Efficient Fairness Testing Through Hash-Based Sampling

Zhenjiang Zhao[1(✉)], Takahisa Toda[1], and Takashi Kitamura[2]

[1] Graduate School of Informatics and Engineering,
University of Electro-Communications, Tokyo, Japan
{zhenjiang,toda}@disc.lab.uec.ac.jp
[2] National Institute of Advanced Industrial Science and Technology (AIST),
Tokyo, Japan
t.kitamura@aist.go.jp

Abstract. There is a growing concern on algorithm fairness, according to wider adoption of machine learning techniques in our daily life. Testing of individual fairness is an approach to algorithm fairness concern. Verification Based Testing (VBT) is a state-of-the-art testing technique for individual fairness, that leverages verification techniques using constraint solving. In this paper, we develop a black-box individual fairness testing technique VBT-X, which applies hash-based sampling techniques to the test case generation part of VBT, aiming to improve its testing ability. Our evaluation by experiments confirms that VBT-X improves the testing ability of VBT by 2.92 times in average.

Keywords: Algorithm fairness · Fairness testing · SAT/SMT solving

1 Introduction

Decision making algorithms based on machine learning (ML) have been more widely adopted in our daily life, in e.g., criminal sentencing [13], financial and insurance [2], hiring [11], (see [19], for more examples). Such algorithmic decision making can overcome some limitations of human decision making, however, there is also a growing concern on fairness of such algorithms, since they tend to be biased, unfairly treating individuals based on sensitive attributes, such as race, gender, and age. For example, COMPAS algorithm, which predicts future criminal, used to determine criminal sentencing, is known to be biased against black defendants [13].

Testing of *individual fairness* is an approach to algorithm fairness concern. *Individual fairness* is a concept of algorithm fairness, which states that an ML classifier should give similar prediction to similar individuals [7]. Testing of individual fairness aims to detect data that violate the concept (called, *discriminatory data*), contained in the given ML classifier under test (CUT). The subject has been studied extensively in previous years, which renders a variety of testing techniques e.g., [1,8,16,23–25,27,28]. These testing techniques respectively use their own search algorithms to generate a set of test cases (i.e., a test set),

M. Papadakis and S. R. Vergilio (Eds.): SSBSE 2022, LNCS 13711, pp. 35–50, 2022.
https://doi.org/10.1007/978-3-031-21251-2_3

which can effectively detect discriminatory data, from the huge input space of the given CUT.

Verification Based Testing (VBT) [24,25], recently developed by Sharma and Wehrheim, is a state-of-the-art black-box testing technique for individual fairness. While VBT detects the presence of discriminatory data in a given CUT, its basic mechanism internally builds a decision tree (DT) classifier represented in SMT (Satisfiability Modulo Theory) constraints as an approximation classifier of CUT, and generates test cases applying SMT solving to the constraints. In the mechanism, a key technical challenge lies on the test generation part, since a technique is required to efficiently search a test set to effectively detect discriminatory data, given the SMT represented DT classifier. VBT proposes two test search techniques, called *data pruning* and *branch pruning*. The more elaborated one, i.e., *branch pruning*, tries to generate diverse test cases by traversing the (SMT-represented) DT, using repetitive calls of SMT solver.

In this paper, we develop an individual fairness testing technique, named VBT-X, by applying the hash-based sampling [3–5,9] in the test generation part of VBT. The hash-based sampling techniques, given a logical formula ϕ, generate diverse solutions of ϕ. Its advantage is the ability to sample diverse solutions at a reasonable computational cost. The techniques have been studied actively, with applications such as probabilistic inference [22], network reliability estimation [6], and verification [18]. Our aim is to leverage its diverse sampling ability in the test generation (i.e., test search) part of VBT, to improve testing ability of VBT. We also devise several enhancement techniques to improve efficiency of VBT-X. Our evaluation confirms that VBT-X achieves a higher testing ability than VBT by 2.92 times in average.

This paper is organized as follows: Sect. 2 reviews the concept of individual fairness, the algorithm of VBT and the hash-based sampling. In Sect. 3, we explain the basic approach of our proposed technique, as Basic VBT-X, and introduce several enhancements to Basic VBT-X, proposing VBT-X. Section 4 is devoted to evaluation of VBT-X by experiments. We discuss related studies in Sect. 5, and mention validity threats of this study in Sect. 6. Section 7 concludes this paper, discussing future work also.

2 Background

This section reviews individual fairness testing (referring to [16]), VBT [27] and hash-based sampling [3–5,9].

2.1 Individual Fairness Testing

Let $P = \{p_1, p_2, \cdots, p_n\}$ be a set of *attributes* (or *parameters*), for $n \in \mathbb{N}$. We use p_i to indicate the i-th attribute in P. Each attribute $p_i \in P$ is associated with a set of *values*, called the *domain* of p_i, denoted by $Dom(p_i)$, such that $(Dom(p_i))_{i \in n}$ is pairwise disjoint. The input space $\mathbb{I}$ of a set of attributes P is the Cartesian product of the domains of $p_1, p_2 \cdots p_n (\in P)$, i.e., $\mathbb{I} = Dom(p_1) \times$

Algorithm 1: VBT algorithm

Data: Classifier (f), Iteration limit ($limit$)
Result: Discriminatory data set (D_{disc})
Step-0: Make a training dataset D_{train} with randomly generated data;
repeat
  Step-1: Make an approximation f' of CUT f by training a decision tree classifier with D_{train} ;
  Step-2: Construct SMT constraints $\phi_{f'}$ from approximation f' ;
  Step-3: Generate test cases by SMT solver;
  Step-4: Execute test cases against CUT f, to detect discriminatory data;
  Step-5: Update the training dataset D_{train} with failing test cases;
until *Iteration exceeds limit* ;

$Dom(p_2) \times \cdots \times Dom(p_n)$. An element I of $\mathbb{I}$ is called a *data item* or *data instance*. We also introduce $P_{prot} \subseteq P$ as the set of protected attributes (e.g., gender, race, age). An ML classifier, whose input space is $\mathbb{I}$, is a function f such that $f(I)$ is the output (i.e., decision) that the classifier f makes for input I.

Definition 1 (Individual discriminatory data and Fairness [27]). *Let ϕ be a classifier under test (CUT), γ be the pre-determined threshold (e.g. chosen by the user), and $I, I' \in \mathbb{I}$. Assume that there exists a non-empty set $Q \subseteq P_{prot}$ s.t. for all $q \in Q$, $I_q \neq I'_q$ and for all $p \in P \backslash Q$, $I_p = I'_p$. If $|f(I) - f(I')| > \gamma$, then I (also I') is called a discriminatory data item of the classifier f, as an instance that manifests the violation of (individual) fairness in f.*

Example 1. Consider an ML classifier f that, taking an individual as input, predicts if the individual gets a loan. Individuals are schemed by three attributes of 'gender', 'income', and 'age', and suppose 'gender' is the protected attribute. Consider the following two individuals I_1 and I_2 that differ only in the protected attribute:

$$I_1 : (gender = male, \ \ income = 1000, \ age = 40) \quad (1)$$
$$I_2 : (gender = female, income = 1000, \ age = 40) \quad (2)$$

Suppose the classifier f gives 1 (Yes) to individual I_1, and 0 (No) to I_2; i.e., $f(I_1) = 1$ and $f(I_2) = 0$. Since we have $|f(I_1) - f(I_2)| > \gamma$ assuming $\gamma = 0$, I_1 (and I_2) is a discriminatory data item.

2.2 Verification Based Testing (VBT)

We briefly review the algorithm of VBT, shown in Algorithm 1. VBT takes the classifier under test (CUT) f, and outputs discriminatory data. Details of the internal mechanism are given by steps as follows:

Step-0: Make a training dataset D_{train} with randomly generated data. This step is executed once at the beginning. The input data instances in D_{train} are generated randomly, and their output labels are obtained by feeding them to CUT f.

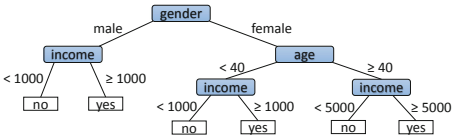

Fig. 1. A decision tree for predicting who gets a loan

Step-1: Make an approximation f' of CUT f by training a decision tree (DT) classifier with D_{train}. For the training in the first iteration, the data set D_{train} created in Step-0 is used. From the second iteration, D_{train} updated in Step-5 is used, where training works as re-training of the approximation f' for refinement. Figure 1 shows an example trained DT (i.e., approximation f').

Step-2: Construct SMT constraints $\phi_{f'}$ from approximation f'. The construction of SMT constraints is designed to check the following: "*Does a discriminatory data instance exist in the given DT?*"

The construction first prepares two variable sequences $x_1^1 \cdots x_1^n$ and $x_2^1 \cdots x_2^n$, where n is the number of attributes and denoted by x_1 and x_2. They express two persons (person 1 and 2) as value assignments for the n variables. Using such variables, the two constraint components '*Unfair*' and '*DecTree*' are built.

The component '*Unfair*' is to check if two persons (x_1, x_2) that are identical except for the protected attribute have different classifier outcomes as follows, where $class_i$ represents the classifier output for individual i:

$$Unfair := \bigwedge_{p \in P \setminus P_{prot}} (x_1^p = x_2^p) \;\&\; \left(\bigvee_{p \in P_{prot}} (x_1^p \neq x_2^p) \right) \;\&\; (class_1 \neq class_2),$$

The component '*DecTree*' specifies that the two persons (x_1, x_2) and classifier's outcomes should conform to the approximation f'. The approximation f' is thus encoded into SMT constrains as follows:

$$DecTree_i(DT) := \bigwedge_{\pi:\ \text{path}} \left(\bigwedge_{1 \leq k < |\pi|} \pi.branch\,(k) \right) \Rightarrow \pi.leaf,$$

where π in the outer conjunction runs over all paths of DT; each conjunct is a predicate of implication form; for the k-th branch node of π, we denote by $\pi.branch(k)$ the (in)equality formula relating the value on an edge to the attribute on the k-th branch node, and by $\pi.leaf$ the (in)equality formula relating the value in the leaf node to the output label.

DT is encoded by conjoining each constraint of two individuals; i.e. $DecTree := DecTree_1 \;\&\; DecTree_2$. The constraint formula $\phi_{f'}$ is constructed as $\phi_{f'} := DecTree \wedge Unfair$. For example, the *DecTree* and *Unfair* constructed from Fig. 1 are respectively expressed in line 1–14 and line 15–17 in Fig. 2.

We can obtain a discriminatory data instance by solving the constraints $\phi_{f'}$; i.e., there exists a discriminatory data instance if the constraint is satisfiable,

$$
\begin{array}{ll}
1 & DecTree_1: \\
2 & (gender_1 = 0)\ \&\ (income_1 < 1000) \Rightarrow (class_1 = 0) \\
3 & (gender_1 = 0)\ \&\ (income_1 >= 1000) \Rightarrow (class_1 = 1) \\
4 & (gender_1 = 1)\ \&\ (age_1 < 40)\ \&\ (income_1 < 1000) \Rightarrow (class_1 = 0) \\
5 & (gender_1 = 1)\ \&\ (age_1 < 40)\ \&\ (income_1 >= 1000) \Rightarrow (class_1 = 1) \\
6 & (gender_1 = 1)\ \&\ (age_1 >= 40)\ \&\ (income_1 < 5000) \Rightarrow (class_1 = 0) \\
7 & (gender_1 = 1)\ \&\ (age_1 >= 40)\ \&\ (income_1 >= 5000) \Rightarrow (class_1 = 1) \\
8 & DecTree_2: \\
9 & (gender_2 = 0)\ \&\ (income_2 < 1000) \Rightarrow (class_2 = 0) \\
10 & (gender_2 = 0)\ \&\ (income_2 >= 1000) \Rightarrow (class_2 = 1) \\
11 & (gender_2 = 1)\ \&\ (age_2 < 40)\ \&\ (income_2 < 1000) \Rightarrow (class_2 = 0) \\
12 & (gender_2 = 1)\ \&\ (age_2 < 40)\ \&\ (income_2 >= 1000) \Rightarrow (class_2 = 1) \\
13 & (gender_2 = 1)\ \&\ (age_2 >= 40)\ \&\ (income_2 < 5000) \Rightarrow (class_2 = 0) \\
14 & (gender_2 = 1)\ \&\ (age_2 >= 40)\ \&\ (income_2 >= 5000) \Rightarrow (class_2 = 1) \\
15 & Unfair: \\
16 & (gender_1 \neq gender_2)\ \&\ (income_1 = income_2)\ \&\ (age_1 = age_2) \\
17 & (class_1 \neq class_2)
\end{array}
$$

Fig. 2. SMT formula used for test case generation

and such an instance can be retrieved from the solution, which is given as a value assignment for the variables in $\phi_{f'}$. For example, the set of the constraints in Fig. 2 is satisfiable. We can thus retrieve the value assignment for two persons (x_1, x_2) and their outcome of classifier from the solution below. Observe that x_1 (and x_2) is a discriminatory data instance for approximation classifier f'.

$$x_1 : [gender_1 = 0, income_1 = 1000, age_1 = 40, class_1 = 1]$$
$$x_2 : [gender_2 = 1, income_2 = 1000, age_2 = 40, class_2 = 0]$$

Step-3: Generate test cases by SMT solver. This step generates numerous test cases using SMT constraints $\phi_{f'}$ constructed in Step-2. VBT uses data instances satisfying $\phi_{f'}$ (i.e., discriminatory data in the approximation f') as test cases. This is based on the idea that discriminatory data in an approximation f' is likely to be one in CUT f, too.

A technical difficulty here arises on how to generate as many test cases as we want using the SMT constraints $\phi_{f'}$. VBT realizes it by two kinds of technique, called *data pruning* and *branch pruning*. The data pruning generates test cases by repeatedly solving the constraints $\phi_{f'}$, while adding blocking clauses in each iteration to block regenerating the test cases that have been generated so far. The branch pruning generates test cases by traversing paths of the DT. It generates a maximum of $2k$ test cases for a DT with k hight (i.e., k attributes) (Algorithm 3 of [25]). Appropriate clauses are added to $\phi_{f'}$ to guide traversing the DT. Test cases are generated by repeatedly solving such constraints.

Step-4: Execute test cases against CUT f to detect discriminatory data. Not all test cases (i.e., discriminatory data in the approximation f') are necessarily discriminatory data in CUT f[1]. Test cases are thus actually tested against f.

[1] Test cases are thus also called *candidates* in [24].

We distinguish success test cases, which are actually discriminatory data for f, from failing ones, which are not.
Step-5: Update the training dataset by adding failing test cases. Failing test cases represent points where the approximated classifier f' differs from CUT f. VBT accumulates such failing test cases in D_{train} for re-training the approximation f' for refinement in Step-1 in the next iteration. By repeatedly refining f', VBT more efficiently detects discriminatory data.

2.3 Hash-Based Sampling

Overview. The concept of hash-based sampling techniques of Boolean formula F is to randomly divide the input space of F (denoted by, $\{0,1\}^n$, where n is the number of variables in F) into "small cells" of roughly equal size, and to pick a solution from one such cell. The partition of the input space is virtually done by determining a random hash function $h\colon \{0,1\}^n \to \{0,1,\ldots,m-1\}$, where let m be the number of cells, so that the inverse images $h^{-1}(0),\ldots,h^{-1}(m-1)$ correspond to the partitioned cells.

A common practice to realize this is to impose random *XOR clauses* on F. Here, an XOR clause is a formula constructed from Boolean variables or constants $(0,1)$ using XOR operators. XOR clauses have effect of restricting the solution space (i.e., the set of all solutions of F) to one randomly chosen cell. Since imposing a single XOR clause on F means roughly halving the solution space of F (and selecting one of them), imposing s XOR clauses means dividing the solution space into 2^s cells of roughly equal size (and selecting one of them). A solution is sampled by applying an off-the-shelf solver to the resulted formula, i.e., the conjunction of XOR clauses and F. We repeat this procedure but with fresh XOR clauses in each repetition, to generate as many samples as we need.

Hash-Based Sampling by Gomes et al. Since the invention of the hash-based sampling by Sipser [26], a variety of techniques for it have been investigated [3–5, 9]. Among them, we review the technique XORSAMPLE by Gomes et al. [9], which captures the essence of the hash-based sampling and is easy to apply and implement in the VBT approach.

Algorithm 2: XORSAMPLE

```
   Parameter : q ∈ (0,1), a positive integer s
   Data: A satisfiable propositional formula F
   Result: A solution of F
1  while True do
2      G ← s XOR clauses, each variable chosen
         with probability q and the constant 1
         with 1/2;
3      if F & G is satisfiable then
4          σ ← GetSolution(F & G);
5          if there is no other solution then
               return σ ;
```

Algorithm 2 shows the algorithm of XORSAMPLE [9]. The steps in each iteration are: Generate XOR clauses (G) so that each clause selects each variable in F with probability q and the constant 1 with probability 1/2; Find a solution for the conjunction of F and G by applying a generic solver; If a solution σ is

found, then find another solution[2] except σ; If it is confirmed that there is no other solution, return σ, and otherwise go to the next iteration. The former case means that σ is a unique solution for $F \wedge G$, i.e., the cell is enough "small". The algorithm terminates only if this case happens.

3 Proposed Method

In this section, we develop VBT-X, a method of integrating the hash-based sampling with VBT. Its basic idea is to apply the essence of the hash-based sampling (explained in Sect. 2.3) to the test generation part (Step-3) of the VBT algorithm (Algorithm 1). The development is presented in two-steps. We first develop the basic method as 'Basic VBT-X' in Sect. 3.1, and next develop several enhancement techniques, presenting 'VBT-X' in Sect. 3.2.

3.1 Basic Method (Basic VBT-X)

Introducing Auxiliary Variables. The first difficulty we encounter in applying the hash-based sampling to VBT is that the variables in SMT constraints are inherently non-binary, i.e., their domains are often integers and real values. Take, for instance, the constraint at line 2 in Fig. 2:

$$(gender_1 = 0) \wedge (income_1 < 1000) \Rightarrow (class_1 = 0). \tag{3}$$

The variable $income_1$ is real-valued, although $gender_1$ and $class_1$ happen to be binary in the current case. In general, the input space of SMT constraints is the Cartesian product of the domains of multi-valued variables. In order to adapt the hash-based sampling to this setup, we need to somehow consider dividing this space into small cells.

To resolve this issue, we introduce auxiliary Boolean variables, called *sampling variables* (denoted by $z_1, z_2, \cdots$), for the node constraints in *DecTree*, and *sampling constraints* that assign the node constraints to the sampling variables using the logical equivalence relation. Based on the setting, we impose random XOR clauses over those sampling variables on an SMT formula, simulating Algorithm 2.

For example, for constraint (3), we introduce the two sampling variables, z_1 and z_3, and two sampling constraints $z_1 \Leftrightarrow (gender_1 = 0)$ and $z_3 \Leftrightarrow (income_1 < 1000)$. Suppose here a single XOR clause, say $z_1 \oplus z_3$, happens to be imposed. Because of the sampling constraints to z_1 and z_3, the effect is that one of $(gender_1 = 0)$ and $(income_1 < 1000)$ is true, but not both of them, and the input space is partitioned into two: one satisfying $z_1 \oplus z_3$ and the other. It is thus expected that random XOR clauses introduced as above bring similar effect as in XORSAMPLE to our SMT setup.

[2] One more run of the solver is sufficient to do this, but we omit the details due to the space limitation.

Automatically Deciding the Number of XOR Clauses (s). To fully automate the testing process of the proposed technique, we decide the number of XOR clauses (s) in the following way: increment s from 0 to 20 until for s XOR clauses (G) randomly generated as in line 1, the formula $\phi'_f \ \& \ I \ \& \ G$ becomes unsatisfiable; let the final value for s be multiplied by 0.5.

Basic Test Case Generation Using XOR Sampling. Based on the preparation explained in Sect. 3.1, Algorithm 3 shows the basic test generation algorithm of our proposed method. The steps are: Introduce sampling variables for all node constraints in *DecTree* and generate the sampling constraints for them (line 1). For instance, the sampling constraints for the SMT formula in Fig. 2 are listed in Fig. 3. Next, estimate the parameter s as explained in Sect. 3.1. In the while loop, generate XOR clauses (G) each time; find a solution for $\phi'_f \ \& \ I \ \& \ G$, if exists, by applying an SMT solver; accumulate it.

Algorithm 3: Test case generation by XOR constraints

Parameter : $q \in (0, 1)$
Data: A positive integer k, a satisfiable SMT formula $\phi_{f'} = DecTree \wedge Unfair$
Result: k (possibly duplicate) solutions of $\phi_{f'}$

1 $I \leftarrow$ Sampling constraints;
2 $s \leftarrow$ estimated the number of XOR clauses
  `// Section 3.1`
3 $Sol \leftarrow \emptyset$ `// multiset`
4 **while** $|Sol| < k$ **do**
5 $G \leftarrow s$ XOR clauses, each sampling variable chosen with probability q, constant 1 with 1/2;
6 **if** $\phi_{f'} \ \& \ I \ \& \ G$ *is satisfiable* **then** $Sol \leftarrow Sol \cup GetSolution(\phi_{f'} \ \& \ I \ \& \ G)$;
7 **return** Sol;

Remark. We remark that (1) the heuristic search (in Sect. 3.1) is ad-hoc and (2) checking of unique solutions (in line 5 of Algorithm 2) is skipped in Algorithm 3. These may affect the degree of uniformity, but there are several reasons for the design choices. First, we find it technically difficult to determine optimal s as well as make solutions unique. Second, the proposed method performs better than VBT even with the ad-hoc search and without the uniqueness checking, as will be shown in Sect. 4, which accomplishes our purpose. Third, modern techniques (e.g., [3–5]) in SAT solving use different techniques, such as independent supports and solution enumeration (BSAT), instead of considering uniqueness of solutions, and they not only lead to large performance gain but also provide a theoretical guarantee of almost-uniformity, which we are more interested in employing but it seems to cause unacceptable overhead if those techniques are simulated in our SMT setup in a straightforward way.

3.2 Enhancement (VBT-X)

Reducing Sampling Variables. Properly determining sampling variables to be used affects the degree of uniformity as well as time required for sampling. We present three ways of reducing sampling variables. The first two leverage the notion of *independent supports*, and we begin with reviewing it.

$z_1 \Leftrightarrow (gender_1 = 0)$ $z_9 \Leftrightarrow (gender_2 = 0)$
$z_2 \Leftrightarrow (gender_1 = 1)$ $z_{10} \Leftrightarrow (gender_2 = 1)$
$z_3 \Leftrightarrow (income_1 < 1000)$ $z_{11} \Leftrightarrow (income_2 < 1000)$
$z_4 \Leftrightarrow (income_1 >= 1000)$ $z_{12} \Leftrightarrow (income_2 >= 1000)$
$z_5 \Leftrightarrow (income_1 < 5000)$ $z_{13} \Leftrightarrow (income_2 < 5000)$
$z_6 \Leftrightarrow (income_1 >= 5000)$ $z_{14} \Leftrightarrow (income_2 >= 5000)$
$z_7 \Leftrightarrow (age_1 < 40)$ $z_{15} \Leftrightarrow (age_2 < 40)$
$z_8 \Leftrightarrow (age_1 >= 40)$ $z_{16} \Leftrightarrow (age_2 >= 40)$

Fig. 3. Sampling constraints of the basic version

Independent Support. Independent support of Boolean formula F [12] is a subset of variables in F such that in every solution of F, the truth values of these variables determine those of the other variables. In the hash-based sampling, it is desirable to focus on as a small independent support as possible and perform sampling by generating XOR clauses over independent support only. This is because XOR clauses have to decouple the dependency between variables in the independent support; if XOR clauses included many other variables, they would bring bias in such a way that the truth values of some variables in drawn samples were unfairly tied. What is worse, it is extremely hard to find a solution of F constrained by long XOR clauses. We will thus consider variables that turn out, from the VBT setup, to have the dependency in their truth values.

Equivalence. Because of the unfairness constraint *Unfair*, some pairs of SMT variables having common non-protected attributes, say age_1 and age_2, must have the same value. Hence, from the following two sampling constraints, the sampling variables z_7 and z_{15} must be logically equivalent: $z_7 \Leftrightarrow (age_1 < 40)$ and $z_{15} \Leftrightarrow (age_2 < 40)$. Clearly, it is sufficient to consider only one of them, say z_7, to be included in XOR clauses, and introduce only the sampling constraint for the variable considered: $z_7 \Leftrightarrow (age_1 < 40)$.

Exclusive OR. Consider the following constraints: $z_7 \Leftrightarrow (age_1 < 40)$ $z_8 \Leftrightarrow (age_1 >= 40)$. Clearly, one of $(age_1 < 40)$ and $(age_1 >= 40)$, is true, but not both of them; the same applies to their sampling variables z_7 and z_8. Hence, it is sufficient to consider only one of z_7 and z_8 to be included in XOR clauses, and introduce only the sampling constraint for it.

Symmetry. Suppose we have a solution σ for the SMT constraints in Fig. 2, that induces the following discriminatory data instance x_1 (and x_2):

$$x_1 : \quad [gender_1 = 0, income_1 = 1000, age_1 = 40, class_1 = 1]$$
$$x_2 : \quad [gender_2 = 1, income_2 = 1000, age_2 = 40, class_2 = 0]$$

The following assignment σ', obtained from σ by swapping x_1 and x_2, is also satisfying the constraints.

$$x_1' : \quad [gender_1 = 1, income_1 = 1000, age_1 = 40, class_1 = 0]$$
$$x_2' : \quad [gender_2 = 0, income_2 = 1000, age_2 = 40, class_2 = 1]$$

This symmetry holds in general because of the construction of *Unfair* and *DecTree*. That is, for any solution σ of $\phi_f' = DecTree \ \& \ Unfair$ for x_1 and x_2, the assignment, σ', obtained from σ by swapping x_1 and x_2 is also satisfying. The truth values of sampling variables differ only in those of the protected attribute, i.e., *gender* in the above case. We do not want to distinguish σ and σ'. We hence do not include all sampling variables of the protected attribute in XOR clauses, and do not introduce the sampling constraints for them. For the running example, the followings are ignored: $z_1 \Leftrightarrow (gender_1 = 0)$, $z_2 \Leftrightarrow (gender_1 = 1)$, $z_9 \Leftrightarrow (gender_2 = 0)$, and $z_{10} \Leftrightarrow (gender_2 = 1)$.

Figure 4 lists all sampling constraints for the version in which all variable reductions are applied.

Shortening XOR Clause Length. As mentioned in Sect. 3.2, short XOR clauses are preferable in practice. The variable reductions given so far are effective for shortening XOR clause length because the expected length is determined by the number of sampling variables and the probability with which each variable is chosen.

We here present another way, which is expected to not sacrifice the degree of uniformity so much. In order to build an XOR clause, for each attribute we randomly choose one from the sampling variables having the attribute in common and determine with given probability q whether or not it is included in the current XOR clause. For instance, we have three sampling variables z_3, z_5, z_7 in Fig. 4. Since z_3 and z_5 have the same attribute *income*, we choose one of z_3 and z_5 at random, and then determine with probability q whether or not it is included. Clearly, the expected length of an XOR clause is related to the number of non-protected attributes.

$$z_3 \Leftrightarrow (income_1 < 1000)$$
$$z_5 \Leftrightarrow (income_1 < 5000)$$
$$z_7 \Leftrightarrow (age_1 < 40)$$

Fig. 4. Sampling constraints of the improved version

4 Evaluation

This section reports our evaluation of the proposed technique by experiments.

For evaluation, we set the following two RQs.

RQ1: Can VBT-X detect discriminatory data more efficiently than VBT?
RQ2: Are the enhancement techniques of VBT-X effective?

RQ1 is the main RQ, since efficient detection of discriminatory data is the main motivation of this work, like other algorithm development for individual fairness testing [1,8,24]. In addition, recall that our work is motivated to improve the VBT framework, which is shown to perform better than other main black-box testing approaches in [24]. RQ2 quantitatively evaluates performance improvement brought by the enhancement techniques explained in Sect. 3.2.

4.1 Experimental Setup

For experiments to run VBT, we use the VBT implementation[3] by the authors of [24]. For all experiments, we use VBT branch pruning for test generation strategy, instead of data pruning, since it is shown in [24] that branch pruning is more efficient. We have implemented the basic version and the improved version of VBT-X (which are respetively called 'Basic VBT-X' and just 'VBT-X'), using Python version 3.8.10 and Scikit-learn version 0.22.1, modifying the original VBT implementation. For a fair comparison, we use the same setup regarding classifiers, datasets, and protected attributes as in [24], which render 16 ($= 4 \times 2 \times 2$) configurations, as follows:

- Classifier: Logistic Regression (LR), Random Forest (RF), Naive Bayes (NB), Decision Tree (DT)
- Dataset: 'Adult' Census Income[4], 'German' Credit Card[5]
- Protected attribute: Gender (Male, Female), Race (White, others), Age

For RQ1, we compare the numbers of detected discriminatory data by VBT and VBT-X within a given execution time limit. We also investigate the cause of the result. We specifically investigate two possibilities for it: the result is mainly caused by difference in (1) the numbers of generated (and hence executed) test cases, and/or (2) precision scores (i.e., hit ratios of discriminatory data over generated test cases) of VBT and VBT-X.

For RQ2, instead of using the heuristic search to decide the number of clauses s explained in Sect. 3.1, we compare Basic VBT-X and VBT-X by executing them with $s = 10$. This is because Basic VBT-X with automatic decision of s runs too slow to detect any discriminatory data for most of the configurations, within our execution time limit. We also investigate the cause of the result, similarly to RQ1. We thus measure (1) numbers of generated test cases, and (2) precision scores of Basic VBT-X and VBT-X.

For all experiments, we set ten minutes (600 s) for the execution time limit. For each configuration, we execute 10 trials and take the average of them. Intel(R) Xeon(R) Silver 4210 CPU @ 2.20 GHz Processor, 32 GB memory, running Ubuntu 20.04.4 LTS.

[3] https://github.com/arnabsharma91/fairCheck.
[4] https://archive.ics.uci.edu/ml/datasets/adult.
[5] https://archive.ics.uci.edu/ml/datasets/statlog+(german+credit+data).

Table 1. The results of experiments. The rows represent configurations, each combined from datasets, classifiers, and Protected features. The columns for 'VBT' and the two versions of 'VBT-X' respectively represent the results of three criteria of the numbers of detected discriminatory data ('#Disc'), the number of generated test cases ('#Tests'), and precision scores ('Prec.'), while their improvement ratios are shown in the next columns (for 'Improvement ratio'). The columns for 'Basic VBT-X (s = 10)' and 'Imp. VBT-X (s = 10)' represent those for Basic VBT-X and Improved VBT-X with $s = 10$, appended with their improvement ratios in the next columns. The bottom row 'avg./total' shows the total numbers (for '#Disc' and '#Test') or averages (for 'Prec.'); and the row '#wins' shows the numbers of configurations that the technique in the column outperforms the competitor in the respective three criteria.

No.	Dataset	Clf.	Prot. feature	VBT			Imp. VBT-X (s: auto)			Improvement ratio			Basic VBT-X (s = 10)			Imp. VBT-X (s = 10)			Improvement ratio		
				#Disc	#Tests	Prec.	#Disc	#Tests	Prec.	#Disc	#Tests	Prec.	#Disc	#Tests	Prec.	#Disc	#Tests	Prec.	#Disc	#Tests	Prec.
1	Adult	LR	Gender	15	269	**0.06**	**28**	**735**	0.04	1.87	2.73	0.67	28	750	**0.04**	**86**	**2015**	**0.04**	3.07	2.69	1.00
2		LR	Race	**69**	1864	**0.04**	52	**2145**	0.03	0.75	1.15	0.75	72	2365	**0.03**	**91**	**4015**	0.02	1.26	1.70	0.67
3		RF	Gender	728	2161	0.34	**1545**	**2925**	**0.53**	2.12	1.35	1.56	1236	1950	**0.63**	**1878**	**3475**	0.54	1.52	1.78	0.86
4		RF	Race	10	1896	0.006	**110**	**2845**	**0.04**	11.00	1.50	6.67	38	1980	0.02	**86**	**3300**	**0.03**	2.26	1.67	1.50
5		NB	Gender	1669	3329	0.5	**4580**	**5865**	**0.78**	2.74	1.76	1.56	2233	4165	0.54	**5033**	**6645**	**0.76**	2.25	1.60	1.41
6		NB	Race	784	3822	0.21	**3837**	**5635**	**0.68**	4.89	1.47	3.24	1780	4120	0.43	**3908**	**6170**	**0.63**	2.20	1.50	1.47
7		DT	Gender	1688	2127	0.79	**5075**	**5685**	**0.89**	3.01	2.67	1.13	2784	3580	0.78	**5777**	**6600**	**0.88**	2.08	1.84	1.13
8		DT	Race	1748	2531	0.69	**5225**	**5960**	**0.88**	2.99	2.35	1.28	1844	3265	0.57	**5040**	**6325**	**0.80**	2.73	1.94	1.40
9	German	LR	Gender	214	1772	**0.12**	**244**	**2205**	0.11	1.14	1.24	0.92	230	2065	**0.11**	**324**	**2875**	**0.11**	1.41	1.39	1.00
10		LR	Age	173	1879	0.09	**289**	**2615**	**0.11**	1.67	1.39	1.22	307	2200	**0.14**	**328**	**2990**	0.11	1.07	1.36	0.79
11		RF	Gender	**168**	1269	**0.13**	92	**1805**	0.05	0.55	1.42	0.38	89	1545	**0.06**	**111**	**2175**	0.05	1.25	1.41	0.83
12		RF	Age	66	1286	0.05	**116**	**1870**	**0.06**	1.76	1.45	1.20	125	1615	**0.08**	**129**	**2180**	0.06	1.03	1.35	0.75
13		NB	Gender	77	1297	**0.06**	**82**	**1850**	0.04	1.06	1.43	0.67	70	1605	0.04	**127**	**2535**	**0.05**	1.81	1.58	1.25
14		NB	Age	165	2674	0.06	**518**	**3245**	**0.16**	3.14	1.21	2.67	421	2820	**0.15**	**523**	**3695**	0.14	1.24	1.31	0.93
15		DT	Gender	1343	2403	0.56	**3942**	**4440**	**0.89**	2.94	1.85	1.59	1770	3085	0.57	**3519**	**5060**	**0.70**	1.99	1.64	1.23
16		DT	Age	1081	2471	0.44	**3505**	**4380**	**0.80**	3.24	1.77	1.82	1930	3185	0.60	**3109**	**4955**	**0.63**	1.61	1.56	1.05
		avg./total		9998	33050	0.26	**29249**	**54205**	**0.38**	2.92	1.64	1.47	14957	40295	0.30	**30069**	**65010**	**0.35**	2.01	1.61	1.16
		#wins		2	0	5	**14**	**16**	**11**	N/A	N/A	N/A	0	0	6	**16**	**16**	**8**	N/A	N/A	N/A

4.2 Results

Table 1 shows the results of experiments, based on which we answer the RQs.

RQ1: Can VBT-X detect more discriminatory data than VBT? From the columns for #Disc of VBT and VBT-X in Table 1, we can observe that VBT-X detects more discriminatory data than VBT, by around 2.92 times in average, for 14 out of 16 configurations, and by upto 11 times for configuration No. 4.

From the columns for '#Tests' and 'Prec.' of VBT, VBT-X and their 'Improvement ratio', we can observe the following: (1) VBT-X generates more test cases than VBT by 1.64 times in average and for all the 16 configurations, and (2) the precision of VBT-X is higher than that of VBT by 1.47 times in average and for 11 out of 16 configurations. We thus may be able to ascribe the above conclusion to both of the number of generated test cases and precision scores.

However, with a finer analysis, we can more likely ascribe the conclusion to the number of generated test cases than the precision score. First, we can say that the improvement in the number of generated test cases (1.64) of VBT-X is higher than that of precision score (1.47). Second, VBT-X wins VBT for all the 16 configurations in the number of test cases, but only for 11 configurations in the precision score. Third, for several configurations (No. 1, 9, 13), although precision score of VBT-X is lower than that of VBT, VBT-X can find more discriminatory data since it generates more test cases.

Answer for RQ1: Yes. VBT-X can detect more discriminatory data than VBT by 2.92 times in average and for more than 87 (= 14/16) % configurations.

RQ2: Are the enhancement techniques of VBT-X effective? From the columns for '#Disc' of 'Basic VBT-X(s = 10)' and 'Imp. VBT-X (s = 10)', we can observe that VBT-X detects more discriminatory data than Basic VBT-X by 2.01 times in average and for all the 16 configurations.

From the columns for '#Tests' and 'Prec.' of Basic VBT-X, VBT-X, and their 'Improvement ratio', we can observe that (1) VBT-X generates more test cases than Basic VBT-X by 1.61 times in average and for all the 16 configurations, and (2) the precision score of VBT-X is higher than that of Basic VBT-X by 1.16 times in average and for 8 out of 16 configurations, while Basic VBT-X wins for 6 configurations. We may ascribe the above conclusion to that VBT-X can generate more test cases, since the improvement on precision score may not be enough significant.

Answer for RQ2: Yes, enhancement techniques explained in Sect. 3.2 are effective, as they improve discriminatory-detecting ability of Basic VBT-X by 2.01 times in average.

5 Related Work

Testing of individual fairness is first tackled by Galhotra et al. in [8]. The main contribution is establishing its concept, including the concepts of similarity of individuals and discriminatory data, which are explained in Sect. 2.1. The concept has become the basis of most existing studies of individual fairness testing, including our study. They also develop a black-box testing algorithm for this concept, named THEMIS, which detects discriminatory data, given a classifier as input.

Udeshi et al. [27] proposed an efficient black-box testing algorithm for individual fairness, improving the algorithm by Galhotra et al. [8], The algorithm enhances efficiency, by structuring it into two steps of global and local search. This two-step structure of the algorithm leverages robustness of ML classifiers.

Another well-known technique for individual fairness testing is SG [1], featured with its efficient testing ability. Its mechanism is similar to VBT, as it internally builds an approximation classifier of the CUT using a decision tree, and apply symbolic execution using SMT solver to generates test cases. However, VBT differs from SG in many details. E.g., SG approximates the CUT in a partial decision tree using local model explainer (LIME [21]), while VBT do so in an entire decision tree using training. Our work applies hash-based sampling technique to VBT, because it is reported that VBT has a higher testing ability than SG [24]; however, our proposed technique is basically applicable to SG, too.

Sharma and Wehrheim developed VBT originally for testing *monotonicity* of ML classifiers [25], which is similar but different concept from individual fairness.

After extend the work [25] to fairness testing as VBT in [24], they further extend VBT in several respects, as a technique called MLCHECK [23]. An extension is to apply other properties than monotonicity and fairness, such as security. Another direction is to use Relu-based Deep Neural Network, (instead of using decision trees,) for making approximation classifier of classifier under test.

Several other recent studies on black-box individual fairness testing are as follows: A technique developed by Morales et al. [16] (CGFT) improves efficacy of AEQUITAS, by applying combinatorial t-way testing (CT) [14] to the global search of AEQUITAS. Patel et al. [20] investigates efficacy of applying combination of CT and a counterfactual explanation technique, called DICE[17].

Although above-mentioned techniques all take the black-box (a. k. a.,, model-agnostic) testing approach, the algorithm proposed by Zhang et al. [28] takes a white-box approach, targeting Deep Neural Networks (DNNs). The algorithm, named Adversarial Discrimination Finder (ADF), employs adversarial sample generation techniques using gradient analysis [10,15]. Although ADF can be only applicable for DNN-based classifiers, their experiments show ADF finds more discriminatory data than AEQUITAS and SG.

6 Validity Threats

Our experiments use two datasets ('Adult' and 'German'), the four classifiers (LR, RF, NB, DT), and two attributes ('Gender' and 'Race'), which are exactly the same as those used in [24]. There are other datasets available in algorithm fairness literature (see e.g., the survey of [19]), countless kinds of classifiers, and more kinds of protected attributes (such as age, nationality). However, it is practically infeasible to experiments all combinations, due to combination explosions. Experiments in most of other studies on fairness testing [8,16,24,27] thus also use two or three datasets, classifiers, and attributes.

VBT-X inherently contains random behaviours, as it samples different data on different executions. This threat is mitigated by taking average over 10 trials for all experiments. In experiments for RQ2, we use $s = 10$ for the number of XOR clauses s for a conservative evaluation, since Basic VBT-X best performs with $s = 10$ by preliminary experiments with $s = 5, 10, 15$. Our experiments use 10 min (600 s) for the execution timeout limit. There is no standard criteria for execution time limit in fairness testing literature, but more studies seem to use several hundred seconds for it; e.g., [24] uses 930 s and [28] uses 500 s. Our timeout setting follows this convention.

7 Conclusion and Future Work

In this paper, we developed a black box testing technique for individual fairness VBT-X, by applying hash-based sampling techniques [3–5,9] to the test generation of VBT, a state-of-the-art fairness testing technique by Sharma and Wehrheim [24,25]. The novelty of this work is to show the mechanism to apply

hash-based sampling, which substantially different approach from VBT, actually works, and performs better than VBT.

There are several directions for future work. One direction is to refine our ad-hoc heuristic search to decide the number of XOR clauses, and improve the degree of uniformity of sampled data in VBT-X, as mentioned in Sect. 7. Several related techniques proposed in SAT solving settings [3–5] may be applicable for the purpose, although we may encounter difficulty to adapt them to our SMT setting. Another direction in the technical side is to apply our proposed technique to MLCheck [23], which uses Deep Nueral Network (DNN) for approximation classifiers in VBT framework, instead of decision trees. We are also interested in applying VBT-X to other properties such as security (e.g., Trojan attack) than fairness as in [23]. The fourth direction is to conduct more thorough experiments to evaluate our proposed techniques using more datasets, classifiers, and protected attributes to generalize obtained results, as explained in Sect. 6.

Acknowledgements. This paper is partly based on results obtained from a project, JPNP20006, commissioned by the New Energy and Industrial Technology Development Organization (NEDO). We are also indebted to the suggestions from peer reviewers of SSBSE22. Although part of provided suggestions has not been adopted due to limited pages, we intend to incorporate these valuable suggestions in the following research.

References

1. Aggarwal, A., Lohia, P., Nagar, S., Dey, K., Saha, D.: Black box fairness testing of machine learning models. In: Proceedings of ESEC/SIGSOFT FSE, pp. 625–635 (2019)
2. Byanjankar, A., Heikkilä, M., Mezei, J.: Predicting credit risk in peer-to-peer lending: a neural network approach. In: Proceedings of SSCI 2015, pp. 719–725. IEEE (2015)
3. Chakraborty, S., Fremont, D.J., Meel, K.S., Seshia, S.A., Vardi, M.Y.: On parallel scalable uniform SAT witness generation. In: Baier, C., Tinelli, C. (eds.) TACAS 2015. LNCS, vol. 9035, pp. 304–319. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46681-0_25
4. Chakraborty, S., Meel, K.S., Vardi, M.Y.: A scalable and nearly uniform generator of SAT witnesses. In: Sharygina, N., Veith, H. (eds.) CAV 2013. LNCS, vol. 8044, pp. 608–623. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39799-8_40
5. Chakraborty, S., Meel, K.S., Vardi, M.Y.: Balancing scalability and uniformity in SAT witness generator. In: Proceedings of DAC 2014, DAC 2014, pp. 1–6 (2014)
6. Duenas-Osorio, L., Meel, K., Paredes, R., Vardi, M.: Counting-based reliability estimation for power-transmission grids. In: Proceedings of AAAI 2017, vol. 31, no. 1, February 2017
7. Dwork, C., Hardt, M., Pitassi, T., Reingold, O., Zemel, R.: Fairness through awareness. In: Proceedings of ITCS 2012, pp. 214–226 (2012)
8. Galhotra, S., Brun, Y., Meliou, A.: Fairness testing: testing software for discrimination. In: Proceedings of ESEC/FSE 2017, pp. 498–510 (2017)
9. Gomes, C.P., Sabharwal, A., Selman, B.: Near-uniform sampling of combinatorial spaces using XOR constraints. In: Proceedings of NIPS 2006, pp. 481–488 (2006)

10. Goodfellow, I.J., Shlens, J., Szegedy, C.: Explaining and harnessing adversarial examples. In: Proceedings of ICLR (2015)
11. Hoffman, M., Kahn, L., Li, D.: Discretion in hiring. Q. J. Econ. **133**(2), 765–800 (2018)
12. Ivrii, A., Malik, S., Meel, K.S., Vardi, M.Y.: On computing minimal independent support and its applications to sampling and counting. Constraints **21**(1), 41–58 (2016). https://doi.org/10.1007/s10601-015-9204-z
13. Angwin, J., Larson, J., Mattu, S., Kirchner, L.: Machine bias (2016). https://www.propublica.org/article/machine-bias-risk-assessments-in-criminal-sentencing
14. Kuhn, R., Kacker, R.: Introduction to Combinatorial Testing. Chapman & Hall CRC (2013)
15. Kurakin, A., Goodfellow, I.J., Bengio, S.: Adversarial examples in the physical world. In: Proceedings of ICLR 2017. OpenReview.net (2017)
16. Perez Morales, D., Kitamura, T., Takada, S.: Coverage-guided fairness testing. In: Lee, R. (ed.) ICIS 2021. SCI, vol. 985, pp. 183–199. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-79474-3_13
17. Mothilal, R.K., Sharma, A., Tan, C.: Explaining machine learning classifiers through diverse counterfactual explanations. In: Proceedings of FACCT 2020, pp. 607–617. ACM (2020)
18. Naveh, Y., et al.: Constraint-based random stimuli generation for hardware verification. In: Proceedings of IAAI 2006, pp. 1720–1727. AAAI Press (2006)
19. Oneto, L., Chiappa, S.: Fairness in machine learning. In: Oneto, L., Navarin, N., Sperduti, A., Anguita, D. (eds.) Recent Trends in Learning From Data. SCI, vol. 896, pp. 155–196. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-43883-8_7
20. Patel, A.R., Chandrasekaran, J., Lei, Y., Kacker, R.N., Kuhn, D.R.: A combinatorial approach to fairness testing of machine learning models. In: Proceedings of IWCT 2022, pp. 1135–1144. IEEE (2022)
21. Ribeiro, M.T., Singh, S., Guestrin, C.: "why should I trust you?": Explaining the predictions of any classifier. In: Proceedings of KDD 2016, pp. 1135–1144. ACM (2016)
22. Roth, D.: On the hardness of approximate reasoning. Artif. Intell. **82**(1), 273–302 (1996)
23. Sharma, A., Demir, C., Ngomo, A.N., Wehrheim, H.: MLCHECK-property-driven testing of machine learning classifiers. In: Proceedings of ICMLA 2021, pp. 738–745 (2021)
24. Sharma, A., Wehrheim, H.: Automatic fairness testing of machine learning models. In: Casola, V., De Benedictis, A., Rak, M. (eds.) ICTSS 2020. LNCS, vol. 12543, pp. 255–271. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-64881-7_16
25. Sharma, A., Wehrheim, H.: Higher income, larger loan? Monotonicity testing of machine learning models. In: Proceedings of ISSTA 2020, pp. 200–210. ACM (2020)
26. Sipser, M.: A complexity theoretic approach to randomness. In: Proceedings of STOC 1983, pp. 330–335. ACM (1983)
27. Udeshi, S., Arora, P., Chattopadhyay, S.: Automated directed fairness testing. In: Proceedings of ASE 2018, pp. 98–108 (2018)
28. Zhang, P., et al.: White-box fairness testing through adversarial sampling. In: Proceedings of ICSE 2020, pp. 949–960 (2020)