Test Input Generation Using Extraction of Bias-Revealing Features Inputs

Timothy Jordan, Aldeida Aleti, Chetan Arora
Monash University, Melbourne, Australia
Email: {tjor0005@student, aldeida.aleti chetan.arora}@monash.edu

Index Terms

Fairness Testing; Input Generation; Machine Learning; Instance Space Analysis

I. Introduction

In the ever-growing utilisation of AI-based systems in sensitive domains - including criminal justice, employment, insurance, education, credit scoring, and loan acceptance - it becomes more apparent for the need of these systems to be fair and unbiased [1]. Identifying and maintaining fairness in AI-based software systems is one of the key focal points within the software development life-cycle[2]. Fairness testing aims to remove fairness bugs within a system which are imperfections that cause the predicted outcome to deviate from the desired fair outcome [2]. It is considered a fairness bug if a systems contains any biases towards a particular race, gender, ethnicity, occupation, or any other sensitive attribute. Bias and unfairness are strongly intertwined in this sense, and therefore the purpose of fairness testing is to ensure sensitive attributes are not treated in a biased manner within a software-based system [1]. That is, there is no discordance between the observed outcome and the desired outcome which upholds fairness conditions [2].

For this project and literature review, we will be focusing on two-phase search-based test input generation in fairness testing. As shown in figure 2, test input generation is one of the key components within the fairness testing workflow [2]. It refers to the automated production of input instances which aim to induce discrimination and uncover fairness bugs within the software system [2]. The two-phased search-based technique is one of many ways to generate inputs for fairness testing. It consists of a global search phase and a local search phase [3]. In Global search phase, the objective is to identify bias instances by conducting a comprehensive exploration of the system's input space [4]. This phase aims to reveal bias instances that manifest at a systematic level, providing a broad perspective on disparities. Subsequently, the local search phases focuses on the refinement of bias detection. During this phase, the goal is to identify additional bias instances that exist in proximity to those previously uncovered in global search [4]. By honing in on the local vicinity of previously detected discriminatory, this phases provides a more granular understanding of potential fairness bugs within the system.

A test input generator generates a comprehensive test suite that is designed to assess the software system's compliance with a specific fairness criterion [2]. Within the domain of fairness testing, there exists a diverse range of distinct fairness definitions, each falling under either the category of individual or group fairness [5]. The selection of a specific fairness metric holds considerable significance as it directly influences the set of conditions an AI-based system must meet to be regarded as unbiased [5].

In the broader scope of algorithm testing, there exists a methodology that is used for algorithm stress testing called instance space analysis (ISA) [6]. ISA allows one to understand and visualise important relationships between features and the algorithm performance [6]. In the context of fairness testing, we are able to utilise ISA to pinpoint key features that influence the individual fairness outcome of an instance. The utilisation of key discriminatory features in test input generation has yet to be explored and can assist in creating test suites that have a higher proportion of bias-revealing inputs compared to existing methodologies.

This literature review offers a detailed description of current research within the field of fairness testing; primarily focusing on various fairness definitions widely applied, the typical workflow established to detect fairness bugs and mitigate them, as well as raising awareness to components of a software system which are subject to fairness testing [2]. This paper then identifies the key research gap in regards to input generation for fairness testing. It will be discussing about how current input generators rely on unsupported assumptions to generate bias-revealing inputs with no knowledge regarding which features of the input space greatly impact biased outcome [3]. I then propose a novel method to integrate ISA within current state-of-the-art test input generation techniques which generates a diverse range of bias-revealing inputs for fairness testing. Furthermore, a research project plan is provided to clarify the research questions we aim to answer in the project.

II. Substantive Literature Review

A. Fairness Definitions

The definition of fairness is a key aspect in setting the fairness conditions an AI-based software system aims to accomplish [2]. Researchers have explored and proposed various fairness definitions over the years, each of them being labelled under two categories: individual fairness and group fairness [5]. The former requires a software system to produce similar outputs for similar individual, whereas the latter expects different demographic groups to produce similar outputs [7]. Sensitive attributes, also known as protected attributes, are key components in fairness testing as they are characteristics we aim to protect against unfairness [2]. Examples of sensitive attribute include gender, age, income, occupation or other personal characteristics [8].

1) Individual Fairness: **Fairness through unawareness** follows the notion that a software system is able to generate fair predictions by excluding sensitive attributes in its decision-making [9]. This definition is simple by design and because of that does not effectively capturing fairness [9]. Under this definition, unfairness in software systems can still emerge as certain non-sensitive attributes can highly correlate to sensitive attributes [9]. For example, the post code of an individual is a non-sensitive attribute but can infer that person's socioeconomic status. Therefore a AI-based system can be biased if varying postcodes produces disparity in results for a loan application.

Fairness through awareness states that, based on a pre-defined similarity metric, instances that are considered similar should have similar outcomes in a software system [10]. The similarity metric is created based on a specific task, but it should reflect what is publicly agreed by society and is open for discussion [10]. This gives the system 'awareness' of how to treat similar individuals for a given task [10]. This definition can be viewed as a general concept and the following individual fairness definitions create a concrete similarity measures.

Causal fairness requires a software system to produce the same prediction for instances that only differ in sensitive attribute values [11]. Formally, the system under test is considered fair if individuals i and j would have the same outcome in a software system if their non-sensitive attributes are the same but their sensitive characteristics differ [11]. Many fairness testing systems test under causal fairness due its easiness in validating discriminatory outcomes [2]. One would only need to test the input by perturbing the sensitive value and running it through the system again to see if the predicted outcome has changed.

Counterfactual fairness proposes that a predicted outcome should remain consistent for an individual across both the real-world context as well as a counterfactual scenario, wherein the individual is affiliated with an alternate demographic group [7]. It is similar to causal fairness in that it highlights unfairness emerging from sensitive attribute changes [11], [7]. However, it adds an extra complexity as, in the counterfactual world, non-sensitive attributes that have a causal relationship to a sensitive attribute would be adjusted accordingly based on the alternate demographic group [7].

2) Group Fairness: **Statistical parity / demographic parity** is a common conception of group fairness and has been used as a metric of success in many papers [12], [13]. It states that a system follows fairness when the probability of a favourable outcome is the equal across all demographic groups [14]. To exemplify, the probability of getting a loan accepting would be the same for an individual that is male as opposed to an individual that is female. Although this definition has evident flaws and limitations as stated in the widely cited paper written by Hardt, et al. (2016) [9]. It is possible for a model to achieve statistical parity while still making unfair predictions for individual instances within a group[9]. At the group level, each group has the probability of being admitted to a university, however at an individual level, a highly qualified group A student might be rejected in favour of a less qualified group B student to maintain proportionality in each group.

Hardt proposed a two definitions of fairness that overcome the shortcomings of statistical parity [2]. **Equalised odds** denotes that a system satisfies fairness when the prediction and sensitive attribute are an independent conditional on the ground truth [9]. In essence, the true positive rate and false positive rate must be equal across the demographic groups [9]. This means knowing the existence of the true prediction does not impact the likelihood of the sensitive attribute and the prediction being related or not related. Hardt also proposes a weaker version of equalised odds that provides better applicability is **equality of opportunity [9]**. It requires a software system to have the same true positive rate for predictions across all demographic groups [9]. Both of these definitions ensure that individual fairness is maintained across groups and allows predictions to be influenced by the sensitive attribute but only through the target outcome [9].

Testing for individual and group fairness comes with their own sets of challenges. Testing for group fairness often comes with more difficulties as there can also exist instances which are in multiple groups

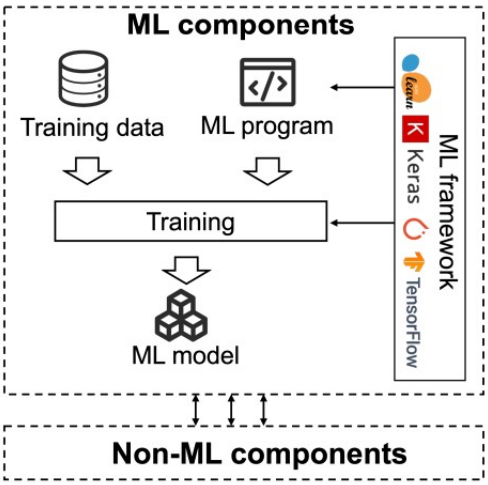


Fig. 1. Fairness Testing Components. Sourced from [2]

simultaneously. Understanding the influence of intersecting demographics complicates the fairness evaluation stage and has not been thoroughly researched.

B. Fairness Testing Components

It is evident that the machine learning (ML) model in AI-based systems is a key aspect to test for fairness, however, fairness testing expands to more than just this component (refer to figure 1). This section of the literature review focuses on what components fairness testing is performed against. Similar to how software testing is conducted on various testable sections in a software system, fairness testing is also conducted on different sections [2]. This includes:

- Training data
- ML program
- ML model
- ML framework
- Non-ML components

These fairness testing components are tested in various ways.

Training data testing Training data refers to the dataset that allows a ML model to learn the relationships between the features and labels [15]. The training data that we train the ML model with significantly impacts the patterns and decision logic being learned [15]. The biases prevalent in the training data is shown to one of the root causes to unfairness in ML software [16]. When the training dataset has a significant class imbalance, where one class (minority) is underrepresented compared to the other classes (majority class), the model may become biased towards the majority class. Training data testing aims to identify and reduce biases found in the data. Researchers have done training data debugging for label bias [17] and feature bias [18] . There is also selection bias in which there is a strong correlation between the predicted outcome and the sensitive attribute whereby modifying the attribute value causes the prediction to change [19] **ML program testing** The ML program encodes the entire pipeline of machine learning including data pre-processing, model selection and training and hyper-parameter tuning [15]. Fairness bugs may arise in any of these processes and so research has been conducted investigate and resolve the fairness impacts for each one of them [20], [21]. **ML model testing** When performing ML model testing, the focus is placed on the ML model and investigating the input-output behaviour to uncover fairness bugs that are present in the model predictions [11]. Fairness testing on ML models can follow a white box, black box and grey box approach. White-box testing requires full access to the decision logic and source code of the model to perform testing; black box testing uncovered fairness bugs in the model based solely on the fundamental aspects of the model and no knowledge of the model's internal workings; grey box testing is in between white box and black box wherein fundamental aspects and limited model knowledge are utilised [22]. **ML framework testing** The ML framework refers to the set of application programming intefaces (APIs) that are essential for streamlining and simplifying the process of building and testing ML systems [2]. The underlying algorithms and API functions that are created by an ML framework (e.g. Keras, Tensorflow, and scikit-learn) have an impact to the way the ML

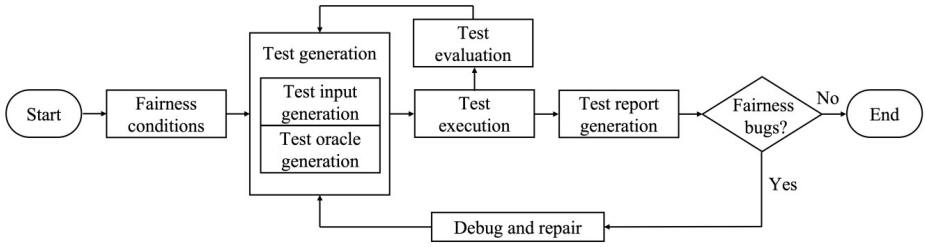


Fig. 2. Fairness Testing Workflow. Sourced from [2]

program and the ML model are designed [23]. Therefore, ML frameworks are an important factor to consider when testing for fairness bugs [23]. **Non-ML component testing** All the aforementioned fairness testing components can be labelled as ML components of a software system. There also exists another component that is segregated from the rest, as shown in fig 1, which is non-ML component. It is possible that unfairness can extend beyond the ML aspect of a software system, although there has been little to no research done to investigate the fairness impact and testing of these components [2].

C. Fairness Testing Workflow

The fairness testing workflow is a common approach to conducting fairness testing on a AI-based software system as shown in figure 2. The initial phase of the process requires software engineers to analyse what users expect from their system under test and thus identify what fairness conditions it needs to uphold [2]. This includes the fairness definition adopted, sensitive attributes considered and the maximum threshold to which bias is tolerated [2]. An automated process of test input generation follows afterwards wherein test inputs are sampled from the collected data or newly created using specific techniques. Following this, a test oracle is identified and created based on the fairness definition chosen [15]. Once these two processes are completed, the test inputs are executed by the software under test with each output being identified as a fair or unfair result through the test oracle mechanism [2]. We then determine the adequacy of the fairness testing system itself, evaluating its efficacy in uprooting fairness bugs [2]. Alongside this, a bug report is generated providing important details to which assist engineers in debugging and repairing the model from the identified fairness bugs [2]. This process of input generation, evaluation, reporting and debugging is a repeated cycle that is continued until the fairness conditions defined at the beginning have been fulfilled.

1) Test Input Generation: Test input generation is one of the main driving forces of the fairness testing workflow. The main purpose of test input generation is to provide test instances that uncover unfairness bugs within the system under test. An efficient and effective test input generator would be able to create numerous inputs that reveal various biases within the system and localises the cause of this issue

Random test input generation assesses a software system by sampling values for non-sensitive attributes randomly under a uniform distribution and iterating over various sensitive attribute values. This technique is able to assess for causal fairness and statistical parity. For causal fairness, the test oracle mechanism would determine that an input is biased if the prediction changes when attribute value is altered. For statistical parity, the probability of receiving positive predictions across demographic groups in the generated inputs. Due to the unguided nature of random testing, this method has a low effectiveness of creating discriminatory instances.

Search-based test input generation is a methodology that involves using meta-heuristic search algorithms to explore the input space and discover inputs that can expose potential fairness bugs. A number of search-based input generation methodologies have been applied in various fairness testing techniques, but most measure the discrimination of the system under test by causal fairness.

A well-known methodology that follows this framework is the two-phase search-based technique. It involves two phases wherein the global search phase looks through the entire input space to create an initial set of instances and local search phase generates specific bias-revealing inputs based on previously found discriminatory instances. Aequitas is the first two-phase search-based test generation strategy proposed which applies the following hypothesis in its local search algorithm: for a discriminatory input that exists in an input space, there exists more discriminatory inputs that are closer to it [3]. This hypothesis is based on the robustness property of machine learning models where inputs that are similar should have similar outputs. In Aequitas' approach, the global search phases uniformly at random samples inputs to test for discrimination, guaranteeing a high probability of uncovering a biased instance if it exists. The local search

algorithm then searches the neighbouring inputs of the uncovered discriminatory instances by performing slight perturbations, ranging from -1 to 1, to one probabilistically chosen attribute .

More efficient search-based fairness testing techniques have been proposed since the introduction of Aequitas. KOSEI is a search-based technique that follows the same approach as Aequitas, but improves upon the local search algorithm. It looks through the entire neighbourhood of discriminatory instances found in the global search phase by modifying all non-sensitive fields to create new discriminatory instances. Their method proved the increase the efficiency of bias detection and reduces the waste reduction made from Aequitas's local search algorithm [24].

Agarwal, et al (2018) from IBM research devised a input generation method, SG, which methodically explores the input space of the system using symbolic execution instead of random sampling. It works by treating variables as symbolic variables, and the program's logic is explored by tracking how these symbolic values change through the program's execution path. Local explainability is then used in interpreting the path taken by an test instance in the form of a decision tree. Decisions are systematically toggled to create new constraints to generate new test instances that take previously unexplored execution paths [25]. In essence, this technique helps identify possible program paths, inputs and conditions that lead to biased outcomes [25]. Then in the local search phase, The added benefit to this method is its utility. SG can be used in black box fairness testing due to how it uses local explainers to explain predicitions of machine learning models by approximating the model's behaviour locally around a specific data point. Without needed to know the model's underlying framework, this technique provides more real-world applicability in scenarios where software engineers may not have access to the internal model architecture. Using local explainers such as Local Interpretable Model-agnostic Explanations (LIME) also gives SG the properity of being model agnostic, allowing various machine learning models - including deep neural networks and ensemble methods - to be tested against[25].

Verification-based test (VBT) input generation involves generating discriminatory inputs through the use of verification tools to verify if inputs violate the given fairness property. This method ensures that the AI-software behaves in accordance with its intended functionality, and it helps identify deviations from expected behaviour. It is able to verify if a suspicious input is fair by following the Satisfiability Modulo Theories (SMT) solving techniques [26]. Sharma, et al. (2021) [27] proposed MLCheck, a VBT algorithm which treats the system under test as a black-box and a white box model is created and trained based on the same training data as the black-box model. Knowing the internal structure of the white-box model we can put this under verification testing. The white-box model and the fairness property are then both encoded as logical formulas using SMT solvers [27]. Sharma then applies the Z3 SMT solver to check for fairness satisfiability, generating test inputs that violate the fairness property. []

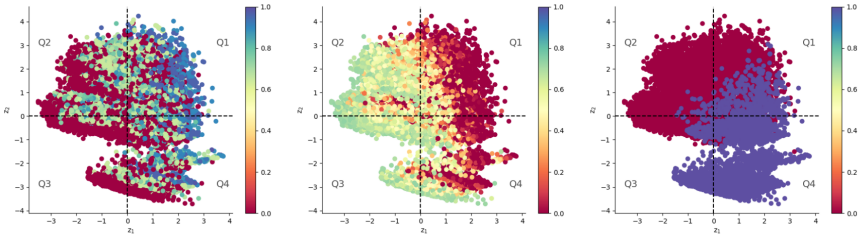
2) Test Oracle Generation: Test oracle generation is a mechanism which determines whether the output of a AI-software system under test matches the expected behaviour which follows the adopted fairness definition [2]. The test oracle serves as a benchmark for evaluating the quality of fairness of the system being tested [17].

3) Debugging and Repairing: The main objective of debugging and repairing is to mitigate or reduce the apparent biases and disparities found within the AI-based system under test [2].Within this stage, there exists pre-processing, in-processing and post-processing algorithms used to reduce bias.

D. Analysing Instance Space

ISA is a methodology first introduced by Kate Smith-Miles and co-workers in 2014 that was created for rigorous stress testing of algorithms [28]. Over the years, the ISA methodology has evolved and been applied to various fields in software engineering including safety-critical driving scenarios [29], automated software repair [30] and automated software testing [31]. This is because it allows one to reveal insightful relationships between input properties and the pre-defined performance of an algorithm [28]. By establishing an instance space, encapsulating all possible test instances within a two-dimensional plane, ISA raises nuanced insights about the weaknesses and strengths of different instance types [6].

A 2D instance space is generated from a selected subset of inputs which serve as representatives of the larger spectrum of possible instances [6]. As shown in figure 3, utilising this instance space makes it feasible to numerically and visually display the mathematical boundary that delineates the theoretical instance space [6]. The ideal projection would be one that establishes linear trends where high values are sectioned at one area and the low values are sectioned in the other area [6]. In reference to figure 3, the distribution of values for feature AV_speed is wish to achieve. This allows one to create linear ML models for each relevant feature and can then be applied to predict the regions within the instance space where there is a high likelihood of good algorithm performance [28], [6]. These regions are also known as algorithm footprints [6]. We can



(a) AV_brake (b) AV_speed (c) is_ped

Fig. 3. Distribution of autonomous vehicle safety-critical features in instance space, with red representing high value and blue representing low value. Sourced from [29]

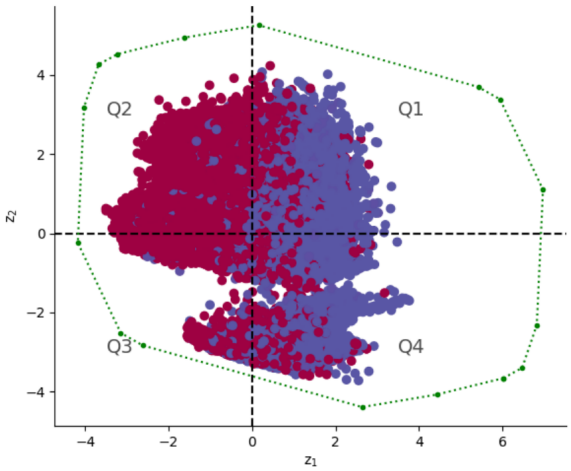


Fig. 4. 2D instance of safety-critical feature value with mathematically defined boundary

extend this ML model to predict algorithm footprints in the broader instance space to assess the diversity and analyse how different features of an instance can impact algorithm footprints [6].

To first begin ISA, one requires to define the four spaces that map the features of a problem instance to the effectiveness [29].

1) Input Space: : A set which includes all possible inputs for a system

2) Feature Space: : A set of relevant features that are extracted from the inputs

3) Instance Space: : the 2D projection space of the *N* features in feature space

4) Performance Space: : defines the formula/mechanism used to evaluate test performance

The additional benefit of ISA is how it allows one to calculate a mathematical boundary of our instance space to identify areas where further tested may be required as shown in figure 4 [29].

III. Summary of State of the Art

Contemporary test input generation methodologies are aimed at producing a suite of inputs designed to efficiently discover fairness bugs and biases inherent in AI-based systems [2]. These methodologies include simple random sampling (Themis) [13], attribute perturbation of previously identified instances of bias (Aequitas and KOSEI) [3], [24], and the use of symbol execution with local explainability (SG) [32]. However, the current state of the art test input generation techniques exhibit a significant gap in the systemic identification of non-protected features that significantly impact biased outcomes. Given the growing demand to ensure fairness in sensitive domains that apply AI-based software, it is all the more important to understand how attribute-outcome relationships commonly found in sensitive domains impact fairness. The examination and manipulation of influential non-sensitive features could potentially yield new insights and enhance the fairness testing methodology to generate more test instances that induce bias.

Another area that lacks research is ensuring the discriminatory instances that are being generated are diverse. This means that all possible areas and borders of the input space where there is a likelihood of producing a biased outcome has been thoroughly explored. Current search-based test input generation techniques explore

randomly the global input space first phase but then in the local search phase only generate instances that are in close proximity to the detected biased instanced found in the former phase . This leaves test scenarios only being generated in certain clusters within the input space, neglecting other areas and edges that may contain bias.

Both of these research gaps can be addressed through the utilisation of instance space analysis. Given how ISA can be applied to any algorithm and performance metric, it is possible to configure the process for fairness testing purposes and reveal key features that impact the generation of a biased outcome [6]. With the knowledge gained from ISA, the local search algorithms used in search-based input generationr techniques can be enhanced. Firstly, knowledge of key bias-revealing features enables gives an indication of which features to focus on when perturbing data, possibly reducing computational costs while still maintaining the same or higher likelihood of generating a bias-revealing

Secondly, ISA can visualise the areas that have been unexplored in the 2D instance space and modify key feature values to populate those areas with a high likelihood of uncovering a biased outcome. Having a diverse test suite for testing will potentially increase the chances of revealing various forms of fairness bugs within the system.

IV. Research Project Plan

Before continuing to the research project plan, it is important to state that the focus of our research is for individual fairness, more specifically causal fairness.

A. Research Questions and Methodologies

To formally address the research gaps aforementioned, 3 research questions are constructed which we aim to answer in our paper. **Research question 1 (RQ1)**: How can we identify significant features of discriminatory instances? To address RQ1, our methodology begins with the retrieval of an existing dataset that inherently exhibits bias. We will also need to clearly define the four spaces ISA utilises to map instance features to the efficacy of test scenarios: test scenario space, T, contains all possible test instances; feature space, F, lists down all revelant features from test scenarios; fairness space, P, which defines the metric that is used to evaluate the fairness of the test scenarios; and instance space, I, which is the 2D projection of the meaningful features. The algorithm that is used by P to identify if an instance uncovers bias checks to see if perturbation of sensitive attribute value of the model under test produces a different outcome. In order to execute this metric, a naive decision tree is built and trained to represent as the system under test. Subsequently, we employ ISA to uncover significant features that induce a discriminatory outcome. To evaluate the efficacy of this approach, a support vector machine (SVM) model is trained to predict if an outcome is biased or not based on the selected key features. Recall, precision, and the F1 score are then calculated. These measures provide insight into the significance of the selected features concerning their impact on the outcome, with higher values indicating a more substantial influence. Having high results in each of the performance metrics would indicate that ISA has successfully captured key features that reveal fairness bugs.

Research question 2 (RQ2): How can we identify bias-revealing inputs more effectively? To address RQ2, we gather multiple state-of-the-art search-based test input generation techniques such as Aequitas, KOSEI and SG. We then seamlessly integrate ISA to help guide the local search phase for each of these methodologies, thereby harnessing its capabilities to steer the test input generation process exclusively based on bias-revealing key features. Within SG's local search phase, it uses a ranking scheme to determine test case generation [25]. With ISA integration, we can modify the ranking scheme to filter the possible test cases to the ones that modify the key features. Due to the similar in approaches between Aequitas and KOSEI, ISA integration will guide their perturbation methods to only change key feature values [3], [24]. Afterwards, the techniques are executed both with and without ISA integration, resulting in a collection of generated test suites. The following phase involves performing a comparative analysis of the test suites, aiming to ascertain the efficiency of each technique. In the context of RQ2, efficiency is defined by two key measurements: firstly, the proportion of instances within a test suite that trigger fairness bugs; secondly, the computational time required for test suite generation. This experiment serves to ascertain whether ISA integration facilitates the identification of a greater number of bias-revealing inputs within a similar time frame.

Research question 3 (RQ3): How can we generate a diverse range of bias-revealing test cases more effectively? To address RQ3, we continue on with the ISA-integration test input generators created from the previous experiment. We modify these generators further by incorporating the use of the mathematical boundary defined by ISA to generate instances near the edge of this border. Pushing the limits of our testing area can potentially reveal new discriminatory instances we have previously never seen before. In order to apply this method into practice, a search algorithm would need to be developed that combines the original

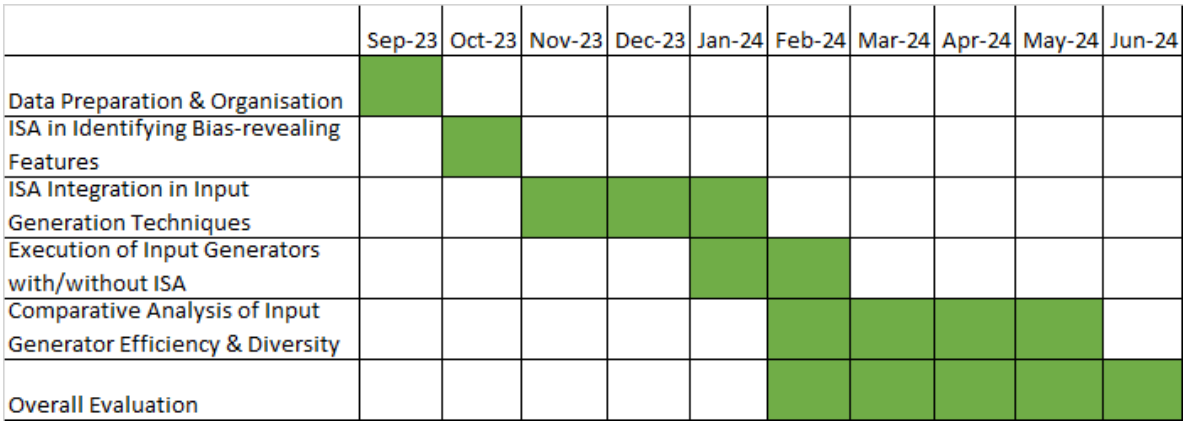


Fig. 5. Project Timeline

local search phase algorithm guided by bias-revealing features and considers the mathematical boundaries of the instance space. In the end, the results would be evaluated based on visual representation of the test suites under 2D projection space. Test suites that incorporate wide searches in the feature space while maintaining a high proportion of bias-revealing inputs would be considered better. Further research is required in order to mathematically represent the diversity of generated test suites.

B. Data Preparation and Organisation

Procurement and preparation of a biased dataset, along with the underlying code bases for Aequitas, KOSEI and SG test input generation techniques are required to begin our research. the developers of Aequitas and KOSEI have made their code bases openly accessible and programmable through Python. Conversely, the source code for the SG input generation technique is not publicly available. However, this challenge can be surmounted through careful examination of their paper , which offers comprehensive instructions regarding algorithms for dynamic symbol execution and utilisation of Python package lime for local explainability [25].

The dataset that will be under testing for each of our research questions will be ProPublica's Correctional Offender Management Profiling for Alternative Sanctions (COMPAS) Recidivism Risk Score Data [4]. This dataset contains 28 features providing information regarding risk assessment in criminal justice. Due to sheer size and number of feature of the data, a number of data pre-processing needs to be performed before utilising the dataset for our research. This involves appropriate handling of missing and null values in the dataset. At the core of each research question is the utilisation of ISA in extracting key features that uncover bias. However, before ISA is conducted, some feature selection is necessary to remove irrelevant and unnecessary features that we do not want to include when creating the instance space [6]. We focus on the influence non-protected attributes have on the biased outcome and so protected attributes are excluded from our feature space. There exists the phenomenon in which a feature is strongly correlated to another feature [33]. The mentioned feature becomes redundant as it does not provide any additional information to the model training and should therefore be excluded as well [33].

C. Limitations

There may be limitations to validating and evaluating the results of our dataset, especially in regards to the mathematically measuring the diversity of test inputs to answer RQ3. Relying on the visual representation of the test suite alone can prove to be unreliable and difficult to compare in situations where there are only marginal differences between 2D instance spaces.

D. Timeline

These fairness testing components are tested in various ways. Figure 5 dictates the timeline of our research project plan. 3 months is dedicates into integration of ISA into input generation techniques in order to provide time to understand the code bases for Aequitas and KOSEI, and most importantly reconstruct the method utilised by SG. Overall evaluation refers to the writing of the research paper as well as the validation of the entire results and confirming if our research questions have been appropriately answered.

V. Conclusion

This review aimed to conduct an comprehensive overview of current research in fairness definitions, fairness testing workflows, fairness testing components and instance space analysis in an attempt to understand the potential of applying ISA in enhancing and diversifying test input generation for fairness testing. The review identified a gap in the literature, exposing the unexplored area of utilising key bias-revealing features to assist in guiding test input generation as well as reduce computational time spent on testing redundant features. The literature showed ISA can be applicable within search-based input generation techniques as it requires a dataset of features and the biased outcome label generated from global phase in order to perform extraction of key features that most significantly reveal fairness bugs. It is proposed that the utilisation of ISA to define discriminatory features and expand our testing areas will lead to an improvement in search-based techniques Aequitas, KOSEI and SG. Furthermore, there is additional research opportunities to investigate the impact of key bias-revealing feature information in improving fairness repair algorithms, specifically the re-weighting of only bias-revealing classes to minimise model restricting and maintaining model performance.

REFERENCES

[1] Y. Brun and A. Meliou, "Software fairness," in *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ser. ESEC/FSE 2018. New York, NY, USA: Association for Computing Machinery, 2018, p. 754–759. [Online]. Available: https://doi.org/10.1145/3236024.3264838

[2] Z. Chen, J. M. Zhang, M. Hort, F. Sarro, and M. Harman, "Fairness testing: A comprehensive survey and analysis of trends," *arXiv preprint arXiv:2207.10223*, 2022.

[3] S. Udeshi, P. Arora, and S. Chattopadhyay, "Automated directed fairness testing," in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, 2018, pp. 98–108.

[4] P. Saleiro, B. Kuester, L. Hinkson, J. London, A. Stevens, A. Anisfeld, K. T. Rodolfa, and R. Ghani, "Aequitas: A bias and fairness audit toolkit," *arXiv preprint arXiv:1811.05577*, 2018.

[5] S. Verma and J. Rubin, "Fairness definitions explained," in *Proceedings of the international workshop on software fairness*, 2018, pp. 1–7.

[6] K. Smith-Miles and M. A. Muñoz, "Instance space analysis for algorithm testing: Methodology and software tools," *ACM Computing Surveys*, vol. 55, no. 12, pp. 1–31, 2023.

[7] M. J. Kusner, J. Loftus, C. Russell, and R. Silva, "Counterfactual fairness," *Advances in neural information processing systems*, vol. 30, 2017.

[8] E. Dai and S. Wang, "Say no to the discrimination: Learning fair graph neural networks with limited sensitive attribute information," in *Proceedings of the 14th ACM International Conference on Web Search and Data Mining*, 2021, pp. 680–688.

[9] M. Hardt, E. Price, and N. Srebro, "Equality of opportunity in supervised learning," *Advances in neural information processing systems*, vol. 29, 2016.

[10] C. Dwork, M. Hardt, T. Pitassi, O. Reingold, and R. Zemel, "Fairness through awareness," in *Proceedings of the 3rd innovations in theoretical computer science conference*, 2012, pp. 214–226.

[11] S. Galhotra, Y. Brun, and A. Meliou, "Fairness testing: testing software for discrimination," in *Proceedings of the 2017 11th Joint meeting on foundations of software engineering*, 2017, pp. 498–510.

[12] M. Zhang, J. Sun, J. Wang, and B. Sun, "Testsgd: Interpretable testing of neural networks against subtle group discrimination," *ACM Transactions on Software Engineering and Methodology*, 2022.

[13] R. Angell, B. Johnson, Y. Brun, and A. Meliou, "Themis: Automatically testing software for discrimination," in *Proceedings of the 2018 26th ACM Joint meeting on european software engineering conference and symposium on the foundations of software engineering*, 2018, pp. 871–875.

[14] S. Barocas and A. D. Selbst, "Big data's disparate impact," *California law review*, pp. 671–732, 2016.

[15] J. M. Zhang, M. Harman, L. Ma, and Y. Liu, "Machine learning testing: Survey, landscapes and horizons," *IEEE Transactions on Software Engineering*, vol. 48, no. 1, pp. 1–36, 2020.

[16] J. Chakraborty, S. Majumder, and T. Menzies, "Bias in machine learning software: Why? how? what to do?" in *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2021, pp. 429–440.

[17] J. Chakraborty, S. Majumder, Z. Yu, and T. Menzies, "Fairway: a way to build fair ml software," in *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2020, pp. 654–665.

[18] Y. Li, L. Meng, L. Chen, L. Yu, D. Wu, Y. Zhou, and B. Xu, "Training data debugging for the fairness of machine learning software," in *Proceedings of the 44th International Conference on Software Engineering*, 2022, pp. 2215–2227.

[19] M. Wick, J.-B. Tristan *et al.*, "Unlocking fairness: a trade-off revisited," *Advances in neural information processing systems*, vol. 32, 2019.

[20] S. Biswas and H. Rajan, "Fair preprocessing: towards understanding compositional fairness of data transformers in machine learning pipeline," in *Proceedings of the 29th ACM joint meeting on European software engineering conference and symposium on the foundations of software engineering*, 2021, pp. 981–993.

[21] M. Hort, J. M. Zhang, F. Sarro, and M. Harman, "Fairea: A model behaviour mutation approach to benchmarking bias mitigation methods," in *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2021, pp. 994–1006.

[22] M. E. Khan and F. Khan, "A comparative study of white box, black box and grey box testing techniques," *International Journal of Advanced Computer Science and Applications*, vol. 3, no. 6, 2012.

[23] M. Nejadgholi and J. Yang, "A study of oracle approximations in testing deep learning libraries," in *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2019, pp. 785–796.

[24] S. Sano, T. Kitamura, and S. Takada, "An efficient discrimination discovery method for fairness testing," in *34th International Conference on Software Engineering and Knowledge Engineering, SEKE 2022*. Knowledge Systems Institute Graduate School, 2022, pp. 200–205.

[25] A. Agarwal, P. Lohia, S. Nagar, K. Dey, and D. Saha, "Automated test generation to detect individual discrimination in ai models," *arXiv preprint arXiv:1809.03260*, 2018.

[26] A. Sharma and H. Wehrheim, "Automatic fairness testing of machine learning models," in *Testing Software and Systems: 32nd IFIP WG 6.1 International Conference, ICTSS 2020, Naples, Italy, December 9–11, 2020, Proceedings 32*. Springer, 2020, pp. 255–271.

[27] A. Sharma, C. Demir, A.-C. N. Ngomo, and H. Wehrheim, "Mlcheck–property-driven testing of machine learning classifiers," in *2021 20th IEEE International Conference on Machine Learning and Applications (ICMLA)*. IEEE, 2021, pp. 738–745.

[28] K. Smith-Miles, D. Baatar, B. Wreford, and R. Lewis, "Towards objective measures of algorithm performance across instance space," *Computers & Operations Research*, vol. 45, pp. 12–24, 2014.

[29] A. Aleti *et al.*, "Identifying safety-critical scenarios for autonomous vehicles via key features," *arXiv preprint arXiv:2212.07566*, 2022.

[30] A. Aleti and M. Martinez, "E-apr: Mapping the effectiveness of automated program repair," *arXiv preprint arXiv:2002.03968*, 2020.

[31] C. Oliveira, A. Aleti, L. Grunske, and K. Smith-Miles, "Mapping the effectiveness of automated test suite generation techniques," *IEEE Transactions on Reliability*, vol. 67, no. 3, pp. 771–785, 2018.

[32] A. Aggarwal, P. Lohia, S. Nagar, K. Dey, and D. Saha, "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*, 2019, pp. 625–635.

[33] K. Yan and D. Zhang, "Feature selection and analysis on correlated gas sensor data with recursive feature elimination," *Sensors and Actuators B: Chemical*, vol. 212, pp. 353–363, 2015.