Fair Algorithms for Hierarchical Agglomerative Clustering

Anshuman Chhabra
Department of Computer Science
University of California
Davis, CA 95616
chhabra@ucdavis.edu

Prasant Mohapatra
Department of Computer Science
University of California
Davis, CA 95616
pmohapatra@ucdavis.edu

Abstract

Hierarchical Agglomerative Clustering (HAC) algorithms are extensively utilized in modern data science and machine learning, and seek to partition the dataset into clusters while generating a hierarchical relationship between the data samples themselves. HAC algorithms are employed in a number of applications, such as biology, natural language processing, and recommender systems. Thus, it is imperative to ensure that these algorithms are *fair*– even if the dataset contains biases against certain *protected groups*, the cluster outputs generated should not be discriminatory against samples from any of these groups. However, recent work in clustering fairness has mostly focused on center-based clustering algorithms, such as *k-median* and *k-means* clustering. Therefore, in this paper, we propose fair algorithms for performing HAC that enforce fairness constraints 1) irrespective of the distance linkage criteria used, 2) generalize to any natural measures of clustering fairness for HAC, 3) work for multiple protected groups, and 4) have competitive running times to vanilla HAC. To the best of our knowledge, this is the first work that studies fairness for HAC algorithms. We also propose an algorithm with lower asymptotic time complexity than HAC algorithms that can rectify existing HAC outputs and make them subsequently fair as a result. Moreover, we carry out extensive experiments on multiple real-world UCI datasets to demonstrate the working of our algorithms.

1 Introduction

Hierarchical Agglomerative Clustering (HAC) refers to a class of greedy unsupervised learning algorithms that seek to build a hierarchy between data points while clustering them in a bottom-up fashion. HAC algorithms are widely utilized in modern data science, with many applications in genetics (Pagnuco et al., 2017), genomics (Pollard and Van Der Laan, 2005), and recommendation systems (Merialdo, 1999), among others. These algorithms also possess two distinct advantages over non-hierarchical or *flat* clustering algorithms: 1) they do not require the number of clusters to be explicitly specified initially, and 2) they output a hierarchy between all samples in the dataset as part of the clustering process.

Recently, the machine learning community has realized the importance of designing *fair* algorithms. Traditional machine learning algorithms do not account for any biases that may be present (against certain minority groups) in the data, and hence, may end up augmenting them. Since machine learning now has many real-world applications, fair variants to many machine learning algorithms are being developed. However, work in designing fair clustering algorithms has mostly been focused on the *k-center*, *k-means*, *k-median*, and *facility location* clustering objectives (Bercea et al., 2018; Chierichetti et al., 2017; Backurs et al., 2019; Bera et al., 2019; Chen et al., 2019; Schmidt et al.,

Preprint. Under review.

2018; Ziko et al., 2019). Recent work has also investigated fair variants of graph partitioning and spectral clustering algorithms (Kleindessner et al., 2019; Anagnostopoulos et al., 2019). However, despite all the advantages of HAC algorithms mentioned in the aforementioned paragraph, there has been no work that proposes fair HAC algorithm variants. In this paper we seek to bridge this gap by making the following contributions:

- To the best of our knowledge, this is the first work that proposes fair algorithms for HAC
- The proposed fair HAC algorithm works for multiple protected groups, and irrespective of the linkage criterion used and we provide results on all the widely used criteria (single-linkage, average-linkage, complete-linkage) for real datasets
- We also provide analysis for the different *fairness costs* defined that our algorithm improves on, and show that it is more *fair* than vanilla HAC algorithms. Moreover, our algorithm achieves an asymptotic time complexity of $\mathcal{O}(fn^3)$ (f is the number of protected groups), which is comparable to $\mathcal{O}(n^3)$ for traditional HAC algorithms as f is usually a small number for most real-world applications
- As an alternative to performing fair HAC through optimization, we also present a post-clustering fairness algorithm. This algorithm *fixes* the output received from an HAC algorithm by subsequently making it fair, and has better asymptotic time complexity compared to our fair HAC algorithm ($\mathcal{O}(fn^2)$)

It is relevant to consider fairness in the context of hierarchical clustering and HAC through a motivating example. Consider we compute a hierarchical clustering of data samples representing movie reviews using a HAC algorithm. Also, assume that the reviews are selected from movies such that we have an equal number of movies directed by male directors, and those directed by female directors. The algorithm will compute a hierarchy linking movies to their reviews' corresponding *sentiments*. Each level of the hierarchy will represent certain sentiments. For example, we could have sentiments represented such as *fun*, *exciting*, *boring*, and *poor acting*. As we go up in the hierarchy, the number of sentiments will be grouped together or coalesced. For example, the earlier sentiments could be eventually grouped together to represent some notion of *positive* (containing *fun, exciting*) or *negative* (containing *boring, poor acting*) sentiments. Now, for these *positive* and *negative* sentiments, we would like to have movies by the male directors and the female directors represented more or less equally, since the dataset has them represented equally. If significantly more movies by male directors are in the positive cluster as opposed to the negative cluster, our algorithm is not fair to the movies made by the women directors. A vanilla HAC algorithm would not account for this *unfairness*, which is what we seek to correct through this work.

There are also more nuances associated with the above example. One could either choose to ensure fairness either for a certain specific set of sentiments or at each set of sentiments in the hierarchy. That is, in the above example, we could make sure that we have equal number of male-directed and female-directed movies for the *positive* and *negative* sentiment clusters, or we could ensure fairness for all the subclusters in the hierarchy, and their associated sentiments. For the latter, we would then try to have an equal number of male-directed and female-directed movies in all sentiment clusters (such as for the *fun, exciting, boring, poor acting* sentiment clusters, and for the clusters in the lower levels of the hierarchy). We cover both these possibilities in our work, and present our algorithm and approach in Section 3.

Another aspect of this problem is to ensure fairness at a lower computational cost if a vanilla HAC algorithm has already been run on a dataset, and the output for that has been obtained. We also cover this in the paper in Section 4. Thus, the paper is organized as follows: Section 2 discusses related work in the field, Section 3 details our proposed algorithm for performing fair HAC, Section 4 delineates how we can improve fairness for existing outputs from vanilla HAC algorithms, Section 5 describes our results on real data, and Section 6 concludes the paper.

2 Related Work

Recently, there has been a lot of work in providing clustering algorithms with fairness guarantees, or in proposing fair variants to existing clustering algorithms. As mentioned before, most of this work has looked at center-based clustering (such as k-means, and k-median clustering) (Bercea et al., 2018; Chierichetti et al., 2017; Backurs et al., 2019; Bera et al., 2019; Chen et al., 2019; Schmidt et al., 2018; Ziko et al., 2019), and spectral methods (Kleindessner et al., 2019; Anagnostopoulos et al., 2019). This line of work seeks to imposes some *fairness* constraints (such as *balance* (Chierichetti et al., 2017)) along with the original clustering distance based objective. Then the goal is to provide algorithms that approximate this fair objective. The first work to do this for k-median and k-center clustering, proposed by Chierichetti et. al, (Chierichetti et al., 2017) was based on the notion of disparate impact (Feldman et al., 2015), and ensured that in the case of two protected groups (red and blue) each cluster formed had points of both groups (colors) in roughly the same amount, measured using a metric known as *balance*. As a result, a lot of work has followed that improves upon these ideas either in terms of approximation rates (Backurs et al., 2019; Bercea et al., 2018), allowing for multiple protected groups (Bercea et al., 2018; Rösner and Schmidt, 2018), extending it to other clustering objectives (Kleindessner et al., 2019; Schmidt et al., 2018), or a combination of these, among others. Some work also looks at alternate notions of fairness for center-based clustering (Chen et al., 2019).

To the best of our knowledge, no work has investigated fairness in the context of hierarchical clustering (and specifically greedy HAC algorithms) so far. Moreover, in this paper, unlike the aforementioned work, we do not work with clustering objectives for hierarchical clustering, and instead seek to improve the greedy HAC algorithms in terms of ensuring fairness constraints themselves. There are multiple reasons for this. While some clustering objectives for hierarchical clustering have been proposed recently following Dasgupta's seminal work (Dasgupta, 2016), such as (Moseley and Wang, 2017; Cohen-Addad et al., 2019), none of these objectives are approximated well-enough by any general distance linkage criteria that are typically used in HAC (except for average-linkage). Moreover, greedy HAC algorithms despite being ad-hoc and heuristic approaches in nature, are very widely utilized in many application areas, especially in the biological sciences. For these reasons, we wanted to ensure fairness for these algorithms specifically, and provide a fair variant to the HAC problem, irrespective of the choice of distance linkage criteria. We also wanted this fair variant to resemble the general HAC algorithms closely, so that it can be readily implemented in applications. Furthermore, our algorithm works for multiple protected groups (but samples can only be assigned to one group at a time).

In the paper, we also provide an algorithm to arrive at a fair clustering from an existing HAC output (Section 4). Work on improving existing general clustering outputs has recently been undertaken (Davidson and Ravi, 2020). However, the approach in (Davidson and Ravi, 2020) could not be directly applied to the HAC problem, since constraints still need to be developed specific to the clustering approach (which is not trivial to do). Instead, we found it simpler to come up with a recursive algorithm to *fix* fairness post-HAC.

3 Performing Fair HAC

First, we need to define the vanilla HAC process formally. Let $X \in \mathbb{R}^{n \times m}$ be our dataset. Then the HAC on X denoted by $HC(X)$ is a hierarchical partitioning of X that is represented by a binary tree T (also called a *dendogram*) of height at most $\lceil \frac{n+1}{2} \rceil$, where each level of T represents a set of disjoint merges between subclusters. Each node of T at any level represents a subcluster of points. A HAC algorithm first considers each of the n samples of X to be singleton subclusters, and then chooses sets of two subclusters to merge together (that is, a point from X can only belong to any one subcluster at any particular level). The lowest level of T are leaves, and comprise of all the n points of X. The root of T is a single node/cluster that contains all of X. Let $C_1, C_2, ..., C_s$ be the subclusters at any level of T. Then $C_1 \cup C_2 \cup ... \cup C_s = X$. The choice of which two subclusters should be merged is made by finding two subclusters C_i and C_j such that they minimize a *linkage criterion* denoted by $D(C_i, C_j)$. There are many linkage criteria that can be used. For example, single-linkage is defined as $D(C_i, C_j) = \min_{x_i \in C_i, x_j \in C_j} d(x_i, x_j)$ and complete-linkage is defined as $D(C_i, C_j) = \max_{x_i \in C_i, x_j \in C_j} d(x_i, x_j)$. In the paper $d(x, y)$ is the Euclidean distance between two points x and y, but other distance metrics can also be used.

Next, we need to define our notion for fairness. Also, it is important to note that in this work we consider data points to only belong to one protected group, that is, *multiple* assignments to protected group for the same point are not allowed. In most works of fairness in clustering, the notion of *balance* is used to ensure that clusters contain at least a minimum number of points from each protected group and no more than a maximum number of points from each protected group (Chierichetti et al., 2017; Bera et al., 2019; Bercea et al., 2018). In this paper, we work with a similar idea, but one which flows more naturally for HAC. We utilize the notion of *proportional fairness* which maintains the same proportion of points of each protected group in a cluster as they are in the entire dataset X (also called the *ideal proportion*). For a protected group with s members the ideal proportion would be s/n. It is also easy to see that this is a general setting for *balance* (Chierichetti et al., 2017)– if we have f protected groups, instead of trying to strive for the *ideal proportion*, we can replace it with n/f, which would perfectly balance all protected groups in each cluster. This is trivial to do, and requires no change to our algorithms. We will now define these ideas mathematically.

Definition 3.1. (**α-Proportional Fairness**) Let $F \in \mathbb{R}^{f \times n}$ be the set of all protected groups where each protected group $g \in F$ is $\{0,1\}^n$. Thus, if a data sample from X belongs to a particular group g then at that index the vector g contains a 1, otherwise a 0. Moreover, a cluster $C = \{x_i | i \in I\}$ where $C \subset X$, and I is the index set containing indices of the points in X which belong to cluster C, that is $X = \{x_i\}_{i=1}^n$. The proportion of group g members in C is denoted by $\delta_g^C = \frac{1}{|C|}\sum_{x_i \in C} g(i)$ and the ideal proportion $\phi_g = \frac{1}{n}\sum_{x_i \in X} g(i)$. Then α-Proportional Fairness for cluster C and protected group g is maintained if the following condition holds: $|\delta_g^C - \phi_g| \leq \alpha$.

Definition 3.2. (**Fairness Cost (FC)**) Let $HC(X)$ be the output of some hierarchical clustering on X. Then the fairness cost on some level with k clusters of the $HC(X)$ tree measures how "close" each cluster of points at this level (denoted by C_i, where $i = \{1, 2, .., k\}$) is to the ideal proportion ϕ_g for each protected group g in F. Mathematically, the Fairness Cost can then be defined as: $\sum_{i=1}^{k}\sum_{g \in F} |\delta_g^{C_i} - \phi_g|$.

Definition 3.3. (**Hierarchical Fairness Cost (HFC)**) The HFC is basically the summation of the FC over all the intermediate clusters that are formed as a result of the HAC process. Let $HC(X) = T$ be the output of some hierarchical clustering on X. Then for the entire HAC tree T, we start with n singleton clusters as leaf nodes, and then keep merging them one at a time to arrive at one cluster (root of T) that contains all nodes. Then, let there be 1 to A intermediate merging stages of the HAC process, and the number of associated clusters associated with stage A be maintained in a set S_A. Then the Hierarchical Fairness Cost is defined as follows: $\sum_{i=1}^{A}\sum_{C \in S_i}\sum_{g \in F} |\delta_g^C - \phi_g|$.

3.1 Fair HAC: The FHAC Algorithm

The goal for our fair algorithm will then be to minimize the FC or the HFC, and ensure that at a level with some clusters, we have at least maintained some α'-Proportional Fairness where α' is some constant value. We also run the algorithm till some k clusters are remaining, and return after that. If the entire tree needs to be computed, $k = 1$. Now, we cannot possibly enforce a fixed α'-Proportional Fairness for each merge throughout the algorithm as in that case the clustering cost in terms of minimizing distance might be too large so no clusters would be chosen for merging. This also might not be possible to achieve in a particular dataset. Thus, we keep tightening the bound on proportional fairness as we start getting closer to k clusters, and conversely we keep loosening the bound on minimizing distance as we start getting closer to k clusters. The greedy Fair Hierarchical Agglomerative Clustering (FHAC) algorithm is described as Algorithm 1. Moreover, it is important to note that Algorithm 1 works irrespective of the choice of linkage criteria.

Algorithm 1 resembles the working of vanilla HAC except for some key distinctions that allow it to be fairer than traditional HAC. The key difference is in incorporating fairness constraints by foregoing the minimum distance linkage criterion constraint to allow for the selection of other clusters that can be merged, and lead to a fairer output tree. We will cover the algorithm in detail now.

Throughout, we maintain a distance matrix $\mathcal{D} \in \mathbb{R}^{n \times n}$ between clusters to improve the runtime of the algorithm, and this can be done by computing the distances using the linkage criterion provided as input. Since we start out with singleton data samples as clusters, in line 2 we find the pairwise distances between all the points in X initially. Line 3 signifies the start of the clustering process with a while loop that checks to see if we have k clusters; if so, we can exit. Here C is the set of clusters and the data points associated with them for that level of the clustering tree. Then for each level, we perform initializations (lines 4-7): $P_g \in \{0,1\}^f$ is a vector that is used to check if we have satisfied the proportionality constraint for a group, and we will discuss this later on. On line 5, $d_{\min}$ signifies the initialization of the distance (computed using the linkage criterion) between the clusters chosen to be merged for this level. It is important to note that it might not be the minimum distance between all these clusters, as we relax the distance constraint so as to allow for fair solutions.

Algorithm 1 Proposed FHAC Algorithm

Input: $X, F, k, D(.,.), \mathcal{Z}_\alpha : \mathbb{R} \to \mathbb{R}, \mathcal{Z}_\beta : \mathbb{R} \to \mathbb{R}$
Output: Fair HAC tree T_{fair}

1: **set** $C \leftarrow X$
2: **compute** $\mathcal{D}_{n_1,n_2} = D(n_1, n_2), \forall (n_1, n_2) \in X \times X$
3: **while** $|C| \geq k$ **do**
4: $\quad P_g = 0, \forall g \in F$
5: $\quad d_{\min} \leftarrow \infty$
6: $\quad \alpha \leftarrow \mathcal{Z}_\alpha(|C| - k)$
7: $\quad \beta \leftarrow \mathcal{Z}_\beta(n - |C|)$
8: $\quad$ **for** each $(c_i, c_j) \in C \times C$, *s.t.* $c_i \neq c_j$ **do**
9: $\quad\quad$ **for** each $g \in F$ **do**
10: $\quad\quad\quad \delta_g^{c_i+c_j} \leftarrow \frac{\delta_g^{c_i}|c_i| + \delta_g^{c_j}|c_j|}{|c_i|+|c_j|}$
11: $\quad\quad\quad$ **if** $|\delta_g^{c_i+c_j} - \phi_g| \leq \alpha$ **then** $P'_g = 1$, **else** $P'_g = 0$
12: $\quad\quad$ **end for**
13: $\quad\quad$ **if** $\sum_{g \in F} P'_g \geq \sum_{g \in F} P_g$ **then**
14: $\quad\quad\quad$ **if** $d_{\min} + \beta > \mathcal{D}_{c_i,c_j}$ **then**
15: $\quad\quad\quad\quad d_{\min} \leftarrow \mathcal{D}_{c_i,c_j}$
16: $\quad\quad\quad\quad P_g \leftarrow P'_g, \forall g \in F$
17: $\quad\quad\quad\quad (c_1^m, c_2^m) \leftarrow (c_i, c_j)$
18: $\quad\quad\quad$ **end if**
19: $\quad\quad$ **end if**
20: $\quad$ **end for**
21: $\quad$ **merge** $c_1^m \leftarrow c_1^m + c_2^m$
22: $\quad$ **update** C **with newly merged clusters**
23: $\quad$ **recompute** $\mathcal{D}_{c_1,c_2} = D(c_1, c_2), \forall (c_1, c_2) \in C \times C$
24: $\quad$ **update** T_{fair} **with merge**
25: **end while**
26: **return** T_{fair}

Next, the proposed Algorithm 1 essentially tightens the fairness constraint (and loosens the distance constraint) as we keep constructing the clustering tree from bottom to top. The fairness constraint tightening is achieved using the function $\mathcal{Z}_\alpha : \mathbb{R} \to \mathbb{R}$ and the distance constraint loosening is achieved using the $\mathcal{Z}_\beta : \mathbb{R} \to \mathbb{R}$. $\mathcal{Z}_\alpha$ and $\mathcal{Z}_\beta$ are both monotonically increasing functions and are parameterized appropriately for the dataset X (their parameterization is discussed later). We compute the actual constraint bounds α and β using $\mathcal{Z}_\alpha(|C|-k)$ (line 6) and $\mathcal{Z}_\beta(n-|C|)$ (line 7), respectively, where C in each iteration of the while loop (line 3) denotes the current state of clusters at some level of the tree. This is intuitively obvious– if both functions are monotonically increasing, then α reduces as we get closer to k clusters, tightening the fairness constraint, whereas β increases as we get closer to k clusters and thus, loosens the distance bound. In the paper, we use $\mathcal{Z}_\alpha(x) = \theta_1 x + \alpha_0$ and $\mathcal{Z}_\beta(x) = \theta_2 x + \beta_0$ for our experiments and results, but any monotonic function can be used.

Line 8 signifies the start of selecting pairs of clusters using a for loop, and then checking to see if they are the optimal pair for merging in this level. In line 9, we also start iterating over all the groups in F so as to ensure that the fairness constraints are met for all of them. Line 10 basically computes the proportion of protected group g members if the current pair of clusters were to be merged. Moreover, this takes constant time since $\delta_g^{c_i}$ and $\delta_g^{c_j}$ (basically δ_g^c for all clusters) can be stored at the end of the iteration of the while loop, and then calculated inside the loop using line 10.

$P_g \in \{0,1\}^f$ and $P'_g \in \{0,1\}^f$ are vectors, that help us keep track of how many protected groups for a potential cluster merge we have met the proportional fairness constraints for so far. If $P'_g = 1$ we have met the proportionality condition for group g in this iteration, otherwise $P'_g = 0$. P_g is the same but maintains global state– that is, it keeps track of the same condition but for the "best" cluster merge pair found so far. Thus, in line 11, we check to see if we have met α-Proportional Fairness if clusters c_i and c_j were to be merged, and then appropriately set the value for P'_g.

Next, once we have done this for all the groups (lines 9-12), we compare the current cluster pair with the "best" cluster merge pair found so far. This is done using P_g (global state) and P'_g (local state) for all groups on line 13. If the current cluster pair is a better choice, we proceed to checking for whether we improve on the minimum distance constraint (relaxed using β) on line 14, and then update variables accordingly. Towards the end of the while loop we recompute $\mathcal{D}$ after the cluster merges take place to reduce the overall lookup (line 23), the clusters to merge (line 21), and then update our set of current clusters C (line 22) before adding it to our clustering tree, T_{fair}.

Unfortunately, the running times for generalized implementations of HAC algorithms are $\mathcal{O}(n^3)$ and prohibitively expensive already. Our FHAC Algorithm (Algorithm 1) attempts to enforce proportional fairness constraints up until we have k clusters in our tree. Despite this, as is evident, it achieves an asymptotic time complexity of $\mathcal{O}(n^2(n-k)|F|)$ or $\mathcal{O}(n^3 f)$ which is comparable to vanilla HAC since f is usually very small in many real-world datasets ($f \leq 10$).

Furthermore, the benefit of employing Algorithm 1 for fair HAC, is that it can be utilized to minimize any cost function that is required for ensuring fairness of the hierarchical clustering process. In the previous section, we defined the Fairness Cost (FC) for the level with k clusters, and the general Hierarchical Fairness Cost (HFC) for the entire tree. Algorithm 1, as we will show next, can minimize either the FC or the HFC by estimating the values of parameters of the functions $\mathcal{Z}_\alpha$ and $\mathcal{Z}_\beta$ for that particular choice of cost.

3.2 Estimating the parameters of $\mathcal{Z}_\alpha$ and $\mathcal{Z}_\beta$

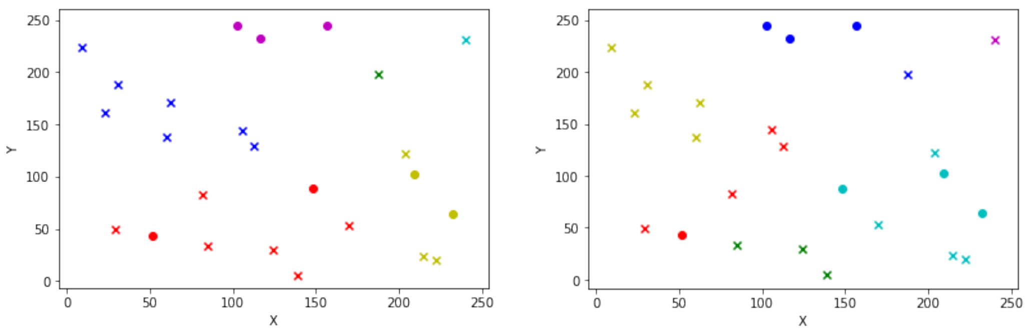
While our method of parameter estimation would hold for any $\mathcal{Z}_\alpha$ and $\mathcal{Z}_\beta$, it is simpler to consider concrete definitions for $\mathcal{Z}_\alpha$ and $\mathcal{Z}_\beta$. As mentioned before, we use $\mathcal{Z}_\alpha(x) = \theta_1 x + \alpha_0$ and $\mathcal{Z}_\beta(x) = \theta_2 x + \beta_0$, for our empirical results, and hence, we have to estimate the parameters α_0, β_0, θ_1, and θ_2. To do this, while any hyperparameter search algorithms can generally be utilized, we use a simple black-box minimization approach (Regis and Shoemaker, 2005) for the search. The approach essentially does the following: it treats the FHAC algorithm as a black-box and attempts to find the parameters α_0^*, β_0^*, θ_1^*, θ_2^* that minimize some objective/cost function (here, we use either the FC for the k clusters or the HFC for the entire tree) using a cubic RBF as a response surface and some potential solutions as candidate points. While this does involve recomputing the entire hierarchical clustering every time to test the suitability of candidate parameters found by the optimization approach (Regis and Shoemaker, 2005) (named CORS), it only runs for a fixed number of iterations t and also only selects c candidate parameters to test the object function where $c << n$. Thus, including hyperparameter search, the runtime of our FHAC algorithm is still $\mathcal{O}(fn^3)$. Moreover, this approach converges (for more details refer to (Regis and Shoemaker, 2005)), and the open-source implementation (Knysh and Korkolis, 2016) worked very well, and proved to be very robust in our experiments. Our experiments and results for minimizing both the FC and the HFC are detailed in the Results section. We can now also show how our approach fares in terms of either the FC or HFC, compared with traditional HAC:

Theorem 3.1. *For appropriately parameterized monotonically increasing functions $\mathcal{Z}_\alpha$ and $\mathcal{Z}_\beta$, Algorithm 1 computes a T_{fair} that is more (or equivalently) fair to the hierarchical clustering tree T obtained from vanilla HAC, according to the chosen cost metric (Fairness Cost or Hierarchical Fairness Cost).*

Proof. Let us first consider choosing the FC as the cost. Given the fact that $\mathcal{Z}_\alpha$ is monotonically increasing and line 11 of Algorithm 1, it is evident that any of the k clusters at the k-cluster level of T_{fair} achieves at least $\mathcal{Z}_\alpha(0)$-Proportional Fairness. Therefore, Fairness Cost for T_{fair} is bounded as:

$$\begin{aligned} FC(T_{fair}) &= \sum_{i=1}^{k} \sum_{g \in F} |\delta_g^{C_i} - \phi_g| \\ &\leq fk\mathcal{Z}_\alpha(0) \end{aligned}$$

Thus, this is the maximum value of FC that can be attained. However, for the case of vanilla HAC, unless the minimum distance clusters selected for merging also are the most optimal choice for



(a) Vanilla HAC clusters

(b) FHAC clusters

Figure 1: Toy Example for Algorithm 1

enforcing fairness, the maximum FC cannot be bounded. Moreover, since parameters of $\mathcal{Z}_\alpha$ and $\mathcal{Z}_\beta$ together minimize FC for dataset X, we have:

$$FC(T_{fair}) \leq FC(T)$$

These arguments can be trivially extended to the case when HFC is chosen as the cost as well. $\square$

3.3 Results on Toy Data

We generate two-dimensional data from a uniform distribution for our toy example. We generate samples between the range $[0, 250]$. There are 25 points in total (n) and $k = 6$. Moreover, there are two protected groups, denoted by 0 and 1, with ideal proportions denoted by $\phi_0 = 0.28$ and $\phi_1 = 0.72$, respectively. We estimate the optimal choice of the parameters for the FHAC algorithm using the CORS optimization approach described above, and then compare how fair the final clusters are for vanilla single-linkage HAC and single-linkage FHAC (Algorithm 1). We compare by iterating over each cluster and for each group, and then summing over the absolute error between the ideal proportion and the obtained proportion. That is, assuming the set of clusters is denoted by $\{C_1, C_2, .., C_k\}$, we find for both cases the Fairness Cost (FC) = $\sum_{i=1}^{k}\sum_{g\in F}|\delta_g^{C_i} - \phi_g|$. The values of the parameters are as follows: $\alpha_0 = 0.49345, \theta_1 = 3.8724, \beta_0 = 9.0568, \theta_2 = 0.20181$. Moreover, FC for vanilla HAC with single-linkage is 3.42, and for FHAC with single-linkage is 2.89. Thus, we find that the fairness achieved by our algorithm is much better, and we obtain proportionally fair clusters as a result. The clusters for the toy example are shown in Figure 1. In both figures the 0 protected group is denoted by the $\circ$ marker, and the 1 protected group is denoted by the $\times$ marker.

4 Making Existing Vanilla HAC Trees Fair

In this section, we consider an alternate way of achieving fairness while performing HAC. In this case, we assume that we already have a tree T from performing regular HAC on the dataset X. The goal is to then come up with a tree T_{fair} that is *fairer* than the tree T with respect to the Fairness Cost (for the level of T with k clusters, as before). We propose a recursive algorithm that achieves this with a time complexity of $\mathcal{O}(fn^2)$. For now we will also consider balanced binary HAC trees for our analysis, but our algorithm can be applied to unbalanced trees as well. As a preliminary, in a balanced HAC binary tree T with n leaf nodes, there are $\log_2(n)$ levels, where each level j has 2^j nodes where $j \in \{1, .. \log_2(n)\}$. Moreover, let the level with k clusters be l_k and the clusters at level l_k be $M_k = \{m_1, m_2, ..., m_k\}$.

The way we present our algorithm is to aid readability, and hence, we first describe some of the notation used. Throughout the algorithm, C_l denotes a set of k sets, such that each set $c_i \in C_l$ represents the clusters at some level l of T that are actually subclusters of $m_i \in M_k$. That is, c_i was merged with some other cluster in the vanilla HAC to eventually become a part of m_k at the level with k clusters. It takes at the most linear time to construct C_{l+1} from C_l (discussed later on). Moreover, for every element x that belongs to the set $c_i \in C_l$, let x_S denote it's sibling in the tree,

x_L it's left child and x_R it's right child. In the context of HAC, x_S denotes the cluster x was merged with to form it's parent cluster, and x_L and x_R are the immediate subclusters that were merged to form x. Also, obtaining x_S, x_L, x_R from x can be done in constant time since we are dealing with a tree. Somewhat similar to Algorithm 1, here we use vectors $P_g^c \in \{0,1\}^f$ to determine if we have met the fairness constraint for group $g \in F$ and cluster $c \in M_k$. Finally, $\epsilon \geq 0$ is a hyperparameter that is used to check for the distance constraint. If ϵ is set to ∞, the distance constraint will not be considered. All of these details are discussed in depth later on. We also assume that we have the distance matrix $\mathcal{D}$ for each level of the tree, thus looking up between-cluster distances should take only constant time. Then, the proposed algorithm is denoted as the Fair Post-HAC (FP-HAC) algorithm, and is presented as Algorithm 2.

Algorithm 2 Proposed FP-HAC Algorithm

Input: T, k, F, $D(.,.)$, $\epsilon \in \mathbb{R}$
Output: Fair HAC tree T_{fair}

1: **set** $l \leftarrow l_k$
2: **obtain** C_{l_k} using l_k and T
3: **set** $C_l \leftarrow C_{l_k}$

4: **function** FIX_TREE(C_l, l, F)
5:     **if** $l > \log_2(n)$ **then return**
6:     **for** all $(c_i, c_j) \in C_l \times C_l$ s.t. $i \neq j$ **do**
7:         **for** all $(x, y) \in c_i \times c_j$ **do**
8:             **obtain** c_i' and c_j' by **swapping** x and y in C_l' (a copy of C_l)
9:             **let** the resulting M_k' clusters of C_l' be m_i' and m_j'
10:             **for** each $g \in F$ **do**
11:                 **if** $|\delta_g^{m_i'} - \phi_g| \geq |\delta_g^{m_i} - \phi_g|$ **then** $P_g^{m_i} = 1$, $P_g^{m_i'} = 0$
12:                 **else** $P_g^{m_i'} = 1$, $P_g^{m_i} = 0$
13:                 **end if**
14:                 **if** $|\delta_g^{m_j'} - \phi_g| \geq |\delta_g^{m_j} - \phi_g|$ **then** $P_g^{m_j} = 1$, $P_g^{m_j'} = 0$
15:                 **else** $P_g^{m_j'} = 1$, $P_g^{m_j} = 0$
16:                 **end if**
17:             **end for**
18:             **if** $\sum_{g \in F} P_g^{m_i} \leq \sum_{g \in F} P_g^{m_i'}$ **and** $\sum_{g \in F} P_g^{m_j} \leq \sum_{g \in F} P_g^{m_j'}$ **then**
19:                 **if** $|\mathcal{D}_{x_S,y} - \mathcal{D}_{x_S,x}| \leq \epsilon$ **and** $|\mathcal{D}_{y_S,x} - \mathcal{D}_{y_S,y}| \leq \epsilon$ **then**
20:                     **replace** $c_i \leftarrow c_i'$ and $c_j \leftarrow c_j'$
21:                 **end if**
22:             **end if**
23:         **end for**
24:     **end for**
25:     **obtain** C_{l+1} by replacing all $x \in c_i$ with (x_L, x_R), $\forall c_i \in C_l$
26:     FIX_TREE($C_{l+1}, l+1, F$)
27: **end function**

28: **update** last level $l = \log_2(n)$ of T **using** final C_l
29: **set** $T_{fair} \leftarrow T$
30: **return** T_{fair}

Algorithm 2 works as follows: on lines 1-3, we initialize the starting level l as l_k, and set C_l as C_{l_k}. Lines 4-28 depict our subroutine FIX_TREE that is used to achieve fairness recursively. The base case on line 5 is simple, if we have covered the maximum depth of the tree, we can return. Next, on line 6 and 7, we essentially set-up the way through which we meet the fairness constraint. Essentially, we will be swapping nodes/clusters between one of the k clusters one at a time, and see if that improves fairness or not. By greedily making swaps that reduce the FC, we will have a *fairer* tree as a result.

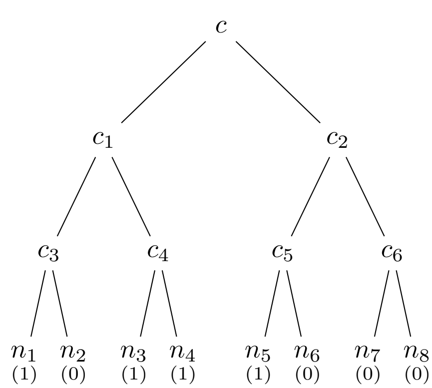
On line 6, we select two separate (of the k) cluster sets from C_l and on line 6 we select two clusters each (x and y) from these cluster sets c_i and c_j respectively. We then create a copy of the cluster set C_l, and denote it as C_l'. Here, we swap the clusters x and y (including all their children/subclusters) on line 8. We denote the modified clusters of the changed M_k cluster set as m'_i and m'_j, respectively (originally m_i and m_j). Next, for all groups in F, we check to see how many groups the swap improved fairness for as opposed to the original case. We do this for both m_i, m'_i, and m_j, m'_j, and appropriately set $P_g^{m_i}(P_g^{m'_i})$ and $P_g^{m_j}(P_g^{m'_j})$ to 0(1) if fairness was improved as a result of the swap, and vice versa (lines 10-17). Then, we check to see if fairness was improved for more groups as a result of the swap, than without the swap (line 18). It is also important to note that similar to Algorithm 1, the current proportion for a group g in cluster c (δ_g^c) can easily be maintained and accessed in constant time.

In case the previous if statement holds true, we check for the distance criterion. As mentioned before, we assume we have the distance matrix for between-cluster distances at each level $\mathcal{D}$ which is utilized and created during the vanilla HAC algorithm itself. Thus, here we utilize the hyperparameter ϵ which is used to set an acceptable threshold to the change in distance as a result of the swap. After the first if statement, on line 19, we check to see if both $|\mathcal{D}_{x_S,y} - \mathcal{D}_{x_S,x}|$ and $|\mathcal{D}_{y_S,y} - \mathcal{D}_{y_S,x}|$ are less than or equal to ϵ, and only then finalize the swap. The sibling of cluster $x(y)$, which is cluster $x_S(y_S)$, is the subcluster that $x(y)$ was chosen to be merged with. Thus, by comparing the increase in distance with the sibling, we can ensure that clusters being swapped are not too "different". Moreover, if ϵ is set to 0, it is probable that no swaps will take place, whereas if ϵ is very large, all swaps will go through that improve fairness. Furthermore, ϵ could also be set to a different value for each tree level and thus allow for better flexibility in making decisions. If this condition is met, we update c_i and c_j with c'_i and c'_j, respectively, finalizing the swap process.

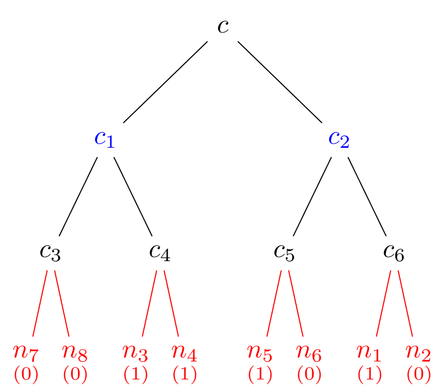
Finally, on line 25, we obtain the set C_{l+1} from C_l, simply by replacing each cluster in each set of C_l with it's children/subclusters, x_L and x_R. Finally, we recursively call FIX_TREE on C_{l+1} and $l+1$ (line 26). Since, only the last level of T needs to be changed (it contains the n data points), to make it fair, we update it in T. We return the updated tree as T_{fair}.

Note: The benefit of using Algorithm 2 to improve fairness of the vanilla HAC tree is that the earlier recursive steps make more *major swaps* between cluster memberships, and progressively *individualized swaps* are made (that are computationally more expensive as well). That is, at a higher level of the tree, a large number of points are swapped collectively, which is less computationally expensive than individually swapping points from each subcluster (which is what happens at lower levels of the tree). Therefore, if at an earlier stage of the recursive algorithm, the fairness constraint for all clusters and their groups is already met (with respect to some α, that is α-Proportional Fairness), we do not need to recur on the more costly lower levels of the tree, and we can exit as is. While we have not mentioned this explicitly in Algorithm 2 to make it more readable, this is the precise reason why Algorithm 2 is beneficial over just making all possible swaps at the last level of the tree itself.

4.1 Toy Examples for Algorithm 2

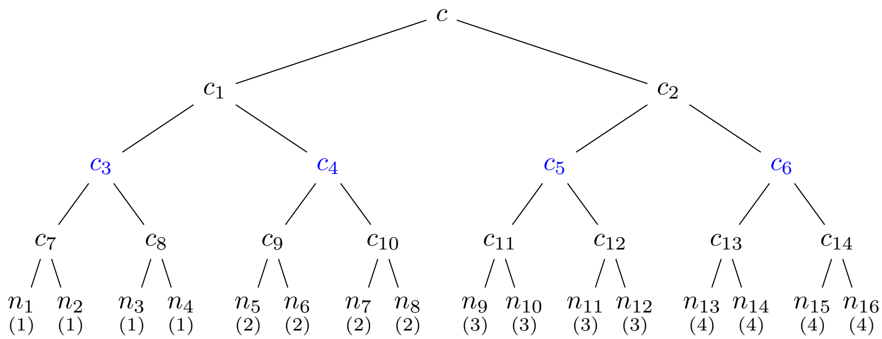


(a) Vanilla HAC Tree T

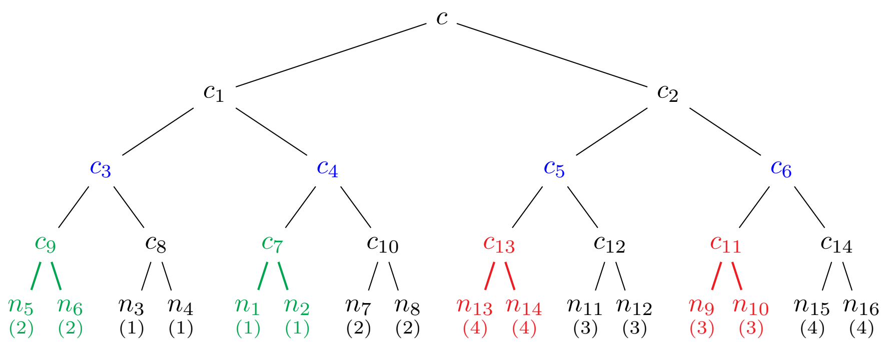


(b) *Fixed* Fair HAC Tree T_{fair}

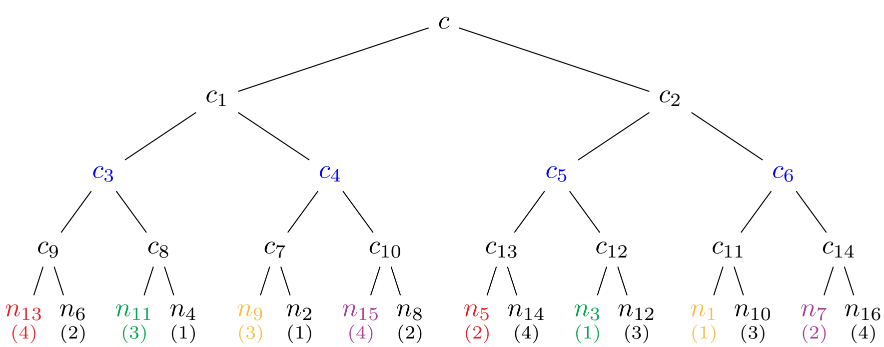
Figure 2: Toy Example #1 for Algorithm 2



(a) Initial vanilla HAC tree T, $k = 4$, $l = l_k = 2$



(b) Intermediate tree: Swapping subclusters at level ($l = 3$)



(c) Final tree: Last level ($l = 4$) swapping of leaf nodes

Figure 3: Toy Example #2 for Algorithm 2

4.1.1 Toy Example #1

We now consider a simple toy example to showcase the working of Algorithm 2. In the example, there are $n = 8$ samples in the dataset X, and there are 2 protected groups in F (denoted by 1 or 0). On running vanilla HAC on X, we get the $HC(X) = T$ tree as shown in Figure 2a. In this example, we consider $k = 2$, thus the two clusters we obtain are c_1 and c_2. The cluster c_1 contains subclusters c_3 and c_4, and the points $n_1(1), n_2(0), n_3(1)$ and $n_4(1)$, and cluster c_2 contains subclusters c_5 and c_6, and the points $n_5(0), n_6(1), n_7(1)$ and $n_8(0)$. Here, the protected groups are written in parantheses following each point. It is easy to see that $\phi_0 = \phi_1 = 0.5$ for X, and that the cluster c_1 and c_2 are not as fair as can be. Also, the hyperparameter ϵ is set to a very large number so that all swaps are permissible, that is $\epsilon \leftarrow \infty$. We thus run Algorithm 2 on T to obtain T_{fair}. Essentially, the problem is solved in only one level of recursion, as the swap between c_3 and c_6, *fixes* fairness in T. After the swap c_1 contains $n_7(0), n_8(0), n_3(1), n_4(1)$, and thus $\delta^{c_1}_{1(0)} = \phi_{1(0)} = 0.5$. Similarly,

c_2 contains $n_5(1), n_6(0), n_1(1), n_2(0)$, and $\delta^{c_2}_{1(0)} = \phi_{1(0)} = 0.5$. The modified HAC tree T_{fair} is shown in Figure 2b, and the updated last level of singleton nodes is highlighted in red.

4.1.2 Toy Example #2

We now consider a more complex example (Figure 3) to showcase the working of Algorithm 2. Here consider $n = 16$ and $f = 4$, and $\epsilon \leftarrow \infty$ as before. We are given $k = 4$, so Algorithm 2 will start recursion from level $l_k = 2$ where we have clusters c_3, c_4, c_5, and c_6. Therefore, Algorithm 2 has to ensure that groups in these clusters come as close to meeting the fairness constraint as possible. Also as can be seen in Figure 3a: $\phi_1 = \phi_2 = \phi_3 = \phi_4 = 0.25$. For the initial HAC tree T given in Figure 3a, c_3 has all the data points that belong to group 1, c_4 has points that belong to group 2, c_5 has points that belong to group 3, and c_6 has points that belong to group 4. Now, on running Algorithm 2 on T, the initial value of $l = l_k = 2$. There is no possible swap that improves fairness at this level, so the algorithm moves on to the next level of the tree. Now $l = 3$, and we have some potential subclusters that can be swapped to improve fairness. It can be seen that the following swaps will be made: $c_7 \Leftrightarrow c_9$, and $c_{11} \Leftrightarrow c_{13}$. The resulting intermediate tree after these swaps are made is shown in Figure 3b, and nodes involved in a swap are highlighted with the same color. That is, in Figure 3b, the red highlighted nodes are c_{13} and c_{11} and their subtrees, and c_9 and c_7 and their subtrees are highlighted in green since these are the respective swaps that went through. As a result, the fairness of each of the 4 clusters at l_k has improved. Next, we will recur to $l = 4$, where all the singleton clusters (data samples) are present as leaf nodes (shown in Figure 3c). Here, the following swaps will be made: $n_5 \Leftrightarrow n_{13}$ (highlighted in red), $n_3 \Leftrightarrow n_{11}$ (highlighted in green), $n_1 \Leftrightarrow n_9$ (highlighted in yellow), and $n_7 \Leftrightarrow n_{15}$ (highlighted in purple). We can see in Figure 3c that the fairness constraint has been met perfectly for all the 4 clusters and all the groups, that is $\delta^{c_i}_g = 0.25$, where $g = \{1, 2, 3, 4\}$ and $i = \{3, 4, 5, 6\}$. Now, the last level containing leaf nodes is copied over in T and the resulting tree T_{fair} is returned as the output by Algorithm 2.

4.2 Theoretical Results

In this section, we detail theoretical results regarding the asymptotic time complexity of Algorithm 2, as well as delineate it's fairness guarantees.

Theorem 4.1. *If a tree T obtained from performing traditional HAC on dataset X, is provided as input to Algorithm 2, the resulting output tree T_{fair} is equivalent or more fair than T with respect to the Fairness Cost at level l_k.*

Proof. In Algorithm 2, we can easily say that the FC for T and T_{fair} can be equivalent in multiple cases. If T is already as fair as possible, no swaps will go through, and hence T and T_{fair} will be equivalent. Or, ϵ could be chosen such that $\epsilon \approx 0$, because of which swaps might also not go through, again resulting in T_{fair} and T being equivalent. Thus, in cases such as these, the FC for both T and T_{fair} will be the same.

Now consider that some swaps do go through. Since each swap is made permanent only if it achieves fairness for more groups than achieved prior to the swap (line 18), it has to reduce the Fairness Cost for the clusters m_i and m_j it was made for. Moreover, since m_i and m_j are located at level l_k of T with k clusters, the FC has to reduce with respect to level l_k (lines 11-15). Thus, it is easy to see that:

$$FC(T_{fair}) \leq FC(T)$$

□

Theorem 4.2. *Algorithm 2 achieves an asymptotic running time complexity of $\mathcal{O}(n^2 f)$ to compute T_{fair} for any arbitrary balanced binary HAC tree T provided as input to it.*

Proof. At each level i of tree T, there are 2^i nodes/clusters. Moreover, at level l_k, there will be 2^k nodes/clusters, and thus, each cluster could be potentially swapped $2^k - 1$ times. We now seek to figure out the total work being done by swapping at any level i. At any level i, we will still have $2^k * (2^k - 1)$ outer swaps between the M_k clusters since we are not going to swap subclusters within the same cluster. On top of this, there are a maximum of 2^i nodes within the same cluster, and a maximum of 2^i possible nodes in the other clusters as well. Also, we would do constant constraint checking work for all groups in F, that is $|F| = f$ times, for each level i. Moreover, apart from this,

Table 1: Description of Datasets

Label	Dataset	n	k	Features	Sensitive Attribute	Protected Groups
`bank1`	`bank`	120	4	age, balance, duration	default	yes ($\phi_0 = 0.5$), no ($\phi_1 = 0.5$)
`bank2`	`bank`	120	4	age, balance, duration	marital	married ($\phi_0 = 0.33$), single ($\phi_1 = 0.33$), divorced ($\phi_2 = 0.33$)
`census`	`census`	110	3	age, education-num, final-weight, capital-gain, hours-per-week	sex	male ($\phi_0 = 0.363$), female ($\phi_1 = 0.637$)
`creditcard`	`creditcard`	125	4	age, bill-amt 1 — 6, limit-bal, pay-amt 1 — 6	education	graduate school ($\phi_0 = 0.24$), university ($\phi_1 = 0.24$), high school ($\phi_2 = 0.36$), others ($\phi_3 = 0.16$)

we do at the most 2^i computations to get C_{l+1} from C_l on line 25. Let the work done at level i be denoted as W_i. Then we can write:

$$W_i \leq (2^k(2^k - 1).2^i.2^i.f) + 2^i$$

Now, since we go through each level recursively, starting from level l_k, we would do the above computations $\log_2(n) - k$ times. Let the total work done by Algorithm 2 be denoted as W. Thus, we can write:

$$\begin{aligned} W &= \sum_{i=1}^{\log_2(n)-k} W_i \\ &\leq \sum_{i=1}^{\log_2(n)-k} 2^k(2^k-1)4^i.f + 2^i \\ &\leq \sum_{i=1}^{\log_2(n)-k} f(2^k(2^k-1)4^i + 2^i) \\ &= \frac{f}{3}((4 - 2^{(2-k)})n^2 + 3(2^{(1-k)})n - 4^{(k+1)} + 2^{(k+2)} - 6) \end{aligned}$$

Thus, the worst-case time complexity for Algorithm 2 is $\mathcal{O}(n^2 f)$.

□

5 Empirical Results

For obtaining results on real data for Algorithm 1 and Algorithm 2, we utilize the UCI datasets similar to the seminal work analyzing clustering fairness by Chierichetti et al. (Chierichetti et al., 2017). We use the `creditcard` (Yeh and Lien, 2009), `bank` (Moro et al., 2014), and `census` (Kohavi, 1996) UCI datasets and create different sub-datasets from these (corresponding to the choice of sensitive attribute). The description of the features used for clustering, the sensitive attributes, and protected groups for each of these datasets is provided in Table 1. As mentioned before, n is the number of samples in the dataset and k represents the number of clusters we will compute the HAC tree upto, and calculate the FC for. We have also open-sourced all the code used in the experiments on Github.

The fairness results (including parameter values) for running Algorithm 1 and vanilla HAC over all the aforementioned datasets and for different linkage criteria when the Fairness Cost (FC) is minimized, are shown in Table 2. The HFC achieved for vanilla HAC, the HFC achieved for Algorithm 1, as

Table 2: Results when FC is minimized

Data	Linkage	α_0	θ_1	β_0	θ_2	FC (Vanilla HAC)	FC (FAHC)
`bank1`	Single	0.926	6.533	2.224	9.815	3.026	**3.0172**
`bank1`	Complete	0.5533	0.762	74.64	2.404	3.017	**2.7593**
`bank1`	Average	0.0	5.693	144.0	13.316	3.035	**3.0256**
`bank2`	Single	0.0	3.5	19.3	0.579	**3.345**	**3.345**
`bank2`	Complete	0.207	3.634	83.896	1.763	1.632	**1.493**
`bank2`	Average	0.766	10.32	150.0	0.379	2.684	**1.69**
`census`	Single	0.357	5.847	19.3	0.965	**2.005**	**2.005**
`census`	Complete	0.7735	5.812	126.96	5.035	0.3084	**0.2658**
`census`	Average	0.0	0.0	150.0	9.332	0.505	**0.242**
`creditcard`	Single	0.357	2.34	19.3	0.386	**4.59**	**4.59**
`creditcard`	Complete	0.357	2.34	19.3	0.386	**1.56**	**1.56**
`creditcard`	Average	0.773	5.812	126.9	5.035	2.566	**2.392**

Table 3: Results when HFC is minimized

Data	Linkage	α_0	θ_1	β_0	θ_2	HFC (Vanilla HAC)	HFC (FAHC)
`bank1`	Single	0.0	0.9403	10.812	0.0	6747.7897	**6524.6078**
`bank1`	Complete	0.0	11.95	0.0	0.0	6365.5709	**6362.5625**
`bank1`	Average	0.7665	3.127	0.0	0.0	6444.5359	**6441.5275**
`bank2`	Single	1.0	14.788	16.680	0.0	9032.1463	**8922.8608**
`bank2`	Complete	0.0	7.104	0.0	0.0	8492.0879	**8490.0651**
`bank2`	Average	0.2087	0.0	12.191	0.0	8634.9503	**8550.0596**
`census`	Single	0.275	15.0	150.0	3.304	4817.7172	**4770.9221**
`census`	Complete	0.06	1.134	88.344	4.263	4500.9533	**4486.0487**
`census`	Average	0.493	7.745	45.284	0.4036	4553.1110	**4551.0302**
`creditcard`	Single	0.265	6.955	123.65	10.0	11062.426	**11030.231**
`creditcard`	Complete	0.0	0.0	150.0	10.0	11062.426	**10314.735**
`creditcard`	Average	0.5075	9.979	120.49	8.337	10532.545	**10506.927**

well as the estimated parameters for $\mathcal{Z}_\alpha$ and $\mathcal{Z}_\beta$ that minimize the HFC for the datasets are shown in Table 3. Moreover, as can be seen in Table 2 and Table 3, Algorithm 1 achieves *fairer* solutions (and in a few cases equivalently fair solutions) to vanilla HAC on real data. For the experiments on *fixing* the clustering to be fair post-HAC, we use a simplified version of Algorithm 2. We utilizing the same philosophy behind the working of the FP-HAC algorithm (Algorithm 2), but directly do all swaps at the leaf nodes instead of working our way through the output tree. This is in a way computationally more expensive but verifies that the algorithm works. Moreover, ϵ is set to ∞. We list out the results on the chosen datasets in Table 4. It can be seen that the algorithm obtains *fairer* (or equally fair) solutions by improving the existing vanilla HAC output.

Table 4: Results for *fixing* fairness post HAC (FP-HAC Algorithm)

Data	Linkage	FC (Vanilla HAC)	FC (FP-HAC)
`bank1`	Single	3.026	**2.342**
`bank1`	Complete	3.017	**2.0**
`bank1`	Average	3.035	**2.0**
`bank2`	Single	**3.345**	**3.345**
`bank2`	Complete	1.632	**1.408**
`bank2`	Average	2.684	**2.0**
`census`	Single	**2.005**	**2.005**
`census`	Complete	**0.3084**	**0.3084**
`census`	Average	0.505	**0.3571**
`creditcard`	Single	4.59	**4.329**
`creditcard`	Complete	1.56	**0.573**
`creditcard`	Average	2.566	**2.344**

6 Conclusion and Future Work

In this paper, we have proposed fair algorithms for performing HAC. To the best of our knowledge, this is the first work detailing fair HAC algorithms. Our proposed FHAC (Fair HAC) algorithm (Algorithm 1 in Section 3) works for multiple protected groups (data points are assigned to only one group at a time, though), and irrespective of the linkage criterion used. To exemplify this, we provide results using single-linkage, average-linkage, and complete-linkage in Algorithm 1 on UCI datasets such as `bank`, `creditcard`, and `census` (Section 5). We first clearly define the costs associated with ensuring fairness (Section 3), and also provide analysis that show that Algorithm 1 is more *fair* than vanilla HAC algorithms with respect to these fairness costs (FC and HFC). Moreover, our algorithm achieves an asymptotic time complexity of $\mathcal{O}(fn^3)$ that is comparable to vanilla HAC algorithms. We also propose a post-clustering fairness algorithm in Section 4 (Algorithm 2) called FP-HAC (Fair Post-HAC). This algorithm *fixes* the output received from an HAC algorithm and makes it fair. Furthermore, FP-HAC has better asymptotic time complexity $\mathcal{O}(n^2 f)$ compared to our fair HAC algorithm ($\mathcal{O}(n^3 f)$) and vanilla HAC ($\mathcal{O}(n^3)$).

For future work, we aim to study fairness in hierarchical clustering in the context of objective functions such as Dasgupta's objective (Dasgupta, 2016), and provide algorithms that approximate the objective with fairness constraints imposed.

References

Aris Anagnostopoulos, Luca Becchetti, Matteo Böhm, Adriano Fazzone, Stefano Leonardi, Cristina Menghini, and Chris Schwiegelshohn. Principal fairness:\\removing bias via projections. *arXiv preprint arXiv:1905.13651*, 2019.

Arturs Backurs, Piotr Indyk, Krzysztof Onak, Baruch Schieber, Ali Vakilian, and Tal Wagner. Scalable fair clustering. *arXiv preprint arXiv:1902.03519*, 2019.

Suman K Bera, Deeparnab Chakrabarty, and Maryam Negahbani. Fair algorithms for clustering. *arXiv preprint arXiv:1901.02393*, 2019.

Ioana O Bercea, Martin Groß, Samir Khuller, Aounon Kumar, Clemens Rösner, Daniel R Schmidt, and Melanie Schmidt. On the cost of essentially fair clusterings. *arXiv preprint arXiv:1811.10319*, 2018.

Xingyu Chen, Brandon Fain, Charles Lyu, and Kamesh Munagala. Proportionally fair clustering. *arXiv preprint arXiv:1905.03674*, 2019.

Flavio Chierichetti, Ravi Kumar, Silvio Lattanzi, and Sergei Vassilvitskii. Fair clustering through fairlets. In *Advances in Neural Information Processing Systems*, pages 5029–5037, 2017.

Vincent Cohen-Addad, Varun Kanade, Frederik Mallmann-Trenn, and Claire Mathieu. Hierarchical clustering: Objective functions and algorithms. *Journal of the ACM (JACM)*, 66(4):1–42, 2019.

Sanjoy Dasgupta. A cost function for similarity-based hierarchical clustering. In *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*, pages 118–127, 2016.

Ian Davidson and SS Ravi. Making existing clusterings fairer: Algorithms, complexity results and insights. 2020.

Michael Feldman, Sorelle A Friedler, John Moeller, Carlos Scheidegger, and Suresh Venkatasubramanian. Certifying and removing disparate impact. In *proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*, pages 259–268, 2015.

Matthäus Kleindessner, Samira Samadi, Pranjal Awasthi, and Jamie Morgenstern. Guarantees for spectral clustering with fairness constraints. *arXiv preprint arXiv:1901.08668*, 2019.

Paul Knysh and Yannis Korkolis. Blackbox: A procedure for parallel optimization of expensive black-box functions. *arXiv preprint arXiv:1605.00998*, 2016.

R Kohavi. Scaling up the accuracy of naive-bayes classifiers: A decision-tree hybrid. 1996.

Arnd Kohrs-Bernard Merialdo. Clustering for collaborative filtering applications. *Intelligent Image Processing, Data Analysis & Information Retrieval*, 3:199, 1999.

Sérgio Moro, Paulo Cortez, and Paulo Rita. A data-driven approach to predict the success of bank telemarketing. *Decision Support Systems*, 62:22–31, 2014.

Benjamin Moseley and Joshua Wang. Approximation bounds for hierarchical clustering: Average linkage, bisecting k-means, and local search. In *Advances in Neural Information Processing Systems*, pages 3094–3103, 2017.

Inti A Pagnuco, Juan I Pastore, Guillermo Abras, Marcel Brun, and Virginia L Ballarin. Analysis of genetic association using hierarchical clustering and cluster validation indices. *Genomics*, 109 (5-6):438–445, 2017.

Katherine S Pollard and Mark J Van Der Laan. Cluster analysis of genomic data. In *Bioinformatics and Computational Biology Solutions Using R and Bioconductor*, pages 209–228. Springer, 2005.

Rommel G. Regis and Christine A. Shoemaker. Constrained global optimization of expensive black box functions using radial basis functions. *Journal of Global Optimization*, 31(1):153–171, Jan 2005. ISSN 1573-2916. doi: 10.1007/s10898-004-0570-0. URL `https://doi.org/10.1007/s10898-004-0570-0`.

Clemens Rösner and Melanie Schmidt. Privacy preserving clustering with constraints. *arXiv preprint arXiv:1802.02497*, 2018.

Melanie Schmidt, Chris Schwiegelshohn, and Christian Sohler. Fair coresets and streaming algorithms for fair k-means clustering. *arXiv preprint arXiv:1812.10854*, 2018.

I-Cheng Yeh and Che-hui Lien. The comparisons of data mining techniques for the predictive accuracy of probability of default of credit card clients. *Expert Systems with Applications*, 36(2): 2473–2480, 2009.

Imtiaz Masud Ziko, Eric Granger, Jing Yuan, and Ismail Ben Ayed. Clustering with fairness constraints: A flexible and scalable approach. *arXiv preprint arXiv:1906.08207*, 2019.