Generalized Reductions: Making any Hierarchical Clustering Fair and Balanced with Low Cost

Marina Knittel [1] **Max Springer** [1] **John Dickerson** [1] **Mohammad Hajiaghayi** [1]

Abstract

Clustering is a fundamental building block of modern statistical analysis pipelines. *Fair* clustering has seen much attention from the machine learning community in recent years. We are some of the first to study fairness in the context of hierarchical clustering, after the results of Ahmadian et al. from NeurIPS in 2020. We evaluate our results using Dasgupta's cost function, perhaps one of the most prevalent theoretical metrics for hierarchical clustering evaluation. Our work vastly improves the previous $O(n^{5/6} poly \log(n))$ fair approximation for cost to a near polylogarithmic $O(n^{\delta} poly \log(n))$ fair approximation for any constant $\delta \in (0, 1)$. This result establishes a cost-fairness tradeoff and extends to broader fairness constraints than the previous work. We also show how to alter existing hierarchical clusterings to guarantee fairness and cluster balance across any level in the hierarchy.

1. Introduction

Fair machine learning, and namely clustering, has seen a recent surge as researchers recognize its practical importance. In spite of the clear and serious impact the lack of fairness in existing intelligent systems has on society (Angwin et al., 2016; Rieke & Bogen, 2018; Ledford, 2019; Sweeney, 2013), and despite significant progress towards fair flat (not hierarchical) clustering (Ahmadian et al., 2020b; Backurs et al., 2019; Bera et al., 2019a;b; Brubach et al., 2020; Chakrabarti et al., 2021; Chen et al., 2019; Chierichetti et al., 2017; Esmaeili et al., 2021; 2020; Kleindessner et al., 2019a; Rösner & Schmidt, 2018), fairness in hierarchical clustering has only received some recent attention (Ahmadian et al., 2020a; Chhabra & Mohapatra, 2020). Thus, we are some of the first to study this problem.

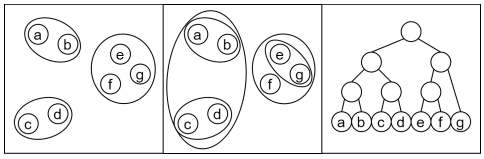

Figure 1: On the left is a 3-clustering, in the center is a hierarchical clustering, and on the right is its dendrogram.

Hierarchical clustering (Figure 1) is the well-known extension to clustering, where we create a hierarchy of subclusters contained within superclusters. This structure forms a tree (a dendrogram), where leaves represent the input data. An internal node v corresponds to the cluster of all the leaves of the subtree rooted at v. The root is the cluster of all data.

Hierarchical clusterings more completely illustrate data relationships than flat clusterings. For instance, they are commonly used in phylogenetics to depict the entire evolutionary history of species, whereas a clustering would only depict species similarities. It also has a myriad of other uses in machine learning applications such as search (Cai et al., 2004; Ferragina & Gulli, 2005; Kou & Lou, 2012), social network analysis (Leskovec et al., 2014; Mann et al., 2008), and image recognition (Arifin & Asano, 2006; Lin et al., 2018; Pan et al., 2016). On top of this, hierarchical clusterings can also be used to solve flat clustering when the number of clusters is not given. To do this, we extract clusterings at different resolutions in the hierarchy that all "agree" (if two points are together in a cluster, then they will also be together in any larger cluster) and select the one that best fits the application.

Hierarchical clusterings can be evaluated using a number of metrics. Perhaps most notably, Dasgupta (2016) introduced cost (Definition 2.2), an intuitive and explainable metric which exhibits numerous desirable properties and has become quite popular and well-respected (Charikar & Chatziafratis, 2017; Charikar et al., 2019; Chatziafratis et al., 2018; Cohen-Addad et al., 2017; Roy & Pokutta, 2016). Unfortunately, it is difficult to approximate, where the best existing solutions require semi-definite programs (Dasgupta, 2016; Charikar & Chatziafratis, 2017), and it is not efficiently

[1]Department of Computer Science, University of Maryland, College Park, USA. Correspondence to: Marina Knittel <mknittel@umd.edu>, Max Springer <mss423@umd.edu>, John Dickerson <john@cs.umd.edu>, Mohammad Hajiaghayi <hajiagha@cs.umd.edu>.

Proceedings of the 40^{th} International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023. Copyright 2023 by the author(s).

$O(1)$-approximable by the Small-Set Expansion Hypothesis (Charikar & Chatziafratis, 2017). The revenue (Moseley & Wang, 2017) and value (Cohen-Addad et al., 2018) metrics, both derived from cost, exhibit $O(1)$-approximability, but are not as explainable or appreciated.

Only two papers have explored fair hierarchical clustering (Ahmadian et al., 2020a; Chhabra & Mohapatra, 2020). Both extend fairness constraints from fair clustering literature that trace back to Chierichetti et al. (2017)'s *disparate impact*. Consider a graph $G = (V, E, w)$, where each point has a color, which represents a protected class (e.g., gender, race, etc.). On two colors, red and blue, they consider a clustering fair if the ratio between red and blue points in each cluster is equal to that in the input data. This ensures that the impact of a cluster on a protected class is proportionate to the class size. The constraint has been further generalized (Ahmadian et al., 2019; Bercea et al., 2019): given a dataset with λ colors and constraint vectors $\vec{\alpha}, \vec{\beta} \in (0, 1)^\lambda$, a clustering is fair if for all $\ell \in [\lambda]$ and every cluster C, $\alpha_\ell |C| \leq \ell(C) \leq \beta_\ell |C|$, were $\ell(C)$ is the number of points in C of color ℓ. Naturally, then, a hierarchical clustering is fair if every non-singleton cluster in the hierarchy satisfies this constraint (with nuances, see Section 2.2), as in Ahmadian et al. (2020a).

This work explores broad guarantees, namely cost approximations, for fair hierarchical clustering. The only previous algorithm is quite complicated and only yields a $O(n^{5/6} \log^{5/4} n)$ fair approximation for cost (where an $O(n)$-approximation is trivial) (Ahmadian et al., 2020a), and it assumes two, equally represented colors. This reflects the inherent difficulty of finding solutions that are low-cost as opposed to high-revenue or high-value, both of which exhibit fair $O(1)$-approximations (Ahmadian et al., 2020a)). Our algorithms improve previous work in quite a few ways: 1) we achieve a near-exponential improvement in approximation factor, 2) our algorithm works on $O(1)$ instead of only 2 colors, 3) our work handles different representational proportions across colors in the initial dataset, 4) we simultaneously guarantee fairness and relative cluster balance, and 5) our methods, which modify a given (unfair) hierarchy, have measurable, explainable, and limited impacts on the structure of the input hierarchy.

1.1. Our Contributions

This work proposes new algorithms for fair and balanced hierarchical clustering. A summary of our work can be found in Table 1.

We introduce four simple hierarchy tree operators which have clear, measurable impacts. We show how to compose them together on a (potentially unbalanced and unfair) hierarchy to yield a fair and/or balanced hierarchy with similar structure. This process clarifies the functionality of our algorithms and illustrates the modifications imposed on the hierarchy. Each of our four proposed algorithms starts with a given γ-approximate (unfair) hierarchical clustering algorithm (i.e., Dasgupta (2016)'s $O(\sqrt{\log n})$-approximation) and then builds on top of each other, imposing a new operator to achieve a more advanced result. Additionally, each algorithm stands alone as a unique contribution.

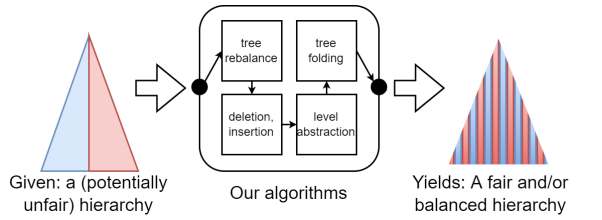

Figure 2: Our algorithms take a potentially unfair hierarchical clustering, apply our tree operators, and yield fair and/or balanced hierarchies.

Our first algorithm produces a $1/6$-relatively balanced hierarchy that $\frac{3}{2}\gamma$-approximates cost (see Theorem 4.1).[1] Here, ϵ-relative balance means that at each split in the hierarchy, a cluster splits in half within a proportional error of up to $1 + \epsilon$ (see Definition 3.1). Starting at the root, the algorithm recursively applies our *tree rebalance* (see Definition 3.4) operator. This restructures the tree by moving some subtree up to become a child of the root. It preserves much of the hierarchy's structure while achieving relative balance.

Our next result refines this to achieve ϵ-relative balance for any $\epsilon \in (0, 1/6)$ that $\frac{9}{2\epsilon}\gamma$-approximates cost (see Theorem 4.5). This can get arbitrarily close to creating a perfectly balanced hierarchy. To achieve this, we simply run our first algorithm and then apply a limited number of *subtree deletion and insertion operators* (see Definition 3.5). This operator selects a subtree, removes it, and reinserts elsewhere. It again preserves much of T's structure.

Third, we propose an algorithm for stochastically fair hierarchical clustering (see Definition 2.4). Under certain stochastic parameterizations and arbitrarily many colors, the algorithm achieves stochastic fairness and $O(\gamma \log n)$-approximates cost (see Theorem 4.9). This is quite impressive, as the best previous fair approximation (albeit, for deterministic colors) was $poly(n)$ (Ahmadian et al., 2020a). To achieve this novel result, we first find an $O(1/\log n)$-relatively balanced hierarchy and then apply our level abstraction operator once to the bottom layers of the hierarchy. This operator removes selected layers, setting much lower descendants of a vertex as direct children. While this removes details in the hierarchy, the remaining structure still agrees with the original tree. This simple addition guaran-

[1] Repeated sparsest cuts achieves this with similar cost. Our algorithms, though, can be used explainably on top of existing unfair algorithms and may perform better as unfair research progresses.

	Qualities Achieved	Approximation	Fairness	Colors	Color ratios	Explainable?
Previous	Deterministic fairness	$O(n^{5/6} poly \log n)$	Perfect	2	50/50 only	No
This work	Deterministic fairness	$O(n^{\delta} poly \log n)$	Approximate	$O(1)$	$O(1)$	Yes
	Stochastic fairness	$O(\log^{3/2} n)$	Approximate	$O(1)$	$O(1)$	Yes
	ϵ-relative balance	$O(\sqrt{\log n}/\epsilon)$	N/A	N/A	N/A	Yes
	$1/6$-relative balance	$O(\sqrt{\log n})$	N/A	N/A	N/A	Yes

Table 1: Our versus previous work. Note $\delta \in (0, 1/6)$ is parameterizable, trading approximation factor for fairness. Our algorithms are explainable in that the alterations made to the hierarchy are clear and well-defined.

tees fairness under stochastic color assignment.

Our main result finds an approximately fair hierarchical clustering that $O(n^{\delta} poly \log n)$-approximates cost (see Theorem 4.14), where $\delta \in (0, 1)$ is a given constant. This is a near-exponential improvement over previous work which only achieves a $O(n^{5/6} poly \log(n))$ approximation on two equally represented colors. On top of that, our algorithm works on many colors, with many different color ratios, and achieves a simultaneously balanced hierarchy in an explainable manner. The algorithm, FairHC (Algorithm 3), builds on top of our stochastic algorithm (parameterized slightly differently, see Section 4.4) before applying a new operator: tree folding. Tree folding maps isomorphic trees on top of each other. In hierarchical clustering, this means taking two subtrees, mapping clusters in one tree to the other, and then merging clusters according to the mapping. Matching up clusters with different proportions of colors helps balance out the color ratios across clusters, which gives us our fairness result.

2. Preliminaries

Our input is a complete weighted graph $G = (V, E, w)$ where $w : E \to \mathbb{R}^+$ is a similarity measure. A hierarchical clustering can be defined as a hierarchy tree T, where its leaves are $\text{leaves}(T) = V$, and internal nodes represent the merging of vertices into clusters and clusters into superclusters.

2.1. Optimization Problem

We use Dasgupta (2016)'s cost function as an optimization metric. For simplicity, we let $n_T(e)$ denote the size of smallest cluster in T containing both endpoints of e. In other words, for $e = (u, v)$, $n_T(e) = |\text{leaves}(T[u \wedge v])|$, where $u \wedge v$ is the lowest common ancestor of u and v in T and $T[u]$ for any vertex u is the subtree rooted at u. We additionally denote $n_T(u) = |\text{leaves}(T[u])|$ for internal node u. Also we let $\text{root}(T)$ be the root of T, and $\text{left}_T(u)$ and $\text{right}_T(u)$ access left and right children respectively. We can evaluate the cost contribution of an edge to a hierarchy.

Definition 2.1. The **cost** of $e \in E$ in a graph $G = (V, E, w)$ in a hierarchy T is $\text{cost}_T(e) = w(e) \cdot n_T(e)$.

We strive to minimize the sum of costs across all edges.

Definition 2.2 (Dasgupta (2016))**.** The **cost** of a hierarchy T on graph $G = (V, E, w)$ is:

$$\text{cost}(T) = \sum_{e \in E} \text{cost}_T(e)$$

Dasgupta (2016) showed that we can assume that all unfair trees optimizing for cost are binary. Note that we must consider non-binary trees when we incorporate fairness as it may not allow binary splits at its lowest levels.

2.2. Fairness and Stochastic Fairness

We consider the fairness constraints based off those introduced by Chierichetti et al. (2017) and extended by Bercea et al. (2019). On a graph G with colored vertices, let $\ell(C)$ count the number of ℓ-colored points in cluster C.

Definition 2.3. Consider a graph $G = (V, E, w)$ with vertices colored one of λ colors, and two vectors of parameters $\alpha, \beta \in (0, 1)^{\lambda}$ with $\alpha_\ell \le \beta_\ell$ for all $\ell \in [\lambda]$. A hierarchy T on G is **fair** if for any non-singleton cluster C in T and for every $\ell \in [\lambda]$, $\alpha_\ell |C| \le \ell(C) \le \beta_\ell |C|$. Additionally, any cluster with a leaf child has only leaf children.

Notice that the final constraint regarding leaf-children simply enforces that a hierarchy must have some "baseline" fair clustering (e.g., a fairlet decomposition (Chierichetti et al., 2017)). Consider a tree that is just a stick with individual leaf children at each depth. While internal nodes may represent fair clusters, you cannot extract any nontrivial fair flat clustering from this, since it must contain a singleton, which is unfair. We view such a hierarchy This is clearly undesirable, and our additional constraint prevents this issue.

In the stochastic problem, points are assigned colors at random. We must ensure that with high probability (i.e., at least $1 - 1/\text{polylog}(n)$) all clusters are fair.

Definition 2.4. Consider the same context as Definition 2.3 with an additional function $p_\ell : v \to (0, 1)$ denoting the probability v has color ℓ such that $\sum_{\ell=1}^{\lambda} p_\ell(v) = 1$ and each vertex has exactly one color. An algorithm is **stochastically fair** if, with high probability, it outputs a fair hierarchy.

3. Tree Properties and Operators

This work is interested in both fair and balanced hierarchies. Balanced trees have numerous practical uses, and in this paper, we show how to use them to guarantee fairness too.

Definition 3.1. A hierarchy T is **ϵ-relatively balanced** if for every pair of clusters C and C' that share a parent cluster C_p with $|C_p| \geq 1/(2\epsilon)$ in T, $(1/2-\epsilon)|C_p| \leq |C|, |C'| \leq (1/2+\epsilon)|C_p|$.

Notice that we only care about splitting clusters C_p with size satisfying $|C_p| \geq 1/(2\epsilon)$. This is because, on smaller clusters, it may be impossible to divide them with relative balance. For instance, if $|C_p| = 3$, we know we can only split it into a 1-sized and 2-sized cluster, yielding a minimum relative balance of $1/6$. For smaller ϵ, we require larger cluster sizes to make this possible.

We will often discuss the "separation" of edges in our proposed operators. It refers to occasions when a point is added to the first cluster that contains both endpoints. We do not care if points are removed.

Definition 3.2. An edge $e = (u, v)$ is (or its endpoints are) **separated** by an operator which changes hierarchy T to T' if $\text{cluster}_T(u \wedge v) \not\supseteq \text{cluster}_{T'}(u \wedge v)$.

Almost definitionally, if an edge is not separated by an operator, then the cluster size at its lowest common ancestor does not increase. Thus, its cost contribution does not increase.

3.1. Tree Operators

Our work uses a number of different tree operations to modify and combine trees (Figure 3). These illustrate exactly how our algorithms alter the input. We show how many operators of each type each of our algorithms use and to what extent they affect the hierarchy through a metric we propose here. Notably, for each proposed algorithm on an input T, it transforms T into output T' by *only applying our four tree operators: tree rebalance, subtree deletion and insertion, level abstraction, and subtree folding.*

Each operator has an associated operation cost, which measures the proportional increase in cost of each edge separated by the operation. We present lemmas that bound the operation cost of each operator in the Appendix.

Definition 3.3. Assume we apply some tree operation to transform T into T'. The **operation cost** is an upper bound Δ such that for any edge e that is separated by the operation, then $\text{cost}_{T'}(e) \leq \Delta\text{cost}_T(e)$.

The first operation is a tree rebalance, which rotates in a descendant of the root to instead be a direct child. This defines our first result in Theorem 4.1, as clever use of the tree rebalance operator allows us to find a relatively balanced tree. This is illustrated in the top left panel of Figure 3.

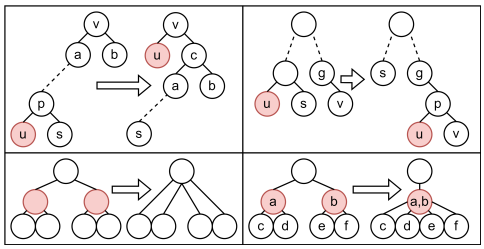

Figure 3: We depict our tree operators: tree rebalance (top left), subtree deletion and insertion (top right), level abstraction (bottom left), and tree folding (bottom right).

Definition 3.4. Consider a binary tree T with internal nodes v, v's descendant u, and v's children a and b. A **tree rebalance** of u at v ($\text{tree\_rebalance}(u, v)$) puts a new node c in between v and sibling nodes a and b. It then removes $T[u]$ from $T[a]$ and sets u to be v's other child.

Tree rebalancing will only yield $1/6$-relatively balanced trees, which is interestingly something Dasgupta (2016)'s sparsest cut algorithm, one of the current best cost approximations, achieves via a similar analysis. To refine this, we use subtree insertion and deletion (Figure 3, top right). At a root with large child a and small child b, we can move a small subtree from a to b to improve the balance.

Definition 3.5. Consider a binary tree T with internal nodes u, some non-ancestor v, u's sibling s, and v's parent g. **Subtree deletion** at u removes $T[u]$ from T and contracts s into its parent. **Subtree insertion** of $T[u]$ at v inserts a new parent p between v and g and adds u as a second child of p. The operator $\text{del\_ins}(u, v)$ deletes u and inserts $T[u]$ at v.

We will also need to abstract away (Figure 3, bottom left) certain levels of the hierarchy to simplify it. This involves taking vertices at depth d_2 and iteratively merging them into their parents until they reach depth d_1. In other words, ignore tree structure between two levels of the hierarchy.

Definition 3.6. Consider a binary tree T with two parameters d_1 and d_2 such that $d_1 < d_2 < \text{height}(T)$. **Level abstraction** between levels $d_1 + 1$ and d_2 ($\text{abstract}(d_1, d_2)$) involves taking all internal nodes between depths $d_1 + 1$ and d_2 in T and contracting them into their parents.

To achieve fairness in Section 4, we use tree folding (Figure 3, bottom right). Given multiple isomorphic trees (ignoring leaves), we map the topologies of the trees together.

Definition 3.7. Consider a set of subtrees $T_1, \ldots, T_k$ of T such that all trees T_i without their leaves have the same topology, and all $\text{root}(T_i)$ have the same parent p in T. This means that for each $i \in [k]$, there is a tree isomorphism $\phi_i : I_i \to I_k$ where I_i and I_k are the internal nodes of the corresponding trees. A **tree folding** of trees $T_1, \ldots, T_k$ ($\text{fold}(T_1, \ldots, T_k)$) modifies T such that all $T_1, \ldots, T_k$ are replaced by a single tree T_f whose $\text{root}(T)$ is made a child

of p and T_f has the same internal topology as I_k such that for any leaf ℓ of any tree T_i with parent p_i in T_i, we set its parent to $\phi_i(p_i)$.

4. Fair and Balanced Reductions

We now present our main algorithms, which sequentially build on top of each other, adding new operators in a limited, measureable capacity to achieve new results.

4.1. Relatively Rebalanced Trees

Our first algorithm guarantees $1/6$-relative balance. It only modifies the tree through a series of limited tree rebalances (Definition 3.4). We can show that this only incurs a small constant factor proportionate increase in cost.

Theorem 4.1. *Given a γ-approximation for cost, we can construct a $\frac{3}{2}\gamma$-approximation for cost which guarantees $\frac{1}{6}$-relative balance. It only modifies the tree by applying tree rebalance operators of operation cost $3/2$, and every edge is only separated by at most one such operator.*

Algorithm 1 RebalanceTree

Input: A hierarchy tree T of size n, with smaller cluster always on the left.
Output: A $\frac{1}{6}$-rebalanced T-tree.

$r, v = \text{root}(T)$
$A = \text{leaves}(\text{left}_T(v))$
while $|A| \geq \frac{2}{3}n$ **do**
  $v \leftarrow \text{left}_T(v)$
  $A \leftarrow \text{leaves}(\text{left}_T(v))$
end while
$T \leftarrow T.\text{tree\_rebalance}(v, r)$
Let $T'_l = \text{RebalanceTree}(T[\text{left}_T(r)])$
Let $T'_r = \text{RebalanceTree}(T[\text{right}_T(r)])$
Return T' with root r with $\text{left}(r) = \text{root}(T'_l)$ and $\text{right}(r) = \text{root}(T'_r)$

This idea was first introduced by Dasgupta (2016) as an analytical tool for their algorithm. However we use it more explicitly here to take any given hierarchy and rearrange it to be balanced. The basic idea is to start with some given tree T. Draw T from top to bottom such that the smaller cluster in a split is put on the left. Let A_1 and B_1 be our first split. Continue working on the left side, splitting A_1 into A_2 and B_2 and so on. Stop when we find the first cluster B_k such that $|B_k| \geq n/3$. This defines our first split: partition V into A_k and $B = \cup_{i=1}^{k} B_i$. Then recurse on each side.

It is not too hard to see that this yields a $\frac{1}{6}$-relatively balanced tree. Our search stopping conditions enforce this.

Lemma 4.2. *Algorithm 1 produces a $\frac{1}{6}$-relatively balanced tree.*

The next property also comes from the fact that once an edge is separated, it will never be separated again.

Lemma 4.3. *In Algorithm 1, every edge is separated by at most one tree rebalance operator.*

Finally, the operation cost of the rebalance operators comes from our stopping threshold.

Lemma 4.4. *In Algorithm 1, every tree rebalance operator has operation cost at most $3/2$.*

In Theorem 4.1, the relative balance comes from Lemma 4.2, the operator properties come from Lemmas 4.3 and 4.4 respectively, and the approximation factor comes from Lemma 4.3 and Lemma 4.4 together.

4.2. Refining Relatively Rebalanced Trees

We now propose a significant extension of Algorithm 1 which allows us to get a stronger balance guarantee. Specifically (where ϵ may be a function of n):

Theorem 4.5. *Given a γ-approximation for cost, we can construct a $\frac{9\gamma}{2\epsilon}$-approximation for cost which guarantees ϵ-relative balance for $0 < \epsilon \leq 1/6$. In addtition to Theorem 4.1, it only modifies the tree by applying subtree deletion and insertion operators of operation cost $\frac{3}{\epsilon}$, and every edge is only separated by at most one such operator.*

To do this, we first apply RebalanceTree. Then, at each split starting at the root, we execute SubtreeSearch (Algorithm 4 in Appendix C.2), which searches for a small subtree below the right child, deletes it, and moves it below the left child in order to reduce the error in the relative balance. If we do this enough, we can reduce the relative balance to ϵ. We call our algorithm, in Algorithm 2, RefineRebalanceTree.

Algorithm 2 RefineRebalanceTree

Input: A $\frac{1}{6}$-relatively balanced hierarchy tree T of size n, with smaller cluster always on the left, and balance parameter $\epsilon \in (0, 1/6)$.
Output: An ϵ-relatively balanced tree.

if $\epsilon \leq 1/(2n)$ **then**
  Return T
end if
$v = \text{root}(T)$
Let $T_{big} = T[\text{left}_T(v)]$, $T_{small} = T[\text{right}_T(v)]$
while $|\text{leaves}(T_{big})| \geq (1/2 + \epsilon)n$ **do**
  $\delta \leftarrow (|\text{leaves}(T_{big})| - n/2)/n$
  Let $T_{big} = \text{SubtreeSearch}(T_{big}, \delta n)$
end while
$T_{big} \leftarrow \text{RefineRebalanceTree}(T_{big}, \epsilon)$
$T_{small} \leftarrow \text{RefineRebalanceTree}(T_{small}, \epsilon)$
Return T' with root r with $\text{left}(r) = \text{root}(T_{big})$ and $\text{right}(r) = \text{root}(T_{small})$

We can show that this algorithm creates a nicely rebalanced tree. SubtreeSearch specifically guarantees a proportional $2/3$ reduction in relative balance (see Appendix C.2). Therefore, enough executions of SubtreeSearch will make the split ϵ-relatively balanced, and recursing down the tree guarantees that the entire tree is ϵ-relatively balanced.

Lemma 4.6. *Algorithm 2 produces an ϵ-rebalanced tree.*

To bound the operators on top of those used by Algorithm 1, note that we only apply the subtree deletion and insertion operators. Additionally, each edge cannot be separated more than once.

Lemma 4.7. *In Algorithm 2, every edge is separated by at most one subtree deletion and insertion operator.*

Finally, we can also limit the operation cost of the subtree deletion and insertion operator. This is because we limit the depth of the SubtreeSearch function as it will never be given a parameter below ϵn.

Lemma 4.8. *In Algorithm 2, every subtree deletion and insertion operator has operation cost at most $3/\epsilon$.*

For Theorem 4.5, the relative balance comes from Lemma 4.6, the operator properties com from Lemmas 4.7 and 4.8 respectively, and the approximation factor comes from Lemma B.2, Lemma 4.7, and Lemma 4.8 together.

4.3. Stochastically Fair Hierarchical Clustering

At this point, we almost have enough tools to solve stochastically fair hierarchical clustering. For this, however, we need a simple application of level abstraction (Definition 3.6). We introduce StochasticallyFairHC, which simply imposes one level abstraction: $T' = T.\text{abstract}(t, h_{max})$ on the bottom levels of the hierarchy. Here, t is a parameter and h_{max} is the max depth in T. Notice that we require the input to be relatively balanced to achieve this result.

Theorem 4.9. *Given a γ-approximation for* cost *and any $\epsilon = 1/(c\log_2 n)$, $c, \lambda = O(1)$, and $\delta \in (0,1)$, in the stochastic fairness setting with $\frac{1}{1-\delta}\alpha_\ell \le p_\ell(v) \le \frac{1}{1+\delta}\beta_\ell$ for all $v \in V$ and $\ell \in [\lambda]$, there is a $e^{4/(c(1-o(1)))} \cdot \frac{3(1-\delta)\ln(cn)}{\delta^2 \min_{\ell\in[\lambda]}\alpha_\ell} \cdot \frac{9\gamma}{2\epsilon}$ fair approximation that succeeds with high probability. On top of the operators of Theorem 4.5, it only modifies the tree by applying one level abstraction of operation cost at most $e^{4/(c(1-o(1)))} \cdot \frac{3\ln(cn)}{a\delta^2}$.*

Theorem 4.9 with constant α_ℓ for all $\ell \in [\lambda]$, c, and δ yields a $O(\gamma \log n)$ approximation. Since $\gamma = O(\sqrt{\log n})$ (Charikar & Chatziafratis, 2017; Dasgupta, 2016), this becomes $O(\log^{3/2} n)$. It is quite impressive, as the best previous fair (albeit, deterministic) approximation was $poly(n)$ (Ahmadian et al., 2020a). Also, δ exhibits an important tradeoff: increasing δ increases success probability but also decreases the range of acceptable $p_\ell(v)$ values.

It might be tempting to suggest applying StochasticallyFairHC to any existing hierarchy, as opposed to one that is ϵ-relatively balanced. However, if we consider, for instance, a highly unbalanced tree where all internal nodes have at least one leaf-child, the algorithm would only merge the bottom t internal nodes into a single cluster, thereby not guaranteeing fairness. The resulting structure would also not be particularly interesting. This is why the rebalancing process is important.

Obviously, since we only apply level abstraction once, edge separation only happens once per edge in the algorithm. To bound the operation cost, we explore the relative size of clusters at a specified depth in the hierarchy. The following guarantee is achieved by considering a root-to-vertex path where we always travel to maximally/minimally sized clusters according to the tree's relative balance.

Lemma 4.10. *Let T be an ϵ-relatively balanced tree and u and v be internal nodes at depth i in T. Then $(1/2-\epsilon)^i n \le n_T(u), n_T(v) \le (1/2+\epsilon)^i n$, which also implies $\frac{n_T(u)}{n_T(v)} \le \frac{(1+2\epsilon)^i}{(1-2\epsilon)^i}$. Additionally, if $i \le \log_{1/2-\epsilon}(x/n)$ for some arbitrary $x \ge 1$ and $\epsilon = 1/(c\log_2 n)$ for a constant c, then the maximum cluster size is at most $e^{4/(c(1-o(1)))}x$.*

This yields our operation cost, since it bounds the size of clusters at certain depths.

Lemma 4.11. *In* StochasticallyFairHC*, the level abstraction has operation cost at most $(1/2+\epsilon)^t n$.*

To get our fairness results, we need to use a Chernoff bound, thus we must guarantee that all internal nodes have sufficiently large size. This too comes from our bounds on cluster sizes.

Lemma 4.12. *In* StochasticallyFairHC*, for any internal node v, $n_{T'}(v) \ge (1/2-\epsilon)^t n$.*

Finally, we must show the fairness guarantee. Since the union of two fair clusters is fair, we only need to show this for the clusters at height 1 in the hierarchy, as this would imply fairness for the rest of the hierarchy. This comes from a Chernoff bound.

Lemma 4.13. *The resulting tree from* StochasticallyFairHC *with $t = \log_{1/2-\epsilon}\left(\frac{3(1-\delta)\ln(cn)}{a\delta^2 n}\right)$ for $a = \min_{\ell\in[\lambda]}\alpha_\ell$ and any $\delta > 0$ is stochastically fair for given parameters α_ℓ, β_ℓ for all colors $\ell \in [\lambda]$ with high probability if with $\frac{1}{1-\delta}\alpha_\ell \le p_\ell(v) \le \frac{1}{1+\delta}\beta_\ell$ for all $v \in V$ and $\ell \in [\lambda]$ for $\lambda = O(1)$.*

This is sufficient to show Theorem 4.9. The fairness is a result of Lemma 4.13, the operator properties are a result of Lemma 4.11 and the obvious fact that we only apply one level abstraction, and the approximation factor comes from Lemma B.3 and Lemma 4.11 together.

4.4. Deterministically Fair Hierarchical Clustering

Finally, we have our main results on the standard, deterministic fair hierarchical clustering problem. This algorithm builds on top of the results from Theorem 4.5 and uses methods similar to Theorem 4.9. In addition to previous algorithms, it uses more applications of level abstraction and introduces tree folding.

Theorem 4.14. *Given a γ-approximation for* cost *over $\ell(V) = c_\ell n = O(n)$ vertices of each color $\ell \in [\lambda]$ with $h = n^\delta$ for any constants c, δ, k, there is an algorithm that yields a hierarchy T' that:*

1. *Is a $e^{\frac{4\log_2 n}{c(1-o(1))\lambda \log_2 h} + \frac{2}{c} + \frac{4}{c(1-o(1))}} \cdot \frac{9c^2\gamma}{4} \cdot n^\delta \log_2^2 n$-approximation for cost.*
2. *Is fair for any parameters for all $\ell \in [\lambda]$: $\beta_\ell \geq c_\ell \left(\frac{e^{4/c}}{kc_\ell} + e^{6/c}\right)^{1/\delta}$ and $\alpha_\ell \leq \frac{c_\ell}{e^{(6/c)\log_h(n)}}$.*

On top of the operators of Theorem 4.9, it only modifies the tree by applying level abstraction of operation cost at most $e^{2/c}n^\delta$ and tree folding of operation cost $ke^{4/(c(1-o(1))}$ on k subtrees, and each edge is separated in at most one level abstraction operator and in at most λ/δ tree fold operators.

This algorithm runs in $O(n^2 \log n)$ time.

Since $\gamma = O(\sqrt{\log n})$, this becomes $O(n^\delta \log^{5/2} n)$ for any constant c, $\delta \in (0, 1)$, and k which greatly improves the previous $O(n^{5/6}\log^{5/4}(n))$-approximation (Ahmadian et al., 2020a). Additionally, the previous work only considered 2 colors with equal representation in the dataset. Our algorithm greatly generalizes this to both more colors and different proportions of representation. While we do not guarantee exact color ratio preservation as the previous work does, our algorithms can get arbitrarily close through parameterization and we no longer require the ratio between colors points in the input to be exactly 1.

In terms of fairness, all of the variables here are parameterizeable constants. Increasing k, c, and δ will all make these values get closer to the true proportions of the colors in the overall dataset, and this can be done to an arbitrary extent. Therefore, based off the parameterization, this allows us to enforce clusters to have pretty close to the same color proportions as the underlying dataset.

The goal of this algorithm is to recursively abstract away the top $\log_2 h$ depth of the tree, where we end up setting $h = n^\delta$. Each time we do this, we get a kind of "frontier clustering", which is an h-sized clustering whose parents in the tree are all the root after level abstraction. Since the subtrees rooted at each cluster have the same topology (besides their leaves, this is due to our level abstraction at the lowest levels in the tree), we can then execute tree folding on any subset of them. We select cluster subtrees to fold together such that, once we merge the appropriate clusters, the clustering at this level will be more fair. Then, as we recurse down the tree, we subsequently either eliminate clusters (via level abstraction) or fold them to guarantee fairness. For more information, see Algorithm 3.

Algorithm 3 FairHC

Input: An $\epsilon = 1/(c\log_2 n)$ relatively balanced hierarchy tree T of size n on red and blue points, and parameters $h = 2^i$ and $k = 2^j$ for some $0 < j < i < \log_{1/2-\epsilon}(1/(2n\epsilon))$
Output: A fair tree.

Let $T \leftarrow T.\text{abstract}(0, i)$
if T is height 1 **then**
  Return T
end if
Let $\mathcal{V}$ be the children of $\text{root}(T)$
for each color $\ell \in [\lambda]$ **do**
  Order $\mathcal{V} = \{v_i\}_{i\in[h]}$ decreasing by $\frac{\ell(\text{leaves}(v_i))}{|\text{leaves}(v_i)|}$
  For all $i \in [k]$, $T \leftarrow T.\text{fold}(\{T'[v_{i+(j-1)k}] : j \in [h/k]\})$
end for
for each child v of $\text{root}(T)$ **do**
  Replace $T[v] \leftarrow \text{FairHC}(T[v])$
end for
Return T'

To see why this creates a fair, low-cost hierarchy, we first bound the metrics on the operators used. When we execute level abstraction, we can leverage relative balance and Lemma 4.10 to show that during FairHC, we can bound the abstraction operation cost.

Lemma 4.15. *In Algorithm 3, the level abstraction has operation cost at most $e^{2/c}h$.*

Our tree folding operation cost bound also comes from the balance of a tree, since any two vertices that are folded together must be at similar depths.

Lemma 4.16. *In Algorithm 3, each tree folding has operation cost at most $ke^{4/(c(1-o(1))}$ and acts on k trees.*

In order to bound the cost, we need to first know how many times an edge will be separated. We notice that an edge that is separated by level abstraction can no longer be separated on a subsequent recursive step. Additionally, the number of tree fold operators is proportionally bounded by the recursive depth, as it only happens λ times each step.

Lemma 4.17. *In Algorithm 3, an edge e is separated by at most 1 level abstraction and $\lambda \log_2(n)/\log_2(h)$ tree folds. The maximum recursion depth is also at most $\log_2(n)/\log_2(h)$.*

Fairness comes from the ordering over ℓ-colored vertices

and the way select subtrees to fold together. One recursive step of FairHC incurs a small constant factor proportionate loss in potential fairness, and the number of times this loss occurs is bounded by the depth of recursion. We desire these fractions to be close to the true color proportions, which we can get arbitrarily close to by setting parameters c, k, and h.

Lemma 4.18. *For an ϵ-relatively balanced hierarchy T over $\ell(V) = c_\ell n = O(n)$ vertices of each color $\ell \in [\lambda]$, Algorithm 3 yields a hierarchical clustering T' such that the amount of each color $\ell \in [\lambda]$ in each cluster (represented by vertex v) is bounded as follows:*

$$\frac{c_\ell}{e^{2\log_h n/c}} \leq \frac{\ell(v)}{n_T(v)} \leq c_\ell \cdot (e^{4/c}/(kc_\ell) + e^{6/c})^{\log_h n}.$$

In Theorem 4.14, fairness is a result of Lemma 4.18, the operator properties are a result of Lemmas 4.15, 4.16, and 4.17, and the approximation factor has already been worked out by Lemma C.5.

5. Experiments

This section validates our algorithms from Section 4. Our simulations demonstrate that our algorithm incurs only a modest loss in the hierarchical clustering objective and exhibits increased fairness. Specifically, the approximate cost increases as a function of Algorithm 3's defining parameters: c, δ, and k.

Datasets. We use two data sets, *Census* and *Bank*, from the UCI data repository (Dua & Graff, 2017). Within each, we subsample only the features with numerical values. To compute the *cost* of a hierarchical clustering we set the similarity to be $w(i,j) = \frac{1}{1+d(i,j)}$ where $d(i,j)$ is the Euclidean distance between points i and j. We color data based on binary (represented as blue and red) protected features: *race* for *Census* and *marital status* for *Bank* (both in line with the prior work of Ahmadian et al. (2020a)). As a result, *Census* has a blue to red ratio of 1:7 while *Bank* has 1:3.

We then subsample each color in each data set such that we retain (approximately) the data's original balance. We use samples of size 256. For each experiment, we do 10 replications and report the average results. We vary the parameters $c \in \{2^i\}_{i=0}^5, \delta \in (\frac{1}{8}, \frac{7}{8})$, and $k \in \{2^i\}_{i=1}^4$ to experimentally validate their theoretical impact on the approximate guarantees of Section 4.

Implementation. The Python code for the following experiments are available in the Supplementary Material. We start by running average-linkage, a popular hierarchical clustering algorithm. We then apply Algorithms 1 - 3 to modify this structure and induce a *fair* hierarchical clustering that exhibits a mild increase in the cost objective.

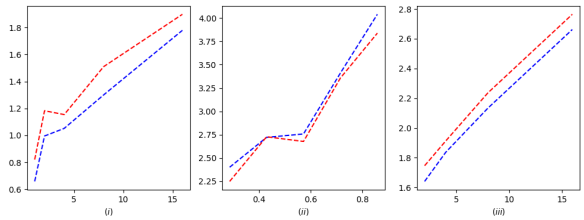

Figure 4: Cost ratio of Algorithm 3 as compared to average-linkage. (i) Ratio increase as a function of the parameter c, (ii) ratio increase as a function of the parameter δ, and (iii) ratio increase as a function of k. Blue lines indicate the result for *Census* dataset whereas red indicates the *Bank* dataset results.

Metrics. In our results we track the approximate cost objective increase as follows: Let G be our given graph, T be average-linkage's output, and T' be Algorithm 3's output. We then measure the ratio $\textsc{Ratio}_{cost} = \frac{cost_G(T')}{cost_G(T)}$.

Results. We first note that the average-linkage algorithm must construct unfair trees since, for each data set, the algorithm induces some monochromatic clusters. Thus, our resultant fair clustering is of considerable value in practice.

In Figure 1, we plot the change in cost ratio as the parameters (c, δ, k) are varied for the two datasets. Supporting our theoretical results, increasing our fairness parameters leads to a modest increase in cost. This is an empirical illustration of our fairness-cost approximation tradeoff according to our parameterization. Note that the results are consistent across tested datasets.

We additionally illustrate the resulting balance of our hierarchical clustering algorithm by presenting the distribution of the cluster ratios of the projected features (blue to red points) in Figure 5 for the *Census* data. The output of average-linkage naturally yields an unfair clustering of the data, yet after applying our algorithm on top this hierarchy we see that the cluster's balance move to concentrate about the underlying data balance of 1:7. An equivalent figure for the *Bank* dataset is provided in the appendix due to space constraints.

References

Ahmadian, S., Epasto, A., Kumar, R., and Mahdian, M. Clustering without over-representation. In *KDD*, pp. 267–275, 2019.

Ahmadian, S., Epasto, A., Knittel, M., Kumar, R., Mahdian, M., Moseley, B., Pham, P., Vassilvitskii, S., and Wang, Y. Fair hierarchical clustering. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H. (eds.), *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing*

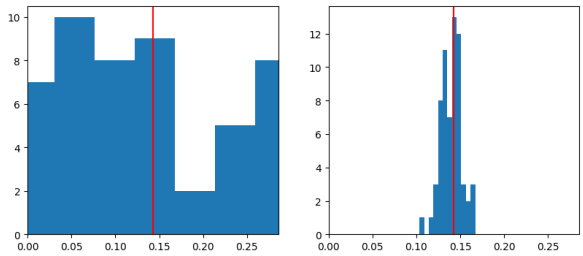

Figure 5: Histogram of cluster balances after tree manipulation by Algorithm 3. The left plot depicts the balances after applying the average- linkage algorithm and the right shows the result of applying our algorithm. The vertical red line indicates the balance of the dataset. Parameters were set to $c = 4, \delta = \frac{3}{8}, k = 4$ for the above clustering result.

Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual, 2020a.

Ahmadian, S., Epasto, A., Kumar, R., and Mahdian, M. Fair correlation clustering. In *AISTATS*, 2020b.

Angwin, J., Larson, J., and Kirchner, L. Machine bias. *ProPublica*, 2016.

Arifin, A. Z. and Asano, A. Image segmentation by histogram thresholding using hierarchical cluster analysis. *Pattern Recognit. Lett.*, 27(13):1515–1521, 2006.

Backurs, A., Indyk, P., Onak, K., Schieber, B., Vakilian, A., and Wagner, T. Scalable fair clustering. In *ICML*, pp. 405–413, 2019.

Barocas, S., Hardt, M., and Narayanan, A. *Fairness and Machine Learning*. www.fairmlbook.org, 2019.

Ben-Porat, O., Sandomirskiy, F., and Tennenholtz, M. Protecting the protected group: Circumventing harmful fairness. In *Thirty-Fifth AAAI Conference on Artificial Intelligence*, pp. 5176–5184. AAAI Press, 2021.

Bera, S., Chakrabarty, D., Flores, N., and Negahbani, M. Fair algorithms for clustering. In *NeurIPS*, pp. 4955–4966, 2019a.

Bera, S. K., Chakrabarty, D., Flores, N., and Negahbani, M. Fair algorithms for clustering. In *NeurIPS*, pp. 4955–4966, 2019b.

Bercea, I. O., Groß, M., Khuller, S., Kumar, A., Rösner, C., Schmidt, D. R., and Schmidt, M. On the cost of essentially fair clusterings. In *APPROX-RANDOM*, pp. 18:1–18:22, 2019.

Brubach, B., Chakrabarti, D., Dickerson, J. P., Khuller, S., Srinivasan, A., and Tsepenekas, L. A pairwise fair and community-preserving approach to k-center clustering. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 1178–1189. PMLR, 2020.

Cai, D., He, X., Li, Z., Ma, W., and Wen, J. Hierarchical clustering of WWW image search results using visual, textual and link information. In *Proceedings of the 12th ACM International Conference on Multimedia*, pp. 952–959. ACM, 2004.

Chakrabarti, D., Dickerson, J. P., Esmaeili, S. A., Srinivasan, A., and Tsepenekas, L. A new notion of individually fair clustering: α-equitable k-center. *CoRR*, abs/2106.05423, 2021.

Charikar, M. and Chatziafratis, V. Approximate hierarchical clustering via sparsest cut and spreading metrics. In *SODA*, pp. 841–854, 2017.

Charikar, M., Chatziafratis, V., Niazadeh, R., and Yaroslavtsev, G. Hierarchical clustering for euclidean data. In *AISTATS*, pp. 2721–2730, 2019.

Chatziafratis, V., Niazadeh, R., and Charikar, M. Hierarchical clustering with structural constraints. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 773–782. PMLR, 2018.

Chen, X., Fain, B., Lyu, L., and Munagala, K. Proportionally fair clustering. In *ICML*, pp. 1032–1041, 2019.

Chhabra, A. and Mohapatra, P. Fair algorithms for hierarchical agglomerative clustering. *arXiv:2005.03197*, 2020.

Chierichetti, F., Kumar, R., Lattanzi, S., and Vassilvitskii, S. Fair clustering through fairlets. In *NIPS*, pp. 5029–5037, 2017.

Cohen-Addad, V., Kanade, V., and Mallmann-Trenn, F. Hierarchical clustering beyond the worst-case. In *NIPS*, pp. 6201–6209, 2017.

Cohen-Addad, V., Kanade, V., Mallmann-Trenn, F., and Mathieu, C. Hierarchical clustering: Objective functions and algorithms. In *SODA*, pp. 378–397, 2018.

Dasgupta, S. A cost function for similarity-based hierarchical clustering. In *STOC*, pp. 118–127, 2016.

Dua, D. and Graff, C. UCI machine learning repository, 2017. URL http://archive.ics.uci.edu/ml.

Esmaeili, S. A., Brubach, B., Tsepenekas, L., and Dickerson, J. Probabilistic fair clustering. In *Advances in Neural Information Processing Systems*, 2020.

Esmaeili, S. A., Brubach, B., Srinivasan, A., and Dickerson, J. Fair clustering under a bounded cost. In *Advances in Neural Information Processing Systems*, pp. 14345–14357, 2021.

Ferragina, P. and Gulli, A. A personalized search engine based on web-snippet hierarchical clustering. In *Proceedings of the 14th international conference on World Wide Web*, pp. 801–810. ACM, 2005.

Kleindessner, M., Awasthi, P., and Morgenstern, J. Fair k-center clustering for data summarization. In *ICML*, pp. 3448–3457, 2019a.

Kleindessner, M., Samadi, S., Awasthi, P., and Morgenstern, J. Guarantees for spectral clustering with fairness constraints. In *ICML*, pp. 3458–3467, 2019b.

Kou, G. and Lou, C. Multiple factor hierarchical clustering algorithm for large scale web page and search engine clickstream data. *Ann. Oper. Res.*, 197(1):123–134, 2012.

Ledford, H. Millions of black people affected by racial bias in health-care algorithms. *Nature*, 574, 2019.

Leskovec, J., Rajaraman, A., and Ullman, J. D. *Mining of Massive Datasets, 2nd Ed*. Cambridge University Press, 2014.

Lin, W., Chen, J., Ranjan, R., Bansal, A., Sankaranarayanan, S., Castillo, C. D., and Chellappa, R. Proximity-aware hierarchical clustering of unconstrained faces. *Image Vis. Comput.*, 77:33–44, 2018.

Mann, C. F., Matula, D. W., and Olinick, E. V. The use of sparsest cuts to reveal the hierarchical community structure of social networks. *Soc. Networks*, 30(3):223–234, 2008.

Moseley, B. and Wang, J. Approximation bounds for hierarchical clustering: Average linkage, bisecting k-means, and local search. In *NIPS*, pp. 3094–3103, 2017.

Pan, T., Lo, L., Yeh, C., Li, J., Liu, H., and Hu, M. Real-time sign language recognition in complex background scene based on a hierarchical clustering classification method. In *IEEE Second International Conference on Multimedia Big Data*, pp. 64–67. IEEE Computer Society, 2016.

Rieke, A. and Bogen, M. Help wanted: An examination of hiring algorithms, equity, and bias. *Upturn*, 2018.

Rösner, C. and Schmidt, M. Privacy preserving clustering with constraints. In *ICALP*, pp. 96:1–96:14, 2018.

Roy, A. and Pokutta, S. Hierarchical clustering via spreading metrics. In *NIPS*, pp. 2316–2324, 2016.

Sweeney, L. Discrimination in online ad delivery. *ACM Queue*, 11(3):10, 2013.

A. Limitations

The main limitations this work suffers from encapsulate the general limitations of study in theoretical clustering fairness. Our work strives to provide algorithms that are applicable to many hierarchical clustering applications where fairness is a concern. However, our work is inherently limited by its focus on a specific fairness constraint (i.e., the extension of disparate impact originally used to study fair clustering (Chierichetti et al., 2017)). While disparate impact has received substantive attention in the clustering community and is seen as one of the primary fairness definitions/constraints (see, e.g., Ahmadian et al., 2020a;b; Bera et al., 2019a; Brubach et al., 2020; Kleindessner et al., 2019b), it is just one of many established fairness constraints for problems in clustering (Chakrabarti et al., 2021; Chen et al., 2019; Esmaeili et al., 2021; Kleindessner et al., 2019a). When applying fair machine learning algorithms to problems, it is not always clear which fairness constraints are the best for the application. This, and the fact that the application of fairness to a problem can cause harm in other ways (Ben-Porat et al., 2021), means that the proposal of theoretical fair machine learning algorithms always has the potential for improper or even harmful use. While this work proposes purely theoretical advances to the field, we direct the reader to (Barocas et al., 2019) for a broader view on the field.

Our results are also limited by the theoretical assumptions that we make. For instance, in the stochastic fairness algorithm, we assume that the probabilities of a vertex being a certain color are within the same bounds across all vertices. This may not be realistic, as there could be higher variance in the distribution of color probabilities, and even though the probabilities may lie outside of our assumed bounds, it still may be tractable to find a low-cost hierarchical clustering.

In our main theorem, we assume that there are only two colors (protected classes), and that they subsume a constant fraction of the general population. The former assumption is clearly limited in that in many cases, protected classes may take on more than two values. The constant fraction assumption is actually highly relevant and is reflected in other clustering literature, but it is a potential limitation that may rule out a handful of applications nevertheless.

Finally, our results are limited to the evaluation of hierarchical clustering quality based off cost. While this is a highly regarded metric for hierarchy evaluation, there may be situations where others are appropriate. It also neglects the practicality of empirical study in that many important machine learning algorithms we use today cannot provide guarantees across all data (which our results necessarily do), but they perform much better on most actual inputs. However, we leave it as an open question to further evaluate the practicality of our algorithms through empirical study.

B. Proofs: Tree Properties and Operators

Here we present all our proofs and theoretical results regarding our tree operator properties.

We start by discussing our tree rebalance operator. Effectively, any edge whose end points are separated by a tree rebalance operator was contained in a cluster of size $n_T(u)$, and now we guarantee they are in a cluster of size $n_T(v)$.

Lemma B.1. *Given a tree T, let $T' = \mathrm{tree\_rebalance}(u, v)$ for a node u and an ancestor node v. The only edges separated by this are $e = (x, y)$ such that $x \in \mathrm{cluster}(u)$ and $y \in \mathrm{cluster}(a) \setminus \mathrm{cluster}(u)$. The operation cost is bounded above by $\Delta = n_T(v)/n_T(p)$, where p is the parent of u.*

Proof of Lemma B.1. Let $e = (x, y)$ be an edge that is separated by a tree rebalance operator $\mathrm{tree\_rebalance}(u, v)$ for some internal nodes u and v. Let's consider when we execute the rebalance. Let $V = \mathrm{cluster}(v)$ be the set of vertices corresponding to V. Traverse down the tree from v to u. Label the clusters we come across $A_1, A_2, \ldots, A_{k-1}$ and their corresponding un-traversed children $B_1, B_2, \ldots, B_{k-1}$. Let $A_k = \mathrm{cluster}(u)$, and B_k be the cluster for its only sibling.

When we rebalance, our first split will now divide V into A_k and $B := \cup_{i \in [k]} B_i$. For e to be separated by the rebalance of u with respect to v, it must be that $x \in \mathrm{cluster}(u)$ and $y \in B = \mathrm{cluster}(a) \setminus \mathrm{cluster}(u)$ (without loss of generality). This means that their lowest common ancestor was on the path between u and v (excluding u), which means the smallest $n_T(e)$ could be is $n_T(p)$ where p is the parent of u. That means $\mathrm{cost}_T(e) = w(e) \cdot n_T(p)$.

In T', their lowest common ancestor is v, thus $n_{T'}(e) = n_{T'}(v) = n_T(v)$, following from the observation that v's cluster does not change. Thus, $\mathrm{cost}_{T'}(e) \leq w(e) \cdot n_T(v)$. Putting these together, we find $\mathrm{cost}_{T'}(e) \leq \frac{n_T(v)}{n_T(p)} \mathrm{cost}_T(e)$. □

For our subtree deletion and insertion, the idea is that an edge that is separated costs at least $n_T(v)$ in the original tree, but may cost up to $n_T(u \wedge v)$ in the modified tree.

Lemma B.2. *Given a tree T, let $T' = \mathrm{del\_ins}(u, v)$ for two nodes u and v, where u is not an ancestor of v. The only edges separated by this are $e = (x, y)$ such that $x \in \mathrm{cluster}(u)$ and $y \in \mathrm{cluster}(u \wedge v) \setminus \mathrm{cluster}(u)$. The operation cost is bounded above by $\Delta = n_T(u \wedge v)/n_T(u)$.*

Proof of Lemma B.2. Let $e = (x, y)$ be an edge that is separated by a subtree deletion and insertion operators $\mathrm{del\_ins}(u, v)$ for some appropriate internal nodes u and v. Let's consider when we execute the subtree deletion and insertion. For x and y to be separated, x must be in $\mathrm{cluster}(u)$ and y must be in $\mathrm{cluster}(u \wedge v) \setminus \mathrm{cluster}(u)$ (without loss

of generality). The first part is true because only the subtree $T[u]$ is moved, otherwise their least common ancestor would be unaffected. The second part is true because otherwise y is either in $T[u]$ too, in which case their relative position remains the same in the subtree, or $y \notin T[u \wedge v]$, in which case still the move still does not affect their least common ancestor (which is higher in the tree than $u \wedge v$).

Now, since $x \in T[u]$ and $y \notin T[u]$, $x \wedge y$ must be an ancestor of u, thus $n_T(e) \geq n_T(u)$. This means that $\text{cost}_T(e) \geq w(e) \cdot n_T(u)$. In T', their least common ancestor must still remain below $u \wedge v$, since all the points in $T[u \wedge v]$ remain somewhere below $u \wedge v$. Also note no points are added to $T[u \wedge v]$ over the two operators. Thus $n_{T'}(e) \leq n'_T(u \wedge v) = n_T(u \wedge v)$. This means $\text{cost}_{T'}(e) \leq w(e) \cdot n_T(u \wedge v)$, so $\text{cost}_{T'}(e) \leq \frac{n_T(u \wedge v)}{n_T(u)} \text{cost}_T(e)$. Thus, $\Delta = \frac{n_T(u \wedge v)}{n_T(u)}$. □

The level abstraction operator is somewhat more complicated, as it modifies entire levels of the tree, instead of individual splits. However, we can still use our notion of operation cost to bound the operator's impact. This just becomes a bit more vague because we have to look at the largest and smallest clusters between depths h_1 and h_2 in T.

Lemma B.3. *Say we apply the level abstraction operator between heights h_1 and h_2 on hierarchy T to yield T'. An edge is separated by the operator if and only if the least common ancestor of its endpoints is between h_1 and h_2. Its operation cost is at most $\Delta \leq \frac{n_T(u)}{n_T(v)}$, where u and v are two clusters that are abstracted away that maximize this ratio.*

Proof of Lemma B.3. Let $e = (x, y)$ be an edge that is separated by a level abstraction operator $\text{abstract}(h_1, h_2)$ for some depths h_1 and h_2 with $h_1 < h_2$. Let's consider when we execute the abstraction. For x and y to be separated, $x \wedge y$ must be merged into its parent by the operator. That means it is between depth h_1 and h_2. Let v be the vertex with the smallest $n_T(v)$ between depths h_1 and h_2. Then $n_T(x \wedge y) \geq n_T(v)$, and so $\text{cost}_T(e) \geq w(e) \cdot n_T(v)$.

The ancestor it eventually gets contracted into must be of depth h_1, because we stop contracting after that point. Although its tree structure is altered below it, its cluster size remains the same since no vertices are moved away or to its subtree. Let u be the vertex with the largest $n_T(u)$ between depths h_1 and h_2. Then we get $n_{T'}(x \wedge y) \geq n_T(u)$, and so $\text{cost}_{T'}(e) \leq w(e) \cdot n_T(u)$.

This has shown us that $\text{cost}_{T'}(e) \leq \frac{n_T(u)}{n_T(v)} \text{cost}_T(e)$. Notice that u and v are precisely the internal nodes that maximize the ratio, so $\text{cost}_{T'}(e) \leq \frac{n_T(u)}{n_T(v)} \text{cost}_T(e)$.

□

Tree folding is a bit more complicated because we are merging multiple clusters on top of each other. Thus we have to factor in the value k on top of considering varying cluster sizes. Ultimately, however, the product of the ratio between cluster size and k bound the proportional increase in cost.

Lemma B.4. *Say we apply the tree folding operator on hierarchy T to yield T'. Its operation cost is at most $\Delta \leq k\frac{n_T(u)}{n_T(v)}$, where u and v are two clusters that are mapped to each other away that maximize this ratio.*

Proof of Lemma B.4. Let $e = (x, y)$ be an edge that is separated by a tree folding operator $\text{fold}(T_1, \ldots, T_k)$ for subtrees $T_1, \ldots, T_k$ of T satisfying the operator conditions. Let's consider when we execute the folding. For x and y to be separated, $x \wedge y$ must be in one of the subtrees, say T_1 without loss of generality. This means $\text{cost}_T(e) \geq w(e) \cdot n_T(x \wedge y)$.

Now we consider the cost in T'. Clearly, $x \wedge y$ becomes the single vertex in T_f corresponding to $\phi_1(x \wedge y)$. A leaf vertex in T_2 (without loss of generality) is only a descendant of $\phi_1(x \wedge y)$ if it has an ancestor a such that $\phi_2(a) = \phi_1(x \wedge y)$. Therefore:

$$n_{T'}(x \wedge y) = n_{T'}(\phi_1(x \wedge y)) \leq \sum_{i \in [k]} n_T(\phi_i^{-1}(\phi_1(x \wedge y))$$

If $u = \max\{n_T(u) : u \in T_i, i \in [k], \phi_i(u) = \phi_1(x \wedge y)\}$, then we further have:

$$n_{T'}(x \wedge y) \leq \sum_{i \in [k]} n_T(u) = k n_T(u)$$

This means that $\text{cost}_{T'}(e) \leq w(e) \cdot k n_T(u)$. Putting these together gives $\text{cost}_{T'}(e) \leq \frac{n_T(x \wedge y)}{n_T(v)} k \text{cost}_T(e) \leq \frac{n_T(u)}{n_T(v)} k \text{cost}_T(e)$ where u and v are the vertices merged together that maximize this ratio.

□

C. Proofs: Results

In this section, we prove all lemmas, theorems, and missing algorithmic discussion regarding our main results.

C.1. RebalanceTree

This section contains the proofs regarding RebalanceTree.

Proof of Lemma 4.2. By our definition of A_i and B_i for all $i \in [k]$, $|A_{k-1}| \geq 2n/3$, implying $|A_k| \geq \frac{1}{2}|A_{k-1}| \geq n/3$, and also that $|A_k| \leq n/3$ since $|B_k| \geq n/3$. Thus $n/3 \leq |A_k|, |B| \leq 2n/3$. Since we rearrange our first split to be

this way, that means our first tree rebalance creates a first split that satisfies the relatively balanced condition. From here, we recurse on each side, guaranteeing that one split after another satisfies the condition. Thus, the entire tree is $\frac{1}{6}$-relatively balanced. □

Proof of Lemma 4.3. Consider an edge $e = (x, y)$ that is first rebalanced at some recursive step in Algorithm 1. By Lemma B.1, x and y must now be separated at the current tree's root. Therefore, at any further level of recursion, only one of e's endpoints will be present, so it cannot be separated again. □

Proof of Lemma 4.4. The rebalance operator is applied to v (the node found) at r. Notice that v's parent p must be such that $n_T(p) \geq \frac{2}{3} n_T(r)$, otherwise the loop would have stopped earlier. Therefore, by Lemma B.1, the operation cost is $n_T(r)/n_T(p) \leq 3/2$. □

Proof of Theorem 4.1. Let T^* be the optimal tree, let T_1 be our guaranteed γ-approximation on T, and let T' be our output. By Lemma 4.2, T' is $1/6$-relatively balanced. By Lemmas 4.3 and 4.4, every edge is separated by at most one tree rebalance operator of length at most $3/2$. Because of this, $\text{cost}_{T'}(e) \leq (3/2)\text{cost}_{T_1}(e)$. Summing over all edges yields $\text{cost}(T) \leq (3/2)\text{cost}(T_1) \leq \gamma \text{cost}(T^*)$. □

C.2. RefineRebalanceTree

This section contains the proofs regarding RefineRebalanceTree as well as the algorithmic description of SubtreeSearch.

As discussed in the body, at a given split, SubtreeSearch traverse the tree below the larger cluster further down in a similar manner until we find a sufficiently small cluster. This cluster must be smaller than the current balance error, ϵ. We simply do this by always traversing to the larger cluster as in Algorithm 4 until its smaller child is sufficiently small. We then remove that subtree, traverse back to the top of the tree, and try to reinsert the subtree by recursing down the right children.

This exhibits nice properties with respect to relative balance.

Lemma C.1. SubtreeSearch *preserves* $\frac{1}{6}$*-relative balance.*

Proof of Lemma C.1. Consider T, the tree at the beginning of the algorithm, and let v be the vertex whose subtree we end up moving. To start, we only consider the deletion, and then we will consider the reinsertion of v's subtree. The only vertices whose corresponding cluster sizes are altered (specifically, reduced) are v's ancestors. Note that they are all right children (i.e., the bigger sibling at the start) and they are reduced by size $n_T(v)$.

Algorithm 4 SubtreeSearch

Input: A $\frac{1}{6}$-relatively balanced hierarchy tree T of size n, with smaller cluster always on the left and error parameter s.
Output: Modified $\frac{1}{6}$-relatively balanced T by a subtree deletion and insertion of a subtree of size between $s/3$ and s

$v = \text{root}(T)$
while $|\text{leaves}(\text{left}_T(v))| > s$ **do**
  $v \leftarrow \text{right}_T(v)$
end while
$v \leftarrow \text{left}_T(v)$

$u \leftarrow \text{root}(T)$
while $|\text{leaves}(\text{right}_T(u))| \geq |\text{leaves}(v)|$ **do**
  $u \leftarrow \text{left}_T(u)$
end while
$T' \leftarrow T.\text{del\_ins}(u, v)$
Return T

Let p be the parent of v. Since $v = \text{left}_T(p)$, we know $n_T(v) \leq \frac{1}{2} n_T(p)$. Since we remove that many vertices, $n_T(p)$ is at worst halved. Since p is a right child, say with sibling node q, $n_T(p) \geq n_T(q)$ at the start. Then at the end, $n_{T'}(p) \geq \frac{1}{2} n_{T'}(q)$. This implies that, in the end, the clusters are between 1/3 and 2/3 the size of their parent. Thus relative balance is held on this split. For ancestor nodes a of p in T, this argument holds since $n_T(a) > n_T(p)$ both before and after, and a is also a right child. Therefore, the entire tree is still $\frac{1}{6}$-relatively balanced after subtree deletion.

Now we consider the second half of the algorithm, where we reinsert $T[v]$. Let u be the vertex we select to insert at, p be its new parent, g be its old parent (now its grandparent), and $r = \text{right}_T(g)$ be its old sibling. Before insertion, we know that $n_T(r) \geq n_T(v)$ by the while loop condition. Since T is $\frac{1}{6}$-relatively balanced still, and r is u's sibling, $n_T(u) \geq \frac{1}{2} n_T(r)$. Since the algorithm did not stop at p, then $n_T(r) \geq n_T(v)$, thus implying $n_T(u) \geq \frac{1}{2} n_T(v)$. Additionally, since the algorithm stopped on u, $n_T(\text{right}_T(u)) \leq n_T(v)$. Since that is u's larger child, $n_T(u) \leq 2n_T(\text{right}_T(u)) \leq 2n_T(v)$. Since v is u's new sibling, and one is not more than twice the size of the other, we have $\frac{1}{6}$-relative balance at that split.

The only other vertices impacted by the insertion are u's ancestors. For an ancestor node a of u in T, the argument also holds since $n_T(a) \geq n_T(p) \geq n_T(v)$ meaning $n_{T'}(a) \leq 2n_T(a)$ and a must be a left (and therefore smaller) child. Therefore, the relative balance is kept at all splits involving ancestors of u, thus we have relative balance. □

Our other guarantee is that we find a subtree of size at least $s/2$ to move. This comes from our first loop's end condition.

Lemma C.2. *In Algorithm 4, $s/3 \leq n_T(v) \leq s$.*

Proof of Lemma C.2. When the first loop stops, this is the first visited vertex whose left child, which ends up being the final v, is at most s. Thus $n_T(v) \leq s$. Since this was the first such instance, if g is the grandparent of v, this means $n_T(\text{left}_T(g)) > s$ since the loop continued after g. Since right children are larger and v's parent p is $\text{right}_T(g)$, $n_T(p) \geq n_T(\text{left}_T(g)) > s$. Since we have $\frac{1}{6}$-relative balance, $n_T(v) \geq \frac{1}{3} n_T(p) \geq \frac{1}{3} s$. □

Proof of Lemma 4.6. At each iteration of Algorithm 2, as long as the relative balance is above ϵ, we move a subtree of size at least $\frac{1}{3}\delta n$ and at most δn by Lemma C.2 where δ is the current relative balance. This means that the relative balance of the first split reduces by a factor of $\frac{2}{3}$, and by Lemma C.1, the rest of the tree remains $\frac{1}{6}$-relatively balanced. This is simply done until the relative balance of the first split is small enough. When we recurse, we are still guaranteed $\frac{1}{6}$-relatively balance, and we can then ensure all sufficiently large splits are ϵ-relatively balanced. □

Proof of Lemma 4.7. Consider an edge e that is first separated by some subtree deletion and insertion operator at some recursive step in Algorithm 2. Notice e must now be separated at the current tree's root. This means that at any further level of recursion, only one of e's end points will be present, so it cannot be separated again. □

Proof of Lemma 4.8. The subtree deletion and insertion operator is applied at u of $T[v]$ when $u \wedge v$ is the root, i.e., $n_T(u \wedge v) = n_T(r) \leq n$ where r is the current tree's root and n is our original data set size. We never allow the algorithm to continue with $\delta \leq \epsilon$, therefore the smallest tree size $T[v]$ that we move is $\frac{1}{3} n\epsilon$ by Lemma C.2. Thus the operation cost is at most $n_T(u \wedge v)/n_T(v) \leq \frac{3}{\epsilon}$ by Lemma B.2. □

Proof of Theorem 4.5. Let T^* be the optimal tree, let T_1 be our $1/6$-relatively balanced $3\gamma/2$ approximation guaranteed by Theorem 4.1, and let T' be our output. By Lemma 4.6, T' is ϵ-relatively balanced. By Lemmas 4.7 and 4.8, every edge is separated by at most one subtree deletion and insertion operator of operation cost at most $3/\epsilon$. Because of this and because of Lemma B.2, $\text{cost}_{T'}(e) \leq \frac{3}{\epsilon}\text{cost}_{T_1}(e)$. Summing over all edges yields $\text{cost}(T) \leq \frac{3}{\epsilon}\text{cost}(T_1) \leq \frac{9\gamma}{2\epsilon}\text{cost}(T^*)$. □

C.3. StochasticallyFairHC

This section contains the proofs regarding StochasticallyFairHC.

Proof of Lemma 4.10. Since T is ϵ-relatively balanced, any cluster A that splits into clusters B and C satisfies $(1/2 - \epsilon)|A| \leq |C| \leq |B| \leq (1/2 + \epsilon)|A|$, without loss of generality. This means that the maximum cluster size that can be found at level i is bounded above by traversing the tree from root down assuming that we always traverse to a maximally sized child, e.g., if p is a parent of w on our path, then $n_T(w) \leq (1/2 + \epsilon) n_T(p)$.

Since we traverse i levels, we get for any i-level vertex u, $n_T(u) \leq (1/2 + \epsilon)^i n$. By the reverse logic (i.e., traversing from the root to a minimally sized child), for any i-level vertex v, $n_T(v) \geq (1/2 - \epsilon)^i n$. Then their ratio must be at most $\frac{n_T(u)}{n_T(v)} \leq \frac{(1+2\epsilon)^i}{(1-2\epsilon)^i}$.

Finally, consider if $i \leq \log_{1/2-\epsilon}(x/n)$. We can just assume it is at the maximum possible level, because this will clearly give the loosest bounds. We already know the smallest cluster size at level i is at least $(1/2 - \epsilon)^i n$, and the ratio between the largest and smallest cluster sizes is at most $\left(\frac{1+2\epsilon}{1-2\epsilon}\right)^i$. Therefore, for a vertex u at level i:

$$n_T(u) \leq \left(\frac{1+2\epsilon}{1-2\epsilon}\right)^{\log_{1/2-\epsilon}(n/x)} (1/2 - \epsilon)^{\log_{1/2-\epsilon}(x/n)} n$$

The second term in the product obviously simplifies to x. For the first term, we can see that since $\epsilon = 1/(c \log_2 n)$:

$$\frac{1+2\epsilon}{1-2\epsilon} = 1 + \frac{4\epsilon}{1-2\epsilon} = 1 + \frac{4}{c(1-2\epsilon)\log_2 n}$$

We can also bound the exponent. Note that we raise a value that is at least 1 to the exponent, so to create an upper bound, we must upper bound the exponent as well. We leverage the fact that $1/(1/2 - \epsilon) > 2$ because $\epsilon \in (0, 1/2)$. This implies $\log_2(1/(1/2 - \epsilon)) > 1$.

$$\begin{aligned}
\log_{1/2-\epsilon}(x/n) &= \frac{\log_2(x/n)}{\log_2(1/2-\epsilon)} \\
&= \frac{\log_2(n/x)}{\log_2(1/(1/2-\epsilon))} \\
&\leq \log_2(n/x) \\
&\leq \log_2(n)
\end{aligned}$$

Where the last step comes from the fact taht $x \geq 1$. We can now put this all together.

$$n_T(u) \leq \left(1 + \frac{4}{c(1-2\epsilon)\log_2 n}\right)^{\log_2(n)} x \leq x \cdot e^{4/(c(1-o(1))}$$

□

Proof of Lemma 4.11. By Lemma 4.10, the smallest cluster at depth $i \geq t$ is at most $(1/2+\epsilon)^t n$. If we assume a trivial cluster size lower bound of 1, this implies for any contracted internal nodes u and v in the level abstraction, $n_T(u)/n_T(v) \leq (1/2+\epsilon)^t n$. □

Proof of Lemma 4.12. By Lemma 4.10, the largest cluster at depth $i \leq t$ in T is at least $(1/2-\epsilon)^t n$. When we execute level abstraction, this cluster size is not changed, but we know all other potentially smaller clusters are contracted into their parents. Thus, this is the smallest cluster size in T'. □

Proof of Lemma 4.13. Consider a vertex v at height 1. By Lemma 4.12, $n_{T'}(v) \geq (1/2-\epsilon)^t n = \frac{3(1-\delta)}{a\delta^2}\ln(cn)$. Fix some $\ell \in [\lambda]$. Let $X_{\ell v}$ count the number of vertices of color ℓ in $\text{leaves}(v)$. Note this is a sum of Bernoullis, so $\mathbb{E}[|X_{\ell v}|] = \sum_{u \in \text{cluster}(v)} p_\ell(u)$. Note that we are given that $p_\ell(u) \geq \frac{1}{1-\delta}\alpha_\ell$ for all u. This gives us the following bounds from Lemma 4.12:

$$\begin{aligned}
\mathbb{E}[X_{\ell v}] \geq& \frac{3(1-\delta)}{a\delta^2}\ln(cn)\cdot\frac{1}{1-\delta}a = \frac{3}{\delta^2}\ln(cn)\\
\mathbb{E}[X_{\ell v}] \geq& \frac{1}{1-\delta}\alpha_\ell n_{T'}(v)\\
\mathbb{E}[X_{\ell v}] \leq& \frac{1}{1+\delta}\beta_\ell n_{T'}(v)
\end{aligned}$$

Then by a Chernoff bound with δ as the error parameter:

$$\begin{aligned}
P(|X_{\ell v} - \mathbb{E}[X_{\ell v}]| \geq \delta\mathbb{E}[X_{\ell v}])&\\
\leq& 2\exp(-\mathbb{E}[X_{\ell v}]\delta^2/3)\\
=& 2\exp(-\frac{3}{\delta^2}\ln(cn)\delta^2/3)\\
=& \frac{2}{cn}
\end{aligned}$$

Thus with probability at least $1-\frac{2}{cn}$:

$$\begin{aligned}
X_{\ell v} - \mathbb{E}[X_{\ell v}] \leq& \delta\mathbb{E}[X_{\ell v}]\\
X_{\ell v} \leq& (1+\delta)\mathbb{E}[X_{\ell v}]\\
\leq& (1+\delta)\cdot\frac{1}{1+\delta}\beta_\ell n_{T'}(v)\\
\leq& \beta_\ell n_{T'}(v)
\end{aligned}$$

In other words, the cluster $\text{leaves}(v)$ satisfies the upper bound for color ℓ. We also find that:

$$\begin{aligned}
-X_{\ell v} + \mathbb{E}[X_{\ell v}] \geq& \delta\mathbb{E}[X_{\ell v}]\\
X_{\ell v} \geq& (1-\delta)\mathbb{E}[X_{\ell v}]\\
\geq& (1-\delta)\cdot\frac{1}{1-\delta}\alpha_\ell n_{T'}(v)\\
\geq& \alpha_\ell n_{T'}(v)
\end{aligned}$$

Which means it also satisfies the upper bound. Let y be the number of internal nodes with leaf-children. Since we already saw the minimum such cluster size is $O(\log n)$ (since $a, \delta = O(1)$), then $y = O(n/\log n)$. Notice, also, that the vertices counted by y are the only ones we need to prove are fair, since taking the union of two fair clusters is fair. Thus, to show this is true for all ℓ and v, we take a union bound over all λ values of ℓ and y values of v. We then find that with probability at least $1 - \frac{2\lambda n/\log n}{cn} = 1 - \frac{2}{\log n}$, all height 1 clusters must be fair, meaning the entire hierarchy must be fair by the union-bound property.

□

Proof of Theorem 4.9. Let T^* be the optimal tree, let T_1 be our ϵ-relatively balanced $\frac{9\gamma}{2\epsilon}$ approximation guaranteed by Theorem 4.5, and let T' be our output. By Lemma 4.13, T' satisfies our fairness constraints. By Lemmas 4.11 and the fact that we only apply one operator, every edge is separated by at most one level abstraction operator of operation cost at most $(1/2+\epsilon)^t n$, but we know from Lemma 4.10 that this is bounded above by $e^{4/(c(1-o(1)))}\cdot\frac{3(1-\delta)\ln(cn)}{a\delta^2}$. Because of this, $\text{cost}_{T'}(e) \leq e^{4/(c(1-o(1)))}\cdot\frac{3(1-\delta)\ln(cn)}{a\delta^2}\text{cost}_{T_1}(e)$. Summing over all edges yields $\text{cost}(T) \leq e^{4/(c(1-o(1)))}\cdot\frac{3(1-\delta)\ln(cn)}{a\delta^2}\text{cost}(T_1) \leq e^{4/(c(1-o(1)))}\cdot\frac{3(1-\delta)\ln(cn)}{a\delta^2}\cdot\frac{9\gamma}{2\epsilon}\text{cost}(T^*)$. □

C.4. FairHC

This section contains the proofs and additional theoretical discussion regarding FairHC.

Lemma C.3. StochasticallyFairHC *with* $t = \log_{1/2-\epsilon}(1/(2n\epsilon))$ *outputs a hierarchy where the* ϵ*-relatively balanced guarantee holds for all splits except those forming the leaves. Additionally, it admits a proportional cost increase of at most* $\frac{1}{2}ce^{4/(c(1-o(1)))}\log_2 n$.

Proof of Lemma C.3. Say T is our input (i.e., it is ϵ-relatively balanced). Notice that StochasticallyFairHC only modifies T's structure below depth $\log_{1/2-\epsilon}(1/(2n\epsilon))$, which means any balance guarantees hold up to that level. By Lemma 4.10, for any vertex v at depth $\log_{1/2-\epsilon}(1/(2n\epsilon))$ or above, $n_T(v) \geq (1/2-\epsilon)^{\log_{1/2-\epsilon}(1/(2n\epsilon))}n = 1/(2\epsilon)$. By the definition of ϵ-relative balance, this means that the balance guarantee holds

for the split at this vertex. Since all internal vertices in the resulting tree T' are at or above this level, all internal vertices except those with leaf children exhibit the relative balance guarantee.

In order to bound the proportional increase in cost, we must bound the operation cost of the level abstraction. The minimum depth in the abstraction is $\log_{1/2-\epsilon}(1/(2n\epsilon))$. By Lemma 4.10, this means the maximum cluster size is at most $e^{4/(c(1-o(1)))}/(2\epsilon) = \frac{1}{2}ce^{4/(c(1-o(1))}\log_2 n$. Since the smallest cluster size involved is at least 1, we can then bound the operation cost by this max cluster size, giving our result. □

Lemma C.4. *If T is ϵ-relatively balanced besides the final layer of splits, then the subtrees rooted at all of the root's children in* FairHC *after tree folding are as well.*

Proof of Lemma C.4. Tree folding only involves overlaying the topology of isomorphic trees (ignoring their leaves). Consider a non-root vertex v in FairHC after tree folding that is also not a parent of leaves. It is the result of merging k vertices $v_1, \ldots, v_k$, and its left and right children l and r are the result of merging $l_1, \ldots, l_k$ and $r_1, \ldots, r_k$ respectively. Due to this:

$$n_{T'}(v) = \sum_{i\in[k]} n_T(v_i)$$
$$n_{T'}(l) = \sum_{i\in[k]} n_T(l_i)$$
$$n_{T'}(r) = \sum_{i\in[k]} n_T(r_i)$$

We also have, by relative balance, for any $i \in [k]$:

$$(1/2-\epsilon)n_T(v_i) \le n_T(l_i), n_T(r_i) \le (1/2+\epsilon)n_T(v_i)$$

A simple combination of these shows that:

$$(1/2-\epsilon)n_T(v) \le n_T(l), n_T(r) \le (1/2+\epsilon)n_T(v)$$

This means the split from v to l and r is relatively balanced. We can apply this to all such splits to find the entire new subtree is relatively balanced. □

Proof of Lemma 4.15. By Lemma 4.10, for any vertex v at depth $i \ge \log_2 h$, $n_T(v) \ge (1/2-\epsilon)^{\log_2 h} n$. This can be further simplified using that $\epsilon = 1/(c\log n)$ and $h \le n$.

$$n_T(v) \ge \frac{1}{2^{\log_2 h}}\left(1-\frac{2}{c\log_2 n}\right)^{\log_2 h} n \ge e^{-2/c}n/h$$

Obviously, the largest $n_T(u)$ for any u within our depth bounds is n. Thus the level abstraction operation cost is at most $n/(e^{-2/c}n/h) = e^{2/c}h$. □

Proof of Lemma 4.16. That it acts on k trees is obvious. To prove the operation cost, consider u, v in trees T_i and T_j respectively where $\phi_i(u) = \phi_j(v)$. Since we are using the tree isomorphism between the trees, this means that u and v have the same height in T_i and T_j respectively, which also means that they had the same height in the original tree T, as the roots of T_i and T_j are both at height $\log_2(h)$ in T. Since u and v are on the same level $i \le \log_{1/2-\epsilon}(1/(2n\epsilon))$, Lemma 4.10 tells us:

$$\frac{n_T(u)}{n_T(v)} \le e^{4/(c(1-o(1))}$$

Since this holds for all such pairs u and v, this also bounds the tree folding operation cost. Note that when we do this operator, the ϵ-relative balance is held by Lemma C.4. Thus, this argument holds across all tree folds in the for loop. □

Proof of Lemma 4.17. If an edge $e = (u, v)$ is separated by the level abstraction, that means $u \wedge v$ is above depth $\log_2 h$. Notice that we recurse on clusters at depth $\log_2 h$, which means on any recursive instance here forward, e will not be contained within the trees, so e cannot be separated. Therefore, e can only be separated by one level abstraction.

Otherwise, notice that the depth of the last internal node is $\log_{1/2-\epsilon}(1/(2n\epsilon))$ by assumption. At each recursive step, we reduce the depth by $\log_2 h$ since we start at subtrees of depth h in the previous tree. Therefore, there are at most $\log_{1/2-\epsilon}(1/(2n\epsilon))/\log_2 h$ levels of recursion. To simplify this, we use similar methods to Lemma 4.16. which allows us to bound the recursive depth by $\log_2(n)/\log_2(h)$.

At each level of recursion, since e is contained in only one tree, it is only separated by λ tree folds. This means e may only be separated by $\lambda\log_2(n)/\log_2(h)$ tree folds. □

Lemma C.5. *For an ϵ-relatively balanced hierarchy T, Algorithm 3 outputs a tree T' such that:*

$$\text{cost}(T') \le e^{\frac{4\lambda\log_2(n)}{c(1-o(1))\log_2 h}+\frac{2}{c}} \cdot h\text{cost}(T)$$

Proof of Lemma C.5. Lemmas 4.15, 4.16, and 4.17 tell us an edge e must only be involved in at most 1 level abstraction of operation cost at most $e^{2/c}h$ and $\lambda\log_2(n)/\log_2(h)$ tree folds of operation cost at most $e^{4/(c(1-o(1))}$ on k trees. By

Lemmas B.3 and B.4, this will incur a total proportional cost increase of:

$$\frac{\text{cost}_{T'}(e)}{\text{cost}_T(e)} \leq (e^{4/(c(1-o(1))})^{\lambda \log_2(n)/\log_2(h)} \cdot e^{2/c}h$$

Which, summed over all edges, is equivalent to the desired result.

□

Lemma C.6. *For an ϵ-relatively balanced hierarchy T over $\ell(V) = c_\ell n = O(n)$ vertices of each color $\ell \in [\lambda]$,* FairHC *before recursion ensures that the clustering induced on each depth-1 internal node v of the output tree T' each have $\frac{c_\ell}{e^{6/c}} \leq \frac{\ell(v)}{\text{leaves}(v)} \leq c_\ell \cdot (e^{4/c}/(kc_\ell) + e^{6/c})$ for each $\ell \in [\lambda]$.*

Proof of Lemma C.6. We start by looking at one tree fold operator. Assume the color we are trying to sort is red. Consider the ordering of the vertices $\{v_i\}_{i\in[h]}$ from Algorithm 3, and let r_i the number of red points from $\text{leaves}(v_i)$ and R be the total number of red vertices.

Fix some i and let v'_i be the root vertex of the resulting subtree in the ith fold (i.e., the one all the subtrees are mapped onto). We know the vertices involved in this were $v_{i+(j-1)k}$ for all $j \in [h/k]$. Because of the ordering, we know that:

$$r_{i+(j-1)k}/n_T(v_{i+(j-1)k}) \leq r_{y+(j-2)k}/n_T(v_{y+(j-2)k}), \quad (1)$$

$$r_{i+(j-1)k}/n_T(v_{i+(j-1)k}) \geq r_{y+jk}/n_T(v_{y+(j-2)k}) \quad (2)$$

for all $y \in [h/k]$ assuming $j > 1$ for (1) and $j < k$ for (2). Since these three vertices are at the same height, say h' (with respect to T after rebalancing and before the algorithm began), Lemma 4.10 gives us that:

$$n_T(v_{i+(j-1)k})/n_T(v_{y+(j-2)k}) \leq \frac{(1+2\epsilon)^{\log_2 n}}{(1-2\epsilon)^{\log_2 n}},$$
$$n_T(v_{i+(j-1)k})/n_T(v_{y+jk}) \geq \frac{(1-2\epsilon)^{\log_2 n}}{(1+2\epsilon)^{\log_2 n}}$$

Combining these with the previous inequalities yield:

$$r_{i+(j-1)k} \leq \frac{(1+2\epsilon)^{\log_2 n}}{(1-2\epsilon)^{\log_2 n}} r_{y+(j-2)k},$$
$$r_{i+(j-1)k} \geq \frac{(1-2\epsilon)^{\log_2 n}}{(1+2\epsilon)^{\log_2 n}} r_{y+jk}$$

for all $y \in [h/k]$. Since $\epsilon = 1/(c\log_2 n)$, this bound can be further simplified to:

$$e^{-4/c} r_{y+jk} \leq r_{i+(j-1)k} \leq e^{4/c} r_{y+(j-2)k}$$

Since these hold for all y, we can say that:

$$\frac{k}{h} e^{-4/c} \sum_{y\in[h/k]} r_{y+jk} \leq r_{i+(j-1)k} \leq \frac{k}{h} e^{4/c} \sum_{y\in[h/k]} r_{y+(j-2)k}$$

Another way to think of this is partitioning the vertices (in order) into contiguous chunks of size h/k. Then $v_{i+(j-1)k}$ is the ith vertex in the jth chunk, and we know it has a lower of fraction of red points than clusters in the previous ($(j-2)$th) chunk and a higher fraction than clusters in the next (jth) chunk.

Now let R_{j-1} be the number of reds in the entire jth chunk (i.e., $R_{j-1} = \sum_{y\in[h/k]} r_{y+(j-1)k}$) Additionally, we can make a comparison between the reds in all chunks and R, namely, $\sum_{j\in[k]} R_{j-1} = R$.

Putting our two previous results together, for our fixed i:

$$\begin{aligned}
\sum_{j\in[k]} r_{i+(j-1)k} &\leq r_i + \frac{k}{h} e^{4/c} \sum_{j\in[k]} R_{j-1} \\
&= r_i + \frac{k}{h} e^{4/c} R, \\
\sum_{j\in[k]} r_{i+(j-1)k} &\geq r_{h-h/k+i} + \frac{k}{h} e^{-4/c} \sum_{j\in[k]} R_{j-1} \\
&= \frac{k}{h} e^{-4/c} R
\end{aligned}$$

Notice that if everything were perfectly balanced, $\frac{k}{h}R$ is exactly the number of reds we would want in $\text{leaves}(v'_i)$. We now must bound r_i. Unfortunately, it could be an entirely red cluster, so this is only bounded by the size of the cluster at depth $\log_2 h$, which we get from Lemma 4.10.

$$\begin{aligned}
r_i &\leq n_T(v_i) \\
&\leq (1/2+\epsilon)^{\log_2 h} n \\
&= 2^{-\log_2 h} (1+2\epsilon)^{\log_2 h} n \\
&\leq e^{2/c} n/h
\end{aligned}$$

Note the final inequality comes from the fact that $h \leq n$ and $\epsilon = 1/(c\log n)$. Now note that we are given $R = c_R n$ for some $c_R = O(1)$. We can sub this in.

$$r_i \le e^{2/c} R/(c_R h)$$

Now, notice we are actually looking for the fraction of red points in the cluster. Since Lemma 4.10 gives us that $n_{T'}(v'_i) \ge k(1-2\epsilon)^{\log_2 h} n/h \ge ke^{-2/c}n/h$ and $n_{T'}(v'_i) \le k(1+2\epsilon)^{\log_2 h} n/h \le ke^{2/c}n/h$ (applying the same logic as the upper bound to $n_T(v_i)$, k times), we get:

$$\begin{aligned}
\frac{\sum_{j\in[k]} r_{i+jk}}{n_{T'}(v'_i)} &\le \frac{e^{2/c}R/(c_R h) + \frac{k}{h}e^{4/c}R}{ke^{-2/c}(n/h)} \\
&= \frac{R}{n}\cdot\left(\frac{e^{4/c}/c_R}{k} + e^{6/c}\right), \\
\frac{\sum_{j\in[k]} r_{i+jk}}{n_{T'}(v'_i)} &\ge \frac{\frac{k}{h}e^{-4/c}R}{ke^{2/c}(n/h)} \\
&= \frac{R}{n}\cdot\frac{1}{e^{6/c}}
\end{aligned}$$

This completes the proof for one tree fold under the observation that $\frac{R}{n} = c_\ell$ if red is ℓ. The same (if not stronger) bounds hold for all subsequent λ tree folds for each color. Note that as we proceed, this bound will not be disrupted since merging two clusters that guarantees the same upper bound on the fraction of red points still guarantees the same bound. □

Proof of Lemma 4.18. Clearly, the most imbalanced clusters in this process will be the clusters in the final level of the hierarchy. By Lemma C.6, when we recurse, we have at most an $\frac{c_\ell}{e^{6/c}} \le \frac{\ell(v)}{\text{leaves}(v)} \le c_\ell \cdot (e^{4/c}/(kc_\ell) + e^{6/c})$ fraction of vertices of color ℓ for each $\ell \in [\lambda]$. Clearly, after at most $\log_2 n/\log_2 h = \log_h n$ recursive levels guaranteed by Lemma 4.17, our bound becomes:

$$\frac{c_\ell}{e^{6t\log_h n/c}} \le \frac{\ell(v)}{\text{leaves}(v)} \le c_\ell \cdot (e^{4/c}/(kc_\ell) + e^{6/c})^{\log_h n}.$$

□

Proof of Theorem 4.14. Let T^* be the optimal tree, let T_1 be our input tree which is a $ce^{4/(c(1-o(1)))}\log_2 n \cdot \frac{9\gamma}{4\epsilon}$ approximation guaranteed by Theorem 4.9 but using $t = \log_{1/2-\epsilon}(1/(2n\epsilon))$ (this was shown more explicitly in Lemma C.3), and let T' be our output. By Lemma 4.18, T' satisfies our fairness constraints. By Lemmas 4.15, 4.16, and 4.17, every edge is separated by at most 1 level abstraction of max operation cost $e^{2/c}n/h$ and $\log_2(n)/\log_2(h)$ tree folds of operation cost at most $e^{4/(c(1-o(1)))}$ on k subtrees. Lemma C.5 immediately tells us:

$$\text{cost}(T') \le e^{\frac{4\lambda\log_2 n}{c(1-o(1))\log_2 h}+\frac{2}{c}} \cdot h\text{cost}(T_1)$$

Combining this with the approximation guaranteed by T_1:

$$\begin{aligned}
&\text{cost}(T') \\
&\le e^{\frac{4\lambda\log_2 n}{c(1-o(1))\log_2 h}+\frac{2}{c}} \cdot hce^{4/(c(1-o(1)))}\log_2 n \cdot \frac{9\gamma}{4\epsilon}\text{cost}(T^*)
\end{aligned}$$

Simplifying and plugging in $h = n^\delta, \epsilon = 1/(c\log_2 n)$ yields the desired result. □

D. Runtime

Here we analyze the runtime of our four algorithms. Recall that before each of these algorithms, we run a black-box cost-approximate hierarchical clustering algorithm as well as all previous algorithms. For simplicity, here we will present the contribution of each algorithm to the runtime.

Theorem 4.1: This algorithm starts at the root, traverses down one side of the tree until a certain sized cluster is found, and then applies a tree rebalance. It then recurses on each child. The length of traversal is bounded by $O(n)$, and a single tree rebalance operation requires some simple constant-time pointer operations. As this is run from each vertex in the tree, the total runtime is $O(n^2)$.

Theorem 4.5: As in the previous algorithm, here we do a computation at each of the $O(n)$ nodes in the tree. At each node, we apply subtree search until the desired balance is achieved. If δ is the current balance, we reduce this to at most $2\delta/3$ at each step. Thus, this will require a total of $O(\log(1/\epsilon))$ steps to complete. Each subtree search operation requires two, $O(n)$-length traversals to find the place to insert and delete. Otherwise, it is constant-time pointer math. Thus, the algorithm runs in $O(n^2\log(1/\epsilon))$ time.

Theorem 4.9: This algorithm is quite, simple, as we are simply deleting some set of low nodes in the tree. Thus it only requires $O(n)$ time.

Theorem 4.14: Again, we execute a computation for at most $O(n)$ nodes in the tree. By a similar logic as before, tree abstraction steps require $O(n)$ time. It is not too hard to see that computing the fraction of red and blue vertices in each considered cluster and then sorting them accordingly also requires $O(n)$ time. Finally, we fold the vertices on top of each other. The isomorphism used for folding can be found by simply indexing the vertices in each subtree, and then applied quite directly, which also takes $O(n)$ time. Thus this requires only $O(n^2)$ time.

Therefore, the entire final algorithm (without the blackbox step) is bounded by the computation time from Theorem 4.5,

which is $O(n^2 \log(1/\epsilon))$. Intuitively, ϵ is bounded by $\epsilon > 1/n$, thus this becomes $O(n^2 \log n)$.

E. Additional Experiments

We here provide the figures and results omitted from the main text due to space constraints.

E.1. Bank Data

We begin with the supplementary figure that complements Figure 5 in the final section. This figure depicts the fairness of each clusterin the hierarchy constructed by Algorithm 3 on the *Bank* data. We see an equivalent concentration about the true balance ratio of 1:3.

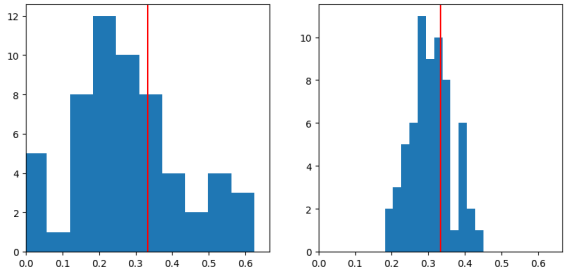

Figure 6: Histogram of cluster balances after tree manipulation by Algorithm 3. The left plot depicts the balances after applying the average- linkage algorithm and the right shows the result of applying our algorithm. The vertical red line indicates the balance of the dataset. Parameters were set to $c = 4, \delta = \frac{3}{8}, k = 4$ for the above clustering result.

E.2. Fairness Results for Parameter Sweep

We additionally provide the fairness results presented in Figure 5 for the other parameter sets plotted in Figure 4 for completeness.

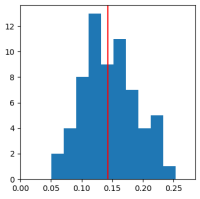

Figure 7: Parameters set to $c = 1, k = 8, \delta = \frac{3}{8}$.

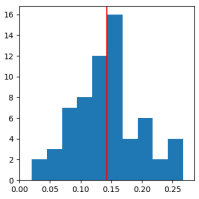

Figure 8: Parameters set to $c = 2, k = 8, \delta = \frac{3}{8}$.

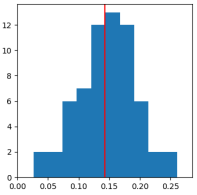

Figure 9: Parameters set to $c = 4, k = 8, \delta = \frac{3}{8}$.

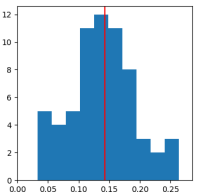

Figure 10: Parameters set to $c = 8, k = 8, \delta = \frac{3}{8}$.

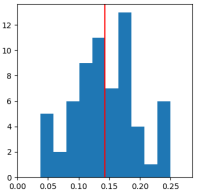

Figure 11: Parameters set to $c = 16, k = 8, \delta = \frac{3}{8}$.

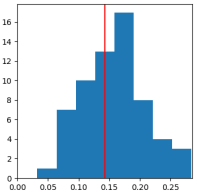

Figure 12: Parameters set to $c = 4, k = 4, \delta = \frac{7}{8}$.

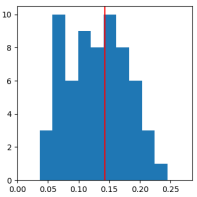

Figure 13: Parameters set to $c = 4, k = 8, \delta = \frac{7}{8}$.

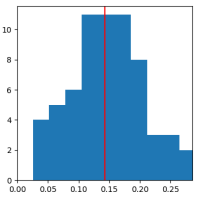

Figure 14: Parameters set to $c = 4, k = 16, \delta = \frac{7}{8}$.

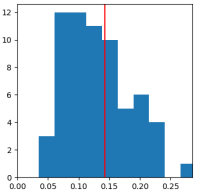

Figure 15: Parameters set to $c = 4, k = 4, \delta = \frac{3}{8}$.

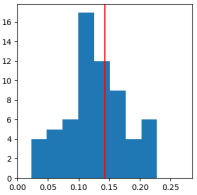

Figure 16: Parameters set to $c = 4, k = 4, \delta = \frac{4}{8}$.

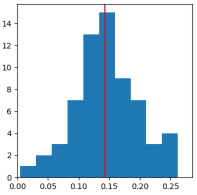

Figure 17: Parameters set to $c = 4, k = 4, \delta = \frac{5}{8}$.

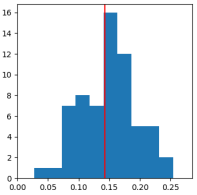

Figure 18: Parameters set to $c = 4, k = 4, \delta = \frac{6}{8}$.

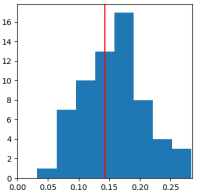

Figure 19: Parameters set to $c = 4, k = 4, \delta = \frac{7}{8}$.

F. Further Experimentation (Rebuttal Draft)

We here present further experimental results on a higher-dimensional data sample as compared to the main text. We first present the histogram of cluster balances after application of the average linkage algorithm for $n = 1024$ samples.

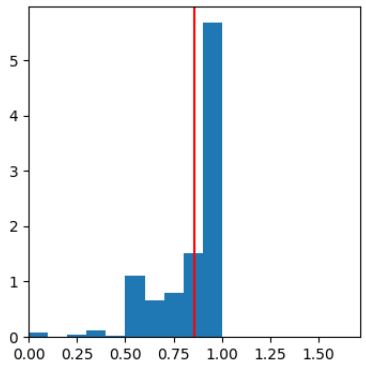

Figure 20: Average linkage balances for $n = 1024$ samples

We now proceed to apply our fair hierarchical clustering algorithm (Algorithm 3) on the constructed base cluster tree from average linkage resulted from the above. The algorithm was run with a wide variety of parameter tuples (c, h, k) where, as noted in the text, $h = n^{\delta}$. To further reiterate: the parameter c is used to define the ε cluster balance, h the number of clusters and k the number of trees folded in the rebalance procedure.

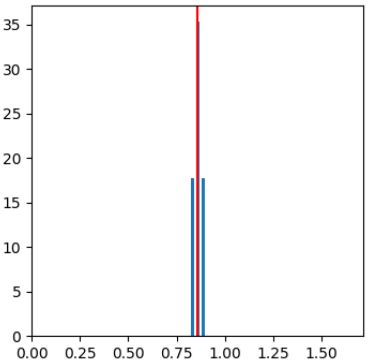

Figure 21: Result of running Algorithm 3 on $n = 1024$ samples with parameter tuple $(c, h, k) = (1, 4, 2)$.

G. Runtime and Cost Experiments

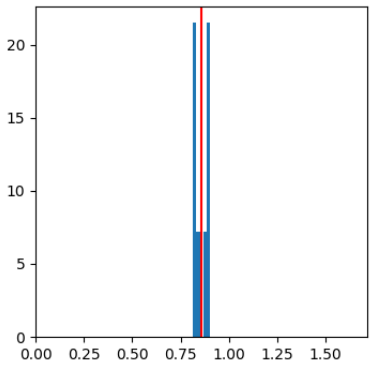

Figure 22: Result of running Algorithm 3 on $n = 1024$ samples with parameter tuple $(c, h, k) = (1, 8, 2)$.

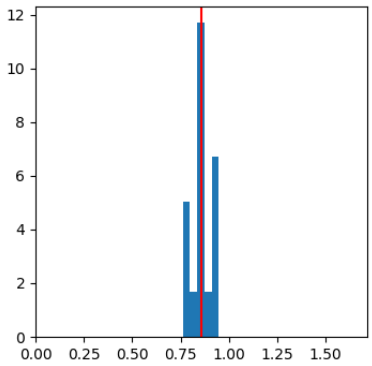

Figure 23: Result of running Algorithm 3 on $n = 1024$ samples with parameter tuple $(c, h, k) = (1, 16, 2)$.

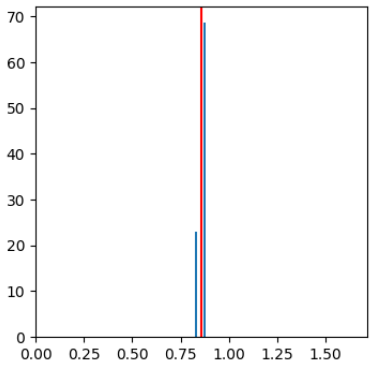

Figure 24: Result of running Algorithm 3 on $n = 1024$ samples with parameter tuple $(c, h, k) = (2, 4, 2)$.

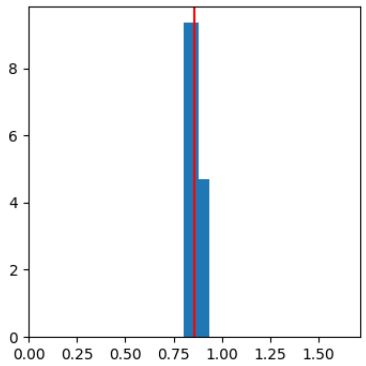

Figure 25: Result of running Algorithm 3 on $n = 1024$ samples with parameter tuple $(c, h, k) = (2, 8, 2)$.

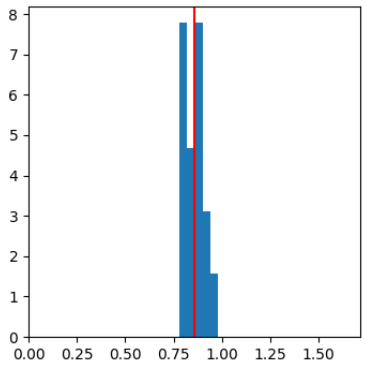

Figure 26: Result of running Algorithm 3 on $n = 1024$ samples with parameter tuple $(c, h, k) = (2, 16, 2)$.

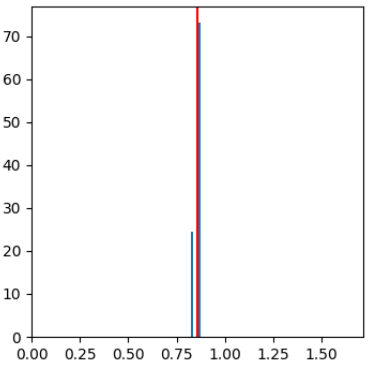

Figure 27: Result of running Algorithm 3 on $n = 1024$ samples with parameter tuple $(c, h, k) = (4, 4, 2)$.

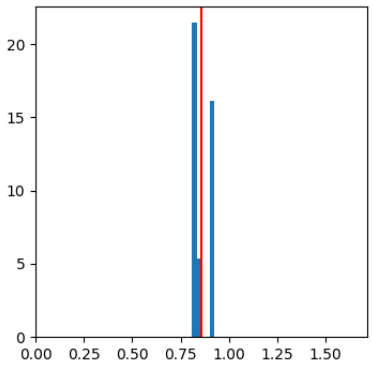

Figure 28: Result of running Algorithm 3 on $n = 1024$ samples with parameter tuple $(c, h, k) = (4, 8, 2)$.

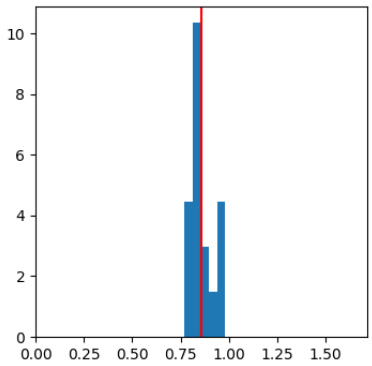

Figure 29: Result of running Algorithm 3 on $n = 1024$ samples with parameter tuple $(c, h, k) = (4, 16, 2)$.

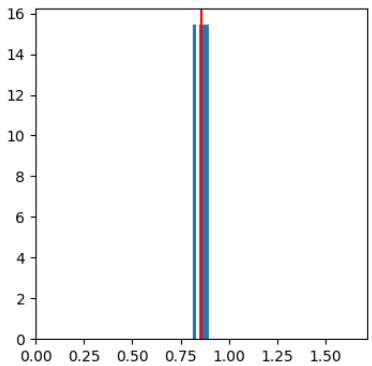

Figure 30: Result of running Algorithm 3 on $n = 1024$ samples with parameter tuple $(c, h, k) = (8, 4, 2)$.

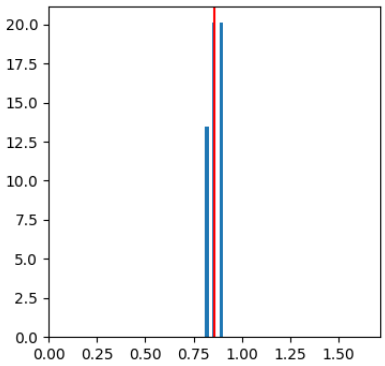

Figure 31: Result of running Algorithm 3 on $n = 1024$ samples with parameter tuple $(c, h, k) = (8, 8, 2)$.

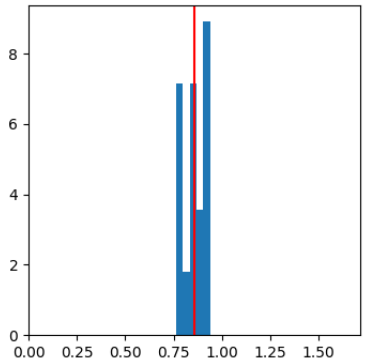

Figure 32: Result of running Algorithm 3 on $n = 1024$ samples with parameter tuple $(c, h, k) = (8, 16, 2)$.

c	h	k	**Runtime** (s)	**Cost**
1	4	2	1.483	129.916
1	8	2	1.351	164.475
1	16	2	1.314	201.268
2	4	2	1.535	146.919
2	8	2	1.351	149.92
2	16	2	1.313	208.099
4	4	2	1.534	150.02
4	8	2	1.353	156.071
4	16	2	1.305	224.063
8	4	2	1.454	136.179
8	8	2	1.302	225.995