially implemented to generate test cases for other requirements (e.g., fairness testing).

Gradient-based adversarial attacks utilize the gradient as a powerful guidance for optimization. Goodfellow *et al.* [22] proposed Fast Gradient Sign Method (FGSM) to generate adversarial examples. They only performed one-step perturbation with the sign of the gradient at each pixel. Kurakin *et al.* [31] extended FGSM by using smaller perturbation steps for multiple iterations, which is called Basic Iterative Method (BIM). Dong *et al.* [13] integrate momentum into BIM to generate more transferable adversarial examples. Papernot *et al.* [40] designed a novel method called Jacobian-based saliency map attack (JSMA). They constructed a saliency map to represent the importance of each pixel for decision-making, which is given by

$$S_{xy}[i] = \begin{cases} 0 \text{ if } \operatorname{sign}(J_{it}(x)) = \operatorname{sign}\left(\sum_{j \neq t} J_{ij}(x)\right) \\ J_{it}(x) * \left|\sum_{j \neq t} J_{ij}(x)\right| \text{ otherwise} \end{cases} \tag{3}$$

where J is the Jacobian matrix of the output y w.r.t. the input x, and t is the target class for the attack. Then they selected the two most significant input features to perturb at each iteration according to the absolute value of saliency value $|S_{xy}[i]|$. Different from FGSM and BIM, JSMA adopts the gradient of the network instead of the gradient of the loss function, which omits the backpropagation through loss function at each iteration. Inspired by JSMA, our work also establish a precise mapping between inputs and outputs of DNNs based on the gradient of the model output to guide the fairness testing.

2.3 Individual Discrimination

It is proved that if a software group discriminates against a set of protected attributes, it must individually discriminate against that set at least as much [20]. Therefore, we only focus on the individual fairness in this work.

Definition 2.1 (Individual discriminatory instance). Let X be a dataset with a set of attributes $A = \{a_i | i \in \{1, ..., n\}\}$. If $a_i \in \mathbb{I}_i$ holds for $1 \leq i \leq n$, the input domain of the corresponding decision-making software $\mathcal{S}$: $\mathbb{I} \rightarrow \{True, False\}$ is $\mathbb{I} = \mathbb{I}_1 \times \mathbb{I}_2 \times \ldots \times \mathbb{I}_n$. We use P to denote the set of protected attributes, and thus $A \setminus P$ is the set of non-protected attributes. We define that an instance $x \in \mathbb{I}$ is an individual discriminatory instance for software $\mathcal{S}$ if there exists an instance $x' \in \mathbb{I}$ such that

- $\exists a \in P, x_a \neq x'_a$
- $\forall a \in A \setminus P, x_a = x'_a$
- $\mathcal{S}(x) \neq \mathcal{S}(x')$

The tuple (x, x') is thus an individual discriminatory instance pair.

Example 2.1. For a dataset with 12 features, we choose gender as the protected attribute and take a pair (x, x') from the dataset as an example:

x: [2, 0, 10, 1, 8, 3, 0, 0, 1, 1, 3, 0]
x': [2, 0, 10, 1, 8, 3, 0, 1, 1, 1, 3, 0]

We highlight gender in red for clarity. If the decision-making software $\mathcal{S}$ makes different predictions for x and x', x is an individual discriminatory instance w.r.t. gender, and (x, x') is an individual discriminatory instance pair. Next, we formally introduce the notion of perturbations and define the problem to be solved.

Definition 2.2 (Perturbation). We define a perturbation function $f : \mathbb{I} \times (A \setminus P) \times \Gamma \rightarrow \mathbb{I}$ where Γ is the set of possible directions for perturbation, e.g., Γ is defined as {-1, 1} for a single discrete attribute. For simplicity, we use $f(x)$ to denote the instance generated from x by perturbation.

Problem definition. For a given dataset X and a DNN model $\mathcal{N}$, we attempt to quickly generate as many diverse individual discriminatory instances as possible that manifests individual fairness violation in $\mathcal{N}$ by perturbing the seed inputs in X and mitigate the individual bias with the generated individual discriminatory instances.

3 APPROACH

In this section, we present our gradient-based algorithm called *Efficient Individual Discriminatory Instances Generator* (EIDIG) for generating individual discriminatory instances, whose high-level workflow is shown in Fig. 2. EIDIG generates individual discriminatory instances in two sequential phases (i.e., a global generation phase and a local generation phase). In the following, we will describe our innovations and improvements in EIDIG in detail. We first demonstrate how to construct a direct and precise mapping from input perturbations to output variations, which is adopted as guidance for generating individual discriminatory instances. We then present how to introduce momentum into the global generation phase to improve the success rate of finding discriminatory seeds. Finally, we elaborate on how we generate as many individual discriminatory instances as possible with a limited budget for gradient calculation in the local generation phase.

3.1 Utilizing Gradient as A Powerful Guidance

To systematically generate abundant test cases evidencing individual fairness violation, we first need to find effective guidance that provides precise information for the search process.

Assume that the model $\mathcal{N}$ classifies an input x as class p for the clarity of illustration. In [51], Zhang *et al.* adopted the gradient of the loss function $\mathcal{L}$ w.r.t. the input features x ($\nabla_x \mathcal{L}(\mathcal{F}(x), y)$) as guidance for generating individual discriminatory instances. The sign of the gradient is the direction in which the function value increases most quickly, so the prediction error $\mathcal{L}(\mathcal{F}(x), y)$ will increase and the confidence for current prediction $\mathcal{F}_p(x)$ will decrease if the attributes are perturbed in the direction of $\nabla_x \mathcal{L}(\mathcal{F}(x), y)$ and vise versa. The impact of each attribute on the model predictions is proportional to the absolute value of the corresponding gradient element when the attribute is perturbed. Then any seed input can

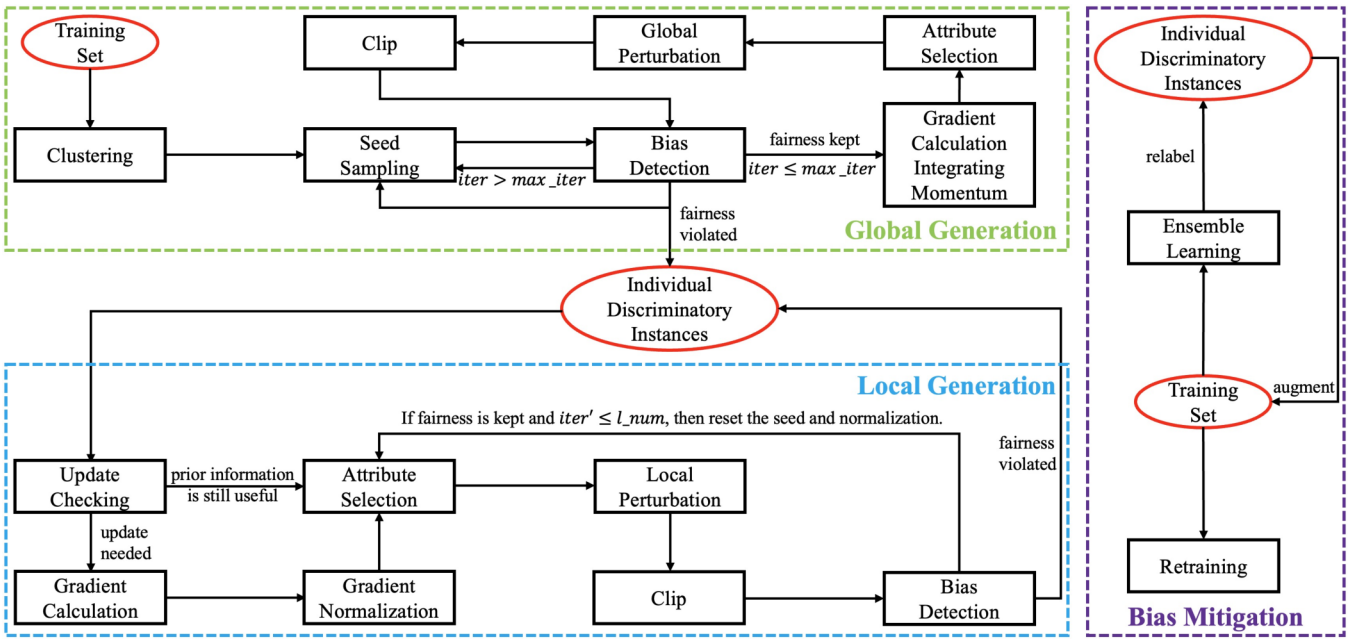


Figure 2: EIDIG workflow of generating individual discriminatory instances for a DNN model.

be systematically perturbed to generate new inputs while getting or keeping a certain property according to the above rules.

In this paper, we take the derivative of the model $\mathcal{N}$ and utilize the gradient of the model output $\mathcal{F}(x)$ w.r.t. the input features x ($\nabla_x \mathcal{F}(x)$) to construct a direct and precise mapping from input perturbations to output variations. Specifically, we only need to calculate the gradient of the class p predicted by the model ($\nabla_x \mathcal{F}_p(x)$) for an input x. In this way, we can increase the confidence for current prediction $\mathcal{F}_p(x)$ by perturbing the input in the direction of $\nabla_x \mathcal{F}_p(x)$ or decrease it by perturbing the input in the opposite direction. Compared with the gradient of the loss function, the gradient of the predicted class saves the backpropagation through the loss function w.r.t. computation, i.e., the derivatives of the loss function w.r.t. the outputs of the output layer's neurons ($\frac{\partial \mathcal{L}(\mathcal{F}(x),y)}{\partial \mathcal{F}(x)}$) and the derivatives of the outputs of these neurons except neuron p w.r.t. the penultimate layer's outputs $\mathcal{H}(x)$ ($\frac{\partial \mathcal{F}(x)}{\partial \mathcal{H}(x)}$ except $\frac{\partial \mathcal{F}_p(x)}{\partial \mathcal{H}(x)}$) are not computed any more according to the chain rule [46].

Example 3.1. We take the network shown in Fig. 1 as an example to illustrate the difference between the gradient of the loss function and the gradient of the predicted class. Assume that an input $x = (x_1, x_2, x_3)$ is labeled as class one, and the model predicts the probabilities $\mathcal{F}(x) = (y_1, y_2)$. We choose the mean squared error as the loss function $\mathcal{L}$. We then calculate the gradient of the loss function and the gradient of the predicted class respectively:

$$\begin{aligned}\nabla_x \mathcal{L} &= \frac{\partial \mathcal{L}(\mathcal{F}(x),y)}{\partial x} = \frac{\partial \mathcal{L}(\mathcal{F}(x),y)}{\partial \mathcal{F}(x)} \cdot \frac{\partial \mathcal{F}(x)}{\partial \mathcal{H}(x)} \cdot \frac{\partial \mathcal{H}(x)}{\partial x} \\ &= \left[\frac{\partial \mathcal{L}(\mathcal{F}(x),y)}{\partial \mathcal{F}_1(x)} \quad \frac{\partial \mathcal{L}(\mathcal{F}(x),y)}{\partial \mathcal{F}_2(x)}\right] \cdot \begin{bmatrix}\frac{\partial \mathcal{F}_1(x)}{\partial \mathcal{H}(x)} \\ \frac{\partial \mathcal{F}_2(x)}{\partial \mathcal{H}(x)}\end{bmatrix} \cdot \frac{\partial \mathcal{H}(x)}{\partial x}\end{aligned} \tag{4}$$

$$\nabla_x \mathcal{F}_p = \frac{\partial \mathcal{F}_1(x)}{\partial x} = \frac{\partial \mathcal{F}_1(x)}{\partial \mathcal{H}(x)} \cdot \frac{\partial \mathcal{H}(x)}{\partial x} \tag{5}$$

Comparing Eq. 4 with Eq. 5, we find that it is not necessary to calculate $\frac{\partial \mathcal{L}(\mathcal{F}(x),y)}{\partial \mathcal{F}_1(x)}$, $\frac{\partial \mathcal{L}(\mathcal{F}(x),y)}{\partial \mathcal{F}_2(x)}$ and $\frac{\partial \mathcal{F}_2(x)}{\partial \mathcal{H}(x)}$ when computing $\nabla_x \mathcal{F}_p$. The former two terms share similar computational process, so here we only compute $\frac{\partial \mathcal{L}(\mathcal{F}(x),y)}{\partial \mathcal{F}_1(x)}$ as an example:

$$\begin{aligned}\frac{\partial \mathcal{L}(\mathcal{F}(x),y)}{\partial \mathcal{F}_1(x)} &= \frac{\partial(\frac{1}{2} \cdot ((\mathcal{F}_1(x)-1)^2 + \mathcal{F}_2^2(x)))}{\partial \mathcal{F}_1(x)} \\ &= \frac{1}{2} \cdot 2 \cdot (\mathcal{F}_1(x)-1) = \mathcal{F}_1(x) - 1\end{aligned} \tag{6}$$

We then compute $\frac{\partial \mathcal{F}_2(x)}{\partial \mathcal{H}(x)}$ with the following expressions:

$$\frac{\partial \mathcal{F}_2(x)}{\partial \mathcal{H}(x)} = \left[\frac{\partial \mathcal{F}_2(x)}{\partial \mathcal{H}_1(x)} \quad \frac{\partial \mathcal{F}_2(x)}{\partial \mathcal{H}_2(x)} \quad \frac{\partial \mathcal{F}_2(x)}{\partial \mathcal{H}_3(x)} \quad \frac{\partial \mathcal{F}_2(x)}{\partial \mathcal{H}_4(x)}\right] \tag{7}$$

These four elements also share similar computational process, and here we take $\frac{\partial \mathcal{F}_2(x)}{\partial \mathcal{H}_1(x)}$ as an example:

$$\begin{aligned}\frac{\partial \mathcal{F}_2(x)}{\partial \mathcal{H}_1(x)} &= \frac{\partial \phi_3(\sum_{k=1}^{4} w_{2,k,2} \cdot \mathcal{H}_k(x))}{\partial \mathcal{H}_1(x)} \\ &= \phi_3'(\sum_{k=1}^{4} w_{2,k,2} \cdot \mathcal{H}_k(x)) \cdot w_{2,1,2}\end{aligned} \tag{8}$$

The computation cost we reduce is proportional to the difficulty of the loss function, the number of neurons in the last two layers, and the difficulty of the activation function used in the output layer. Note that DNNs deployed in the industries can exploit well-designed loss functions, have a huge width and adopt non-saturating activation functions. Therefore, the guidance used in our work can significantly reduce time consumption for generating individual discriminatory instances.

Algorithm 1 Global Generation

Input: Training set X, model under test $\mathcal{N}$, protected attributes set P, input domain $\mathbb{I}$, c_num, g_num, η, max_iter, s_g.
Output: Diverse seeds g_id violating individual fairness.

1: $g\_id = \emptyset$
2: clusters = Clustering(X, c_num)
3: **for** i from 0 to g_num **do**
4: Sample a seed x from clusters in a round-robin fashion
5: $m = m' =$ ZerosLike(x)
6: **for** $iter$ from 0 to max_iter **do**
7: $similar\_x = \{x' \in \mathbb{I} | \exists a \in P, x'_a \neq x_a; \forall a \in A \setminus P, x'_a = x_a\}$
8: **if** x violates individual fairness **then**
9: $g\_id = g\_id \cup x$
10: **Break**
11: **end if**
12: $x' = \text{argmax}\{\|\mathcal{F}(x) - \mathcal{F}(x')\|_2 | x' \in similar\_x\}$
13: $m = \eta * m + \partial \mathcal{F}(x)_{\text{pred}} / \partial x$
14: $m' = \eta * m' + \partial \mathcal{F}(x')_{\text{pred}} / \partial x'$
15: $dir =$ ZerosLike(x)
16: **for** $a \in A \setminus P$ **do**
17: **if** $\text{sign}(m_a) = \text{sign}(m'_a)$ **then**
18: $dir_a = (-1) * sign(m_a)$
19: **end if**
20: **end for**
21: $x = x + s\_g * dir$
22: $x =$ Clip$(x, \mathbb{I})$
23: **end for**
24: **end for**
25: **return** g_id

3.2 Boosting Global Generation with Momentum

Algorithm 1 presents the skeleton algorithm for global generation. The purpose that we adopt a global generation phase is to accelerate the process of generating individual discriminatory instances and diversify the generated instances.

To cover diverse instances, we first cluster the original training set X with clustering algorithms like K-Means [35] (line 2). We then sample a seed x from the clusters in a round-robin fashion (line 4).

As defined in Definition 2.1, the key to identifying an individual discriminatory instance is to successfully find an individual discriminatory instance pair. We thus collect all instances that differ only in protected attributes from x as a set $similar\_x$ (line 7), the size of which is the number of all possible combinations of the selected protected attributes in $\mathbb{I}$ except x. Afterward, we check whether x violates the individual fairness by exploring $similar\_x$ (lines 8-11). If there is an instance that gets a different prediction from x, then x is an individual discriminatory instance according to Definition 2.1, and we continue to sample another seed input. Otherwise, we iteratively perturb x with the guidance of the gradient of the predicted class until an individual discriminatory instance is generated or the maximum iteration max_iter is reached (lines 12-22). We traverse $similar\_x$ and select an instance x' from it such that $\|\mathcal{F}(x) - \mathcal{F}(x')\|_2$ is maximized (line 12). The intuition is that x and x' are more likely to be split by the decision boundary of $\mathcal{N}$ after perturbation if the Euclidean distance between them is maximized.

Momentum optimization [42] is a technique for speeding up iterative methods by adding the local gradient to the momentum vector at each iteration. For each instance in the potential pair (x, x'), We compute the gradient of the predicted class w.r.t. the input vector and integrate the previous momentum term multiplied by a decay factor η into the local gradient(lines 13,14). The momentum term enables the memorization of previous gradients, which helps to stabilize update directions and escape from poor local minima or maxima [15]. Therefore, the introduction of momentum boosts the iterative search process and improves the success rate of discovering the individual discrimination during global generation.

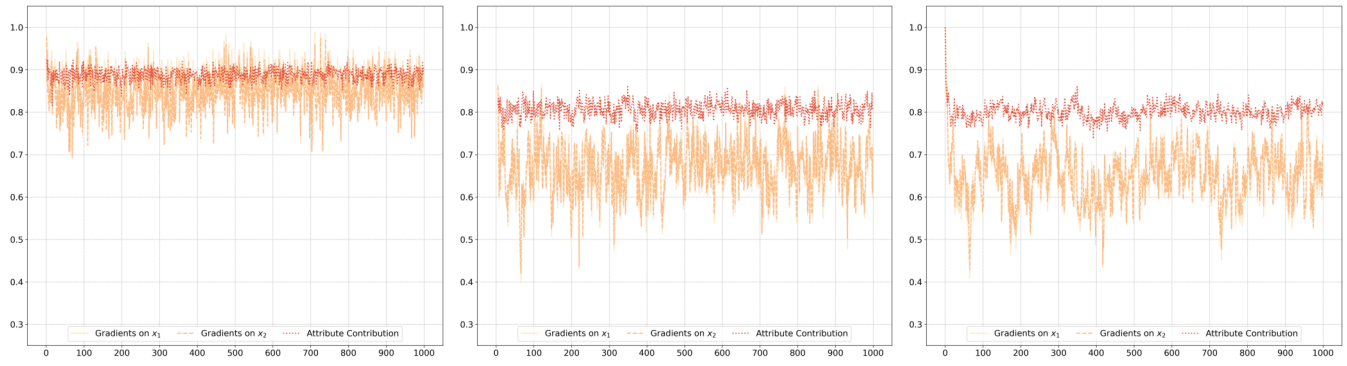
After the momentum vectors m and m' are calculated, the problem is how to choose attributes and directions for perturbation such that $\mathcal{N}(f(x)) \neq \mathcal{N}(f(x'))$. Note that the same perturbation should be added to both x and x' at each iteration in order to keep the pair only different in the selected protected attributes. To solve this problem, Zhang *et al.* [51] tries to maximize the difference between $\mathcal{N}(f(x))$ and $\mathcal{N}(f(x'))$ via EM algorithm [12] and hence selects the attributes on which the gradients of the loss function for the pair have the same sign to perturb. Afterward, They perturb these chosen attributes in the same direction with the gradient of the loss function to make $\mathcal{N}$ reduce its confidence for the current predictions. Similarly, we utilize the gradient of the model output integrating a momentum term to guide the optimization process. We also select the non-protected attributes whose corresponding momentums have the same sign, and then perturb these attributes in the opposite direction of the momentum vectors (lines 15-21). Eventually, we clip the generated instance to keep it within the input domain $\mathbb{I}$ (line 22). Intuitively, the pair are iteratively perturbed towards the decision boundary of $\mathcal{N}$ to maximize the likelihood that they get different predictions from $\mathcal{N}$.

The individual discriminatory instances generated during global generation (i.e., g_id) will be taken as the seed inputs for local generation.

3.3 Accelerating Local Generation by Reducing Frequency of Gradient Calculation

During local generation, we attempt to rapidly generate as many individual discriminatory instances as possible in the neighborhood of the seeds g_id generated by global generation. Our motivation is from the robustness of well-trained classifiers [17, 47], which requires that similar valid inputs are mapped to similar or the same outputs. Note that during global generation, we maximally change the model outputs for the potential individual discriminatory instance pair at each iteration to quickly discover a discriminatory input, while in the local generation phase, we minimally change the model outputs to maintain the original predictions from the model for the pair. In this way, the perturbed pair still get different model predictions while keeping only different in some protected attributes, i.e., a new individual discriminatory instance is generated around the original one. Therefore, we prefer to choose the non-protected attribute whose contribution to the model prediction is small for perturbation.

Recall that we integrate the momentum term into the global generation phase since it is necessary to consider the overall trend of optimization when a big step is taken at each iteration. In contrast,



(a) Successive iterations (b) Every 5 iterations (c) Every iteration with the initial one

Figure 3: Cosine similarity between the information at the current and previous iterations during local generation of ADF.

Algorithm 2 Local Generation

Input: Model under test $\mathcal{N}$, protected attributes set P, input domain $\mathbb{I}$, g_id, l_num, s_l, $update\_interval$.
Output: Individual discriminatory instances l_id around g_id.

1: $l\_id = \emptyset$
2: **for** $x \in g\_id$ **do**
3: $counts = update\_interval$
4: **for** i from 0 to l_num **do**
5: **if** $counts = update\_interval$ **then**
6: $similar\_x = \{x' \in \mathbb{I} | \exists a \in P, x'_a \neq x_a; \forall a \in A \setminus P, x'_a = x_a\}$
7: Select $x' \in \{x' \in similar\_x | \mathcal{N}(x) \neq \mathcal{N}(x')\}$
8: $grad = \partial \mathcal{F}(x)_{\text{pred}} / \partial x$
9: $grad' = \partial \mathcal{F}(x')_{\text{pred}} / \partial x'$
10: $prob = \text{Normalization}(grad, grad', P)$
11: $counts = 0$
12: **end if**
13: $counts = counts + 1$
14: Select $a \in A \setminus P$ with probability $prob_a$
15: Select $dir \in \{-1, 1\}$ with uniform probability
16: $x_a = x_a + s\_l * dir$
17: $x = \text{Clip}(x, \mathbb{I})$
18: **if** x violates individual fairness **then**
19: $l\_id = l\_id \cup x$
20: **else**
21: Reset(x); Reset($prob$)
22: $counts = 0$
23: **end if**
24: **end for**
25: **end for**
26: **return** l_id

information guiding attribute selection and perturbation is likely to be highly correlated at each iteration due to minor perturbation in the local generation phase. To confirm this, we empirically show the correlation in Fig. 3 by plotting the cosine similarity between iterations along the local search trajectory of ADF as [27] did for iterative attacks. We find that the attribute contribution measured by ADF is almost constant during the local search on a single discriminatory seed. Therefore, there is no need to update gradient normalization at each iteration like ADF, and we prefer to recalculate the attribute contributions every few iterations or even never update it. In this way, the frequency of gradient calculation are drastically decreased with a very limited loss in the number of discriminatory instances found.

Algorithm 2 shows the details of local generation. For each seed x in g_id, we can always find a similar instance x' such that $\mathcal{N}(x) \neq \mathcal{N}(x')$ (lines 6,7). We then utilize the gradient of the model output to guide attribute selection and perturbation like ADF (lines 8-10, 14-16). We first form a probability distribution on the non-protected attributes as the attribute contributions, which calculates the sum of the absolute values of the gradients as an indicator of the influence on the model predictions for each non-protected attribute, and then normalizes the corresponding reciprocals as the probability distribution for attribute selection because we tend to choose the attribute with less impact on the model predictions. After choosing an attribute according to the probability distribution, we determine the perturbation direction based on uniformly random distribution. After perturbation, we also clip the instance x within the input domain $\mathbb{I}$. If x violates the individual fairness after perturbation, we repeat the process with the generated discriminatory input. Otherwise, the instance is reset to the original value. The main difference is that the constant $update\_interval$ is introduced to control the interval steps between gradient calculation instead of simply updating the attribute contributions at each iteration.

At this point, we have generated abundant individual discriminatory instances (i.e., $g\_id \cup l\_id$), which can be leveraged to mitigate bias for the original model. In the next section, we experimentally show effectiveness and efficiency of EIDIG.

4 EVALUATION

In this section, we evaluate how EIDIG performs. Our experimental evaluation answers the research questions below.

RQ1: How effective and efficient is EIDIG in generating individual discriminatory instances?
RQ2: Does utilizing the gradient of the model output as guidance improve the ability of EIDIG to generate individual discriminatory instances?

Table 1: DNN models under test and accuracy variation after retraining

Dataset	Model	Input Domain	Accuracy/F1-score (before)	Accuracy/F1-score (after)		
				ADF	EIDIG-5	EIDIG-∞
Census Income	Six-layer Fully-connected Neural Network	2.96×10^9	84.32%/0.6258	83.66%/0.5632	84.12%/0.6131	84.11%/0.6324
German Credit	Six-layer Fully-connected Neural Network	1.01×10^{11}	78.25%/0.5756	77.25%/0.5517	77.00%/0.5619	74.50%/0.5750
Bank Marketing	Six-layer Fully-connected Neural Network	2.34×10^8	89.22%/0.2756	89.03%/0.3351	89.14%/0.3199	89.00%/0.3291

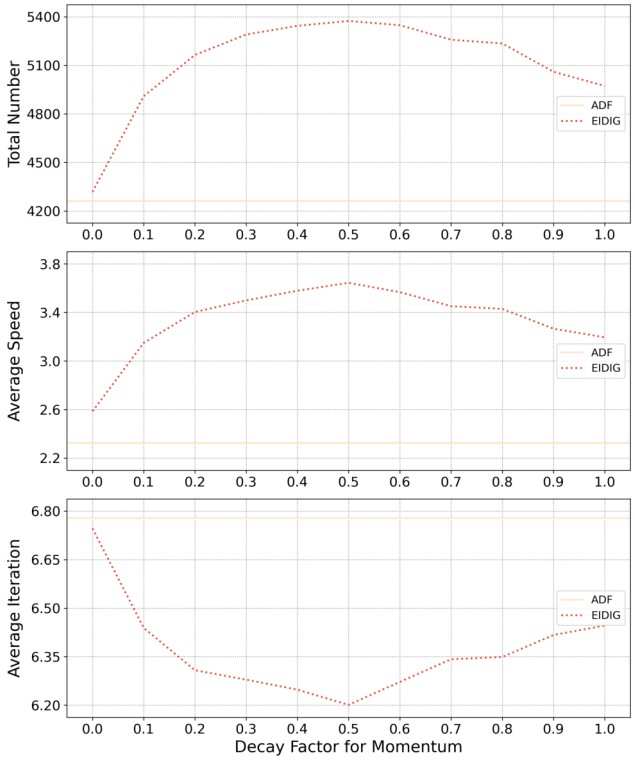


Figure 4: Effect of decay factor during global generation.

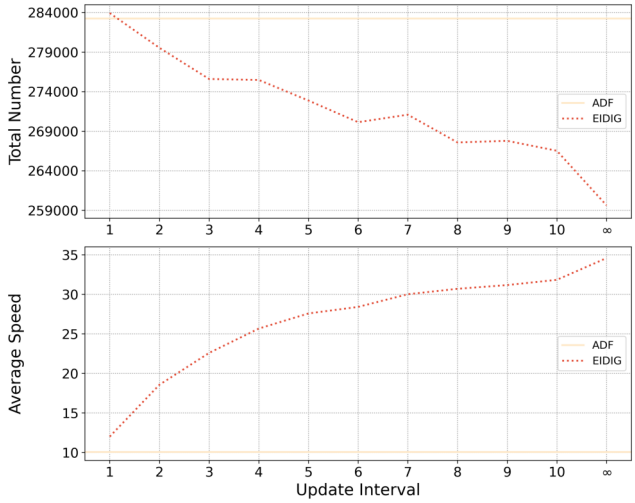


Figure 5: Effect of update interval during local generation.

RQ3: Does momentum boost the iterative search during global generation?

RQ4: Does EIDIG achieve a significant speedup while ensuring effectiveness during local generation with a low frequency of gradient calculation?

RQ5: How useful are the generated test inputs for improving the fairness of the model?

To answer these research questions, we systematically designed our experiments comparing EIDIG with the state-of-the-art method ADF [51]. Note that Zhang *et al.* proved that ADF significantly outperforms other methods, including Themis [20], AEQUITAS [50], and Symbolic Generation (SG) [2]. Themis and AEQUITAS are lightweight, but they suffer from the high duplicate rate when plentiful search attempts are made and miss many combinations of non-protected attributes that may violate individual fairness [2, 51]. SG seeks to solve these problems by combining local explanation and symbolic execution, but it is relatively heavy and less scalable. The most recent work ADF introduces the gradient of the loss function as guidance for discovering individual discrimination and shows state-of-the-art performance in efficiency and effectiveness. Therefore, we only choose ADF for baseline comparison.

4.1 Experimental Setup

Datasets and Models. We evaluate EIDIG on three real-world datasets, which are most commonly used in individual fairness testing [2, 20, 50, 51], including Census Income[2], German Credit[3], and Bank Marketing[4]. The prediction tasks of these datasets are to determine whether a person meets a certain condition. Here we first preprocess the datasets with the binning method to convert all attributes to categorical ones, e.g., discretize age according to the human life cycle. Afterward, we train six-layer fully-connected DNN models for consistency with ADF, and the details of these models are listed in Table 1. In this context, we choose age, race, and gender as the protected attributes. To ensure reasonability, We have verified that these three attributes are not heavily correlated with any other attributes in the subject datasets by applying Spearman's rank-order correlation [45]. The previous research [2, 20, 50, 51] only conducted experiments on a single protected attribute. However,

[2]https://archive.ics.uci.edu/ml/datasets/adult
[3]https://archive.ics.uci.edu/ml/datasets/statlog+(german+credit+data)
[4]https://archive.ics.uci.edu/ml/datasets/bank+marketing

the bias towards some subgroups can be hidden or compromised if each protected attribute is considered in isolation [6]. To fully expose the bias in the models under test, we also combine some protected attributes together as additional benchmarks (e.g., gender and race on Census Income dataset). For ease of presentation, We denote each benchmark as 'X-x', where 'X' is the capital initial of the dataset name, and 'x' is the initial of the protected attribute.

Configurations. During global generation, we use K-Means [35] to cluster the training set with the cluster number *c_num* set to 4 as SG and ADF did. Fig. 4 shows the average effect of decay factor on all 10 benchmarks given 1000 seeds for each benchmark (600 for German Credit Dataset). When the decay factor η is set to 0.5 for the momentum term, EIDIG achieves best performance in terms of generation number, generation speed, and convergence speed. Therefore, we set the decay factor η to 0.5, i.e., past gradient information decays away to half its original impact after each iteration. We also set the maximum iterations *max_iter* to 10, since we find that less than 5 iterations need to be taken for most of the seeds if an individual discriminatory instance can be identified around them. During local generation, Fig. 5 shows the average effect of the gradient update interval on all 10 benchmarks given 100 discriminatory seeds for each benchmark. When the update interval increases, the number of generated discriminatory instances decreases slightly, but the generation speed grows rapidly. We empirically set *update_interval* to 5 and ∞, and the corresponding methods are referred to as EIDIG-5 and EIDIG-∞ respectively. The step sizes of perturbation *s_g* and *s_l* are fixed to 1.0, i.e., the minimum step for the categorical attributes. Additionally, we implement EIDIG-G that improves ADF only by utilizing the gradient of the model output as guidance for a comprehensive comparison.

All experiments were run on a personal computer with 16 GB RAM, AMD Ryzen 7 3700X 3.60GHz CPU and NVIDIA GTX 1650super GPU in Ubuntu20.10, and we take the average on 5 rounds for all empirical results below.

4.2 Results and Discussion

We conduct a comprehensive comparison between EIDIG and ADF. Notice that both ADF and EIDIG comprise a global generation phase and a local generation phase. To this end, we also compare them phase by phase other than a complete comparison to give a close investigation.

RQ1(Effectiveness and Efficiency). We select 1000 seeds during global generation and then generate 1000 instances in the vicinity of each individual discriminatory instance successfully identified before during local generation (i.e., $g\_num = l\_num = 1000$). Note that there are only 600 instances in the training set of the German Credit dataset, so we use the whole training set as the seeds for global generation. The comparison results are shown in Fig. 6 and Fig. 7 labeled with 'Two phases'. Fig. 6 shows the number of individual discriminatory instances generated by each approach. Fig. 7 shows the number of individual discriminatory instances generated per second. We find that both EIDIG-5 and EIDIG-∞ stably outperform ADF w.r.t. the number and speed of individual discrimination generation.

We first show effectiveness of EIDIG. On average, EIDIG-5 (EIDIG-∞) explores 24.75% (19.74%) larger search space than ADF when the

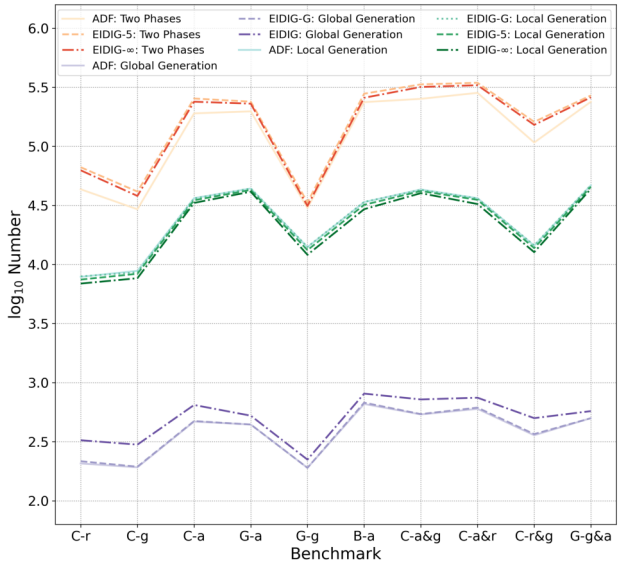


Figure 6: Number of generated individual discriminatory instances.

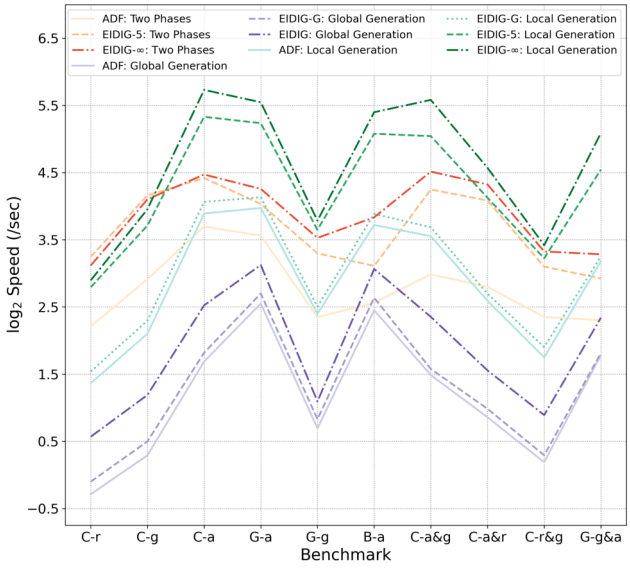


Figure 7: Speed of generating individual discriminatory instances.

maximum number of search attempts is fixed. Specifically, EIDIG-5 (EIDIG-∞) generates 25.78% (19.11%) more individual discriminatory instances than ADF on average. We also study the ratio of found discriminatory instances to seed inputs ($g\_num = l\_num = 100$, averaged on 10 benchmarks). As shown in Fig. 8, Both ADF and EIDIG generate instances at an almost linear trend, and EIDIG clearly generates more individual discriminatory instances than ADF given a set of seeds.

Another improvement we have made is a significant increase in the efficiency of individual discrimination generation due to the

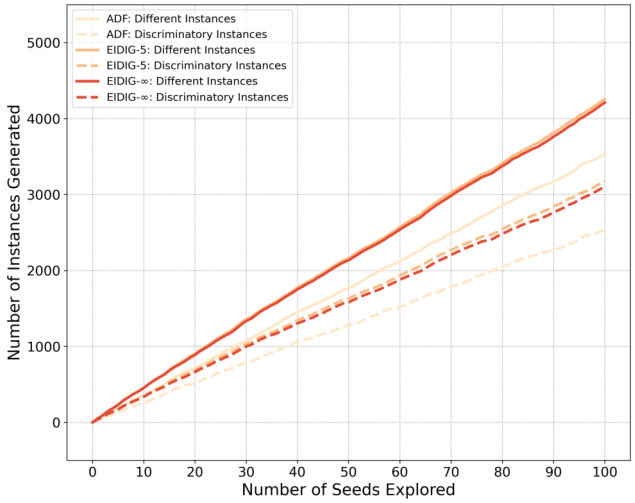


Figure 8: Effectiveness comparison with ADF.

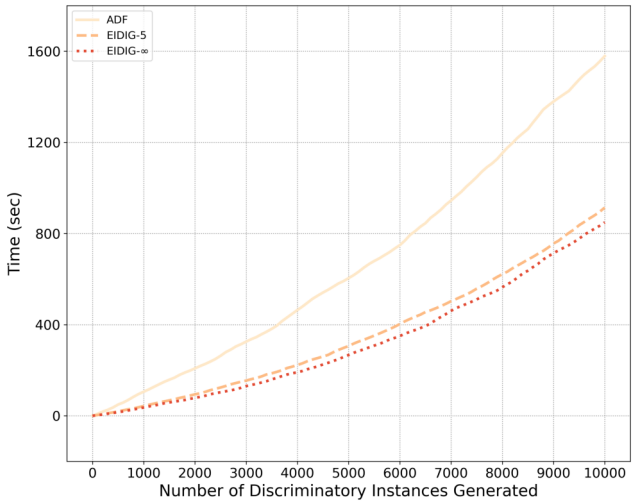


Figure 9: Efficiency comparison with ADF.

absence of the backpropagation through the loss function, faster convergence in the global search, and fewer counts of gradient calculation needed in the local search. In our experiments, EIDIG-5 (EIDIG-∞) achieves an average speedup of 81.29% (121.49%) for generating individual discriminatory instances when compared with ADF. Fig. 9 shows the time consumption for generating 10000 individual discriminatory instances averaged on 10 benchmarks. both EIDIG-5 and EIDIG-∞ take much less time than ADF to generate the same number of individual discriminatory instances.

We conclude that EIDIG significantly outperforms ADF both in terms of effectiveness and efficiency.

RQ2(Guidance). For global generation, we sample 1000 seeds (600 for German Credit dataset) from the clustered training set in a round-robin fashion. We use the same set of seeds for ADF, EIDIG-G, and EIDIG. The lines labeled with 'Global Generation' in Fig. 6 and Fig. 7 show the comparison results for global generation. Compared

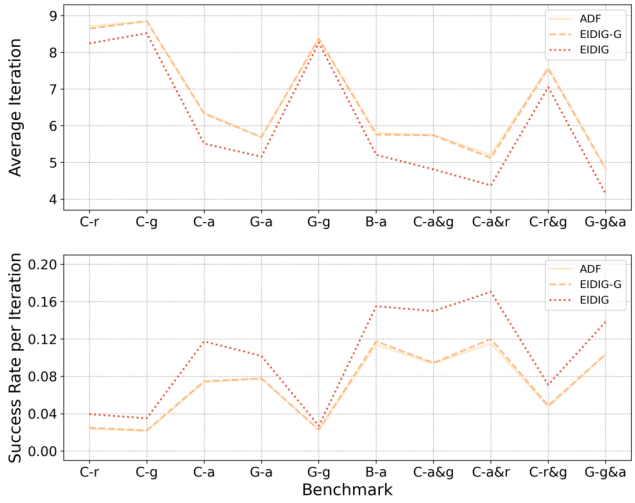


Figure 10: Convergence speed for global generation.

with ADF, EIDIG-G generates 1.42% more individual discriminatory instances and achieves a speedup of 9.73% on average.

For local generation, we generate 100 diverse individual discriminatory instances as the seed inputs for ADF, EIDIG-G, EIDIG-5, and EIDIG-∞. The experimental results for local generation are also shown in Fig. 6 and Fig. 7. We find that with *l_num* set to 1000, EIDIG-G generates almost the same individual discriminatory instances, and achieves a speedup of 10.56% on average when compared with ADF.

The more direct and precise guidance speeds up both the global generation phase and the local generation phase, which is one of the factors accounting for efficiency of EIDIG.

We conclude that utilizing the gradient of the model output as guidance improves the speed of generating individual discriminatory instances.

RQ3(Global Generation). As shown in Fig. 6 and Fig. 7, EIDIG further improves EIDIG-G in terms of the number and speed of individual discrimination generation during global generation. After introducing the momentum term into global generation, EIDIG generates 29.15% more individual discriminatory instances and accelerates the iterative process of finding individual discriminatory instances by 59.46% on average when compared with ADF. To further understand the improvements from momentum, we show the average iteration for each seed input and the average number of individual discriminatory instances generated at each iteration with *max_iter* fixed to 10 in Fig. 10. We find that ADF and EIDIG-G share almost the same results w.r.t. average iteration, so utilizing the gradient of the model output as guidance does not boost the convergence speed for iterative search. However, EIDIG takes 8.72% fewer iterations and have a 44.79% higher success rate of identifying a new individual discriminatory instance at each iteration on average when compared with ADF. Thus, the integration of momentum enables EIDIG to markedly improve the ability to generate individual discriminatory instances on the basis of EIDIG-G. By means of momentum, EIDIG successfully generates individual discriminatory instances around some seeds that ADF fails to handle. Recall

Table 2: Fairness improvement with retraining

Benchmark	Before(%)	After(%)		
		ADF	EIDIG-5	EIDIG-∞
C-r	9.33±0.06	2.61±0.03	**1.00±0.02**	1.24±0.02
C-g	3.81±0.04	1.24±0.02	**0.89±0.02**	1.27±0.02
C-a	11.78±0.07	4.45±0.03	**2.28±0.03**	2.64±0.03
G-a	28.70±0.08	3.93±0.04	4.20±0.04	**3.64±0.04**
G-g	10.23±0.05	3.64±0.04	3.41±0.03	**3.21±0.04**
B-a	10.36±0.06	3.73±0.04	**2.83±0.03**	3.22±0.04
C-a&g	15.54±0.07	5.70±0.04	**3.18±0.03**	3.90±0.03
C-a&r	21.20±0.08	7.18±0.04	**3.31±0.03**	3.92±0.04
C-r&g	13.16±0.07	3.87±0.03	**1.90±0.03**	2.51±0.03
G-g&a	38.42±0.08	7.58±0.06	7.63±0.05	**6.90±0.06**

that the goal of global generation is to rapidly generate a set of diverse individual discriminatory instances, and EIDIG significantly outperforms ADF in this context.

We conclude that the momentum term stabilizes the update direction, accelerates the iterative search, and increases the success rate of identifying a new individual discriminatory instance during global generation.

RQ4(Local Generation). The experimental results of local generation given 100 discriminatory seeds are shown in Fig. 6 and Fig. 7. EIDIG-5 (EIDIG-∞) generates 3.71% (9.16%) fewer individual discriminatory instances, but achieves a speedup of 164.12% (242.89%) on average when compared with ADF, which means that EIDIG further achieves a significant speedup with a limited loss in the number of instances generated during local generation. Recall that EIDIG-5 recalculate gradients and attribute contributions every 5 iterations, and EIDIG-∞ never recalculate them. Reducing frequency of gradient calculation notably accelerates our algorithms while still enabling EIDIG to generate abundant discriminatory instances due to the heavy correlation between the guidance information of successive iterations.

We conclude that reducing frequency of gradient calculation during local generation enables EIDIG to find abundant individual discriminatory instances with a significant speedup.

RQ5(Usefulness). The next step is to mitigate bias for the original model. It is shown that a neural network gets more robust to adversarial examples after being trained on a mixture of clean and adversarial examples [47, 48]. Similarly, we also leverage the individual discriminatory instances generated before to mitigate bias in the model. To utilize the generated individual discriminatory instances, we need to attach more fair labels to them first. Hence, we train five common models as an ensemble model, including k-Nearest Neighbors [18], Multilayer Perceptron [25], Support Vector Machine [9], Random Forest [26], and Gaussian Naive Bayes [25]. As [51] did, we randomly sample 5% of the individual discriminatory instances and then relabel them with majority voting [33]. Afterward, we add these instances to the original training set and then retrain the model under test on the augmented training set. The experimental results of retraining are listed in Table 2, where columns 'Before' and 'After' are the statistical estimation of the percentage of discriminatory inputs in the input domain before and after retraining. To evaluate the model bias, we randomly sample 10000 instances to estimate the proportion of individual discriminatory instances for 100 rounds, and also compute the 95% confidence interval. It is found that EIDIG-5 (EIDIG-∞) achieves a fairness improvement of 81.15% (80.03%) versus 72.97% for ADF. Additionally, we record the accuracy and F1-score [23] of retrained models on the original test set in Table 1. We find that the accuracy loss of the models under test is very limited after retraining.

We conclude that the individual discriminatory instances generated by EIDIG are useful for bias mitigation through retraining.

5 RELATED WORK

In this section, we review the evolution in related domains and position our work on fairness testing.

Fair Machine Learning. Numerous attempts have been targeted to combatting bias in machine learning. These methods fall under three categories, which are pre-processing, in-processing, and post-processing respectively [37]. Pre-processing techniques try to remove the bias from the data before training [7, 10]. In-processing techniques achieve fairness by regularizing the model or modifying the learning procedure [4, 29]. Post-processing calibrates the model's prediction during inference by enforcing prediction distribution to approach either the training distribution or a specific fairness metric [14, 41]. Our work is complementary to these approaches for bias detection and mitigation.

Fairness Testing. Bias in software is common even when fairness is an explicit goal during design, and thus fairness testing is a critical step before deployment [20, 50]. Galhotra *et al.* [20] formally defined software fairness testing, introduced a causality-based measure of the algorithmic discrimination, and also proposed Themis to perform these measurements by randomly sampling test suites. Although Themis is a black-box approach, it is ineffective due to the absence of any systematic test case generation technique. Unlike Themis, Udeshi *et al.* [50] systematically designed a directed test generation module AEQUITAS to generate individual discriminatory instances for machine learning models. During global search, AEQUITAS also randomly samples from the input domain to identify a set of individual discriminatory instances. During local search, AEQUITAS takes the discriminatory inputs generated before as seeds and searches the neighborhood of them with the guidance of a global adaptive probability distribution, which represents the likelihood of successfully generating an individual discriminatory instance by perturbing a non-protected attribute. However, the global probability distribution maintained by AEQUITAS hardly applies to all specific inputs, since the contribution of each non-protected attribute to the model inference can vary for different inputs. Agarwal *et al.* [2] presented Symbolic Generation (SG) to find individual discrimination. SG combines local explanation that constructs a linear model (e.g., decision tree) to approximate the model under test with symbolic execution that aims to systematically explore the input space. Therefore, SG is relatively heavy when compared with Themis and AEQUITAS, because SG is largely

dependent on the development of local model explainers and symbolic execution solvers, which makes SG less scalable. Zhang *et al.* [51] proposed Adversarial Discrimination Finder (ADF) to generate individual discriminatory instances through adversarial sampling with a two-phase generation framework. ADF utilizes the gradient of the loss function as guidance for perturbation. During global generation, ADF rapidly discovers some individual discriminatory instances by iteratively perturbing the seed inputs towards the decision boundary. During local generation, ADF searches the vicinity of each found discriminatory input according to an input-specific probability distribution. The non-protected attributes with a smaller impact on the model output correspond to a higher probability of being chosen for perturbation.

Our approach EIDIG inherits and improves ADF via constructing a more direct mapping from input perturbations to output variations, integrating momentum into the global search, and fully exploiting prior information in the local search. The direct and precise mapping effectively reduces computation cost for each search iteration, the introduction of momentum improves the success rate of identifying individual discriminatory instances during global generation, and the full exploitation of prior information significantly decreases frequency of gradient calculation during local generation.

There are also some recent studies targeted at fairness testing for high-dimensional data, such as images or texts. Tian *et al.* [49] defined NAPVD (Neuron Activation Probability Vector Distance) and identified class-based confusion and bias errors in DNN-driven image classification software based on NAPVD. Ma *et al.* [36] employed metamorphic testing to test NLP models for fairness violation. Soremekun *et al.* [44] leveraged context-free grammars to generate discriminatory inputs for NLP models.

Adversarial Attacks. Apart from the gradient-based methods reviewed in Section 2.2, there are also other methods (e.g., optimization-based methods) for generating adversarial examples. Szegedy *et al.* [47] used an optimization method L-BFGS to optimize the distance between adversarial examples and real examples. C&W Attack [8] formulates generating adversarial examples as image distance minimization problem and solves it with the optimizer Adam [30]. Moosavi-Dezfooli *et al.* [38] proposed DeepFool to iteratively project the original input to the linear approximation of the decision boundary of adversarial examples. Adversarial attacks focus on test case generation for testing the robustness and safety of DNNs, while fairness testing aims to generate test suites for testing the fairness of DNNs. Therefore, these two research domains are highly correlated. In the future, more novel adversarial attack methods can be introduced into fairness testing.

6 CONCLUSION

In this paper, we propose an effective and efficient algorithm EIDIG for discovering individual discrimination in differentiable models through gradient-based heuristic search. EIDIG utilizes the gradient of the model output as powerful guidance for the two-phase search. During global generation, we rapidly identify a set of diverse individual discriminatory instances by iteratively perturbing the potential individual discriminatory instance pairs towards the decision boundary, and we boost the iterative search with momentum. During local generation, we exploit the robustness of well-trained deep neural networks and search the neighborhood of the found discriminatory seeds for more individual discriminatory instances with a low frequency of gradient calculation. Our experimental results show that EIDIG achieves state-of-the-art performance for individual discrimination generation.

ACKNOWLEDGMENTS

This work is funded by National Key R&D Program of China (2020AAA0107800) and NSFC Project (No. 61672012).

REFERENCES

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. 2016. Tensorflow: A system for large-scale machine learning. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation* (Savannah, GA, USA) *(OSDI'16)*. USENIX Association, USA, 265–283. https://dl.acm.org/doi/10.5555/3026877.3026899

[2] Aniya Aggarwal, Pranay Lohia, Seema Nagar, Kuntal Dey, and Diptikalyan Saha. 2019. Black box fairness testing of machine learning models. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 625–635. https://doi.org/10.1145/3338906.3338937

[3] Julia Angwin, Jeff Larson, Surya Mattu, and Lauren Kirchner. 2016. Machine bias: there's software used across the country to predict future criminals. and it's biased against blacks. propublica 2016. https://www.propublica.org/article/machine-bias-risk-assessments-in-criminal-sentencing

[4] Richard Berk, Hoda Heidari, Shahin Jabbari, Matthew Joseph, Michael Kearns, Jamie Morgenstern, Seth Neel, and Aaron Roth. 2017. A convex framework for fair regression. *arXiv preprint arXiv:1706.02409* (2017). https://arxiv.org/abs/1706.02409

[5] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. 2016. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316* (2016). https://arxiv.org/abs/1604.07316

[6] Joy Buolamwini and Timnit Gebru. 2018. Gender shades: Intersectional accuracy disparities in commercial gender classification. In *Conference on fairness, accountability and transparency*. PMLR, 77–91. http://proceedings.mlr.press/v81/buolamwini18a.html

[7] Flavio P Calmon, Dennis Wei, Bhanukiran Vinzamuri, Karthikeyan Natesan Ramamurthy, and Kush R Varshney. 2017. Optimized pre-processing for discrimination prevention. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*. 3995–4004. https://dl.acm.org/doi/10.5555/3294996.3295155

[8] Nicholas Carlini and David Wagner. 2017. Towards evaluating the robustness of neural networks. In *2017 ieee symposium on security and privacy (sp)*. IEEE, 39–57. https://doi.org/10.1109/SP.2017.49

[9] Corinna Cortes and Vladimir N Vapnik. 1995. Support-Vector Networks. *Machine Learning* 20, 3 (1995), 273–297. https://doi.org/10.1023/A:1022627411411

[10] Brian d'Alessandro, Cathy O'Neil, and Tom LaGatta. 2017. Conscientious classification: A data scientist's guide to discrimination-aware classification. *Big data* 5, 2 (2017), 120–134. https://doi.org/10.1089/big.2016.0048

[11] Jeffrey Dastin. 2018. Amazon scraps secret AI recruiting tool that showed bias against women. *San Fransico, CA: Reuters. Retrieved on October* 9 (2018), 2018. https://www.reuters.com/article/us-amazon-com-jobs-automation-insight-idUSKCN1MK08G

[12] Arthur P Dempster, Nan M Laird, and Donald B Rubin. 1977. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)* 39, 1 (1977), 1–22. https://doi.org/10.1111/j.2517-6161.1977.tb01600.x

[13] Yinpeng Dong, Fangzhou Liao, Tianyu Pang, Hang Su, Jun Zhu, Xiaolin Hu, and Jianguo Li. 2018. Boosting adversarial attacks with momentum. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 9185–9193. https://doi.org/10.1109/CVPR.2018.00957

[14] Mengnan Du, Fan Yang, Na Zou, and Xia Hu. 2020. Fairness in deep learning: A computational perspective. *IEEE Intelligent Systems* (2020). https://doi.org/10.1109/MIS.2020.3000681

[15] Włodzisław Duch and Jerzy Korczak. 1998. Optimization and global minimization methods suitable for neural networks. *Neural computing surveys* 2 (1998), 163–212.

[16] Cynthia Dwork, Moritz Hardt, Toniann Pitassi, Omer Reingold, and Richard Zemel. 2012. Fairness through awareness. In *Proceedings of the 3rd innovations in*

theoretical computer science conference. 214–226. https://doi.org/10.1145/2090236.2090255

[17] Alhussein Fawzi, Omar Fawzi, and Pascal Frossard. 2018. Analysis of classifiers' robustness to adversarial perturbations. *Machine Learning* 107, 3 (2018), 481–508. https://doi.org/10.1007/s10994-017-5663-3

[18] Evelyn Fix and Joseph L Hodges. 1951. Discriminatory analysis: nonparametric discrimination: consistency properties. *Report No.4, USAF School of Aviation Medicine, Randolph Field, Texas, Feb* (1951). https://doi.org/10.2307/1403797

[19] Kang Fu, Dawei Cheng, Yi Tu, and Liqing Zhang. 2016. Credit card fraud detection using convolutional neural networks. In *International Conference on Neural Information Processing*. Springer, 483–490. https://doi.org/10.1007/978-3-319-46675-0_53

[20] Sainyam Galhotra, Yuriy Brun, and Alexandra Meliou. 2017. Fairness testing: testing software for discrimination. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*. 498–510. https://doi.org/10.1145/3106237.3106277

[21] Noah J Goodall. 2016. Can you program ethics into a self-driving car? *IEEE Spectrum* 53, 6 (2016), 28–58. https://doi.org/10.1109/MSPEC.2016.7473149

[22] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. 2014. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572* (2014). https://arxiv.org/abs/1412.6572

[23] Cyril Goutte and Eric Gaussier. 2005. A probabilistic interpretation of precision, recall and F-score, with implication for evaluation. In *European conference on information retrieval*. Springer, 345–359.

[24] Moritz Hardt, Eric Price, and Nathan Srebro. 2016. Equality of opportunity in supervised learning. In *Proceedings of the 30th International Conference on Neural Information Processing Systems* (Barcelona, Spain) *(NIPS'16)*. Curran Associates Inc., Red Hook, NY, USA, 3323–3331. https://dl.acm.org/doi/10.5555/3157382.3157469

[25] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. 2008. The Elements of Statistical Learning: Data Mining, Inference, and Prediction. (2008). https://doi.org/10.1007/BF02985802

[26] Tin Kam Ho. 1998. The random subspace method for constructing decision forests. *IEEE Transactions on Pattern Analysis & Machine Intelligence* 20, 8 (1998), 832–844. https://doi.org/10.1109/34.709601

[27] Andrew Ilyas, Logan Engstrom, and Aleksander Madry. 2018. Prior convictions: Black-box adversarial attacks with bandits and priors. *arXiv preprint arXiv:1807.07978* (2018). https://arxiv.org/abs/1807.07978

[28] Kyle D Julian, Jessica Lopez, Jeffrey S Brush, Michael P Owen, and Mykel J Kochenderfer. 2016. Policy compression for aircraft collision avoidance systems. In *2016 IEEE/AIAA 35th Digital Avionics Systems Conference (DASC)*. IEEE, 1–10. https://doi.org/10.1109/DASC.2016.7778091

[29] Toshihiro Kamishima, Shotaro Akaho, Hideki Asoh, and Jun Sakuma. 2012. Fairness-aware classifier with prejudice remover regularizer. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 35–50. https://doi.org/10.1007/978-3-642-33486-3_3

[30] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014). https://arxiv.org/abs/1412.6980

[31] Alexey Kurakin, Ian Goodfellow, Samy Bengio, et al. 2016. Adversarial examples in the physical world. arXiv:1607.02533 [cs.CV] https://arxiv.org/abs/1607.02533

[32] Matt J Kusner, Joshua R Loftus, Chris Russell, and Ricardo Silva. 2017. Counterfactual fairness. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*. 4069–4079. https://dl.acm.org/doi/10.5555/3294996.3295162

[33] Louisa Lam and SY Suen. 1997. Application of majority voting to pattern recognition: an analysis of its behavior and performance. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans* 27, 5 (1997), 553–568. https://doi.org/10.1109/3468.618255

[34] Geert Litjens, Thijs Kooi, Babak Ehteshami Bejnordi, Arnaud Arindra Adiyoso Setio, Francesco Ciompi, Mohsen Ghafoorian, Jeroen Awm Van Der Laak, Bram Van Ginneken, and Clara I Sánchez. 2017. A survey on deep learning in medical image analysis. *Medical image analysis* 42 (2017), 60–88. https://doi.org/10.1016/j.media.2017.07.005

[35] Stuart Lloyd. 1982. Least squares quantization in PCM. *IEEE transactions on information theory* 28, 2 (1982), 129–137. https://doi.org/10.1109/TIT.1982.1056489

[36] Pingchuan Ma, Shuai Wang, and Jin Liu. 2020. Metamorphic testing and certified mitigation of fairness violations in nlp models. In *Proceedings of the 29th International Joint Conference on Artificial Intelligence (IJCAI). 458–465*. https://doi.org/10.24963/ijcai.2020/64

[37] Ninareh Mehrabi, Fred Morstatter, Nripsuta Saxena, Kristina Lerman, and Aram Galstyan. 2019. A survey on bias and fairness in machine learning. *arXiv preprint arXiv:1908.09635* (2019). https://arxiv.org/abs/1908.09635

[38] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. 2016. Deepfool: a simple and accurate method to fool deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2574–2582. https://doi.org/10.1109/CVPR.2016.282

[39] Vinod Nair and Geoffrey E Hinton. 2010. Rectified linear units improve restricted boltzmann machines. In *Icml*. https://dl.acm.org/doi/10.5555/3104322.3104425

[40] Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z Berkay Celik, and Ananthram Swami. 2016. The limitations of deep learning in adversarial settings. In *2016 IEEE European symposium on security and privacy (EuroS&P)*. IEEE, 372–387. https://doi.org/10.1109/EuroSP.2016.36

[41] Geoff Pleiss, Manish Raghavan, Felix Wu, Jon Kleinberg, and Kilian Q Weinberger. 2017. On fairness and calibration. (2017), 5684–5693. https://dl.acm.org/doi/10.5555/3295222.3295319

[42] Boris T Polyak. 1964. Some methods of speeding up the convergence of iteration methods. *Ussr computational mathematics and mathematical physics* 4, 5 (1964), 1–17. https://doi.org/10.1016/0041-5553(64)90137-5

[43] Nripsuta Ani Saxena, Karen Huang, Evan DeFilippis, Goran Radanovic, David C Parkes, and Yang Liu. 2019. How do fairness definitions fare? Examining public attitudes towards algorithmic definitions of fairness. In *Proceedings of the 2019 AAAI/ACM Conference on AI, Ethics, and Society*. 99–106. https://doi.org/10.1145/3306618.3314248

[44] Ezekiel Soremekun, Sakshi Udeshi, and Sudipta Chattopadhyay. 2020. Astraea: Grammar-based Fairness Testing. *arXiv preprint arXiv:2010.02542* (2020). https://arxiv.org/abs/2010.02542

[45] C. Spearman. 1904. General intelligence, objectively determined and measured. *The American Journal of Psychology* 15, 2 (1904), 201–292. https://doi.org/10.2307/1412107

[46] Earl William Swokowski. 1979. *Calculus with analytic geometry*. Taylor & Francis.

[47] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. 2013. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199* (2013). https://arxiv.org/abs/1312.6199

[48] Yuchi Tian, Kexin Pei, Suman Jana, and Baishakhi Ray. 2018. Deeptest: Automated testing of deep-neural-network-driven autonomous cars. In *Proceedings of the 40th international conference on software engineering*. 303–314. https://doi.org/10.1145/3180155.3180220

[49] Yuchi Tian, Ziyuan Zhong, Vicente Ordonez, Gail Kaiser, and Baishakhi Ray. 2020. Testing DNN image classifiers for confusion & bias errors. In *ICSE '20: 42nd International Conference on Software Engineering*. https://doi.org/10.1145/3377811.3380400

[50] Sakshi Udeshi, Pryanshu Arora, and Sudipta Chattopadhyay. 2018. Automated directed fairness testing. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*. 98–108.

[51] Peixin Zhang, Jingyi Wang, Jun Sun, Guoliang Dong, Xinyu Wang, Xingen Wang, Jin Song Dong, and Dai Ting. 2020. White-box fairness testing through adversarial sampling. In *International Conference on Software Engineering (ICSE 2020), Seoul, South Korea*. 23–29. https://doi.org/10.1145/3377811.3380331