APPENDIX

A ADDITIONAL PRELIMINARIES

Ensuring Pareto-optimality: A major challenge when formulating fair algorithms is that those may generate Pareto-dominated solutions [74]. That is, those may produce a fair solution that are worse for all groups, including minorities, compared to the unfair ones. In particular, in a utility assignment setting, let the utility assigned to each group $\mathbf{g}_i$ by the fair algorithm be u_i. There may exist another (unfair) assignment that assigns $u'_i > u_i$, $\forall \mathbf{g}_i \in \mathcal{G}$. This usually can happen when fairness is defined as the parity between different groups, without further specifications.

In this paper, the requirements 2 and 3 (single and pairwise fairness) have been specifically defined in a way to prevent generating Pareto-dominated fair solutions. To better explain the rational behind our definitions, let us consider pairwise fairness (the third requirement). Only requiring equal collision probability between various groups, the fairness constraint would translate to $Pr[h(p_i) = h(q_i)] = Pr[h(p_k) = h(q_k)]$, where p_i and q_i belong to group $\mathbf{g}_i$. Now let us consider a toy example with two groups $\{\mathbf{g}_1, \mathbf{g}_2\}$, two buckets $\{b_1, b_2\}$, and n points where half belong to $\mathbf{g}_1$. Let the hashmap $\mathcal{H}$ map each point $p \in \mathbf{g}_1$ to b_1 and each point $q \in \mathbf{g}_2$ to b_2. In this example, the collision probability between a random pair of points is 0.5, simply because each bucket contains half of the points. It also satisfies the collision probability equality between pairs of points from the same groups: $Pr[h(p_i) = h(q_i)] = Pr[h(p_k) = h(q_k)] = 1$.

This, however, is the worst assignment for both groups as their pairs *always collide*. In other words, it is fair, in a sense that it is equally bad for both groups. Any other hashmap would have a smaller collision probability for both groups and, even if not fair, would be more beneficial for both groups. In other words, this is a fair but Pareto-dominated solution.

In order to ensure Pareto-optimality while developing fair hashmaps, in requirements 2 and 3 (single and pairwise fairness), not only we require the probabilities to be equal, but also we require them to be equal to the *best case*, where the collision probability is $\frac{1}{m}$. As a result, we guarantee that (1) our hashmap is fair and (2) no other hashmap can do better for any of the groups.

Calculation of probabilities: Given a hashmap, we give the exact close forms for computing the collision probability, the single fairness, and the pairwise fairness. Let $\alpha_{i,j}$ be the number of items from group i at bucket j and let $n_j = \sum_{i=1}^{k} \alpha_{i,j}$. The probabilities are calculated as following, collision probability: $\sum_{j=1}^{m} \left(\frac{n_j}{n}\right)^2$, single fairness: $\sum_{j=1}^{m} \frac{\alpha_{i,j}}{|\mathbf{g}_i|} \cdot \frac{n_j}{n}$, pairwise fairness: $\sum_{j=1}^{m} \left(\frac{\alpha_{i,j}}{|\mathbf{g}_i|}\right)^2$.

B EXTENDED DETAILS OF DISCREPANCY-BASED HASHMAPS

B.1 Approximate collision probability and single fairness

Let P_w be an ordering of points in P and let $\mathcal{H}$ be a hashmap constructed on P_w. Recall that Pr_i is defined as the pairwise fairness of $\mathcal{H}$. Let Cp be the collision probability and Sp_i the single fairness of group $\mathbf{g}_i$. We show that finding a hashmap/partition with γ-discrepancy leads to a hashmap with bounded collision probability, single fairness, and pairwise fairness. The proof of the next Lemma can be found in Appendix C.

Lemma 4. *Let $\mathcal{H}$ be a hashmap satisfying γ-discrepancy. Then $Cp \leq \frac{1+\gamma}{m}$, $\frac{1-\gamma}{m} \leq Sp_i \leq \frac{1+\gamma}{m}$ and $Pr_i \leq \frac{1+\gamma}{m}$ for each group $\mathbf{g}_i$.*

Next, we propose ranking-based algorithms for finding a hashmap satisfying γ-discrepancy for the smallest γ. The first one is a slow algorithm with optimum discrepancy. The second one is a faster approximation algorithm.

Slower algorithm with optimum discrepancy.

It is sufficient to find the minimum γ so that a partition of exactly m buckets satisfies γ-discrepancy. We use dynamic programming to design our algorithm. We consider the ordering of items P_w for a vector $w \in \mathcal{W}$. Let $T[i, j]$ be the discrepancy of the optimum partition among the first i projections using j buckets. We define the recursive relation

$$T[i, j] = \min_{x<i} \max\{T[x-1, j-1], Error(P_w^{x,i})\},$$

where $P_w^{x,i} \subset P_w$ contains the projections of P from the x-th to the i-th largest one, and $Error(P_w^{x,i})$ is the discrepancy of one bucket containing $P_w^{x,i}$.

The algorithm is described as follows. Assume that we want to compute $T[i, j]$ having already computed $T[i', j']$ for $i' < i$ and $j' < j$. We describe an efficient way to find and update $Error(P_w^{x,i})$ considering the different values of x. We define a max heap M of size k. For each group $\mathbf{g}_i$ we create the pair $(\mathbf{g}_i, 1)$ in M. We also define an array B of size m with $B[i] = 0$. Finally, we keep the minimum error found so far $E = \infty$ and the location $X = \infty$. For each $x \leq i$, assume that $P_w[x]$ belongs to group $\mathbf{g}_i$. We set $B[i] = B[i] + 1$ and we update $(\mathbf{g}_i, |\frac{B[i]}{|\mathbf{g}_i|/m} - 1|)$ in M. Let $(\mathbf{g}_j, \varepsilon_j)$ be the pair at the top of M. Let $e_x = \max\{T[x-1, j-1], \varepsilon_j\}$ be the discrepancy. If $e_x < E$ then we update $E = e_x$ and $X = x$. In the end of this process we return $T[n, m]$. By straightforward modifications, we can return the actual partition at the same time.

From the recursive relation and the implementation of finding $Error(P_w^{x,i})$, the algorithm we propose is correct. For the running time, we have $O(nm)$ cells in T, and in each cell we spend $O(n \log k)$ time to check if there is a valid partition. In total the algorithm runs in $O(n^2 m \log k)$ time. Hence, we prove the next theorem.

Theorem 6. *Let P_w be the input set of tuples in $\mathbb{R}$ in sorted order. There exists an algorithm that finds an $(\varepsilon^*, 1)$-hashmap in $O(n^2 m \log k)$ time, where ε^* is the minimum discrepancy of an m-sized partition in P_w.*

By slight modifications, we can also have the following result.

Corollary 1. *Let P_w be the projection of P onto w in sorted order and let ε be a parameter. There exists an algorithm that finds an $(\varepsilon, 1)$-hashmap, if there exists an ε-discrepancy partition of P_w, in $O(n^2 m)$ time.*

We run the algorithm we propose for every $w \in \mathcal{W}$ so we conclude with the next theorem.

Theorem 7. *Let P be a set of n tuples in $\mathbb{R}^d$. There exists an algorithm that finds an $(\varepsilon_D, 1)$-hashmap in $O(n^{d+2}m\log k)$ time, satisfying ε_D-approximation in collision probability and single fairness.*

Faster algorithm with near-optimum discrepancy.

While the previous algorithm will find an ε_D-unfair hashmap, where $\varepsilon_D \leq \varepsilon_R$ comparing to the algorithm proposed in Section 3 with optimum collision probability and single fairness, it is much slower. Next, we present an algorithm that returns a $(2(1+\delta)\varepsilon_D, 1)$-hashmap using possibly less than m buckets. Overall the hashmap returned by this algorithm is slightly more unfair, however the hashmap is found much faster. Again, we assume that we are given a $w \in \mathcal{W}$.

We need a simple tree-based data structure Φ with the following properties: i) Given P_w Φ is constructed in Φ in $O(n)$ time, ii) Φ has space $O(n)$, iii) given $l \in \mathbb{N}, r \in \mathbb{N}$ with $r > l$ and a group $\mathbf{g}_i$, $\Phi(i, l, r)$ returns the number of tuples in $P_w \cap \mathbf{g}_i$ with rank at least l and at most r, in $O(\log n)$ time, and iv) when two consecutive tuples in P_w swap, update Φ in $O(\log n)$ time.

In a high level, we guess the ε parameter and we try to construct buckets allowing at most $2(1+\varepsilon)\frac{|\mathbf{g}_i|}{m}$ and at least $(1-\varepsilon)\frac{|\mathbf{g}_i|}{m}$ tuples from each group $\mathbf{g}_i$. If such a partition with at most m buckets is possible then we repeat the process for a smaller parameter ε. On the other hand, if such partition is not possible, we repeat the process for a larger parameter ε.

We discretize the range $[0, 1]$ creating the discrete values of discrepancy $E = \{0, \frac{1}{n}, (1+\delta)\frac{1}{n}, (1+\delta)^2\frac{1}{n}, \ldots, 1\}$, where $|E| = O(\frac{\log n}{\delta})$. We run a binary search on E. In this way, we assume that we know the optimum ε^* up to a multiplicative error $(1+\delta)$. This is true because the smallest non-zero discrepancy is at least $\frac{m}{n} > \frac{1}{n}$. Let $\gamma \in E$ be the current parameter we check in the binary search.

Let x be the index of the leftmost tuple in P_w that have not covered by a bucket/interval. Initially, $x = 1$. We find the maximal bucket such that a group $\mathbf{g}_i$ contains exactly $2(1+\gamma)\frac{|\mathbf{g}_i|}{m}$ tuples. In particular, we find the right endpoint of the current bucket by running a binary search in the range $[x+1, n]$. Let y be the position we consider. For each group $\mathbf{g}_i$, we run $\Phi(i, x+1, y)$. If there exists a group $\mathbf{g}_\ell$ with more than $2(1+\gamma)\frac{|\mathbf{g}_\ell|}{m}$ tuples between $x+1$ and y we continue the binary search for values less than y. Otherwise we continue with values greater than y. When we find the position y, we set $x = y$ and we continue with constructing the next bucket. After constructing at most m buckets we check whether there are still tuples not assigned to a bucket. If yes, then we increase γ and continue the binary search in E. If not, then we decrease γ and we continue the binary search in E. We return the minimum error that we find a valid partition (and the partition itself) of at most m buckets.

Correctness. Let γ be the largest value such that $\gamma \geq \varepsilon^*$. It also holds that $\gamma \leq (1+\delta)\varepsilon^*$. We prove that in each bucket we place at most $2(1+\gamma)\frac{|\mathbf{g}_i|}{m}$ and at least $(1-\gamma)\frac{|\mathbf{g}_i|}{m}$ tuples for each group $\mathbf{g}_i$. If this is true then using the proof of Lemma 3 we show that we return a $(2(1+\delta)\varepsilon^*)$-unfair hashmap with collision probability at most $\frac{1+2(1+\delta)\varepsilon^*}{m}$ and single fairness in the range $[\frac{1-(1+\delta)\varepsilon^*}{m}, \frac{1+2(1+\delta)\varepsilon^*}{m}]$.

The upper bound $2(1+\gamma)\frac{|\mathbf{g}_i|}{m}$ for each group $\mathbf{g}_i$ for every bucket follows straightforwardly from the description of the algorithm. For the lower bound we argue as follows. In each iteration we construct a bucket that contains exactly $2(1+\gamma)\frac{|\mathbf{g}_i|}{m}$ tuples from one group $\mathbf{g}_i$. Any optimum solution contains at most $(1+\varepsilon^*)\frac{|\mathbf{g}_i|}{m}$ tuples in any bucket. Hence, the bucket we construct in each iteration fully contains at least one bucket from the optimum solution. From this argument, it straightforwardly follows that for each group $\mathbf{g}_\ell$ we have at least $(1-\gamma)\frac{|\mathbf{g}_\ell|}{m}$ tuples in every bucket we construct.

Running time. The set E contains $O(\frac{\log n}{\delta})$ elements so the binary search for the error γ takes $O(\log \frac{\log n}{\delta})$ iterations. In each iteration, we construct a bucket running $O(\log n)$ queries on data structure Φ for each group. Hence, the total running time of the algorithm is $O(\log(\frac{\log n}{\delta})mk\log^2 n)$.

Lemma 5. *Let P_w be the input set of tuples in $\mathbb{R}$ in sorted order, Φ a data structure for counting queries over P_w, and a parameter δ. There exists an algorithm that finds a $(2(1+\delta)\varepsilon^*, 1)$-hashmap with at most m buckets in $O(\log(\frac{\log n}{\delta})mk\log^2 n)$ time such that the collision probability is at most $\frac{1+2(1+\delta)\varepsilon^*}{m}$ and the single fairness is in the range $[\frac{1-(1+\delta)\varepsilon^*}{m}, \frac{1+2(1+\delta)\varepsilon^*}{m}]$. ε^* is the minimum discrepancy of an m-sized partition in P_w.*

After initializing Φ, we run the algorithm we propose for every $w \in \mathcal{W}$ so we conclude with the next theorem.

Theorem 8. *Let P be a set of n tuples in $\mathbb{R}^d$, and a parameter $\delta \in (0, 1)$. There exists an algorithm that finds a $(2(1+\delta)\varepsilon_D, 1)$-hashmap with at most m buckets in $O(n^d\log(\frac{\log n}{\delta})mk\log^2 n)$ time such that the collision probability is at most $\frac{1+2(1+\delta)\varepsilon_D}{m}$ and the single fairness is within $[\frac{1-(1+\delta)\varepsilon_D}{m}, \frac{1+2(1+\delta)\varepsilon_D}{m}]$.*

B.2 Faster randomized algorithm for small m

All algorithms proposed so far depend on n^d, which is too large for $d > 3$. Here we propose a faster randomized algorithm to return an $((1+\delta)\varepsilon_D+\gamma, 1)$-hashmap, in $O(n+\log(\frac{1}{\delta})\frac{1}{\gamma^{2d+4}}k^{d+2}m^{d+3}\log^{d+2} n)$ time, i.e., the running time depends linearly on n.

We use the notion of γ-approximation. Let w be a vector in $\mathbb{R}^d$, and let P_w be the projections of all points in P onto w. We also use P_w to denote the ordering of the points on w. Let $\mathcal{I}_w$ be the set of all possible intervals on the line supporting w. Let $(P_w, \mathcal{I}_w)$ be the range space with elements P_w and ranges $\mathcal{I}_w$. Let A_i be a random (multi-)subset of $\mathbf{g}_i$ of size $O(\gamma^{-2}\log\phi^{-1})$ where at each step a point from $\mathbf{g}_i$ is chosen uniformly at random (with replacement). It is well known [33, 53] that A_i is a γ-approximation of $(\mathbf{g}_i, \mathcal{I}_w)$ with probability at least $1-\phi$, satisfying

$$\left|\frac{|\mathbf{g}_i \cap I|}{|\mathbf{g}_i|} - \frac{|A_i \cap I|}{|A_i|}\right| \leq \gamma,$$

for every $I \in \mathcal{I}_w$.

As we had in the previous section, we discretize the range $[0, 1]$ creating the discrete values of discrepancy $E = \{0, \frac{1}{n}, (1+\delta)\frac{1}{n}, \ldots, 1\}$. We run a binary search on E. Let $\alpha \in E$ be the discrepancy we consider in the current iteration of the binary search. We design a randomized algorithm such that, if $\alpha > \varepsilon_D$ it returns a valid hashmap. Otherwise, it does not return a valid hashmap. The algorithm returns the correct answer with probability at least $1 - 1/n^{O(1)}$.

Algorithm. For each i, we get a sample $A_i \subseteq \mathbf{g}_i$ of size $O(\frac{m^2}{\gamma^2}\log n^{d+2})$. Let $A = \cup_i A_i$. Then we run the binary search we described in the previous paragraph. Let α be the discrepancy we currently check in the binary search. Let $\mathcal{W}_A$ be the set of all combinatorially different vectors with respect to A as defined in Section 3. For each $w \in \mathcal{W}_A$ we define the ordering A_w. Using Corollary **??** we search for a partition of m buckets on A_w such that every bucket contains $[(1-\alpha-\gamma/2)\frac{|A_i|}{m}, (1+\alpha+\gamma/2)\frac{|Ai|}{m}]$ for each $\mathbf{g}_i$. If it returns one such partition, then we continue the binary search for smaller values of α. Otherwise, we continue the binary search for larger values of α. In the end, we return the partition/hashmap of the last vector w that Corollary **??** returned a valid hashmap.

Analysis. Let w be a vector in $\mathbb{R}^d$ and the ordering P_w. Let $\mathcal{I}_w$ be the intervals defined on the line supporting the vector w. By definition, A_i is a $\gamma/(2m)$-approximation of $(P_w, \mathcal{I}_w)$ with probability at least $1 - 1/n^{d+2}$. Since there are $k < n$ groups and $O(n^d)$ combinatorially different vectors (with respect to P) using the union bound we get the next lemma.

Lemma 6. *A_i is a $\gamma/(2m)$-approximation in $(P_w, \mathcal{I}_w)$ for every $w \in \mathbb{R}^d$ and every $i \in [1,k]$ with probability at least $1 - 1/(2n)$.*

Let w^* be a vector in $\mathbb{R}^d$ such that there exists an ε_D-discrepancy partition of P_{w^*}. Let w' be the vector in $\mathcal{W}_A$ that belongs in the same cell of the arrangement of $\mathcal{A}(\Lambda_A)$ with w^*, where Λ_A is the set of dual hyperplanes of A. We notice that the ordering $A_{w'}$ is exactly the same with the ordering A_{w^*}.

By definition, there exists a partition of m buckets/intervals on P_{w^*} such that each interval contains $[(1-\varepsilon_D)\frac{|\mathbf{g}_i|}{m}, (1+\varepsilon_D)\frac{|\mathbf{g}_i|}{m}]$ items from group $\mathbf{g}_i$, for every group $\mathbf{g}_i$. Let $I_1, I_2, \dots I_m$ be the m intervals defining the ε_D-discrepancy partition. By the definition of A, it holds that

$$|\frac{|\mathbf{g}_i \cap I_j|}{|\mathbf{g}_i|} - \frac{|A_i \cap I_j|}{|A_i|}| \le \gamma/(2m) \Leftrightarrow$$

$$\frac{1-\varepsilon_D-\gamma/2}{m} \le \frac{|A_i \cap I_j|}{|A_i|}| \le \frac{1+\varepsilon_D+\gamma/2}{m}$$

for every $i \in [1,k]$ and $j \in [1,m]$ with probability at least $1 - 1/n^d$. Since, the ordering of A_{w*} is the same with the ordering of $A_{w'}$ it also holds that there are $I'_1, \dots, I'_m$ on the line supporting w' such that

$$\frac{1-\varepsilon_D-\gamma/2}{m} \le \frac{|A_i \cap I'_j|}{|A_i|}| \le \frac{1+\varepsilon_D+\gamma/2}{m}$$

with probability at least $1 - 1/n^d$. Let $\alpha \in E$ be the discrepancy we currently check in the binary search such that $\alpha > \varepsilon_D$. Since there exists a $(\varepsilon_D + \gamma/2)$-discrepancy partition in $A_{w'}$ it also holds that the algorithm from Corollary **??** executed with respect to vector w' will return a partition satisfying $[(1-\alpha-\gamma/2)\frac{|A_i|}{m}, (1+\alpha+\gamma/2)\frac{|Ai|}{m}]$ for each $\mathbf{g}_i$ with probability at least $1 - 1/n^d$.

Next, we show that any hashmap returned by our algorithm satisfies $(1+\delta)\varepsilon_D + \gamma$ discrepancy, with high probability. Let $\alpha \in E$ be the discrepancy in the current iteration of the binary search such that $\alpha \in [\varepsilon_D, (1+\delta)\varepsilon_D]$ (there is always such α in E). Let $\bar{w} \in \mathcal{W}_A$ be the vector such that our algorithm returns a valid partition with respect to $A_{\bar{w}}$ (as shown previously). Let $\bar{I}_1, \dots, \bar{I}_m$ be the intervals on the returned valid partition defined on the line supporting $\bar{w}$ such that

$$(1-\alpha-\gamma/2)\frac{|A_i|}{m} \le |\bar{I}_j \cap A_i| \le (1+\alpha+\gamma/2)\frac{|A_i|}{m}, \quad (2)$$

for each $\mathbf{g}_i$ and $j \in [1,m]$. Using Lemma **??**, we get that

$$|\frac{|\mathbf{g}_i \cap \bar{I}_j|}{|\mathbf{g}_i|} - \frac{|A_i \cap \bar{I}_j|}{|A_i|}| \le \gamma/(2m) \quad (3)$$

for every $i \in [1,k]$ and every $j \in [1,m]$ with probability at least $1 - 1/(2n)$. From Equation (**??**) and Equation (**??**) it follows that

$$\frac{1-(1+\delta)\varepsilon_D-\gamma}{m} \le \frac{|\mathbf{g}_i \cap \bar{I}_j|}{|\mathbf{g}_i|} \le \frac{1+(1+\delta)\varepsilon_D+\gamma}{m}.$$

for every $i \in [1,k]$ and $j \in [1,m]$ with probability at least $1 - 1/n$. The correctness of the algorithm follows.

We construct A in $O(\frac{km^2}{\gamma^2}\log n)$ time. Then the binary search runs for $O(\log\frac{\log n}{\delta})$ rounds. In each round of the binary search we spend $O(\frac{k^d m^{2d}}{\gamma^{2d}}\log^{d+1} n)$ time to construct $\mathcal{A}(\Lambda_A)$. The algorithm from Corollary **??** takes $O(\frac{k^2m^5}{\gamma^4}\log^2 n)$ time. Overall, the algorithm runs in $O(n + \log(\frac{\log n}{\delta})\frac{k^{d+2}m^{2d+5}}{\gamma^{2d+4}}\log^{d+2} n)$ time.

Theorem 9. *Let P be a set of n tuples in $\mathbb{R}^d$ and parameters δ, γ. There exists an $O(n + \log(\frac{\log n}{\delta})\frac{k^{d+2}m^{2d+5}}{\gamma^{2d+4}}\log^{d+2} n)$ time algorithm such that, with probability at least $1-\frac{1}{n}$ it returns an $((1+\delta)\varepsilon_D+\gamma, 1)$-hashmap $\mathcal{H}$ with collision probability at most $\frac{1+(1+\delta)\varepsilon_D+\gamma}{m}$ and single fairness in the range $[\frac{1-(1+\delta)\varepsilon_D-\gamma}{m}, \frac{1+(1+\delta)\varepsilon_D+\gamma}{m}]$.*

C MISSING PROOFS

Lemma 1. *While CDF-based hashmap satisfies collision probability, hence single fairness, it may not satisfy pairwise fairness.*

Proof: Let n_j be the number of tuples in the j-th bucket, and let $\alpha_{i,j}$ be the number of tuples of group $\mathbf{g}_i$ in the j-th bucket for every $i \le k$ and $j \le m$. From [63], it is always the case that each bucket contains the same number of tuples, i.e., $\frac{n_j}{n} = \frac{1}{m}$ for every $j = 1, \dots, m$. Hence, the collision probability is $\sum_{j=1}^m \left(\frac{n_j}{n}\right)^2 = \sum_{j=1}^m \frac{1}{m^2} = \frac{1}{m}$. Furthermore, the single fairness for each group $\mathbf{g}_i$ is $\sum_{j=1}^m \frac{\alpha_{i,j}}{|\mathbf{g}_i|} \cdot \frac{n_j}{n} = \frac{1}{m\cdot|\mathbf{g}_i|}\sum_{j=1}^m \alpha_{i,j} = \frac{1}{m}$. Hence, the CDF-based hashmap satisfies both the collision probability and the single fairness.

Next, we show that CDF-based hashmap does not always satisfy pairwise fairness using a counter-example. Let $k = 2$, $m = 2$, $|\mathbf{g}_1| = 3$, $|\mathbf{g}_2| = 3|$, and $n = 6$. Assume the 1-dimensional tuples $\{1,2,3,4,5,6\}$ where the first 3 of them belong to $\mathbf{g}_1$ and the last three belong to $\mathbf{g}_2$. The CDF-based hashmap will place the first three tuples into the first bucket and the last three tuples into the second bucket. By definition, the pairwise fairness is 1 (instead of 1/2) for both groups. □

Lemma 2. *In the binary demographic groups cases, where $\mathcal{G} = \{\mathbf{g}_1, \mathbf{g}_2\}$ and $r = |\mathbf{g}_1|$, the expected number of bins added by Sweep&Cut is bounded above by $2\left(\frac{r(n-r)}{n} + m\right)$.*

Proof: SWEEP&CUT adds at most $m-1$ boundaries between the neighboring pairs that both belong to the same group, simply because a boundary between neighboring pair can only happen when moving from one bucket to the next while there are m buckets.

In order to find the upper-bound on the number of bins added, in the following we compute the expected number of neighboring pairs from different groups. Consider the in the sorted list of points $\langle p_1, \cdots, p_n \rangle$ based on the attribute x. The probability that a point p_i belongs to group g_1 is

$$Pr_1 = Pr(\mathbf{g}(p_i) = \mathbf{g}_1) = \frac{r}{n}$$

Now consider two consecutive points p_i, p_{i+1}, in the list. The probability that these two belong to different groups is $2Pr_1(1 - Pr_1)$. Hence, the expected number of such pairs is

$$E[\text{pairs from diff. groups}] = \sum_{i=1}^{n-1} 2Pr_1(1-Pr_1) = 2(n-1)\frac{r}{n}(1-\frac{r}{n}) < \frac{2r(n-r)}{n}$$

Therefore,

$$E[\text{No. bins}] \leq E[\text{pairs from diff. groups}] + 2m < 2\Big(\frac{r(n-r)}{n} + m\Big)$$

□

Lemma 3. *Let $\mathcal{H}$ be a hashmap satisfying γ-discrepancy. Then $Cp \leq \frac{1+\gamma}{m}$, and $\frac{1-\gamma}{m} \leq Sp_i \leq \frac{1+\gamma}{m}$ and $Pr_i \leq \frac{1+\gamma}{m}$ for each group $\mathbf{g}_i$.*

Proof: Let n_j be the number of items in bucket j and let $\alpha_{i,j}$ be the number of items from group i in bucket j. Notice that $\sum_{j=1}^{m} \alpha_{i,j} = |\mathbf{g}_i|$ and $\sum_{j=1}^{m} n_j = n$. We have

$$\begin{aligned} Pr_i &= \sum_{j=1}^{m} \left(\frac{\alpha_{i,j}}{|\mathbf{g}_i|}\right)^2 = \sum_{j=1}^{m} \frac{\alpha_{i,j}}{|\mathbf{g}_i|} \cdot \frac{\alpha_{i,j}}{|\mathbf{g}_i|} \leq \sum_{j=1}^{m} \frac{(1+\gamma)\frac{\mathbf{g}_i}{m}}{|\mathbf{g}_i|} \cdot \frac{\alpha_{i,j}}{|\mathbf{g}_i|} \\ &= \frac{1+\gamma}{m} \sum_{j=1}^{m} \frac{\alpha_{i,j}}{|\mathbf{g}_i|} = \frac{1+\gamma}{m}. \end{aligned}$$

Similarly we show that

$$Cp = \sum_{j=1}^{m} \left(\frac{n_j}{n}\right)^2 \leq \frac{1+\gamma}{m}.$$

Using the same arguments it also holds that

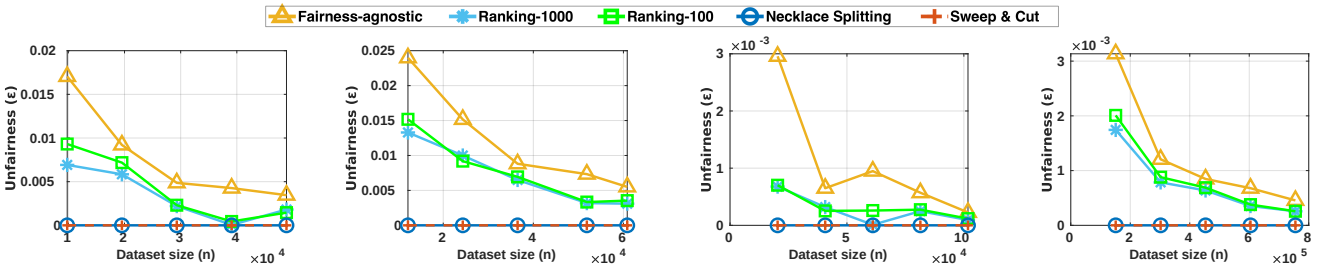
$$Sp_i = \sum_{j=1}^{m} \frac{\alpha_{i,j}}{|\mathbf{g}_i|} \frac{n_j}{n} \leq \frac{1+\gamma}{m}$$

and

$$Sp_i \geq \frac{1-\gamma}{m}.$$

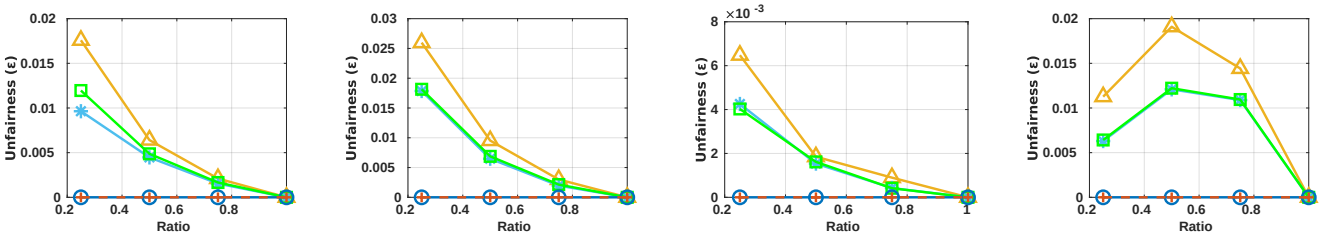
□

D EXTENDED EXPERIMENT RESULTS



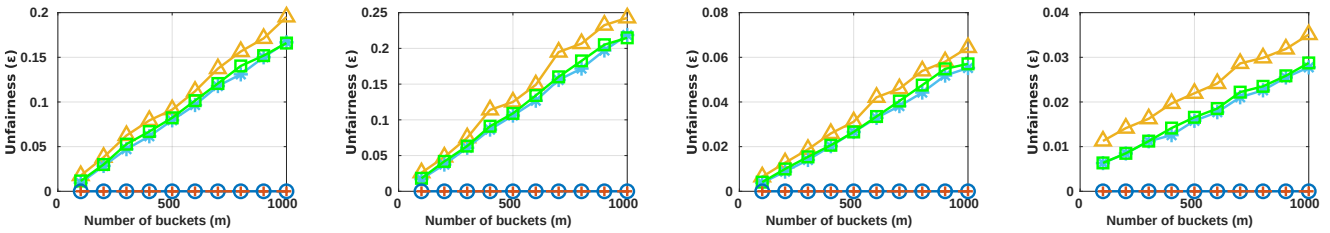
(a) Adult (b) Compas (c) Diabetes (d) ChicagoPop

Figure 34: Effect of varying dataset size n on unfairness



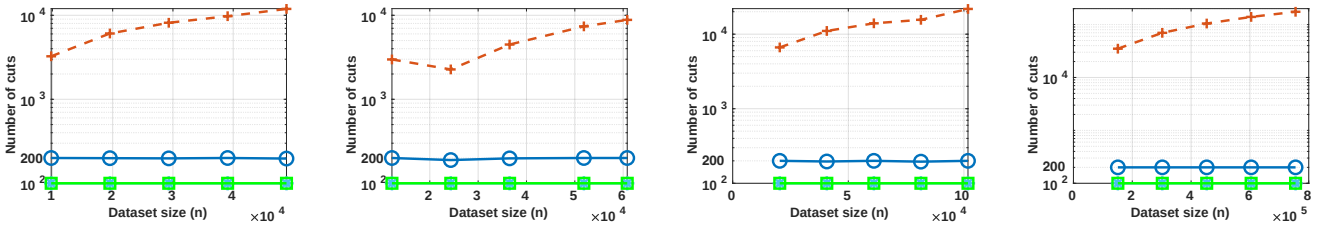
(a) Adult (b) Compas (c) Diabetes (d) ChicagoPop

Figure 35: Effect of varying majority to minority ratio on unfairness



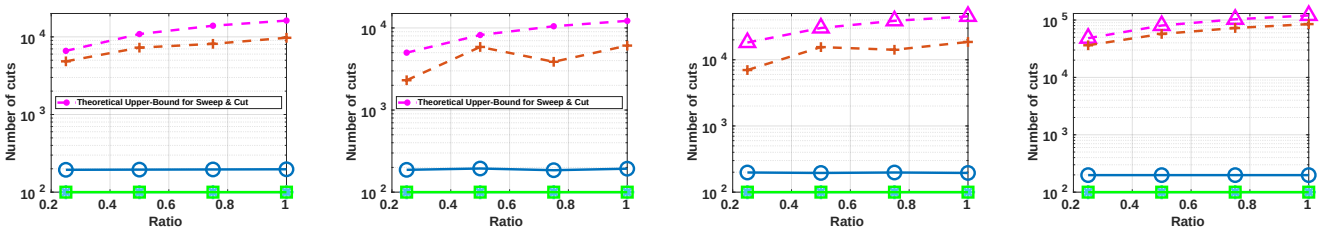
(a) Adult (b) Compas (c) Diabetes (d) ChicagoPop

Figure 36: Effect of varying number of buckets m on unfairness



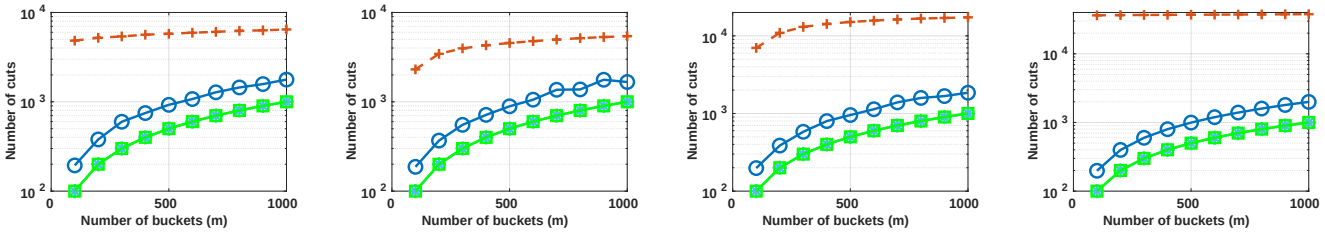
(a) Adult (b) Compas (c) Diabetes (d) ChicagoPop

Figure 37: Effect of varying dataset size n on space



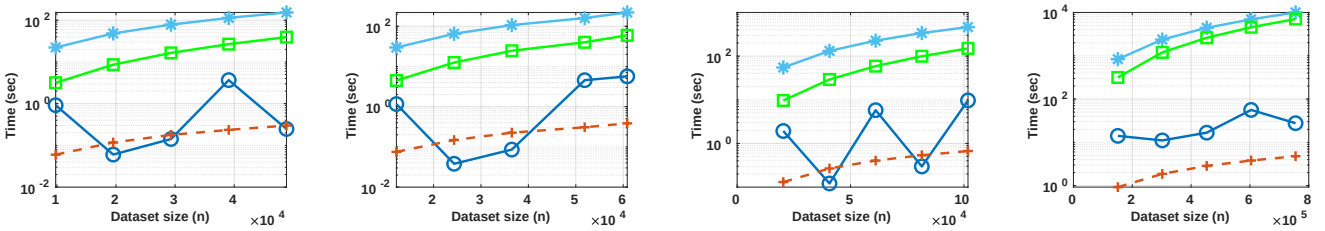
(a) Adult (b) Compas (c) Diabetes (d) ChicagoPop

Figure 38: Effect of varying majority to minority ratio on space



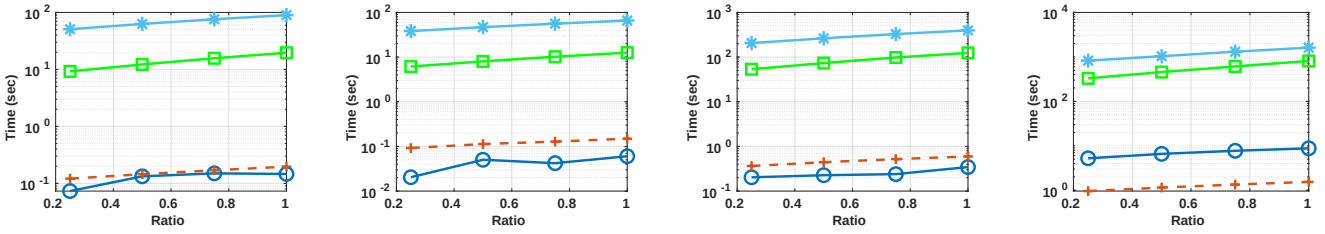
(a) Adult (b) Compas (c) Diabetes (d) ChicagoPop

Figure 39: Effect of varying number of buckets m on space



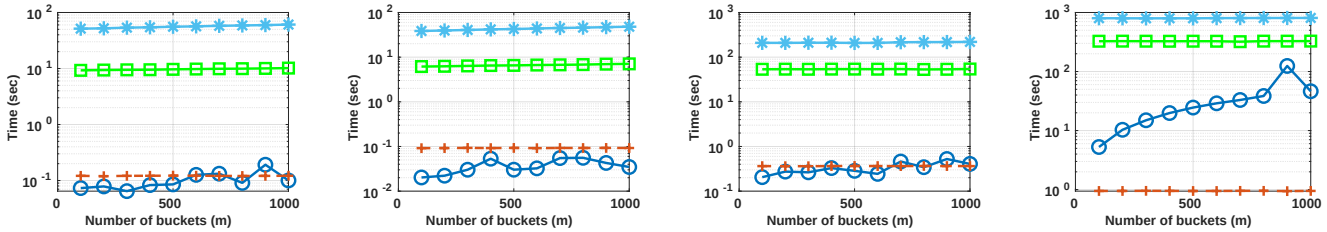
(a) Adult (b) Compas (c) Diabetes (d) ChicagoPop

Figure 40: Effect of varying dataset size n on preprocessing time



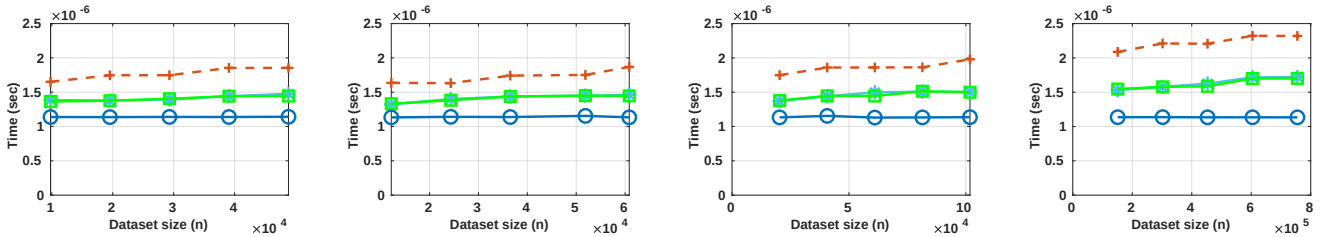
(a) Adult (b) Compas (c) Diabetes (d) ChicagoPop

Figure 41: Effect of varying majority to minority ratio on preprocessing time



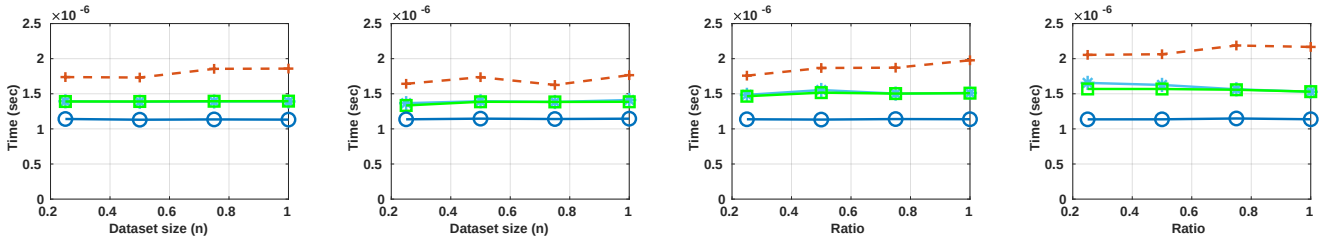
(a) Adult (b) Compas (c) Diabetes (d) ChicagoPop

Figure 42: Effect of varying number of buckets m on preprocessing time



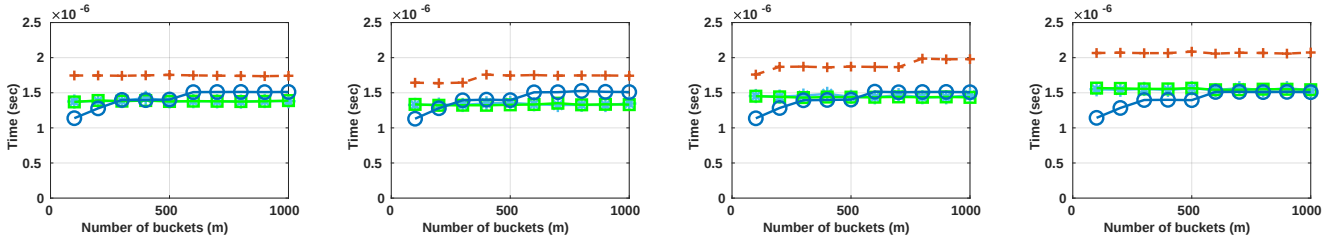
(a) Adult (b) Compas (c) Diabetes (d) ChicagoPop

Figure 43: Effect of varying dataset size n on query time



(a) Adult (b) Compas (c) Diabetes (d) ChicagoPop

Figure 44: Effect of varying majority to minority ratio on query time



(a) Adult (b) Compas (c) Diabetes (d) ChicagoPop

Figure 45: Effect of varying number of buckets m on query time

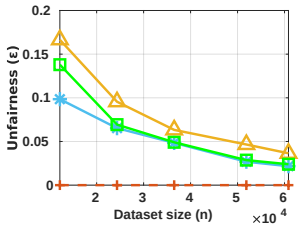


Figure 46: Effect of varying dataset size n on unfairness, COMPAS, race

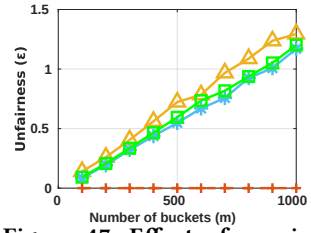


Figure 47: Effect of varying number of buckets m on unfairness, COMPAS, race

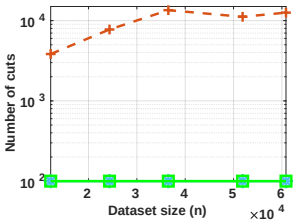


Figure 48: Effect of varying dataset size n on space, COMPAS, race

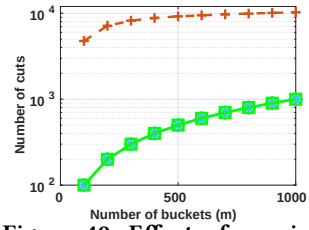


Figure 49: Effect of varying number of buckets m on space, COMPAS, race

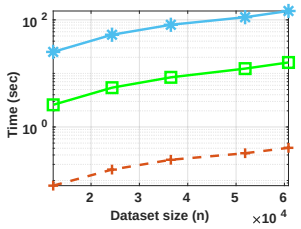


Figure 50: Effect of varying dataset size n on preprocessing time, COMPAS, race

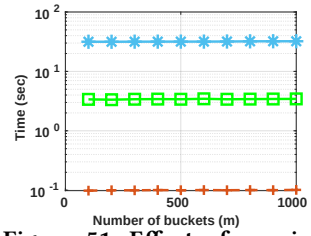


Figure 51: Effect of varying number of buckets m on preprocessing time, COMPAS, race

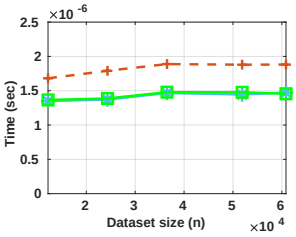


Figure 52: Effect of varying dataset size n on query time, COMPAS, race

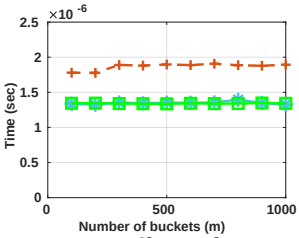


Figure 53: Effect of varying number of buckets m on query time, COMPAS, race

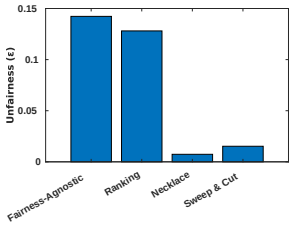


Figure 54: Learning setting: Unfairness evaluation over held out data, ADULT

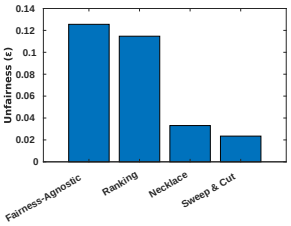


Figure 55: Learning setting: Unfairness evaluation over held out data, COMPAS

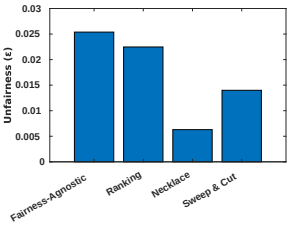


Figure 56: Learning setting: Unfairness evaluation over held out data, DIABETES

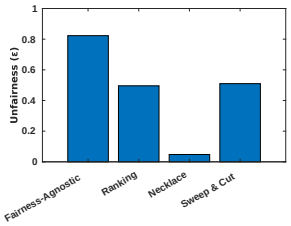


Figure 57: Learning setting: Unfairness evaluation over held out data, CHICAGOPOP