

iFlipper: Label Flipping for Individual Fairness

Hantian Zhang*

Georgia Institute of Technology
hantian.zhang@cc.gatech.edu

Ki Hyun Tae*

KAIST

kihyun.tae@kaist.ac.kr

Jaeyoung Park

KAIST

jypark@kaist.ac.kr

Xu Chu

Georgia Institute of Technology
xu.chu@cc.gatech.edu

Steven Euijong Whang

KAIST

swhang@kaist.ac.kr

ABSTRACT

As machine learning becomes prevalent, mitigating any unfairness present in the training data becomes critical. Among the various notions of fairness, this paper focuses on the well-known individual fairness, which states that similar individuals should be treated similarly. While individual fairness can be improved when training a model (in-processing), we contend that fixing the data before model training (pre-processing) is a more fundamental solution. In particular, we show that *label flipping is an effective pre-processing technique for improving individual fairness*.

Our system iFlipper solves the optimization problem of minimally flipping labels given a limit to the individual fairness violations, where a violation occurs when two similar examples in the training data have different labels. We first prove that the problem is NP-hard. We then propose an approximate linear programming algorithm and provide theoretical guarantees on how close its result is to the optimal solution in terms of the number of label flips. We also propose techniques for making the linear programming solution more optimal without exceeding the violations limit. Experiments on real datasets show that iFlipper significantly outperforms other pre-processing baselines in terms of individual fairness and accuracy on unseen test sets. In addition, iFlipper can be combined with in-processing techniques for even better results.

ACM Reference Format:

Hantian Zhang, Ki Hyun Tae, Jaeyoung Park, Xu Chu, and Steven Euijong Whang. 2023. iFlipper: Label Flipping for Individual Fairness. In *SIGMOD '23, June 20–25, 2023, Seattle, USA*. ACM, New York, NY, USA, 20 pages. <https://doi.org/10.1145/1122445.1122456>

1 INTRODUCTION

Machine learning (ML) impacts our everyday lives where applications include recommendation systems [12], job application [15], and face recognition [10]. Unfortunately, ML algorithms are also known to reflect or even reinforce bias in the training data and thus make unfair decisions [6, 46]. This issue draws concerns from

*Equal contribution and co-first authors

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

SIGMOD '23, Seattle, USA

© 2023 Association for Computing Machinery.

ACM ISBN 978-1-4503-XXXX-X/18/06...\$15.00

<https://doi.org/10.1145/1122445.1122456>

both the public and research community, so algorithms have been proposed to mitigate bias in the data and improve the fairness of ML models.

There are several prominent notions of fairness, and we focus on individual fairness [17], which states that *similar individuals must be treated similarly*. Suppose that two applicants are applying to the same school. If the two applicants have similar application materials, then it makes sense for them to obtain the same or similar outcomes. Likewise if two individuals are applying for a loan and have similar financial profiles, it is fair for them to be accepted or rejected together as well. In addition to individual fairness, the other prominent fairness notions include group fairness and causal fairness. Group fairness [3, 50, 53] focuses on the parity between two different sensitive groups (e.g., male versus female), and causal fairness [28] looks at fairness from a causal perspective (e.g., does gender affect an outcome?). These are orthogonal notations of fairness and do not capture the individual fairness we focus on.

How can one improve a model's individual fairness? There are largely three possible approaches: fixing the data before model training (pre-processing), changing the model training procedure itself (in-processing), or updating the model predictions after training (post-processing). Among them, most of the literature focuses on in-processing [43, 48, 49] and more recently pre-processing techniques [29, 30, 52]. We contend that pre-processing is important because biased data is the root cause of unfairness, so fixing the data is the more fundamental solution rather than having to cope with the bias during or after model training. The downside of pre-processing is that one cannot access the model and has to address the bias only using the training data. Due to this challenge, only a few pre-processing techniques for individual fairness have been proposed, which we will compare with in the experiments.

We propose *label flipping as a way to mitigate data bias for individual fairness* and assume a binary classification setting where labels have 0 or 1 values. Given a training set of examples, the idea is to change the labels of some of the examples such that similar examples have the same labels as much as possible. Which examples are considered similar is application dependent and non-trivial and can be learned from input data [22, 33, 45] or obtained from annotators (e.g., humans). This topic is important for individual fairness, but not the focus of this work where we assume the criteria is given as an input. For example, one may compute the Euclidean distance between two examples and consider them similar if the distance is within a certain threshold. As the training set becomes more individually fair, the trained model becomes fairer as well (see Section 4.2). Our label flipping approach is inspired by the robust training literature of learning from noisy labels [42] where

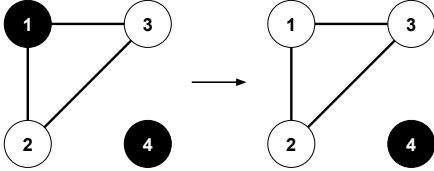


Figure 1: Label flipping example for individual fairness using a graph representation. Two similar nodes have an edge, and color indicates the label. By flipping the label of node 1, all pairs of similar nodes have the same label, which is considered individually fair.

the labeling itself may be imperfect. The standard approach for handling such labels is to ignore or fix them [41]. In our setting, we consider any label that reduces individual fairness as biased and would like to fix it.

We can use a graph representation to illustrate label flipping as shown in Figure 1. Each node represents a training example, and its color indicates the original label (black indicates label 1, and white indicates 0). Two similar nodes (defined in Section 2.1) are connected with an edge, and a violation occurs when an edge is connecting two nodes with different labels. In Figure 1, there are four nodes where only the nodes 1, 2, and 3 are similar to each other. Each edge also has an associated weight, which reflects the similarity of the two nodes. For simplicity, let us assume that all weights are 1. We can see that there are two “violations” of fairness in this dataset: (1,2) and (1,3) because there are edges between them, and they have different colors. After flipping the label of node 1 from 1 to 0, we have no violations.

Our key contribution is in formulating and solving a constrained label flipping problem for the purpose of individual fairness. Just like in a robust training setup, we assume labelers can make labeling mistakes in a biased fashion. Here the effective solution is to debias the labels by flipping them. One consequence of label flipping is an accuracy-fairness trade-off where the model’s accuracy may diminish. As an extreme example, if we simply flip all the labels to be 0, then a trained model that only predicts 0 is certainly fair, but inaccurate to say the least. Even if we carefully flip the labels, we still observe a trade-off as we detail in Section 4.2. We thus formulate the optimization problem where the objective is to minimize the number of label flipping while limiting the *total error*, which is the total degree of the individual fairness violations (see Definition 2). The optimization can be formally stated as an instance of mixed-integer quadratic programming (MIQP) problem, and we prove that it is NP-hard. We then transform the problem to an approximate linear programming (LP) problem for efficient computation. Interestingly, we show that our LP algorithm has theoretical guarantees on how close its result is to the optimal solution in terms of the number of flips performed. We then further optimize the solution given by the LP algorithm to reduce the number of flips while ensuring the total error does not exceed the given limit.

We call our approach iFlipper and empirically show how its label flipping indeed results in individually-fair models and significantly outperforms other pre-processing baselines on real datasets. In particular, the state-of-the-art baselines [29, 30] use representation learning to place similar examples closer in a feature space. We show that iFlipper has better accuracy-fairness trade-off curves and

is also significantly more efficient. We also compare iFlipper with baselines (e.g., greedy approach and k-means clustering) for solving the optimization problem and show that iFlipper is superior. Finally, we demonstrate how iFlipper can be integrated with in-processing techniques [48] for even better results. We release our code as a community resource [1].

The rest of the paper is organized as follows.

- We formulate the label flipping optimization as an MIQP problem and prove that it is NP-hard (Section 2).
- We propose iFlipper, which solves this problem by converting it into an approximate LP algorithm that has theoretical guarantees and present an additional optimization to further improve the solution given by the LP solver (Section 3).
- We evaluate iFlipper on real datasets and show how it outperforms other pre-processing baselines and can be integrated with in-processing techniques for better results (Section 4).

2 PROBLEM DEFINITION

2.1 Preliminaries

We focus on a *binary classification* setting, and assume a training dataset $D = \{(x_i, y_i)\}_{i=1}^n$ where x_i is an example, and y_i is its label having a value of 0 or 1. A binary classifier h can be trained on D , and its prediction on a test example x is $h(x)$.

Individual Fairness [17] states that similar individuals must be treated similarly. The criteria for determining if two examples are similar depends on the application, and we assume it is given as input, though research on how to automatically discover such similarity measures exists [22, 33, 45]. For example, for each x , we may consider all the examples within a certain distance to be similar. Alternatively, we may consider the top- k nearest examples to be similar. In our work, we assume as input a given similarity measure, which can be applied on the training set to produce a similarity matrix $W_{ij} \in \mathbb{R}^{n \times n}$ where $W_{ij} > 0$ if and only if x_i and x_j are deemed similar according to the measure. If $W_{ij} > 0$, we set it to be the similarity of x_i and x_j , although any other positive value can be used as well. In order to satisfy individual fairness, we introduce the notion of *individual fairness violation* as follows.

DEFINITION 1. Individual Fairness Violation. Given a similarity matrix W on a training set, an individual fairness violation occurs when $W_{ij} > 0$, but $y_i \neq y_j$. The magnitude of the violation is defined to be W_{ij} .

DEFINITION 2. Total Error. Given a similarity matrix W , the total error is the sum of all W_{ij} values where $y_i \neq y_j$.

Our goal is to reduce the total error to be within a maximum allowed amount.

DEFINITION 3. *m*-Individually Fair Dataset. Given a similarity matrix W , a dataset is considered *m*-individually fair if the total error is at most m .

A smaller m naturally translates to an *m*-individually fair dataset that is likely to result in a more individually fair model on the unseen test set as we demonstrate in Section 4.2.

Incomplete Similarity Matrix. If we only have partial knowledge of the similarities, iFlipper can be naturally adapted to utilize or

fix the partial data. The straightforward solution is to treat the partial similarity as if it is complete information where the pairs with unknown similarities are not included in our optimization formulation. Another solution is to “extrapolate” the partial similarity matrix to reconstruct the full similarity matrix. For example, one can learn similarity functions using the partial data. In this case, iFlipper’s performance will largely depend on how accurate the learned similarity function is.

Evaluating the Fairness of a Trained Model h . To evaluate the final individual fairness of a trained model h , we compute the widely-used *consistency score* [30] of model predictions on an unseen test set as defined below. Here W_{ij} is the similarity matrix on the unseen test set.

$$\text{Consistency Score} = 1 - \frac{\sum_i \sum_j |h(x_i) - h(x_j)| \times W_{ij}}{\sum_i \sum_j W_{ij}}$$

Intuitively, if the model is trained on an individually-fair dataset, the predictions on the test set between similar individuals tend to be the same, so the consistency score on the test set increases. In the extreme case, a consistency score of 1 indicates that all similar pairs of test examples get the same predictions from the model.

Local Sensitive Hashing. We construct a similarity matrix W using locality sensitive hashing (LSH) [4] instead of materializing it entirely and thus avoid performing $O(n^2)$ comparisons. We exploit the fact that W is sparse, because most examples are dissimilar. This approach is similar to blocking techniques in entity resolution [13].

2.2 Label Flipping Optimization Problem

We define the label flipping optimization problem for individual fairness. Given a training dataset D and a limit m of total error allowed, our goal is to flip the minimum number of labels in D such that it becomes *m -individually fair*. This statement can be formalized as a mixed-integer quadratic programming (MIQP) problem:

$$\begin{aligned} (\text{MIQP}) \quad & \min \quad \sum_{i=1}^n (y_i - y'_i)^2 \\ \text{s.t.} \quad & \sum_{i=1}^n \sum_{j=1}^n W_{ij} (y_i - y_j)^2 \leq m \\ & y_i \in \{0,1\}, \forall i \end{aligned} \tag{1}$$

where y_i indicates an output label, and y'_i is its original value. Intuitively, we count the number of flips ($(y_i - y'_i)^2 = 1$) while ensuring that the total error ($(y_i - y_j)^2 = 1$ and $W_{ij} > 0$) is within the limit m . We call a solution *feasible* if it satisfies the error constraints in Equation 1, but may or may not be optimal.

MIQP is an NP-hard problem in general, and we prove that our specific instance of the MIQP problem is also NP-hard.

Theorem 1. *The MIQP problem in Equation 1 is NP-hard.*

The full proof for Theorem 1 is in Section A. The key idea is to reduce the well-known NP-hard *at most k -cardinality s-t cut* problem [9] to our MIQP problem.

2.3 Baseline Algorithms

Our key contribution is to propose algorithms for the label flipping problem that not only scales to large datasets, but also provides

feasible and high-quality solutions. We present three naïve algorithms that are efficient, but may fail to produce feasible solutions. In comparison, our method in Section 3 always produces feasible solutions with theoretical guarantees.

Greedy Algorithm. The greedy algorithm repeatedly flips labels of nodes that reduce the total error the most. The algorithm terminates if the total error is m or smaller, or if we cannot reduce the error anymore. For example, suppose we start from the graph in Figure 1 where there are initially two violations (recall we assume that $W_{ij} = 1$ for all edges for simplicity) and set $m = 1$. We need to determine which label leads to the largest reduction in error when flipped. We can see that flipping node 1 will reduce the total error by 2 while flipping the other nodes does not change the total error. Hence, the greedy algorithm flips node 1 to reduce the total error to 0 and then terminates. While the greedy approach seems to work for Figure 1, in general it does not always find an optimal result and may fail to produce a feasible solution even if one exists. Consider the example in Figure 2 where there is one violation between nodes 2 and 3. Again, we assume that $W_{ij} = 1$ for all edges for simplicity. If we set $m = 0$, the feasible solution is to flip nodes 1 and 2 or flip nodes 3 and 4 together. Unfortunately, the greedy algorithm immediately terminates because it only flips one node at a time, and no single flip can reduce the error.

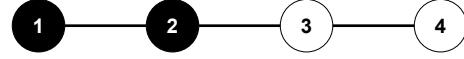


Figure 2: A graph where the greedy algorithm fails to find a feasible solution with zero violations.

The computational complexity of the greedy algorithm is $O(n^2)$ because we flip at most $O(n)$ labels, and each time a label is flipped, we update the total error of each neighbor of the node.

Gradient Based Algorithm. The second naïve approach is to use a Lagrangian multiplier to move the error constraint into the objective function and solve the problem via gradient descent. This approach is common in machine learning, and the following equation shows the problem setup:

$$\begin{aligned} (\text{Lagrangian}) \quad & \min \quad \sum_{i=1}^n (y_i - y'_i)^2 + \lambda \sum_{i=1}^n \sum_{j=1}^n W_{ij} (y_i - y_j)^2 \\ & y_i \in [0,1], \forall i \end{aligned} \tag{2}$$

where λ is a hyperparameter that controls the trade-off between fairness and accuracy. A higher λ favors better fairness. Although this gradient based algorithm is efficient, it shares the same problem as the greedy algorithm where it may get stuck in a local minima and thus fail to find a feasible solution.

Clustering Based Algorithm. The third approach is to use a clustering algorithm for clustering examples and assigning the same label to each cluster. We use k-means as a representative clustering algorithm. If $k = 1$, then all examples will have the same label, and we can simply choose the majority label of the examples. If k is set correctly, and the clustering is perfect, then only the similar examples will be clustered together. However, this approach is unsuitable for our problem, which is to flip the minimum number of labels to have at most m total error. Reducing the total error to 0 is not the primary goal as there may be a large degradation in

accuracy. To cluster for our purposes, one would have to adjust k to find the clusters with just the right amount of total error, but this fine-tuning is difficult as we show in Section 4.4.

3 IFLIPPER

We explain how iFlipper converts the MIQP problem into an approximate linear program (LP) problem and produces a feasible solution with theoretical guarantees. The conversion is done in two steps: from MIQP to an equivalent integer linear program (ILP) using linear constraints (Section 3.1) and from the ILP problem to an approximate LP problem (Section 3.2). We present iFlipper’s algorithm to solve the approximate LP problem and show why its result always leads to a feasible solution (Section 3.3). We then provide theoretical guarantees on how far the result is from the optimal solution of the ILP problem, and propose a reverse-greedy approach to further optimize the solution (Section 3.4). Finally, we present iFlipper’s overall workflow with a complexity analysis (Section 3.5).

3.1 From MIQP to Equivalent ILP

We convert the MIQP problem to an equivalent ILP problem. We first replace the squared terms in Equation 1 to absolute terms:

$$\begin{aligned} \min \quad & \sum_{i=1}^n |y_i - y'_i| \\ \text{s.t.} \quad & \sum_{i=1}^n \sum_{j=1}^n W_{ij} |y_i - y_j| \leq m \\ & y_i \in \{0,1\}, \forall i \end{aligned} \quad (3)$$

The resulting formulation is equivalent to the original MIQP because y_i and y'_i have binary values.

To convert this problem into an equivalent ILP problem, we replace each absolute term with an XOR expression, which can be expressed as four linear constraints. For two binary variables x and y , one can easily see that $|x - y| = x \text{ XOR } y$. Also, each expression $z = x \text{ XOR } y$ is known to be equivalent to the following four linear constraints: $z \leq x + y$, $z \geq y - x$, $z \geq x - y$, and $z \leq 2 - x - y$. For example, if $x = 1$ and $y = 1$, z is bounded by $z \leq 2 - x - y$ and is thus 0. For other combinations of x and y , z will be bounded to its correct XOR value as well. We thus introduce the auxiliary variables z_i and z_{ij} to represent $y_i \text{ XOR } y'_i$ and $y_i \text{ XOR } y_j$, respectively, and obtain the following integer linear programming (ILP) problem:

$$\begin{aligned} (\text{ILP}) \quad \min \quad & \sum_{i=1}^n z_i \\ \text{s.t.} \quad & \sum_{i=1}^n \sum_{j=1}^n W_{ij} z_{ij} \leq m \\ & y_i, z_i \in \{0,1\}, \forall i, \quad z_{ij} \in \{0,1\}, \forall i, j \\ & z_i - y_i \leq y'_i, \quad z_i - y_i \geq -y'_i \\ & z_i + y_i \geq y'_i, \quad z_i + y_i \leq 2 - y'_i \\ & z_{ij} - y_i - y_j \leq 0, \quad z_{ij} - y_i + y_j \geq 0 \\ & z_{ij} + y_i - y_j \geq 0, \quad z_{ij} + y_i + y_j \leq 2 \end{aligned} \quad (4)$$

Since the ILP problem is equivalent to the MIQP problem, we know it is NP-hard as well. In the next section, we convert the

ILP problem to an approximate linear program (LP), which can be solved efficiently.

3.2 From ILP to Approximate LP

We now relax the ILP problem to an approximate LP problem. At this point, one may ask why we do not use existing solvers like CPLEX [14], MOSEK [32], and Gurobi [20], which also use LP relaxations. All these approaches repeatedly solve LP problems using branch-and-bound methods and have time complexities exponential to the number of variables in the worst case. In Section 4.4, we demonstrate how slow the ILP solvers are. Our key contribution is finding a near-exact solution to the original ILP problem by solving the LP problem *only once*.

We first replace the integer constraints in Equation 4 with range constraints to obtain the following LP problem:

$$\begin{aligned} (\text{LP}) \quad \min \quad & \sum_{i=1}^n z_i \\ \text{s.t.} \quad & \sum_{i=1}^n \sum_{j=1}^n W_{ij} z_{ij} \leq m \\ & y_i, z_i \in [0,1], \forall i, \quad z_{ij} \in [0,1], \forall i, j \\ & z_i - y_i \leq y'_i, \quad z_i - y_i \geq -y'_i \\ & z_i + y_i \geq y'_i, \quad z_i + y_i \leq 2 - y'_i \\ & z_{ij} - y_i - y_j \leq 0, \quad z_{ij} - y_i + y_j \geq 0 \\ & z_{ij} + y_i - y_j \geq 0, \quad z_{ij} + y_i + y_j \leq 2 \end{aligned} \quad (5)$$

Now the violation between two points becomes the product of the weight of the edge W_{ij} and the absolute difference between the two labels $z_{ij} = |y_i - y_j|$. Although this problem can be solved more efficiently than the ILP problem, its result cannot be used as is because of the continuous values. Hence, we next convert the result into a binary solution that is close to the optimal solution of the ILP problem. A naive method is to round the continuous values to their nearest integers, but this does not guarantee a feasible solution.

Example 1. Suppose we start from the graph in Figure 3a. Just like previous examples, we again assume that W_{ij} is 1 for simplicity. The total error in Figure 3a is 4, and we set $m = 2$. Solving the LP problem in Equation 5 can produce the solution in Figure 3b where the labels of nodes 1 and 4 are flipped from 1 to 0.5 (gray color). One can intuitively see that the labels are minimally flipped while the total error is exactly 2. However, rounding the labels changes the two 0.5’s back to 1’s as shown in Figure 3c, resulting in an infeasible solution, because the total error becomes 4 again.

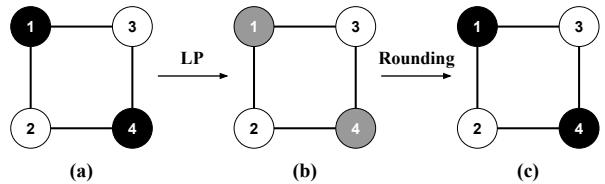


Figure 3: Performing simple roundings on the optimal solution’s values of the LP problem may result in an infeasible solution for the ILP problem.

3.3 Constructing a Feasible Solution

We now explain how to construct a feasible integer solution for the ILP problem (Equation 4) from an optimal solution of the LP problem (Equation 5). We first prove a surprising result that any optimal solution y^* can be converted to another optimal solution \check{y} where all of its variables have one of the three values: 0, 1, or some $\alpha \in (0, 1)$. That is, α is between 0 and 1, but not one of them. Next, we utilize this property and propose an adaptive rounding algorithm that converts \check{y} into a feasible solution whose values are only 0's and 1's.

Optimal Solution Conversion. We prove the following lemma for converting an optimal solution to our desired form.

Lemma 2. *Given an optimal solution y^* of Equation 5, we can always convert y^* to a new solution \check{y} where \check{y} is also optimal and only has 0, 1, or a unique $\alpha \in (0, 1)$ as its values.*

PROOF. Suppose there are at least two distinct values α and β that are not 0 or 1 in y^* . We can combine all the y^* nodes whose value is α to one node while maintaining the same total error because an α - α edge has no violation (see Figure 4 for an illustration). We call this collection an α -cluster. Likewise, we can generate a β -cluster.

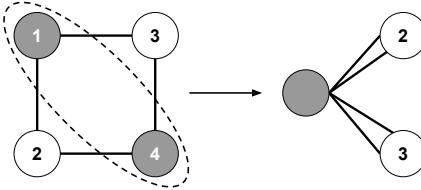


Figure 4: Continuing from Figure 3, recall that the optimal LP solution has nodes 1 and 4 with 0.5 labels. We can combine nodes 1 and 4 to construct a 0.5-cluster connected to nodes 2 and 3 with four edges as shown on the right. The total error is still 2.

Without loss of generality, let us assume that $0 < \alpha < \beta < 1$, and the sum of the pairwise node similarities between the two clusters is E . Suppose an α -cluster and a β -cluster have A_0 and B_0 nodes whose initial labels are 0, respectively, and A_1 and B_1 nodes whose initial values are 1, respectively. Let $N_\alpha = A_0 - A_1$ and $N_\beta = B_0 - B_1$. Now suppose there are U nodes connected to the α -cluster by an edge $W_{a_i\alpha}$ and V nodes connected to the β -cluster by an edge $W_{b_i\beta}$, whose values satisfy

$$0 \leq a_1 \leq \dots \leq a_k < \alpha < a_{k+1} \leq \dots \leq a_U \leq 1 \text{ and}$$

$$0 \leq b_1 \leq \dots \leq b_l < \beta < b_{l+1} \leq \dots \leq b_V \leq 1.$$

Note that there is no connected node with a value of α or β by construction. Let $S_\alpha = \sum_{i=1}^k W_{a_i\alpha} - \sum_{i=k+1}^U W_{a_i\alpha}$ and $S_\beta = \sum_{i=1}^l W_{b_i\beta} - \sum_{i=l+1}^V W_{b_i\beta}$. Let us also add the following nodes for convenience: $a_0 = 0$, $a_{U+1} = 1$, $b_0 = 0$, and $b_{V+1} = 1$. We can then reduce at least one unique non-0/1 value in y^* , by either changing α to a_k or a_{k+1} , or changing β to b_l or b_{l+1} , or changing both α and β to the same value, while keeping the solution optimal using the following Lemmas 2.1 and 2.2. The key idea is that at least one of the conversions allows the solution to have the same number of label flippings and the same or even smaller amount of total

Algorithm 1: iFlipper’s LP solution conversion algorithm.

Input: Optimal solution y^* for the LP problem, similarity matrix W
Output: Transformed optimal solution \check{y} where each value is one of $\{0, \alpha, 1\}$

```

1  $\check{y} = y^*$ ;
   // There are  $T$  unique non-0/1 values in  $y^*$ 
2  $T = \text{GETNUMCLUSTERS}(y^*)$ ;
3 while  $T > 1$  do
4    $\text{ZeroClusterList} = \text{GETZEROCLUSTERS}(\check{y})$ ;
5   for  $\alpha$  in  $\text{ZeroClusterList}$  do
6     // Details are in Algorithm 2
7      $\text{TRANSFORMWITHONECLUSTER}(\check{y}, \alpha, W)$ ;
8      $T = T - 1$ ;
9   if  $T > 1$  then
10     $\alpha, \beta = \text{GETNONZEROTWOCLOUDERS}(\check{y})$ ;
     // Details are in Algorithm 2
11     $\text{TRANSFORMWITHTWOCLUSTERS}(\check{y}, \alpha, \beta, W)$ ;
12     $T = \text{GETNUMCLUSTERS}(\check{y})$ ;
   // There is at most one unique non-0/1 value in  $\check{y}$ 
13 return  $\check{y}$ ;

```

error, keeping the solution optimal. The conversion process only depends on N_α , S_α , N_β , S_β , and E . The complete proof and detailed conditions for each case are given in Sections B and C.

Lemma 2.1. *For an α -cluster with $N_\alpha = 0$ in the optimal solution, we can always convert α to either a_k or a_{k+1} while maintaining an optimal solution. As a result, we can reduce exactly one non-0/1 value in the optimal solution.*

Lemma 2.2. *For an α -cluster with $N_\alpha \neq 0$ and a β -cluster with $N_\beta \neq 0$ in the optimal solution, we can always convert (α, β) to one of $(a_k, \beta + \frac{N_\alpha}{N_\beta}(a_k - \alpha))$, $(a_{k+1}, \beta - \frac{N_\alpha}{N_\beta}(a_{k+1} - \alpha))$, $(\alpha + \frac{N_\beta}{N_\alpha}(\beta - b_l), b_l)$, $(\alpha - \frac{N_\beta}{N_\alpha}(b_{l+1} - \beta), b_{l+1})$, or $(\frac{\alpha N_\alpha + \beta N_\beta}{N_\alpha + N_\beta}, \frac{\alpha N_\alpha + \beta N_\beta}{N_\alpha + N_\beta})$, while maintaining an optimal solution. As a result, we can reduce at least one non-0/1 value in the optimal solution.*

We can repeat this adjustment until the solution \check{y} has at most one unique value α that is neither 0 nor 1. \square

Based on Lemma 2, Algorithm 1 shows how to convert the optimal LP solution y^* to another optimal solution \check{y} whose values are in $\{0, \alpha, 1\}$. We first obtain all the unique non-0/1 values in y^* and for each value v construct a v -cluster. We then pick a cluster (say the α -cluster) with $N_\alpha = 0$ and change α using the $\text{TRANSFORMWITHONECLUSTER}$ function, which implements the converting process of Lemma 2.1 (see Section B for details) until there are no more such clusters (Lines 4–7). Among the rest of the clusters, we choose two (say the α -cluster and β -cluster) and transform them using the $\text{TRANSFORMWITHTWOCLUSTERS}$ function, which implements the combining process of Lemma 2.2 described in Section C (Lines 8–11). We repeat these steps until there is at most one non-0/1 value in \check{y} . The details of the $\text{TRANSFORMWITHONECLUSTER}$ and $\text{TRANSFORMWITHTWOCLUSTERS}$ functions are shown in Algorithm 2.

Algorithm 2: Transformation functions in Algorithm 1.

```

1 Function TRANSFORMWITHONECLUSTER( $y, \alpha, W$ ):
    // Replace all  $\alpha$  with  $\alpha_{new}$  to reduce one
    // unique non-0/1 value
2    $a_k, a_{k+1}, S_\alpha = \text{GETONECLUSTERINFO}(y, \alpha, W);$ 
3   if  $S_\alpha \leq 0$  then
4     |  $\alpha \leftarrow a_{k+1};$ 
5   else
6     |  $\alpha \leftarrow a_k;$ 
    // Now  $\alpha \in \{a_k, a_{k+1}\}$ 
7 Function TRANSFORMWITHTWOCLUSTERS( $y, \alpha, \beta, W$ ):
    // Replace all  $(\alpha, \beta)$  with  $(\alpha_{new}, \beta_{new})$  using
    // one of the five cases in Lemma 2.2 to
    // reduce at least one unique non-0/1 value
8    $a_k, a_{k+1}, N_\alpha, S_\alpha, b_l, b_{l+1}, N_\beta, S_\beta, E =$ 
    GETTWOCLUSTERSINFO( $y, \alpha, \beta, W$ );
9    $X, Y = \frac{N_\alpha}{N_\beta}, \frac{(S_\alpha-E)N_\beta - (S_\beta+E)N_\alpha}{N_\beta};$ 
    // Details are in our technical report [2]
10  if  $X < 0, Y \leq 0$  then
11    | if  $X + 1 \leq 0$  then
12      | |  $\alpha \leftarrow \alpha + \min(a_{k+1} - \alpha, -\frac{N_\beta}{N_\alpha}(b_{l+1} - \beta));$ 
13    else
14      | |  $\alpha \leftarrow \alpha + \min(a_{k+1} - \alpha, -\frac{N_\beta}{N_\alpha}(b_{l+1} - \beta), \frac{N_\beta(\beta-\alpha)}{N_\alpha+N_\beta});$ 
15      | |  $\beta \leftarrow \beta - \frac{N_\alpha}{N_\beta}\epsilon_\alpha;$ 
16  else if  $X < 0, Y > 0$  then
17    | if  $X + 1 \geq 0$  then
18      | |  $\alpha \leftarrow \alpha - \min(\alpha - a_k, -\frac{N_\beta}{N_\alpha}(\beta - b_l));$ 
19    else
20      | |  $\alpha \leftarrow \alpha - \min(\alpha - a_k, -\frac{N_\beta}{N_\alpha}(\beta - b_l), -\frac{N_\beta(\beta-\alpha)}{N_\alpha+N_\beta});$ 
21      | |  $\beta \leftarrow \beta + \frac{N_\alpha}{N_\beta}\epsilon_\alpha;$ 
22  else if  $X > 0, Y \leq 0$  then
23    | |  $\alpha \leftarrow \alpha + \min(a_{k+1} - \alpha, \frac{N_\beta}{N_\alpha}(\beta - b_l), \frac{N_\beta(\beta-\alpha)}{N_\alpha+N_\beta});$ 
24    | |  $\beta \leftarrow \beta - \frac{N_\alpha}{N_\beta}\epsilon_\alpha;$ 
25  else if  $X > 0, Y > 0$  then
26    | |  $\alpha \leftarrow \alpha - \min(\alpha - a_k, \frac{N_\beta}{N_\alpha}(b_{l+1} - \beta));$ 
27    | |  $\beta \leftarrow \beta + \frac{N_\alpha}{N_\beta}\epsilon_\alpha;$ 
    // Now  $(\alpha, \beta)$  is one of the cases in Lemma 2.2

```

Example 2. To illustrate Algorithm 1, consider Figure 3a again where $m = 2$. Say we obtain an optimal solution from an LP solver as shown in Figure 5a where nodes 1 and 4 have the labels $\alpha = 0.1$ and $\beta = 0.9$, respectively, while the other nodes have 0 labels. This solution uses one flip and has a total error of 2. We first construct an α -cluster and a β -cluster where each cluster only contains a single node, and there is no edge between them as shown in Figure 5b. We then convert (α, β) using the TRANSFORMWITHTWOCLUSTERS. As a result, we set (α, β) to $(0.5, 0.5)$ and reduce one non-0/1 unique value in the solution. The solution now has only one non-0/1 value (i.e., 0.5) and still has a total error of 2 while using one flip.

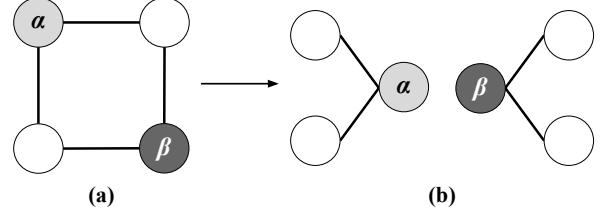


Figure 5: Cluster construction example for Algorithm 1.

Algorithm 3: iFlipper's adaptive rounding algorithm.

```

Input: Transformed optimal solution  $\check{y}$  for the LP problem
        whose values are in  $\{0, \alpha, 1\}$ , similarity matrix  $W$ 
Output: Feasible binary integer solution  $\bar{y}$ 
        // Get the total similarity values of  $(0-\alpha)$  and
         $(1-\alpha)$  label pairs
1  $M_{0\alpha}, M_{1\alpha} = \text{GETSIMILARITYSUMOFPAIRS}(\check{y}, W);$ 
2 if  $M_{0\alpha} \leq M_{1\alpha}$  then
3   |  $\bar{\alpha} \leftarrow 1;$ 
4 else
5   |  $\bar{\alpha} \leftarrow 0;$ 
    // Replace  $\alpha$  with  $\bar{\alpha} \in \{0, 1\}$ 
6  $\bar{y} = \text{APPLYROUNDING}(\check{y}, \bar{\alpha});$ 
7 return  $\bar{y};$ 

```

Adaptive Rounding into a Binary Solution. We now convert \check{y} into a feasible integer solution with only 0's and 1's. If we use simple rounding to change α to 0 or 1 as in the naïve method in Section 3.2, the resulting integer solution may not be feasible like the example in Figure 3c. Fortunately, the fact that we only have three possible values allows us to propose an adaptive rounding algorithm (Algorithm 3) that guarantees feasibility. We first denote M_{ab} as the sum of similarity values of $(a-b)$ label pairs with edges. For example, $M_{0\alpha}$ is the sum of similarities of similar node pairs whose labels are 0 and α , respectively. Then the key idea is to round α to 1 if the $(1-\alpha)$ label pairs have a higher similarity sum than that of the $(0-\alpha)$ label pairs (i.e., $M_{0\alpha} \leq M_{1\alpha}$), and round α to 0 otherwise. For example, consider Figure 3b again. Here $\alpha = 0.5$, $M_{0\alpha} = 4$, and $M_{1\alpha} = 0$. We have $M_{0\alpha} > M_{1\alpha}$, so rounding α to 0 will result in a feasible solution with zero violations.

We prove the correctness of Algorithm 3 in Lemma 3.

Lemma 3. Given an optimal solution \check{y} of Equation 5 where each value is one of $\{0, \alpha, 1\}$, applying adaptive rounding (Algorithm 3) always results in a feasible solution.

PROOF. The total error in \check{y} can be expressed as a function of α :

$$\begin{aligned} \text{Total Error} &= \sum_{i=1}^n \sum_{j=1}^n W_{ij} \check{z}_{ij} = \sum_{i=1}^n \sum_{j=1}^n W_{ij} |\check{y}_i - \check{y}_j| \\ &= f(\alpha) = M_{01} + \alpha M_{0\alpha} + (1 - \alpha) M_{1\alpha} \leq m \end{aligned} \quad (6)$$

In addition, the error constraint in Equation 5 is satisfied because \check{y} is an optimal solution.

We now want to round α to either 0 or 1 while satisfying the error constraint. If $M_{0\alpha} \leq M_{1\alpha}$, we have $f(1) = M_{01} + M_{0\alpha} \leq M_{01} + \alpha M_{0\alpha} + (1 - \alpha) M_{1\alpha} \leq m$ ($\because 0 < \alpha < 1$), so setting $\alpha = 1$

guarantees a feasible solution. On the other hand, if $M_{1\alpha} < M_{0\alpha}$, we have $f(0) = M_{01} + M_{1\alpha} < M_{01} + \alpha M_{0\alpha} + (1 - \alpha)M_{1\alpha} \leq m$ ($\because 0 < \alpha < 1$), so setting $\alpha = 0$ ensures feasibility. \square

Lemma 3 shows that the rounding choice of α depends on how $M_{0\alpha}$ and $M_{1\alpha}$ compare instead of α 's value itself. This result explains why simple rounding as in the naïve method does not necessarily lead to a feasible solution.

In the next section, we analyze how accurate the rounded solution is compared to the optimal solution of the ILP problem and investigate whether it can be further improved.

3.4 Optimality and Improvement

We analyze the optimality of Algorithm 3 and propose a technique called *reverse greedy* for further optimizing it without exceeding the total error limit.

Optimality Analysis. We prove the theoretical bounds of Algorithm 3 in Lemma 4:

Lemma 4. *For a given optimal solution \bar{y} of Equation 5, let us denote N_{ab} as the number of nodes in \bar{y} where the label a was flipped to b . Then the objective value of the output \bar{y} from Algorithm 3 is at most C more than the optimal objective value of the original ILP problem where the value of C depends on α :*

$$\begin{aligned}\alpha = 1 \rightarrow C &= (1 - \alpha)(N_{0\alpha} - N_{1\alpha}) \\ \alpha = 0 \rightarrow C &= \alpha(N_{1\alpha} - N_{0\alpha})\end{aligned}\quad (7)$$

PROOF. Since the optimal solution of the ILP problem is always one of the feasible solutions of the LP problem, the optimal objective values (OPT) of the two optimization problems should satisfy:

$$OPT_{LP} \leq OPT_{ILP} \quad (8)$$

The objective value of \bar{y} can then be expressed as follows:

$$OPT_{LP}(\bar{y}) = \sum_{i=1}^n \bar{z}_{ij} = (N_{01} + N_{10}) + \alpha N_{0\alpha} + (1 - \alpha)N_{1\alpha} \quad (9)$$

We first consider the case where $\alpha = 1$. Here the objective value of \bar{y} is $OPT_{LP}(\bar{y}) = (N_{01} + N_{10}) + N_{0\alpha}$. We can then derive the bound on $OPT_{LP}(\bar{y})$ from Equation 8 and Equation 9 as follows:

$$\begin{aligned}OPT_{LP}(\bar{y}) - OPT_{LP}(\bar{y}) &\leq OPT_{ILP} - OPT_{LP}(\bar{y}) \\ (1 - \alpha)(N_{1\alpha} - N_{0\alpha}) &\leq OPT_{ILP} - OPT_{LP}(\bar{y}) \\ OPT_{LP}(\bar{y}) &\leq OPT_{ILP} + (1 - \alpha)(N_{1\alpha} - N_{0\alpha})\end{aligned}\quad (10)$$

We note that $(1 - \alpha)(N_{1\alpha} - N_{0\alpha})$ is always non-negative because \bar{y} is an optimal solution, i.e., $OPT_{LP}(\bar{y}) \leq OPT_{LP}(\bar{y})$.

Similarly, if $\alpha = 0$, we can obtain the objective value bound of $\alpha(N_{1\alpha} - N_{0\alpha})$. \square

Reverse Greedy Algorithm. Lemma 4 shows that the bound may sometimes be loose because it depends on α and the number of nodes whose labels are flipped to α . There is a possibility that Algorithm 3 flips nodes unnecessarily, which results in a total error smaller than m (see Section 4.5 for empirical results). Hence, we propose an algorithm called *reverse greedy* (Algorithm 4), which unflips flipped labels as long as the total error does not exceed m . As the name suggests, we run the greedy algorithm in Section 2.2 in the reverse direction where for each iteration, we unflip nodes that

Algorithm 4: iFlipper's reverse greedy algorithm.

Input: Feasible binary integer solution \bar{x} , total error limit m
Output: Improved binary integer solution \tilde{x}

```

1  $\tilde{x} = \bar{x};$ 
2  $flipped\_labels = GETFLIPPEDLABEL(\tilde{x});$ 
3  $total\_error = GETTOTALERROR(\tilde{x});$ 
4 while  $total\_error \leq m$  do
    // Unflip flipped labels of the nodes so that
    // the total error is increased the least
    5  $\tilde{x} = FLIPLABELLEASTVIOLATION(\tilde{x}, flipped\_labels);$ 
    6  $flipped\_labels = GETFLIPPEDLABEL(\tilde{x});$ 
    7  $total\_error = GETTOTALERROR(\tilde{x});$ 
8 return  $\tilde{x};$ 

```

increase the error the least to recover the original labels as much as possible. Thus, reverse greedy can only improve the optimality of Algorithm 3.

Example 3. Continuing from our example using Figure 3 and $m = 2$, suppose we run Algorithm 3 on Figure 3b and round the 0.5 labels to 0 to obtain the feasible solution in Figure 6a. A total of two flips are performed compared to the initial graph Figure 3a. However, there exists an optimal solution like Figure 6b where only one flip is necessary. Running Algorithm 4 will unflip nodes 1 or 4, and unflipping node 1 produces this solution.

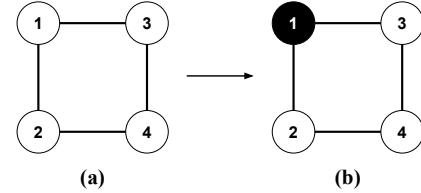


Figure 6: The reverse greedy algorithm can further optimize a rounded solution from Algorithm 3.

The time complexity of reverse greedy is the same as the greedy algorithm, i.e., $O(n^2)$, but in practice requires fewer iterations than greedy because the total error is relatively close to m .

3.5 Putting Everything Together

We now describe the overall workflow of iFlipper. For a given ILP problem, we first convert it into an approximate LP problem. We then solve the approximate LP problem and convert its optimal solution to another optimal solution that only has values in $\{0, \alpha, 1\}$ using Algorithm 1. We then apply adaptive rounding using Algorithm 3. Finally, we possibly improve the rounded solution by unflipping labels using Algorithm 4.

Complexity Analysis. iFlipper consists of four parts: solving the approximate LP problem, the converting algorithm, the adaptive rounding algorithm, and the reverse-greedy algorithm. The interior-points algorithm for solving LP has a time complexity of $O(n^{2+\epsilon})$, where ϵ is smaller than 1. The converting algorithm has a time complexity of $O(n^2)$: in the worst case, we may need to combine clusters n times, and for each combining process, we need to check all neighbors of a particular cluster, which is up to n points. For the adaptive rounding algorithm, the time complexity is also $O(n^2)$,

because we need to count the number of $(1, \alpha)$ edges and $(0, \alpha)$ edges, which is $O(n^2)$ in the worst case. The reverse greedy algorithm, as analyzed in Section 3.4, is $O(n^2)$. Overall, the complexity is bounded by the LP solver, which is $O(n^{2+\epsilon})$. This result may look worse than the greedy algorithm, but iFlipper runs much faster than greedy in practice. The reason is that we use efficient LP solvers, and the empirical running times of the adaptive rounding and reverse greedy algorithms are much faster than their theoretical worst case complexities. We show the empirical runtimes in Section 4.

4 EXPERIMENTS

In this section, we evaluate iFlipper on real datasets and address the following key questions:

- Is there an accuracy-fairness trade-off for iFlipper?
- How does iFlipper compare with various baselines in terms of model accuracy, individual fairness, and efficiency?
- How useful is each component of iFlipper?
- Does the LP solution conversion run correctly?
- Can iFlipper be integrated with in-processing techniques?

We implement iFlipper in Python and use Scikit-learn [36] and PyTorch [35] libraries for model training. For performing optimization, we use two software packages: CPLEX [14] and MOSEK [32]. We run all experiments on Intel Xeon Gold 5115 CPUs.

4.1 Setting

Datasets. We experiment on the following three popular datasets in the fairness literature. We randomly split each dataset into training, test, and validation sets. We then use the same feature preprocessing as in IBM’s AI Fairness 360 toolkit [7] where examples with missing values are discarded. See Table 1 for more details.

- COMPAS [5]: Contains 6,167 people examples and is used to predict criminal recidivism rates. The features include gender, race, and criminal history.
- AdultCensus [25]: Contains 45,222 people examples and is used to predict whether an individual’s income exceeds \$50K per year. The features include gender, age, and occupation.
- Credit [16]: Contains 1,000 examples and is used to predict an individual’s credit risk. The features contain race, credit amount, and credit history.
- Synthetic [51]: We generate 200,000 samples with two attributes (x_1, x_2) and a binary label y following a process introduced in [51]. Tuples with a positive label $(x_1, x_2, y = 1)$ are drawn from a Gaussian distribution and, tuples with a negative label $(x_1, x_2, y = 0)$ are drawn from another Gaussian distribution.

We also considered two additional fairness datasets: Bank [31] and LSAC [47]. However, these datasets exhibit fewer individual fairness issues as shown in Section E. Hence, we do not include them in our main experiments because the fairness improvements are not as significant.

Similarity Matrices. We consider two similarity matrices using Euclidean distance (i.e., $d(x_i, x_j) = \|x_i - x_j\|^2$). When measuring distances, we follow previous approaches [29, 52] and exclude sensitive attributes like gender and race. The rationale is that individuals who have similar information other than the sensitive

Dataset	# Train	# Test	# Valid	Sen. Attr	k	T	θ
COMPAS	3,700	1,850	617	gender	20	3	0.05
AdultCensus	27,133	13,566	4,523	gender	20	3	0.1
Credit	700	201	199	age	20	7	0.05
Synthetic	100,000	60,000	40,000	N/A ¹	20	3	0.05

Table 1: Settings for the four datasets.

attributes values must be treated similarly. We quantify the similarity as $W_{ij} = e^{-\theta d(x_i, x_j)}$ where $\theta > 0$ is a scale parameter. Our design follows a related work on individual fairness that also uses similarity matrices [37]. See Table 1 for the configurations.

- *kNN-based:* Considers (x_i, x_j) as a similar pair if x_i is one of x_j ’s nearest neighbors or vice versa:

$$W_{ij} = \begin{cases} e^{-\theta d(x_i, x_j)} & \text{if } x_i \in NN_k(x_j) \text{ or } x_j \in NN_k(x_i) \\ 0 & \text{otherwise} \end{cases}$$

where $NN_k(x_j)$ denotes the set of k nearest examples of x_j in a Euclidean space.

- *threshold-based:* Considers (x_i, x_j) as a similar pair if their distance is smaller than a threshold T :

$$W_{ij} = \begin{cases} e^{-\theta d(x_i, x_j)} & \text{if } d(x_i, x_j) \leq T \\ 0 & \text{otherwise} \end{cases}$$

We use the FALCONN LSH library [4] to efficiently construct both similarity matrices. For all datasets, we set the LSH hyperparameters (number of hash functions and number of hash tables) to achieve a success probability of 0.98 on a validation set where 98% of similar pairs of examples are in the same buckets.

There are two important choices to make for constructing the similarity matrix: the similarity/distance function (e.g., Euclidean) and similarity/distance thresholds (e.g., k and T). The similarity function depends on the user’s domain knowledge on what similarity means when defining individual fairness. For example, if the user thinks each feature contributes the same to individual fairness, then Euclidean distance would be a good choice, otherwise the users can consider distances like Mahalanobis distance. The thresholds can be tuned based on the following trade-off: a more relaxed threshold makes the similarity matrix denser, which slows down the optimizations as more edges need to be considered. Hence, the goal is to keep enough information about the similarities, but ignore enough edges with small weights to make iFlipper run fast enough.

Measures. We evaluate a model’s accuracy by computing the portion of correctly-predicted data points. We evaluate individual fairness using the consistency score [30] defined in Section 2.1, which measures the consistency of predictions on the test set between similar individuals. For both scores, higher values are better. We report mean values over five trials for all measures.

Hyperparameter Tuning. iFlipper provides a total error limit hyperparameter m , which impacts the trade-off between the model accuracy and fairness where a lower m results in better fairness, but worse accuracy (see Section 4.3.1). Given a desired consistency score, we can use binary search to adjust m by comparing the consistency score of the trained model with the desired score.

¹There is no sensitive attribute to exclude when measuring distances.

Methods Compared. We compare iFlipper with the following existing pre-processing algorithms for individual fairness:

- *LFR* [52]: A fair representation learning algorithm that optimizes between accuracy, group fairness, and individual fairness in terms of data reconstruction loss.
- *iFair* [29]: A fair representation learning algorithm that optimizes accuracy and individual fairness. Compared to LFR, iFair does not optimize group fairness, which enables the classifier to have better individual fairness. If two data points are close to each other in the input space, iFair aims to map them close to each other in the feature space as well.
- *PFR* [30]: A fair representation learning algorithm that tries to learn an embedding of the fairness graph. The fairness graph used in PFR is similar to the one used in iFlipper. Similar to iFair, PFR optimizes individual fairness while preserving the original data by mapping similar individuals to nearby points in the learned representation. Among the three baselines, only PFR is able to support a general similarity matrix. PFR uses an efficient trace optimization algorithm, which can learn representations much faster than iFair.

For all baselines, there are multiple hyperparameters that can balance the competing objectives. We start with the same sets of hyperparameters as described in their papers, and tune them to provide the best results.

Optimization Solutions Compared. We compare iFlipper with the following optimization baselines described in Sections 2.3 and 3.1.

- *Greedy*: Flips labels that reduce the total error the most.
- *Gradient*: Solves an unconstrained optimization problem.
- *kMeans*: Applies k-means clustering and, for each cluster, make its examples have the majority label.
- *ILP Solver*: Solves the ILP problem exactly using CPLEX [14], which is a state-of-the-art solver.

For *Gradient* and *kMeans*, we tune λ and k , respectively, to satisfy the total error limit. We use CPLEX to solve ILP problems as it is faster than MOSEK. When solving LP problems in iFlipper, however, CPLEX turns out to be much slower than MOSEK due to its long construction time, so we use MOSEK instead.

An interesting behavior of MOSEK is that empirically all of its optimal solutions only contain values in $\{0, \alpha, 1\}$ without having to run our Algorithm 1. Note that this behavior is limited to the label flipping problems we are solving and does not necessarily occur for other LP problems. We suspect that MOSEK’s algorithm effectively contains the functionality of Algorithm 1. We also make the same observations when using CPLEX for our LP problems. We find these results encouraging because it means that Algorithm 1 is not a significant runtime overhead and can even be tightly integrated with the LP solving code. A detailed analysis of MOSEK’s code is an interesting future work. For any other solver that does not have this behavior, one can always run Algorithm 1 on its solutions. Hence to make sure our Algorithm 1 is correct, we separately evaluate it in Section 4.6.

Model Setup. We use logistic regression (LR), random forest (RF), and neural network (NN) models for our experiments. We tune the model hyperparameters (e.g., learning rate and maximum depth of the tree) such that the model has the highest validation accuracy.

Our goal is to pre-process the training data to make these models train fairly.

4.2 Accuracy and Fairness Trade-off

Figure 7 shows the trade-off between fairness and accuracy for the COMPAS dataset when running iFlipper with different allowed amounts of total error. The x-axis is accuracy, and the y-axis is the consistency score. There are two curves, which correspond to two different similarity matrices: kNN-based and threshold-based. Each data point on the curve has two other results: (total error after repairing, number of flips). As we flip more labels, there is less total error in the repaired dataset, and the resulting model has a higher consistency score on the test set, which means it has better individual fairness. However, the accuracy drops as we flip more labels. The trade-off curves of the other datasets are similar and shown in Section D

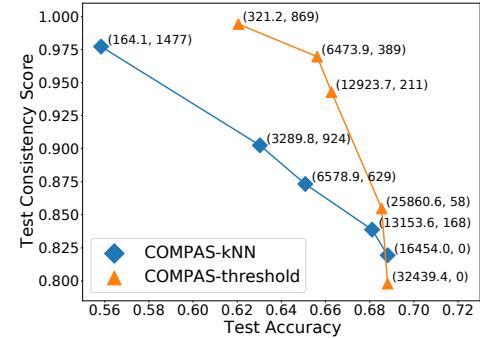


Figure 7: Accuracy-fairness trade-off curves for iFlipper. For each data point, we also show the total error after repairing and the number of flips in parentheses.

4.3 Performance and Runtime Results

We now compare iFlipper with other baselines using the three datasets and two similarity matrices.

4.3.1 Accuracy and Fairness Comparison. We first compare the accuracy and fairness results of iFlipper with the other methods. Figure 8 shows the trade-off results with logistic regression where the x-axis is the accuracy, and the y-axis is the consistency score on a test set. *Original* is where we train a model on the original data with no data pre-processing. For LFR, iFair, and PFR, we employ 20, 45, and 15 different hyperparameter sets, respectively, as described in their papers. For iFlipper, we use 10–11 different m values to control accuracy and fairness. For a clear visualization, we omit some points in the bottom left region of the graph for methods other than iFlipper. As a result, iFlipper significantly outperforms the baselines in terms of both test accuracy and consistency score for all cases, which demonstrates that our label flipping approach is effective in improving individual fairness while maintaining high accuracy. We also observe that iFlipper performs better than the three baselines on both similarity matrices. This result is important because the similarity matrix may vary depending on the application. The results for a random forest and neural network are in Section H, and the key trends are similar to Figure 8 where iFlipper shows a better trade-off than the baselines. We note that the advantage of iFlipper compared to the baselines is relatively less clear

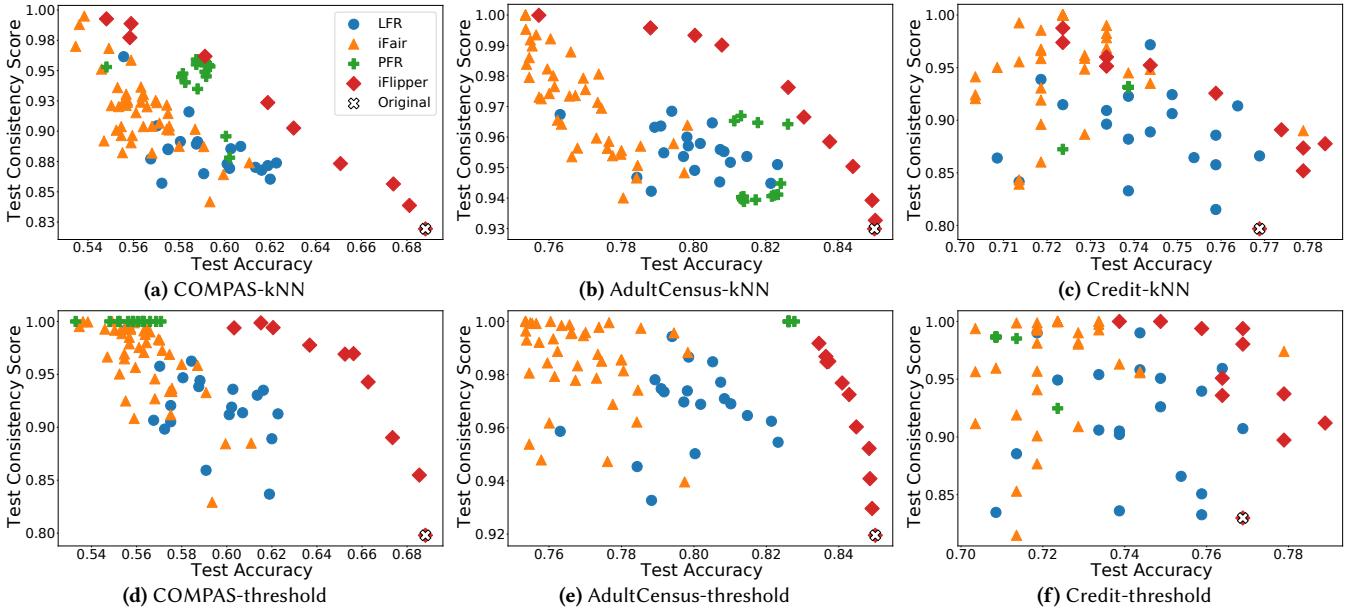


Figure 8: Accuracy-fairness trade-offs of logistic regression on the three datasets using the two similarity matrices. In addition to the four methods LFR, iFair, PFR, and iFlipper, we add the result of model training without any pre-processing and call it “Original.” As a result, only iFlipper shows a clear accuracy and fairness trade-off.

Dataset	Adult-kNN			Adult-threshold			
	Model	LR	RF	NN	LR	RF	NN
(C. Score)	(0.94)	(0.90)	(0.90)	(0.95)	(0.83)	(0.95)	
LFR	.815	.827	.806	.821	.827	.806	
iFair	.797	.820	.805	.798	.820	.806	
PFR	.826	.808	.844	.826	.808	.829	
iFlipper	.844	.851	.848	.845	.850	.845	

Table 2: Accuracy comparison of methods with similar individual fairness on different models. In order to align the individual fairness, for each model we make the methods have similar consistency scores on the validation data (C. Score) by tuning their hyperparameters.

on the Credit dataset mainly because the data is small, making the variance of the results higher.

Table 2 shows a more detailed comparison with the baselines using the AdultCensus dataset. For each ML model, we fix the consistency score (denoted as “C. Score”) to be some value as shown below each model name in Table 2 in parentheses. Then for each method, we report the test accuracy when we tune its hyperparameter such that the trained model has the highest validation accuracy while achieving the fixed consistency score on the validation set. As a result, iFlipper consistently shows the highest test accuracy compared to the other baselines for all models.

In comparison to the other baselines, iFlipper’s accuracy-fairness trade-off is also much cleaner as shown in Figure 8 where the other baselines have noisy trade-offs and even overlapping results for different hyperparameter sets. The benefit of having a clean trade-off is that iFlipper can easily balance between accuracy and fairness to obtain a desired fairness by varying the total error limit m . In Figure 8, we also observe that iFlipper is flexible and can obtain a wide range of test consistency scores, all the way up to nearly 1.0.

Datasets	Sim. Matrix	Avg. Runtime (sec)			
		LFR	iFair	PFR	iFlipper
COMPAS	kNN	786	29,930	0.41	5.70
	threshold	786	29,930	0.35	8.69
Adult-Census	kNN	700	21,321†	15.62	140
	threshold	700	21,321†	14.98	685
Credit	kNN	22.78	394	0.01	1.74
	threshold	22.78	394	0.01	0.68

Table 3: Runtime results of LFR, iFair, PFR, and iFlipper on the COMPAS, AdultCensus, and Credit datasets. For each method, we show the average runtime for all experiments in Figure 8. The symbol † indicates that we reduce the size of the training data for better efficiency.

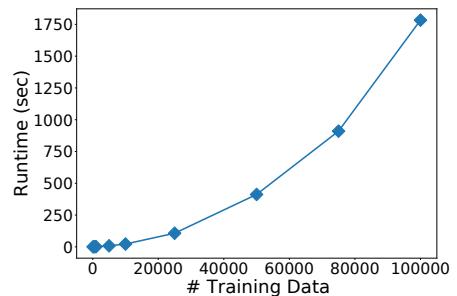


Figure 9: Runtime results of iFlipper on synthetic datasets.

4.3.2 Runtime Comparison. We evaluate the efficiency of iFlipper and the three baselines in Table 3. For each method, we show the average runtime for all experiments in Figure 8. We note that the runtimes of LFR and iFair do not depend on the similarity matrix.

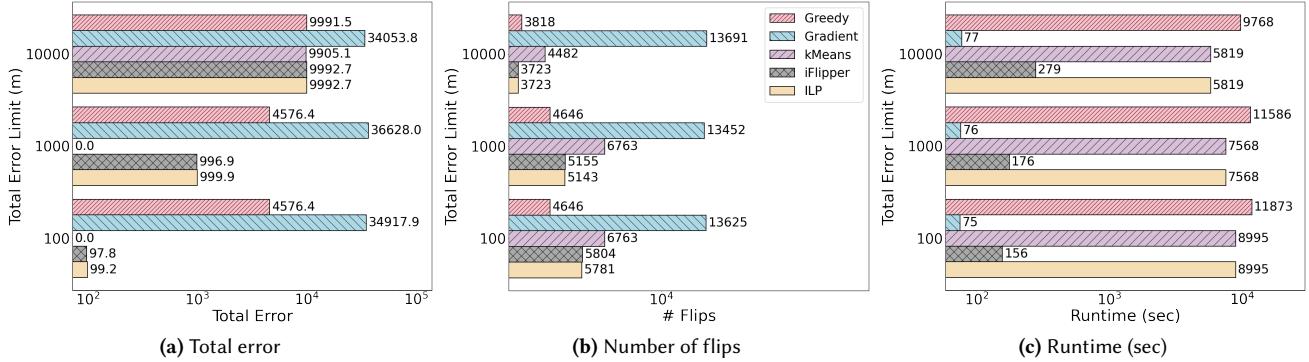


Figure 10: A detailed comparison of iFlipper against the three naïve optimization solutions (*Greedy*, *Gradient*, and *kMeans*) and ILP solver on the AdultCensus dataset where we use the kNN-based similarity matrix. Here the initial amount of total error is 65,742.5. We show the results for three different total error limits (m).

For the iFair experiments on the AdultCensus dataset, we lower the training data using uniform sampling to fit iFair because fitting iFair on the entire data takes more than 24 hours. We confirm that this relaxation does not hurt the original performance reported in the iFair paper [29]. As a result, iFlipper is much faster than LFR and iFair for all cases. Although PFR seems to be the fastest, it performs much worse than iFlipper in terms of accuracy and fairness as shown in Figure 8.

Another observation is that as the dataset size increases, iFlipper’s runtime increases following the time complexity analysis in Section 3.5. For a detailed evaluation, we conduct the experiment using random subsets of the synthetic dataset. For each experiment, we set the total error limit m to 20% of the initial amount of total error. Figure 9 shows runtime results of iFlipper on datasets of different sizes. As a result, we observe a quadratic increase in running time as the size of the training data increases, which is consistent with our theoretical analysis.

4.4 Optimization Solution Comparison

We now compare iFlipper with other optimization solutions: *Greedy*, *Gradient*, *kMeans*, and an ILP solver. Figure 10 makes a comparison in terms of optimality and efficiency on the AdultCensus dataset where we use the kNN-based similarity matrix. We set the total error limit m to three different values for a detailed comparison. Note that all three subfigures within Figure 10 use the same legends.

We first compare the resulting total error of the methods in Figure 10a. Obviously, the optimal solution from the ILP solver is the closest to the total error limit. We observe that iFlipper never exceeds the target. On the other hand, both *Greedy* and *Gradient* are unable to find a feasible solution in some cases. For example, *Greedy* and *Gradient* cannot reduce the total error to less than 4,576.4 and 34,053.8, respectively. This result is expected because these methods may fall into local minima as we explained in Section 2.3. Also, *kMeans* does find feasible solutions, but flips too many labels unnecessarily. For example, for $m = 100$ and $m = 1,000$, *kMeans* has zero violations using more flips than iFlipper, which defeats the purpose of the optimization altogether.

We also compare the number of flips in Figure 10b. As a result, iFlipper produces solutions that are close to the optimal ones. When $m = 100$ and $1,000$, *Greedy* shows the fewest number of label flips

because it fails to find a feasible solution. *Gradient* has the largest number of flips. We suspect this poor performance is due to the fact that *Gradient* (1) solves an unconstrained optimization problem that makes it hard to satisfy the limit on total error, and (2) provides continuous values, which introduces rounding errors. *kMeans* also performs worse than iFlipper because its clustering cannot be fine-tuned to have close to m total error.

Finally, we compare the average runtimes (i.e., wall clock times in seconds) of each method in Figure 10c. As a result, iFlipper is much faster than the ILP solver as it solves an approximate LP problem only once. In addition, *Greedy* and *kMeans* are slower than iFlipper because they use many iterations to reduce the total error and adjust the number of clusters, respectively. *Gradient* is the most efficient, but the result is not meaningful because the total error and the numbers of flips are even worse than *Greedy*.

We also perform the above experiments on the COMPAS dataset, and the results are similar (see Section F).

4.5 Ablation Study

We perform an ablation study in Figure 11 to demonstrate the effectiveness of each component in iFlipper using the AdultCensus dataset and kNN-based similarity matrix. The results for the COMPAS dataset are similar and shown in Section G. As we explained in Section 4.1, MOSEK always produces an optimal LP solution that only has values in $\{0, \alpha, 1\}$, so we do not have an ablation study without the LP solution conversion. Instead, we separately evaluate the conversion algorithm in Section 4.6. Again, the fact that MOSEK effectively contains the conversion functionality indicates that the conversion overhead is not significant and that a tight integration with the LP solving is possible. We thus compare iFlipper (LP solver with both adaptive rounding and reverse greedy algorithms) with the following variants: (1) LP solver with simple rounding (LP-SR); and (2) LP solver with adaptive rounding (LP-AR). We also consider an optimal solution from the ILP solver to compare the optimality of each solution.

Figure 11a shows that the adaptive rounding algorithm always provides a feasible solution while simple rounding to a fractional optimal solution results in an infeasible solution as we discussed in Lemma 3. However, the rounding algorithm does flip more labels than the optimal solution as shown in Figure 11b, resulting in

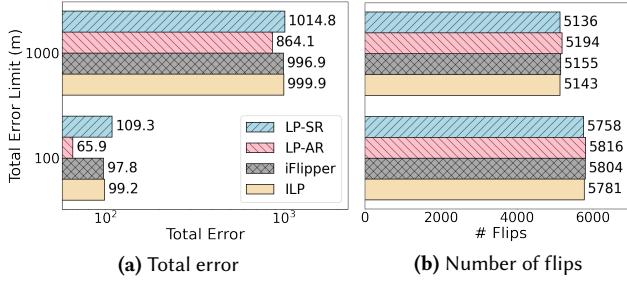


Figure 11: Ablation study for iFlipper on the AdultCensus dataset and kNN-based similarity matrix.

an unintended fairness improvement and accuracy drop. In this case, the reverse greedy algorithm in iFlipper is used to reduce the optimality gap with the optimal solution by recovering as many original labels as possible without exceeding the total error limit.

Table 4 shows the average runtime of each component (LP solver, adaptive rounding, and reverse greedy) in iFlipper in Figure 11. As a result, the runtime for solving the LP problem is dominant, which demonstrates that iFlipper is able to provide a near-exact solution with minimal time overhead.

Method	Avg. Runtime (sec)
LP Solver (effectively includes Alg. 1)	153.61
+ Adaptive Rounding (Alg. 3)	0.48
+ Reverse Greedy (Alg. 4)	13.26

Table 4: Avg. runtimes of iFlipper’s components in Figure 11.

4.6 Algorithm 1 Evaluation

We now evaluate the LP solution conversion (Algorithm 1) separately using the three datasets and kNN-based similarity matrices. For all datasets, we start with an LP problem solution by assigning random values and perform the conversion. As a result, we successfully convert random solutions to new solutions whose values are in $\{0, \alpha, 1\}$ where the exact value of α for each dataset is shown in Table 5. In addition, each converted solution always has the same number of label flips and a smaller total error. This result is consistent with Lemma 2, which says the conversion maintains the optimality of the original solution. Finally, the conversion is much faster than the overall runtime of iFlipper (see Table 3) even in this worst-case scenario where the labels are randomly initialized.

Dataset	α	Before Conv.		After Conv.		Time(s)
		Tot. Err.	# Flips	Tot. Err.	# Flips	
COMPAS	0.471	12645.1	1,838	1038.2	1,838	2.47
AdultCensus	0.494	93519.7	13,514	3164.4	13,514	93.63
Credit	0.448	2447.0	361	128.7	361	0.25

Table 5: iFlipper’s conversion algorithm (Algorithm 1) on LP problem solutions that have random labels.

4.7 Compatibility with In-processing Method

In this section, we demonstrate how iFlipper can be combined with an in-processing algorithm to further improve individual fairness.

We evaluate iFlipper with SenSR [48], which is an in-processing fair training algorithm that is robust to sensitive perturbations to the inputs, on the AdultCensus dataset. For iFlipper, we use the same distance function in SenSR, which computes the distance using the features that are not correlated to sensitive attributes and use the threshold $T = 2$ to construct the similarity matrix. For a fair comparison, we report both our consistency score and the GR-Con. (gender and race consistency) / S-Con. (spouse consistency) metrics used by SenSR to evaluate the individual fairness. Here Con. measures the consistency between $h(x_i)$ and $h(x_j)$ when x_i and x_j are the same except the sensitive attributes. To evaluate Con., we use the same method in SenSR where the test data examples are duplicated, but assigned different sensitive attribute values. As a result, Table 6 shows that using both methods gives the best individual fairness for both metrics while having little accuracy drop. Thus, iFlipper complements in-processing algorithms by removing the bias inherent in the data during the pre-processing step.

Dataset	Method	Test Acc.	Con. Score	GR/S-Con.
Adult- Census	Original	0.855	0.917	0.919 / 0.867
	iFlipper	0.853	0.955	0.931 / 0.907
	SenSR	0.836	0.953	0.990 / 0.945
	Both	0.829	0.960	0.992 / 0.984

Table 6: Using iFlipper, SenSR, or both on the AdultCensus dataset.

5 RELATED WORK

Various fairness measures have been proposed to capture legal and social issues [34, 44]. The prominent measures include individual fairness [17], group fairness [3, 50, 53], and causal fairness [28]. Individual fairness captures the notion that similar individuals should be treated similarly. Group fairness measures like equal opportunity [21], equalized odds [21], and demographic parity [18] ensure that two sensitive groups have similar statistics. Causal fairness identifies causal relationships among attributes. Although optimizing for a different objective function, the causality-based methods are often used to improve group or individual fairness as well. All these measures complement each other, and there is a known conflict between group fairness and individual fairness [8, 19] that they cannot be satisfied at the same time because of their different assumptions. Our primary focus is on individual fairness.

Recently, many unfairness mitigation techniques for individual fairness [7] have been proposed where they can be categorized into pre-processing [29, 30, 52], in-processing [43, 48, 49], and post-processing [37] techniques depending on whether the fix occurs before, during, or after model training, respectively. Among the categories, we focus on pre-processing because fixing the data can solve the root cause of unfairness.

The recent pre-processing works LFR [52], iFair [29], and PFR [30] all propose fair representation learning algorithms that optimize accuracy and individual fairness. Using an adjacency matrix that represents a fairness graph, the goal is to optimize for a combined objective function with reconstruction loss and individual fairness loss based on the fairness graph. In comparison, iFlipper takes the

alternative approach of flipping labels to fix bias, which results in better accuracy-fairness trade-offs and efficiency.

It is also important to understand the in-processing and post-processing techniques. SenSR [48] enforces the model to be invariant under certain sensitive perturbations to the inputs using adversarial learning. Several extensions have been proposed as well. SenSeI [49] enforces invariance on certain sensitive sets and uses a regularization-based approach for jointly optimizing accuracy and fairness, and BuDRO [43] extends the fairness loss used in SenSR to use gradient boosting. For post-processing, GLIF [37] formulates a graph smoothening problem and uses Laplacian regularization to enforce individual fairness. In comparison, iFlipper can complement all these techniques as we demonstrated with SenSR.

Pre-processing techniques for other fairness measures have been proposed as well. For group fairness, Kamiran et al. [23] and Calmon et al. [11] change features of the training data to remove dependencies between the sensitive attribute and label and thus achieve statistical parity. It is worth noting that Kamiran et al. [23] also proposed label flipping (called massaging) to reduce bias in the data for group fairness, but not for individual fairness, which is our focus. There is also a possibility of extending iFlipper to support group fairness. However, we do not believe iFlipper can directly support group fairness with the current optimization objective, so an interesting direction is to develop new label flipping techniques that can support group fairness notions beyond statistical parity by formulating the right optimization problems.

For causal fairness, Capuchin [40] makes a connection between multivalued dependencies (MVDs) and causal fairness, and proposes a data repair algorithm for MVDs using tuple inserts and deletes that also ensures causal fairness.

Finally, there is a line of recent work on data programming [38, 39], which generates probabilistic labels for machine learning datasets by combining weak labels. In comparison, iFlipper focuses on generating (flipping) labels with the different purpose of satisfying individual fairness constraints. iFlipper can be potentially used in combination with data programming systems to provide an individual fairness guarantee on the weakly-supervised labels. For example, the labels generated by data programming paradigm may have uncertainty due to the combination of multiple weak labels. A data programming model can output a label that is say 80% positive and 20% negative. Here we can view the label as having a continuous value of 0.8. iFlipper already supports continuous similarity matrices when solving its LP problems (c.f. Section 3.1), and thus can support continuous labels in a similar fashion.

6 CONCLUSION AND FUTURE WORK

We proposed iFlipper, which flips labels in the training data to improve the individual fairness of trained binary classification models. iFlipper uses pairwise similarity information and minimizes the number of flipped labels to limit the total error to be within a threshold. We proved that this MIQP optimization problem is NP-hard. We then converted the problem to an approximate LP problem, which can be solved efficiently. Our key finding is that the proposed LP algorithm has theoretical guarantees on how close its result is to the optimal solution in terms of the number of label flips. In addition, we further optimized the algorithm without exceeding

the total error limit. We demonstrated on real datasets that iFlipper outperforms pre-processing unfairness mitigation baselines in terms of fairness and accuracy, and can complement existing in-processing techniques for better results.

In the future, we would like to support multi-class classification and regression tasks. For multi-class classification, we can group the labels to favorable and unfavorable labels and make the problem a binary classification one. Alternatively, we can solve a one-vs-all problem for each class (binary classification) and use a softmax to calculate the final label. For regression, the action becomes changing the label instead of flipping it. Since the current optimization problem can be extended to support continuous labels, we plan to change the optimization setup to support regression problems.

ACKNOWLEDGMENTS

Ki Hyun Tae, Jaeyoung Park, and Steven Euijong Whang were supported by a Google Research Award and by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIT) (No. NRF-2018R1A5A1059921 and NRF-2022R1A2C2004382).

REFERENCES

- [1] iFlipper Github repository. <https://github.com/khtae8250/iFlipper>.
- [2] iFlipper: Label flipping for individual fairness. <https://github.com/khtae8250/iFlipper/blob/main/techreport.pdf>.
- [3] A. Agarwal, A. Beygelzimer, M. Dudik, J. Langford, and H. Wallach. A reductions approach to fair classification. *arXiv preprint arXiv:1803.02453*, 2018.
- [4] A. Andoni, P. Indyk, T. Laarhoven, I. Razenshteyn, and L. Schmidt. Practical and optimal lsh for angular distance. In *NeurIPS*, page 1225–1233, 2015.
- [5] J. Angwin, J. Larson, S. Mattu, and L. Kirchner. Machine bias: There’s software used across the country to predict future criminals. And its biased against blacks., 2016.
- [6] S. Barocas and A. D. Selbst. Big data’s disparate impact. *Calif. L. Rev.*, 104:671, 2016.
- [7] R. K. E. Bellamy, K. Dey, M. Hind, S. C. Hoffman, S. Houde, K. Kannan, P. Lohia, J. Martino, S. Mehta, A. Mojsilovic, S. Nagar, K. N. Ramamurthy, J. T. Richards, D. Saha, P. Sattigeri, M. Singh, K. R. Varshney, and Y. Zhang. AI fairness 360: An extensible toolkit for detecting, understanding, and mitigating unwanted algorithmic bias. *CoRR*, abs/1810.01943, 2018.
- [8] R. Binns. On the apparent conflict between individual and group fairness. In *FAT**, pages 514–524, 2020.
- [9] M. Bruglieri, F. Maffioli, and M. Ehrgott. Cardinality constrained minimum cut problems: complexity and algorithms. *Discret. Appl. Math.*, 137(3):311–341, 2004.
- [10] J. Buolamwini and T. Gebru. Gender shades: Intersectional accuracy disparities in commercial gender classification. In *FAT*, pages 77–91. PMLR, 2018.
- [11] F. Calmon, D. Wei, B. Vinzamuri, K. N. Ramamurthy, and K. R. Varshney. Optimized pre-processing for discrimination prevention. In *NeurIPS*, pages 3992–4001, 2017.
- [12] A. J. Chaney, B. M. Stewart, and B. E. Engelhardt. How algorithmic confounding in recommendation systems increases homogeneity and decreases utility. In *RecSys*, pages 224–232, 2018.
- [13] P. Christen. *Data Matching - Concepts and Techniques for Record Linkage, Entity Resolution, and Duplicate Detection*. Data-Centric Systems and Applications. Springer, 2012.
- [14] Cplex IBM ILOG. V12. 1: User’s manual for cplex. *International Business Machines Corporation*, 46(53):157, 2009.
- [15] J. Dastin. Amazon scraps secret ai recruiting tool that showed bias against women. *reuters* (2018), 2018.
- [16] D. Dua and C. Graff. UCI machine learning repository, 2017.
- [17] C. Dwork, M. Hardt, T. Pitassi, O. Reingold, and R. Zemel. Fairness through awareness. In *ITCS*, pages 214–226, 2012.
- [18] M. Feldman, S. A. Friedler, J. Moeller, C. Scheidegger, and S. Venkatasubramanian. Certifying and removing disparate impact. In *KDD*, pages 259–268, 2015.
- [19] S. A. Friedler, C. Scheidegger, and S. Venkatasubramanian. The (im) possibility of fairness: Different value systems require different mechanisms for fair decision making. *CACM*, 64(4):136–143, 2021.
- [20] Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2022.
- [21] M. Hardt, E. Price, and N. Srebro. Equality of opportunity in supervised learning. In *NeurIPS*, pages 3315–3323, 2016.

- [22] C. Ilvento. Metric learning for individual fairness, 2020.
- [23] F. Kamiran and T. Calders. Data preprocessing techniques for classification without discrimination. *Knowledge and Information Systems*, 33(1):1–33, 2012.
- [24] E. J. Kim, S. Kratsch, M. Pilipczuk, and M. Wahlström. Solving hard cut problems via flow-augmentation. In *SODA*, pages 149–168. SIAM, 2021.
- [25] R. Kohavi. Scaling up the accuracy of naive-bayes classifiers: A decision-tree hybrid. In *KDD*, pages 202–207, 1996.
- [26] V. Kolmogorov and R. Zabin. What energy functions can be minimized via graph cuts? *IEEE TPAMI*, 26(2):147–159, 2004.
- [27] S. Kratsch, S. Li, D. Marx, M. Pilipczuk, and M. Wahlström. Multi-budgeted directed cuts. *Algorithmica*, 82(8):2135–2155, 2020.
- [28] M. Kusner, J. Loftus, C. Russell, and R. Silva. Counterfactual fairness. In *NeurIPS*, pages 4069–4079, 2017.
- [29] P. Lahoti, K. P. Gummadi, and G. Weikum. ifair: Learning individually fair data representations for algorithmic decision making. In *ICDE*, pages 1334–1345, 2019.
- [30] P. Lahoti, K. P. Gummadi, and G. Weikum. Operationalizing individual fairness with pairwise fair representations. *Proc VLDB Endow*, 13(4):506–518, Dec. 2019.
- [31] S. Moro, P. Cortez, and P. Rita. A data-driven approach to predict the success of bank telemarketing. *Decision Support Systems*, 62:22–31, 2014.
- [32] MOSEK ApS. *MOSEK Optimizer API for Python*, 2019.
- [33] D. Mukherjee, M. Yurochkin, M. Banerjee, and Y. Sun. Two simple ways to learn individual fairness metrics from data. In *ICML*, volume 119, pages 7097–7107, 2020.
- [34] A. Narayanan. Translation tutorial: 21 fairness definitions and their politics. In *FAccT*, volume 1170, 2018.
- [35] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library. In *NeurIPS*, pages 8024–8035, 2019.
- [36] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *JMLR*, 12:2825–2830, 2011.
- [37] F. Petersen, D. Mukherjee, Y. Sun, and M. Yurochkin. Post-processing for individual fairness. In *ICLR*, 2021.
- [38] A. J. Ratner, S. H. Bach, H. Ehrenberg, J. Fries, S. Wu, and C. Ré. Snorkel: Rapid training data creation with weak supervision. *PVLDB*, 11(3):269–282, Nov. 2017.
- [39] A. J. Ratner, S. H. Bach, H. R. Ehrenberg, and C. Ré. Snorkel: Fast training set generation for information extraction. In *Proceedings of the 2017 ACM International Conference on Management of Data, SIGMOD ’17*, page 1683–1686, New York, NY, USA, 2017. Association for Computing Machinery.
- [40] B. Salimi, L. Rodriguez, B. Howe, and D. Suciu. Interventional fairness: Causal database repair for algorithmic fairness. In *SIGMOD*, pages 793–810, 2019.
- [41] H. Song, M. Kim, and J. Lee. SELFIE: refurbishing unclean samples for robust deep learning. In *ICML*, volume 97, pages 5907–5915, 2019.
- [42] H. Song, M. Kim, D. Park, and J. Lee. Learning from noisy labels with deep neural networks: A survey. *CoRR*, abs/2007.08199, 2020.
- [43] A. Vargo, F. Zhang, M. Yurochkin, and Y. Sun. Individually fair gradient boosting. In *ICLR*, 2021.
- [44] S. Verma and J. Rubin. Fairness definitions explained. In *Proceedings of the International Workshop on Software Fairness, FairWare ’18*, page 1–7, New York, NY, USA, 2018. Association for Computing Machinery.
- [45] H. Wang, N. Grgic-Hlaca, P. Lahoti, K. P. Gummadi, and A. Weller. An empirical study on learning fairness metrics for compas data with human supervision. *arXiv preprint arXiv:1910.10255*, 2019.
- [46] S. E. Whang, K. H. Tae, Y. Roh, and G. Heo. Responsible ai challenges in end-to-end machine learning. *IEEE Data Eng. Bull.*, 2021.
- [47] L. F. Wightman and H. Ramsey. *LSAC national longitudinal bar passage study*. Law School Admission Council, 1998.
- [48] M. Yurochkin, A. Bower, and Y. Sun. Training individually fair ML models with sensitive subspace robustness. In *ICLR*, 2020.
- [49] M. Yurochkin and Y. Sun. Sensei: Sensitive set invariance for enforcing individual fairness. In *ICLR*, 2021.
- [50] M. B. Zafar, I. Valera, M. Gomez Rodriguez, and K. P. Gummadi. Fairness beyond disparate treatment & disparate impact: Learning classification without disparate mistreatment. In *WWW*, pages 1171–1180, 2017.
- [51] M. B. Zafar, I. Valera, M. G. Rogriguez, and K. P. Gummadi. Fairness Constraints: Mechanisms for Fair Classification. In A. Singh and J. Zhu, editors, *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, volume 54 of *Proceedings of Machine Learning Research*, pages 962–970. PMLR, 20–22 Apr 2017.
- [52] R. Zemel, Y. Wu, K. Swersky, T. Pitassi, and C. Dwork. Learning fair representations. In *ICML*, volume 28, pages 325–333, 2013.
- [53] H. Zhang, X. Chu, A. Asudeh, and S. B. Navathe. Omnipfair: A declarative system for model-agnostic group fairness in machine learning. In *SIGMOD*, pages 2076–2088, 2021.

A PROOF FOR THEOREM 1

We continue from Section 2.2 and provide a full proof for Theorem 1.

Theorem 1. *The MIQP problem in Equation 1 is NP-hard.*

PROOF. We prove that the MIQP problem in Equation 1 is NP-hard by showing its sub-problem where W_{ij} is binary ($W_{ij} \in \{0, 1\}^{n \times n}$) is NP-hard. We prove the sub-problem is NP-hard by reducing it from the well-known *at most k-cardinality s-t cut problem*. Given an undirected graph $G = (V, E)$, the *at most k-cardinality minimum s-t cut problem* is to find the minimum sum of edge weights in the cut edge set C to divide the vertex set V into two sets V_1 and V_2 , where $s \in V_1$ and $t \in V_2$, and the cut edge set $C = \{(v_1, v_2) \in E : v_1 \in V_1, v_2 \in V_2\}$ has at most k edges. This problem is known to be NP-hard even if all the edge weights are 1 [9, 24, 27]. Furthermore, the *existence of at most k-cardinality s-t cut problem* is also known to be NP-hard [9].

We will show the reduction as a three-step approach. First, given an instance of the *at most k-cardinality s-t cut problem* on $G = (V, E)$ with all edge weights equal to 1, we show how to create $k + 1$ label flipping problems in Equation 1 accordingly. Second, we show how a solution to any of the $k + 1$ MIQP problems represents an $s-t$ cut to the original graph cut problem. Third, we establish that if there exists an *at most k-cardinality s-t cut*, it must be one of the $k + 1$ $s-t$ cuts obtained in the previous step.

Step 1: We show how to create MIQP problems for a given *at most k-cardinality s-t cut problem* on $G = (V, E)$. We create a variable y_i for each $v_i \in V - \{s, t\}$. We make $W_{ij} = 1$ for each edge $(i, j) \in E$ in the graph, and $W_{ij} = 0$ for other pairs of (i, j) . If a node v_i is only connected to s (t), we make $y'_i = 1(0)$ and add $(y_i - y'_i)^2$ to the objective function. That is, we set the initial label of nodes directly connected to s (t) as 1 (0). If a node v_i is directly connected to both s and t , we add two terms to the objective function for this node, one with $y'_i = 0$ and the other one with $y'_i = 1$. If a node v_i is not directly connected to s or t , we do not add any terms to the objective. Now that we have defined all y_i , their initial assignments y'_i , and the weight W_{ij} , we vary m from 0 to k to create $k + 1$ instances of MIQP problems with different allowed amounts of total error as the constraints.

The intuition for the above process of creating $k + 1$ MIQP problem is that an $s-t$ cut is a binary partition of a graph and can be viewed as a binary-valued labeling process. The nodes that are connected to the source s have label 1 and the nodes that are connected to sink t have label 0. A cut consists of two types of edges: e_f and e_v . A flip edge $e_f : (v_1, v_2) : v_1 \in \{s, t\}, v_2 \in V - \{s, t\}$ is an edge between s (t) and other nodes. If an e_f edge exists in a cut, that means a node which was connected to s (t) is no longer in the same partition as s (t), which can be represented as a flip of label. See the $(s, 1)$ edge in Figure 12 as an example for flip edges. A violation edge $e_v : (v_1, v_2) : v_1 \in V - \{s, t\}, v_2 \in V - \{s, t\}$ is an edge between two nodes that are not s or t . If an e_v edge exists in a cut, that means two nodes that are supposed to be similar and have the same label ended up in two different partitions, which can be represented as a violation. See the $(1, 2)$ edge in Figure 12 as an example for violation edges. The cardinality of the cut equals to the sum of the counts of the two types of edges, which equals to the number of flips and the total error in the MIQP solution. Our mapping from

graph cutting to label flipping is inspired by a similar process used in image segmentation [26] that aims at using graph cutting to split an image into foreground and background.

Step 2: We show how a solution to any of the $k+1$ MIQP problems can be mapped to an $s-t$ cut. After solving a corresponding MIQP, we put all nodes with $y_i = 1$ in V_1 and $y_i = 0$ in V_2 as the resulting $s-t$ cut. The cardinality of the cut equals to the sum of the number of label flips and the total error in the label flipping problem.

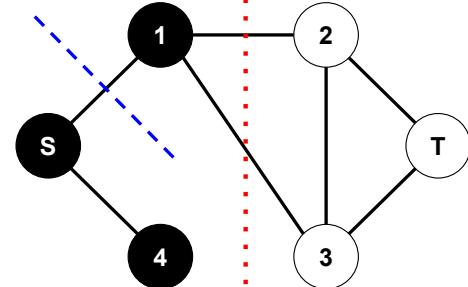


Figure 12: An example for the $s-t$ cut. The cut represented by the blue dashed line consists of one e_f edge, and the cut represented by the red dotted line consists of two e_v edges.

In Figure 12, after removing s and t , we get the same graph representation as Figure 1. By solving the label flipping problem with m equal to 0 or 1, the resulting label assignment is $y_4 = 1$ and $y_1 = y_2 = y_3 = 0$. The corresponding cut is shown as the blue dashed line in Figure 12. The cut is $V_1 = \{s, 4\}$, $V_2 = \{1, 2, 3, t\}$, and $C = \{(s, 1)\}$. This label assignment has one flip (node 1 flips to label 0) and zero violations. This flip is represented as an e_f edge $(s, 1)$ in the cut. If we solve the problem with $m = 2$, the result is $y_1 = y_4 = 1$ and $y_2 = y_3 = 0$. The corresponding cut is shown as the red dotted line in Figure 12. The cut is $V_1 = \{s, 1, 4\}$, $V_2 = \{2, 3, t\}$, and $C = \{(1, 2), (1, 3)\}$. This label assignment has zero flips and a total error of 2, represented by two e_v edges in the cut: $(1, 2)$ and $(1, 3)$.

Step 3: Since the cardinality of the cut equals to the sum of the number of label flips and the total error in the label flipping problem, finding an *at most k-cardinality s-t cut* can be solved by finding a label assignment where the sum of flips and the total error equals to at most k . To find such a label assignment, we can repeatedly solve the MIQP problem $k + 1$ times, for all m values in $\{0, \dots, k\}$, and check if the sum of label flips and the total error is less than or equal to k . The *k*-cardinality minimum $s-t$ cut, if exists, must equal to the result of the MIQP problem with at least one of the possible m values. Therefore, if the label flipping MIQP problem is not NP-hard, then the *k*-cardinality minimum $s-t$ cut problem would also not be NP-hard, which contradicts the known results. \square

B PROOF FOR LEMMA 2.1

We continue from Section 3.3 and provide a full proof for Lemma 2.1. Here we restate Lemma 2.1 with the problem setup:

Lemma 2.1. *Suppose an α -cluster has A_0 points whose initial labels are 0 and A_1 points whose initial values are 1. Let $N_\alpha = A_0 - A_1$.*

Now suppose there are U nodes connected to the α -cluster (shown in Figure 13) by an edge $W_{a_i\alpha}$ satisfying

$$0 \leq a_1 \leq \dots \leq a_k < \alpha < a_{k+1} \leq \dots \leq a_U \leq 1.$$

Note that there is no connected node with a value of α by construction. Let $S_\alpha = \sum_{i=1}^k W_{a_i\alpha} - \sum_{i=k+1}^U W_{a_i\alpha}$. Let us also add the following nodes for convenience: $a_0 = 0$ and $a_{U+1} = 1$. For an α -cluster with $N_\alpha = 0$ in the optimal solution, we can always convert α to either a_k or a_{k+1} while maintaining an optimal solution. As a result, we can reduce exactly one non-0/1 value in the optimal solution.

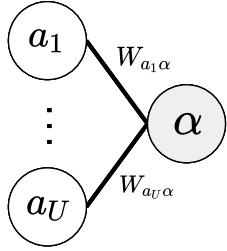


Figure 13: Problem setup for Lemma 2.1.

PROOF. We compute the initial number of flips and total error in Figure 13 as follows:

$$\# \text{Flips} = \alpha A_0 + (1 - \alpha) A_1 = A_1 + \alpha N_\alpha = A_1 (\because N_\alpha = 0)$$

$$\begin{aligned} \text{Total Error} &= \sum_{i=1}^U W_{a_i\alpha} |a_i - \alpha| = S_\alpha \alpha + C \\ (C &= - \sum_{i=1}^k W_{a_i\alpha} a_i + \sum_{i=k+1}^U W_{a_i\alpha} a_i) \end{aligned} \quad (11)$$

We first observe that the number of flips is independent of the α value. Hence, even if we change α to an arbitrary value, the solution still has the same objective value.

Now consider a small positive value ϵ_α . Suppose we change α by ϵ_α such that

$$0 \leq a_1 \leq \dots \leq a_k \leq \alpha + \epsilon_\alpha \leq a_{k+1} \leq \dots \leq a_U \leq 1 \quad (12)$$

Similarly, we also change α by $-\epsilon_\alpha$ such that

$$0 \leq a_1 \leq \dots \leq a_k \leq \alpha - \epsilon_\alpha \leq a_{k+1} \leq \dots \leq a_U \leq 1 \quad (13)$$

Note that such ϵ_α always exists because $a_k < \alpha < a_{k+1}$. If we change α to $\alpha + \epsilon_\alpha$ while satisfying Equation 12, the total error becomes $S_\alpha(\alpha + \epsilon_\alpha) + C$. Similarly, if we change α to $\alpha - \epsilon_\alpha$ while satisfying Equation 13, the total error becomes $S_\alpha(\alpha - \epsilon_\alpha) + C$. Hence, the change in the total error for each case can be computed as follows:

$$\begin{aligned} \alpha + \epsilon_\alpha \rightarrow \Delta(\text{Total Error}) &= S_\alpha \epsilon_\alpha \\ \alpha - \epsilon_\alpha \rightarrow \Delta(\text{Total Error}) &= -S_\alpha \epsilon_\alpha \end{aligned} \quad (14)$$

From Equation 14, we observe that one of the transformations always maintains or even reduces the total error according to the sign of S_α , i.e., the solution is feasible. Specifically, if $S_\alpha \leq 0$, we change α to a_{k+1} by setting $\epsilon_\alpha = a_{k+1} - \alpha$, otherwise we change α to a_k by setting $\epsilon_\alpha = \alpha - a_k$. As a result, we can remove one non-0/1 value α while maintaining an optimal solution. \square

C PROOF FOR LEMMA 2.2

We continue from Section 3.3 and provide a full proof for Lemma 2.2. Here we restate Lemma 2.2 with the problem setup:

Lemma 2.2. Let us assume that $0 < \alpha < \beta < 1$, and the sum of the pairwise node similarities between the two clusters between the two clusters is E . Suppose an α -cluster and a β -cluster have A_0 and B_0 points whose initial labels are 0, respectively, and A_1 and B_1 points whose initial values are 1, respectively. Let $N_\alpha = A_0 - A_1$ and $N_\beta = B_0 - B_1$. Now suppose there are U nodes connected to the α -cluster by an edge $W_{a_i\alpha}$ and V nodes connected to the β -cluster by an edge $W_{b_i\beta}$ satisfying

$$\begin{aligned} 0 \leq a_1 \leq \dots \leq a_k &< \alpha < a_{k+1} \leq \dots \leq a_U \leq 1 \text{ and} \\ 0 \leq b_1 \leq \dots \leq b_l &< \beta < b_{l+1} \leq \dots \leq b_V \leq 1. \end{aligned}$$

Note that there is no connected node with a value of α or β by construction. Let $S_\alpha = \sum_{i=1}^k W_{a_i\alpha} - \sum_{i=k+1}^U W_{a_i\alpha}$ and $S_\beta = \sum_{i=1}^l W_{b_i\beta} - \sum_{i=l+1}^V W_{b_i\beta}$. Let us also add the following nodes for convenience: $a_0 = 0$, $a_{U+1} = 1$, $b_0 = 0$, and $b_{V+1} = 1$. For an α -cluster with $N_\alpha \neq 0$ and a β -cluster with $N_\beta \neq 0$ in the optimal solution, we can always convert (α, β) to one of $(a_k, \beta + \frac{N_\alpha}{N_\beta}(a_k - \alpha))$, $(a_{k+1}, \beta - \frac{N_\alpha}{N_\beta}(a_{k+1} - \alpha))$, $(\alpha + \frac{N_\beta}{N_\alpha}(\beta - b_l), b_l)$, $(\alpha - \frac{N_\beta}{N_\alpha}(b_{l+1} - \beta), b_{l+1})$, or $(\frac{\alpha N_\alpha + \beta N_\beta}{N_\alpha + N_\beta}, \frac{\alpha N_\alpha + \beta N_\beta}{N_\alpha + N_\beta})$, while maintaining an optimal solution. As a result, we can reduce at least one non-0/1 value in the optimal solution.

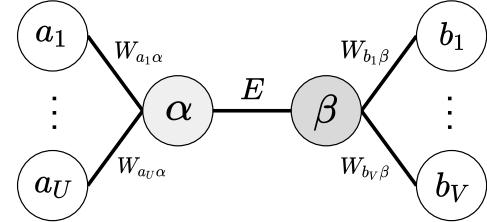


Figure 14: Problem setup for Lemma 2.2.

PROOF. We compute the initial number of flips and total error in Figure 14 as follows:

$$\# \text{Flips} = \alpha A_0 + (1 - \alpha) A_1 + \beta B_0 + (1 - \beta) B_1 = A_1 + B_1 + \alpha N_\alpha + \beta N_\beta$$

$$\begin{aligned} \text{Total Error} &= \sum_{i=1}^U W_{a_i\alpha} |a_i - \alpha| + \sum_{i=1}^V W_{b_i\beta} |b_i - \beta| + E(\beta - \alpha) \\ (C &= - \sum_{i=1}^k W_{a_i\alpha} a_i + \sum_{i=k+1}^U W_{a_i\alpha} a_i - \sum_{i=1}^l W_{b_i\beta} b_i + \sum_{i=l+1}^V W_{b_i\beta} b_i) \end{aligned} \quad (15)$$

Now consider two small positive values ϵ_α and ϵ_β . Both N_α and N_β are non-zero, so we have only two cases: $\frac{N_\alpha}{N_\beta} < 0$ and $\frac{N_\alpha}{N_\beta} > 0$.

Case 1 $\frac{N_\alpha}{N_\beta} < 0$: Suppose we change α by ϵ_α and β by ϵ_β , respectively, such that

$$0 \leq a_1 \leq \dots \leq a_k \leq \alpha + \epsilon_\alpha \leq a_{k+1} \leq \dots \leq a_U \leq 1$$

$$0 \leq b_1 \leq \dots \leq b_l \leq \beta + \epsilon_\beta \leq b_{l+1} \leq \dots \leq b_V \leq 1 \quad (16)$$

$$\alpha + \epsilon_\alpha \leq \beta + \epsilon_\beta$$

We then compute the number of flips for $(\alpha + \epsilon_\alpha, \beta + \epsilon_\beta)$ as $A_1 + B_1 + (\alpha + \epsilon_\alpha)N_\alpha + (\beta + \epsilon_\beta)N_\beta$. In order to have the same number of flips as the initial value in Equation 15, $(\epsilon_\alpha, \epsilon_\beta)$ should satisfy $\epsilon_\alpha N_\alpha + \epsilon_\beta N_\beta = 0$.

Similarly, we also change α by $-\epsilon_\alpha$ and β by $-\epsilon_\beta$, respectively, such that

$$\begin{aligned} 0 &\leq a_1 \leq \dots \leq a_k \leq \alpha - \epsilon_\alpha \leq a_{k+1} \leq \dots \leq a_U \leq 1 \\ 0 &\leq b_1 \leq \dots \leq b_l \leq \beta - \epsilon_\beta \leq b_{l+1} \leq \dots \leq b_V \leq 1 \\ \alpha - \epsilon_\alpha &\leq \beta - \epsilon_\beta \end{aligned} \quad (17)$$

In this case, $(\alpha - \epsilon_\alpha, \beta - \epsilon_\beta)$ also maintains the same number of label flips if $\epsilon_\alpha N_\alpha + \epsilon_\beta N_\beta = 0$.

From now, we consider $(\epsilon_\alpha, \epsilon_\beta)$ that also satisfies $\epsilon_\alpha N_\alpha + \epsilon_\beta N_\beta = 0$, i.e., $\epsilon_\beta = -\frac{N_\alpha}{N_\beta} \epsilon_\alpha$. Note that such ϵ_α and ϵ_β always exist because $a_k < \alpha < a_{k+1}$, $b_l < \beta < b_{l+1}$, and $\frac{N_\alpha}{N_\beta} < 0$. Therefore, both $(\alpha + \epsilon_\alpha, \beta + \epsilon_\beta)$ and $(\alpha - \epsilon_\alpha, \beta - \epsilon_\beta)$ have the same number of label flipping as the initial number.

If we change (α, β) to $(\alpha + \epsilon_\alpha, \beta + \epsilon_\beta)$ while satisfying Equation 16 and $\epsilon_\alpha N_\alpha + \epsilon_\beta N_\beta = 0$, the total error becomes $(S_\alpha - E)(\alpha + \epsilon_\alpha) + (S_\beta + E)(\beta + \epsilon_\beta) + C$. Similarly, if we change (α, β) to $(\alpha - \epsilon_\alpha, \beta - \epsilon_\beta)$ while satisfying Equation 17 and $\epsilon_\alpha N_\alpha + \epsilon_\beta N_\beta = 0$, the total error becomes $(S_\alpha - E)(\alpha - \epsilon_\alpha) + (S_\beta + E)(\beta - \epsilon_\beta) + C$. Hence, the change in the total error for each case can be computed as follows:

$$\begin{aligned} (\alpha + \epsilon_\alpha, \beta + \epsilon_\beta) \rightarrow \Delta(\text{Total Error}) &= (S_\alpha - E)\epsilon_\alpha + (S_\beta + E)\epsilon_\beta \\ &= \frac{(S_\alpha - E)N_\beta - (S_\beta + E)N_\alpha}{N_\beta} \epsilon_\alpha \\ (\alpha - \epsilon_\alpha, \beta - \epsilon_\beta) \rightarrow \Delta(\text{Total Error}) &= -(S_\alpha - E)\epsilon_\alpha - (S_\beta + E)\epsilon_\beta \\ &= -\frac{(S_\alpha - E)N_\beta - (S_\beta + E)N_\alpha}{N_\beta} \epsilon_\alpha \end{aligned} \quad (18)$$

From Equation 18, we observe that one of the transformations always maintains or even reduces the total error according to the sign of $\frac{(S_\alpha - E)N_\beta - (S_\beta + E)N_\alpha}{N_\beta}$, i.e., the solution is feasible.

- If $\frac{(S_\alpha - E)N_\beta - (S_\beta + E)N_\alpha}{N_\beta} \leq 0$, we change (α, β) to $(\alpha + \epsilon_\alpha, \beta + \epsilon_\beta)$ so that the solution is still optimal. Recall $(\epsilon_\alpha, \epsilon_\beta)$ satisfies the three inequalities and one condition: $\alpha + \epsilon_\alpha \leq a_{k+1}$, $\beta + \epsilon_\beta \leq b_{l+1}$, $\alpha + \epsilon_\alpha \leq \beta + \epsilon_\beta$, and $\epsilon_\alpha N_\alpha + \epsilon_\beta N_\beta = 0$. Among the possible $(\epsilon_\alpha, \epsilon_\beta)$, we choose the upper bound of ϵ_α and the corresponding ϵ_β ($\epsilon_\beta = -\frac{N_\alpha}{N_\beta} \epsilon_\alpha$). To get an upper bound of ϵ_α , we find the equality conditions for each inequality and take the smallest value among them. If $1 + \frac{N_\alpha}{N_\beta} \leq 0$, the last inequality ($\alpha + \epsilon_\alpha \leq \beta + \epsilon_\beta$) always hold because $\epsilon_\alpha \leq \epsilon_\beta$. Hence, we consider only the first two inequalities and set ϵ_α to $\min(a_{k+1} - \alpha, -\frac{N_\beta}{N_\alpha}(b_{l+1} - \beta))$. On the other hand, if $1 + \frac{N_\alpha}{N_\beta} > 0$, we set ϵ_α to $\min(a_{k+1} - \alpha, -\frac{N_\beta}{N_\alpha}(b_{l+1} - \beta), \frac{N_\beta(\beta - \alpha)}{N_\alpha + N_\beta})$ from the three inequalities. As a result, we can convert (α, β) to $(a_{k+1}, \beta - \frac{N_\alpha}{N_\beta}(a_{k+1} - \alpha))$,

$(\alpha - \frac{N_\beta}{N_\alpha}(b_{l+1} - \beta), b_{l+1})$, or $(\frac{\alpha N_\alpha + \beta N_\beta}{N_\alpha + N_\beta}, \frac{\alpha N_\alpha + \beta N_\beta}{N_\alpha + N_\beta})$, which is one of the cases in Lemma 2.2.

- If $\frac{(S_\alpha - E)N_\beta - (S_\beta + E)N_\alpha}{N_\beta} > 0$, we change (α, β) to $(\alpha - \epsilon_\alpha, \beta - \epsilon_\beta)$ so that the solution is still optimal. Recall $(\epsilon_\alpha, \epsilon_\beta)$ satisfies the three inequalities and one condition: $a_k \leq \alpha - \epsilon_\alpha$, $b_l \leq \beta - \epsilon_\beta$, $\alpha - \epsilon_\alpha \leq \beta - \epsilon_\beta$, and $\epsilon_\alpha N_\alpha + \epsilon_\beta N_\beta = 0$. Among the possible $(\epsilon_\alpha, \epsilon_\beta)$, we choose the upper bound of ϵ_α and the corresponding ϵ_β ($\epsilon_\beta = -\frac{N_\alpha}{N_\beta} \epsilon_\alpha$). To get an upper bound of ϵ_α , we find the equality conditions for each inequality and take the smallest value among them. If $1 + \frac{N_\alpha}{N_\beta} \geq 0$, the last inequality ($\alpha - \epsilon_\alpha \leq \beta - \epsilon_\beta$) always holds because $\epsilon_\alpha \geq \epsilon_\beta$. Hence, we consider only the first two inequalities and set ϵ_α to $\min(\alpha - a_k, -\frac{N_\beta}{N_\alpha}(\beta - b_l))$. On the other hand, if $1 + \frac{N_\alpha}{N_\beta} < 0$, we set ϵ_α to $\min(\alpha - a_k, -\frac{N_\beta}{N_\alpha}(\beta - b_l), -\frac{N_\beta(\beta - \alpha)}{N_\alpha + N_\beta})$ from the three inequalities. As a result, we can convert (α, β) to $(a_k, \beta + \frac{N_\alpha}{N_\beta}(\alpha - a_k))$, $(\alpha + \frac{N_\beta}{N_\alpha}(\beta - b_l), b_l)$, or $(\frac{\alpha N_\alpha + \beta N_\beta}{N_\alpha + N_\beta}, \frac{\alpha N_\alpha + \beta N_\beta}{N_\alpha + N_\beta})$, which is one of the cases in Lemma 2.2.

Case 2: $\frac{N_\alpha}{N_\beta} > 0$: The proof is similar to the proof for Case 1 except that we consider $(\alpha + \epsilon_\alpha, \beta - \epsilon_\beta)$ and $(\alpha - \epsilon_\alpha, \beta + \epsilon_\beta)$ instead of $(\alpha + \epsilon_\alpha, \beta + \epsilon_\beta)$ and $(\alpha - \epsilon_\alpha, \beta - \epsilon_\beta)$. We now write the full proof for Case 2 for completeness.

Suppose we change α by ϵ_α and β by $-\epsilon_\beta$, respectively, such that

$$\begin{aligned} 0 &\leq a_1 \leq \dots \leq a_k \leq \alpha + \epsilon_\alpha \leq a_{k+1} \leq \dots \leq a_U \leq 1 \\ 0 &\leq b_1 \leq \dots \leq b_l \leq \beta - \epsilon_\beta \leq b_{l+1} \leq \dots \leq b_V \leq 1 \\ \alpha + \epsilon_\alpha &\leq \beta - \epsilon_\beta \end{aligned} \quad (19)$$

We then compute the number of flips for $(\alpha + \epsilon_\alpha, \beta - \epsilon_\beta)$ as $A_1 + B_1 + (\alpha + \epsilon_\alpha)N_\alpha + (\beta - \epsilon_\beta)N_\beta$. In order to have the same number of flips as the initial value in Equation 15, $(\epsilon_\alpha, \epsilon_\beta)$ should satisfy $\epsilon_\alpha N_\alpha - \epsilon_\beta N_\beta = 0$.

Similarly, we also change α by $-\epsilon_\alpha$ and β by ϵ_β , respectively, such that

$$\begin{aligned} 0 &\leq a_1 \leq \dots \leq a_k \leq \alpha - \epsilon_\alpha \leq a_{k+1} \leq \dots \leq a_U \leq 1 \\ 0 &\leq b_1 \leq \dots \leq b_l \leq \beta + \epsilon_\beta \leq b_{l+1} \leq \dots \leq b_V \leq 1 \\ \alpha - \epsilon_\alpha &\leq \beta + \epsilon_\beta \end{aligned} \quad (20)$$

In this case, $(\alpha - \epsilon_\alpha, \beta + \epsilon_\beta)$ also maintains the same number of label flips if $\epsilon_\alpha N_\alpha - \epsilon_\beta N_\beta = 0$.

From now, we consider $(\epsilon_\alpha, \epsilon_\beta)$ that also satisfies $\epsilon_\alpha N_\alpha - \epsilon_\beta N_\beta = 0$, i.e., $\epsilon_\beta = \frac{N_\alpha}{N_\beta} \epsilon_\alpha$. Note that such $(\epsilon_\alpha$ and $\epsilon_\beta)$ always exist because $a_k < \alpha < a_{k+1}$, $b_l < \beta < b_{l+1}$, and $\frac{N_\alpha}{N_\beta} > 0$. Therefore, both $(\alpha + \epsilon_\alpha, \beta - \epsilon_\beta)$ and $(\alpha - \epsilon_\alpha, \beta + \epsilon_\beta)$ have the same number of label flipping as the initial number.

If we change (α, β) to $(\alpha + \epsilon_\alpha, \beta - \epsilon_\beta)$ while satisfying Equation 19 and $\epsilon_\alpha N_\alpha - \epsilon_\beta N_\beta = 0$, the total error becomes $(S_\alpha - E)(\alpha + \epsilon_\alpha) + (S_\beta + E)(\beta - \epsilon_\beta) + C$. Similarly, if we change (α, β) to $(\alpha - \epsilon_\alpha, \beta + \epsilon_\beta)$ while satisfying Equation 20 and $\epsilon_\alpha N_\alpha - \epsilon_\beta N_\beta = 0$, the total error becomes $(S_\alpha - E)(\alpha - \epsilon_\alpha) + (S_\beta + E)(\beta + \epsilon_\beta) + C$. Hence, the change in the total error for each case can be computed as follows:

$$\begin{aligned}
(\alpha + \epsilon_\alpha, \beta - \epsilon_\beta) &\rightarrow \Delta(\text{Total Error}) = (S_\alpha - E)\epsilon_\alpha - (S_\beta + E)\epsilon_\beta \\
&= \frac{(S_\alpha - E)N_\beta - (S_\beta + E)N_\alpha}{N_\beta}\epsilon_\alpha \\
(\alpha - \epsilon_\alpha, \beta + \epsilon_\beta) &\rightarrow \Delta(\text{Total Error}) = -(S_\alpha - E)\epsilon_\alpha + (S_\beta + E)\epsilon_\beta \\
&= -\frac{(S_\alpha - E)N_\beta - (S_\beta + E)N_\alpha}{N_\beta}\epsilon_\alpha
\end{aligned} \tag{21}$$

From Equation 21, we observe that one of the transformations always maintains or reduces the total error according to the sign of $\frac{(S_\alpha - E)N_\beta - (S_\beta + E)N_\alpha}{N_\beta}$, i.e., the solution is feasible.

- If $\frac{(S_\alpha - E)N_\beta - (S_\beta + E)N_\alpha}{N_\beta} \leq 0$, we can change (α, β) to $(\alpha + \epsilon_\alpha, \beta - \epsilon_\beta)$ so that the solution is still optimal. Recall $(\epsilon_\alpha, \epsilon_\beta)$ satisfies the three inequalities and one condition: $\alpha + \epsilon_\alpha \leq a_{k+1}, b_{l+1} \leq \beta - \epsilon_\beta, \alpha + \epsilon_\alpha \leq \beta + \epsilon_\beta$, and $\epsilon_\alpha N_\alpha - \epsilon_\beta N_\beta = 0$. Among the possible $(\epsilon_\alpha, \epsilon_\beta)$, we choose the upper bound of ϵ_α and the corresponding ϵ_β ($\epsilon_\beta = \frac{N_\alpha}{N_\beta}\epsilon_\alpha$). To get an upper bound of ϵ_α , we find the equality conditions for each inequality and take the smallest value among them. Specifically, we set ϵ_α to $\min(a_{k+1} - \alpha, \frac{N_\beta}{N_\alpha}(\beta - b_l), \frac{N_\beta(\beta - \alpha)}{N_\alpha + N_\beta})$ and ϵ_β . As a result, we can convert (α, β) to $(a_{k+1}, \beta - \frac{N_\alpha}{N_\beta}(a_{k+1} - \alpha), \alpha + \frac{N_\beta}{N_\alpha}(\beta - b_l), b_l)$, or $(\frac{\alpha N_\alpha + \beta N_\beta}{N_\alpha + N_\beta}, \frac{\alpha N_\alpha + \beta N_\beta}{N_\alpha + N_\beta})$, which is one of the cases in Lemma 2.2.
- If $\frac{(S_\alpha - E)N_\beta - (S_\beta + E)N_\alpha}{N_\beta} > 0$, we can change (α, β) to $(\alpha - \epsilon_\alpha, \beta + \epsilon_\beta)$ so that the solution is still optimal. Recall $(\epsilon_\alpha, \epsilon_\beta)$ satisfies the three inequalities and one condition: $a_k \leq \alpha - \epsilon_\alpha, \beta + \epsilon_\beta \leq b_{l+1}, \alpha - \epsilon_\alpha \leq \beta + \epsilon_\beta$, and $\epsilon_\alpha N_\alpha - \epsilon_\beta N_\beta = 0$. Among the possible $(\epsilon_\alpha, \epsilon_\beta)$, we choose the upper bound of ϵ_α and the corresponding ϵ_β ($\epsilon_\beta = \frac{N_\alpha}{N_\beta}\epsilon_\alpha$). To get an upper bound of ϵ_α , we find the equality conditions for each inequality and take the smallest value among them. In this case, the last inequality $(\alpha - \epsilon_\alpha \leq \beta + \epsilon_\beta)$ always hold. Hence, we consider only the first two conditions and set ϵ_α to $\min(\alpha - a_k, \frac{N_\beta}{N_\alpha}(b_{l+1} - \beta))$. As a result, we can convert (α, β) to either $(a_k, \beta + \frac{N_\alpha}{N_\beta}(a_k - \alpha))$ or $(\alpha - \frac{N_\beta}{N_\alpha}(b_{l+1} - \beta), b_{l+1})$, which is one of the cases in Lemma 2.2.

We summarize the main results for each case below and conclude that (α, β) can be transformed into one of the five cases in Lemma 2.2 while maintaining an optimal optimal. As a result, we remove at least one of α and β . If both converted values already exist in the solution, we can even reduce two non-0/1 values.

- If $\frac{(S_\alpha - E)N_\beta - (S_\beta + E)N_\alpha}{N_\beta} \leq 0, 1 + \frac{N_\alpha}{N_\beta} \leq 0$, we convert (α, β) to $(\alpha + \epsilon_\alpha, \beta + \epsilon_\beta)$ where $\epsilon_\alpha = \min(a_{k+1} - \alpha, -\frac{N_\beta}{N_\alpha}(b_{l+1} - \beta))$ and $\epsilon_\beta = -\frac{N_\alpha}{N_\beta}\epsilon_\alpha$.
- If $\frac{(S_\alpha - E)N_\beta - (S_\beta + E)N_\alpha}{N_\beta} \leq 0, 1 + \frac{N_\alpha}{N_\beta} > 0$, we convert (α, β) to $(\alpha + \epsilon_\alpha, \beta + \epsilon_\beta)$ where $\epsilon_\alpha = \min(a_{k+1} - \alpha, -\frac{N_\beta}{N_\alpha}(b_{l+1} - \beta), \frac{N_\beta(\beta - \alpha)}{N_\alpha + N_\beta})$ and $\epsilon_\beta = -\frac{N_\alpha}{N_\beta}\epsilon_\alpha$.

- If $(\frac{N_\alpha}{N_\beta} < 0, \frac{(S_\alpha - E)N_\beta - (S_\beta + E)N_\alpha}{N_\beta} > 0, 1 + \frac{N_\alpha}{N_\beta} \geq 0)$, we convert (α, β) to $(\alpha - \epsilon_\alpha, \beta - \epsilon_\beta)$ where $\epsilon_\alpha = \min(\alpha - a_k, -\frac{N_\beta}{N_\alpha}(\beta - b_l))$ and $\epsilon_\beta = -\frac{N_\alpha}{N_\beta}\epsilon_\alpha$.
- If $(\frac{N_\alpha}{N_\beta} < 0, \frac{(S_\alpha - E)N_\beta - (S_\beta + E)N_\alpha}{N_\beta} > 0, 1 + \frac{N_\alpha}{N_\beta} < 0)$, we convert (α, β) to $(\alpha - \epsilon_\alpha, \beta - \epsilon_\beta)$ where $\epsilon_\alpha = \min(\alpha - a_k, -\frac{N_\beta}{N_\alpha}(\beta - b_l), -\frac{N_\beta(\beta - \alpha)}{N_\alpha + N_\beta})$ and $\epsilon_\beta = -\frac{N_\alpha}{N_\beta}\epsilon_\alpha$.
- If $(\frac{N_\alpha}{N_\beta} > 0, \frac{(S_\alpha - E)N_\beta - (S_\beta + E)N_\alpha}{N_\beta} \leq 0)$, we convert (α, β) to $(\alpha + \epsilon_\alpha, \beta - \epsilon_\beta)$ where $\epsilon_\alpha = \min(a_{k+1} - \alpha, \frac{N_\beta}{N_\alpha}(\beta - b_l), \frac{N_\beta(\beta - \alpha)}{N_\alpha + N_\beta})$ and $\epsilon_\beta = \frac{N_\alpha}{N_\beta}\epsilon_\alpha$.
- If $(\frac{N_\alpha}{N_\beta} > 0, \frac{(S_\alpha - E)N_\beta - (S_\beta + E)N_\alpha}{N_\beta} > 0)$, we convert (α, β) to $(\alpha - \epsilon_\alpha, \beta + \epsilon_\beta)$ where $\epsilon_\alpha = \min(\alpha - a_k, \frac{N_\beta}{N_\alpha}(b_{l+1} - \beta))$ and $\epsilon_\beta = \frac{N_\alpha}{N_\beta}\epsilon_\alpha$.

□

D TRADE-OFF FOR OTHER DATASETS

We continue from Section 4.2 and show the trade-off results on the AdultCensus and Credit datasets in Figure 15. The results are similar to the COMPAS dataset in Figure 7 where there is a clear trade-off between accuracy and fairness.

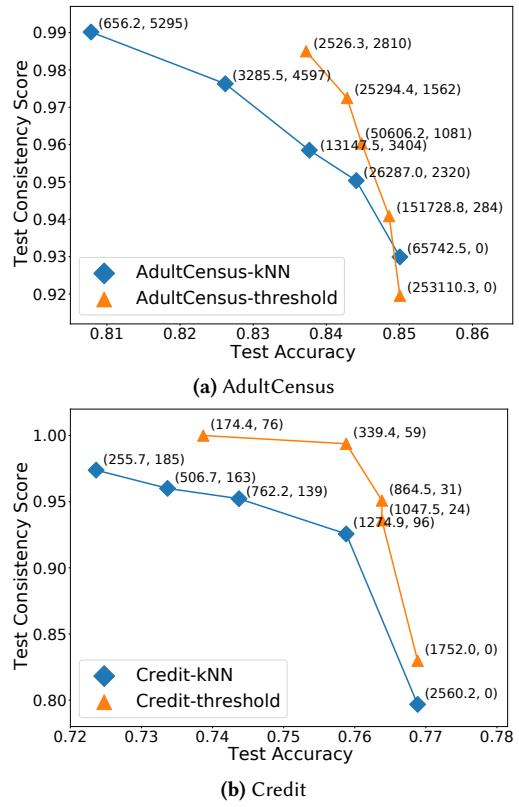


Figure 15: Trade-off curves on the AdultCensus and Credit datasets.

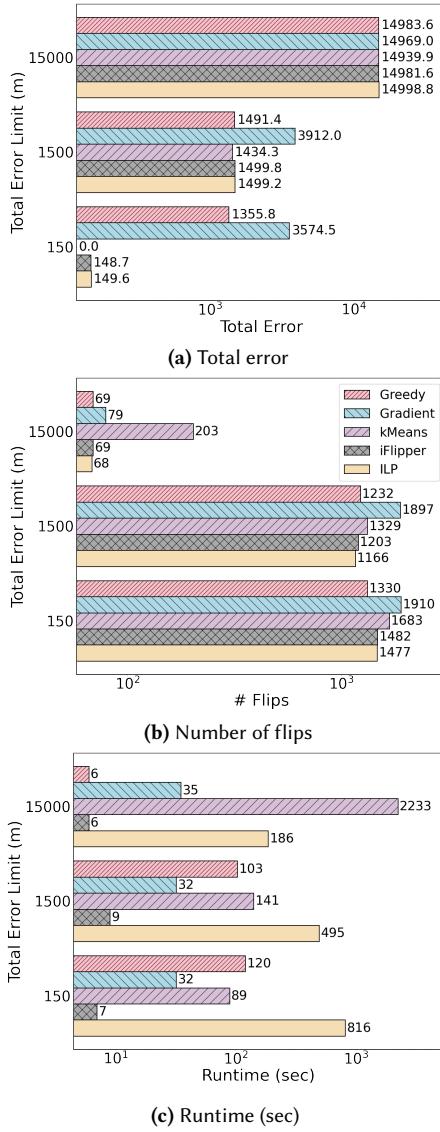


Figure 16: A detailed comparison of iFlipper against three naïve solutions (*Greedy*, *Gradient*, and *kMeans*) and ILP solver on the COMPAS dataset where we use the kNN-based similarity matrix. Here the initial amount of total error is 16,454.0. We show the results for three different total error limits (m). All three subfigures use the same legends.

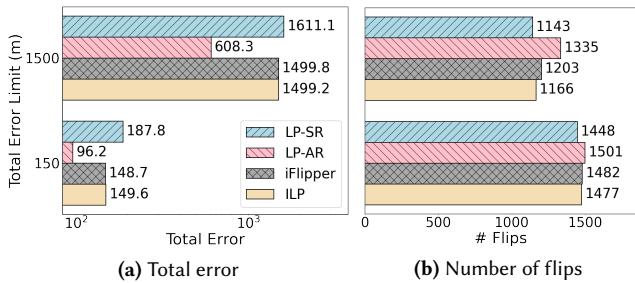


Figure 17: Ablation study for iFlipper on the COMPAS dataset and the kNN-based similarity matrix.

E BANK AND LSAC DATASETS RESULTS

We continue from Section 4.1 and show experimental results on the Bank and LSAC datasets when training a logistic regression model. Table 7 shows consistency scores with respect to the threshold-based similarity matrix. As a result, both datasets have almost 1.0 consistency scores, which means that they are already inherently fair in terms of individual fairness. The reason is that these datasets have much smaller total errors compared to other fairness datasets.

Dataset	Test Accuracy	Consistency Score
Bank	0.956	0.997
LSAC	0.825	0.986

Table 7: Accuracy and fairness results using logistic regression on the Bank and LSAC datasets.

F OPTIMIZATION SOLUTIONS FOR COMPAS

We continue from Section 4.4 and perform the same experiments on the COMPAS dataset where we use the kNN-based similarity matrix. In Figure 16, the key trends are still similar to Figure 10 where iFlipper (1) always satisfies the total error limit while *Greedy* and *Gradient* result in infeasible solutions for some cases while *kMeans* returns feasible solutions, but flips too many labels (Figure 16a), (2) provides the solution closest to the optimal in terms of the number of label flips (Figure 16b), and (3) is much faster than other optimization solutions (Figure 16c). Compared to the AdultCensus results in Figure 10, iFlipper is the most efficient because the COMPAS dataset is relatively small and has a smaller total error.

G ABLATION STUDY FOR COMPAS DATASET

We continue from Section 4.5 and provide the ablation study for the COMPAS dataset where we use the kNN-based similarity matrix in Figure 17. The observations are similar to those of Figure 11 where both adaptive rounding and reverse greedy algorithms are necessary for iFlipper to provide a near-exact solution. In addition, Table 8 shows the average runtime of each component in iFlipper in Figure 17 and the results are similar to Table 4 where the proposed algorithms are efficient in practice.

Method	Avg. Runtime (sec)
LP Solver (effectively includes Alg. 1)	5.87
+ Adaptive Rounding (Alg. 3)	0.09
+ Reverse Greedy (Alg. 4)	2.19

Table 8: Avg. runtimes of iFlipper’s components in Figure 17.

H COMPARISON WITH OTHER ML MODELS

In Section 4.3, we compared iFlipper with the baselines using logistic regression. In this section, we perform the same experiments using random forest and neural network models. Figure 18 and Figure 19 are the trade-off results using the random forest and neural network, respectively. The key trends are still similar to Figure 8 where iFlipper consistently outperforms the baselines in terms of accuracy and fairness. The results clearly demonstrate that how iFlipper’s pre-processing algorithm benefits various ML models.

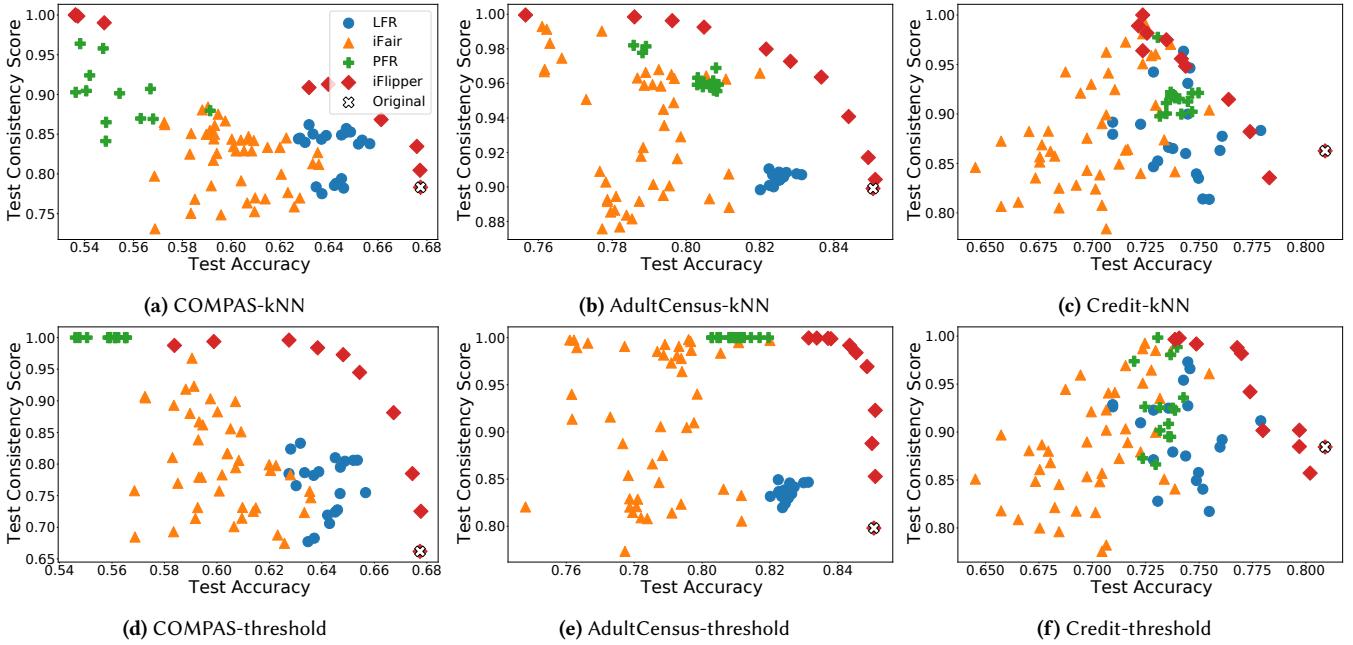


Figure 18: Accuracy-fairness trade-offs of random forest on the three datasets using the two similarity matrices. In addition to the four methods LFR, iFair, PFR, and iFlipper, we add the result of model training without any pre-processing and call it “Original.” As a result, only iFlipper shows a clear accuracy and fairness trade-off.

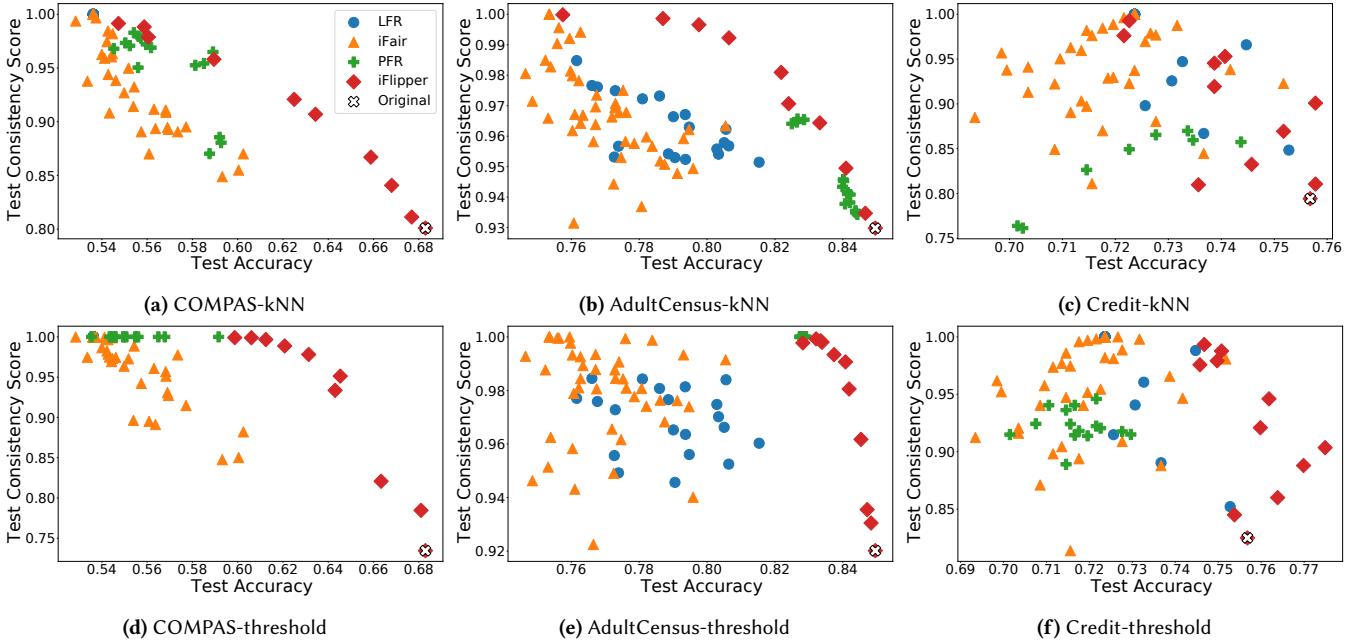


Figure 19: Accuracy-fairness trade-offs of neural network on the three datasets using the two similarity matrices. In addition to the four methods LFR, iFair, PFR, and iFlipper, we add the result of model training without any pre-processing and call it “Original.” As a result, only iFlipper shows a clear accuracy and fairness trade-off.