A NEW FRAMEWORK FOR FAIR k-MEDIAN: STATIC AND DYNAMIC

Anonymous authors
Paper under double-blind review

ABSTRACT

Concerns over bias in AI-driven decision-making have made fairness a prominent issue in machine learning. In this work, we study fair clustering under the k-median objective, following a popular notion of fairness suggested by Chierichetti et al. (2017), which requires each cluster to maintain a balanced representation of input classes. Building on their reduction, which transforms fair k-median into its vanilla counterpart via fairlet decompositions, we give a general framework that converts any polynomial-time fairlet decomposition into an almost-linear time algorithm with only a constant-factor loss in approximation. Through the extension of our framework, we obtain the first efficient dynamic algorithm for maintaining a fair decomposition under point insertions and deletions. We validate our algorithms on real-world datasets, demonstrating competitive performance against existing baselines.

1 INTRODUCTION

Clustering is a fundamental task in machine learning. One inherent problem in clustering is the presence of bias in the underlying data and the way it can be further amplified through downstream decisions. We consider a natural notion of fairness for clustering suggested in Chierichetti et al. (2017) under the k-median objective. Specifically, we are given a set of data points $P \subseteq \mathbb{R}^d$ and aim to find a partition of P into k subsets $C_1, ..., C_k$ together with corresponding centers $S = \{s_1, ..., s_k\} \subset P$, where the goal is to minimize the objective function $\sum_{i=1}^{k} \sum_{p \in C_i} \|p - s_i\|$. In the fair version of the problem, we are additionally given two parameters $r, b > 0$, as well as a partitioning of P into red and blue sets R, B. The (r, b)-*fair k-median* problem aims to find a clustering that minimizes the k-median objective, subject to the constraint that the *balance* of each cluster C_i, defined as $\min\{\frac{|R \cap C_i|}{|B \cap C_i|}, \frac{|B \cap C_i|}{|R \cap C_i|}\}$, is at least r/b.

In this setting, the colors in the fairness formulation frequently correspond to so-called sensitive (or protected) attributes. Such attributes arise naturally in fair clustering applications, and are particularly relevant when they represent characteristics such as gender or race, since ignoring the group fairness constraints may lead to undesirable outcomes, such as recommendations that reinforce demographic biases (Frisch et al., 2021; Peiretti & Pensa, 2025), underrepresentation of protected groups in student grouping and learner profiling (Quy et al., 2021; Song et al., 2024), targeted advertising that disproportionately affects certain groups Speicher et al. (2018); Ali et al. (2019), among others.

While there has been extensive work on designing fast algorithms for the vanilla k-median problem (see e.g., Jain & Vazirani (2001); Mettu & Plaxton (2003; 2004); Thorup (2004); Cohen-Addad et al. (2021); Jiang et al. (2026); Dupre la Tour & Saulpic (2026)), this fair variant has been comparatively less explored. In particular, Chierichetti et al. (2017) presented a computationally expensive quadratic-time algorithm achieving a constant-factor approximation for fair k-median clustering. Several follow-ups focus on improving the ratio, but at the cost of high-degree polynomial time, and/or slight violation of the fairness constraint Bera et al. (2019); Bercea et al. (2019); Wu et al. (2024a). The only known fast algorithm was presented by Backurs et al. (2019), which runs in *near-linear time*, albeit with significantly weaker $O(\log n)$-approximation. This naturally raises the question of how efficiently one can compute a fair k-median clustering while retaining strong approximation guarantees.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

On the other hand, given the ever-evolving nature of datasets, it is desirable to design efficient dynamic algorithms for fair clustering that outperform the baseline of recomputing the set of centers from scratch after every update, where updates consist of point insertions and deletions. While there is a substantial body of work on dynamic algorithms for the *fairness-oblivious* k-median problem Lattanzi & Vassilvitskii (2017); Henzinger & Kale (2020); Fichtenberger et al. (2021); Bateni et al. (2023); Cruciani et al. (2024); Goranci et al. (2026), Bhattacharya et al. (2025) no dynamic algorithms are known for its fair counterpart.

Our Contributions. To address this efficiency challenge for fair k-median, we focus on a notion called *fairlet decompositions* introduced by Chierichetti et al. (2017), which serves as a bridge to reduce fair k-median to vanilla k-median. In general, an (r, b)-fairlet decomposition is a clustering of the dataset into possibly many balanced clusters, each of size at most $r + b$. It is shown that, after collapsing each cluster into a weighted point, computing a *vanilla* k-median on this new dataset readily yields a solution for fair clustering Chierichetti et al. (2017).

Our main results are frameworks for efficiently constructing $(1, t)$-fairlet decompositions in both static and dynamic settings. In Theorem 1.1, we give a static framework that transforms any *polynomial-time* algorithm for (r, b)-fairlet decomposition into an algorithm that runs in *almost-linear* time. While conceptually similar frameworks for fair k-median have been studied before, particularly coresets Huang et al. (2019); Braverman et al. (2022); Bandyapadhyay et al. (2024); Huang et al. (2025), we emphasize that we allow k to be part of the input and coresets essentially reduce n to $\mathrm{poly}(k)$, which cannot lead to almost-linear time as in our guarantee, in the worst case when k is large.

Theorem 1.1 (Acceleration framework)**.** *Suppose there exists an algorithm that, given any weighted bi-colored instance consisting of n points in $\mathbb{R}^2$ with total weight W, computes an α-approximate (r, b)-fairlet decomposition in time $\tilde{O}(\mathrm{poly}(n) + W)$. Then, there exists a randomized algorithm that, given an unweighted bi-colored set $P \subseteq \mathbb{R}^2$ of n points, computes an (r, b)-fairlet decomposition in time $\tilde{O}(n^{1+O(\epsilon)})$ with expected approximation ratio $(\alpha(r + b))^{O(1/\varepsilon)}$.*

In particular, using the above theorem on top of a natural weighted generalization of the $(1, t)$-fairlet decomposition algorithm from Chierichetti et al. (2017), we obtain a constant-approximate $(1, t)$-fairlet decomposition in time almost-linear time.

The strength of our framework is also showcased in the dynamic setting. By subtly modifying the algorithm of Chierichetti et al. (2017) and carefully examining its behavior under updates when substituted into our framework, we obtain the following dynamic result.

Theorem 1.2 (Dynamic implementation)**.** *For points in $\mathbb{R}^2$ and a fixed integer parameter $t > 1$, there exists a randomized algorithm maintaining a $t^{O(1/\epsilon)}$ -approximate $(1, t)$-fair decomposition in amortized update time $O(n^{\epsilon})$ under point insertions and deletions.*

Finally, by combining the above results with existing static (e.g., Jain & Vazirani (2001)) and dynamic (e.g., Bhattacharya et al. (2025)) algorithms for k-median, we obtain the following two results for static and dynamic $(1, t)$-fair k-median.

Corollary 1.3. *For points in $\mathbb{R}^2$ and integer parameters $t > 1, k > 0$, there exist*

1. *a static $(1, t)$-fair k-median algorithm with running time $\tilde{O}(n^{1+\epsilon}/\epsilon)$, and*
2. *a dynamic $(1, t)$-fair k-median algorithm with amortized update time $\tilde{O}(n^{\epsilon}/\epsilon)$,*

both with expected approximation ratio $O_{\epsilon,t}(1)$.

Experiments. We validate our dynamic algorithm on standard real-world datasets: Bank (Moro et al., 2014), Adult (Becker & Kohavi, 1996), and Diabetes (Clore et al., 2014). Since we are not aware of any previous dynamic algorithm designed for fair clustering, we employ the original fairlet decomposition (Chierichetti et al., 2017) as well as a near-linear time fairlet decomposition (Backurs et al., 2019) as baselines. Since these baselines are static algorithms, we recompute their output from scratch after each update. We observe that our algorithm achieves a comparable objective value to these slow yet accurate algorithms, whereas offering ˜10x speedup.

1.1 TECHNICAL OVERVIEW

Static Framework. State-of-the-art approaches for constructing fairlet decompositions follow two distinct paradigms. Chierichetti et al. (2017) reduce fairlet decompositions to a graph min-cost flow problem, achieving strong approximation guarantees at the cost of enumerating all pairwise distances, which inherently requires quadratic running time. In contrast, Backurs et al. (2019) employ tree embeddings to obtain a compact metric representation, at the cost of a $\log n$ approximation loss, which is inherent even in $\mathbb{R}^2$.

To overcome the barrier on the construction time and approximation error of existing approaches, we employ a divide-and-conquer approach based on a quad-tree, similar to the approaches used for solving the Euclidean matching problem (see e.g. Agarwal & Varadarajan (2004); Indyk (2007)). The main idea is to reduce the problem to weighted sub-instances of size $n^{O(\epsilon)}$ while preserving sufficient structural information about the metric, for any $\epsilon > 0$. This allows us to avoid an approximation loss that depends on the input size.

More specifically, we construct a quad-tree via nested grids whose cell side lengths decrease by a factor of n^{ϵ} between consecutive levels. For each non-empty cell containing more than n^{ϵ} points, all points are shifted to the center of their respective sub-cells. This yields weighted sub-instances on $n^{O(\epsilon)}$ points, solvable efficiently using any (r, b)-fair decomposition algorithm with running time $O(\mathrm{poly}(n) + W)$, where W denotes the total weight. Points assigned to fairlets that do not cross sub-cell boundaries are recursively propagated to lower levels. Each point contributes once per level to both terms in the running time. Our approach crucially exploits the low dimensionality of the metric, as for such inputs the quad-tree consists of $O(1/\epsilon)$ layers, resulting in a total running time of $O(n^{O(\epsilon)}/\epsilon)$.

Dynamic Implementation. At a high level, our dynamic algorithm maintains the output of an instantiation of Theorem 1.1 with a natural weighted generalization of Chierichetti et al. (2017) plugged in. Namely, Chierichetti et al. (2017) showed that the fairlet decomposition problem corresponding to each cell C induces a minimum cost flow problem on a corresponding graph G_C. We show that a single point update in a cell can be handled by $O(t)$ path and cycle augmentations in the residual graph of G_C, resulting in $O(tn^{O(\epsilon)})$ update time per affected cell. Updates are then propagated top-down along the quad-tree.

The main challenge is to bound the number of affected cells during the propagation of an update. In general, the fairlet decomposition induced by the flow problem may change by $\Omega(n^{\epsilon})$ edges, potentially branching into $\mathrm{poly}(n)$ cells across $O(1/\epsilon)$ levels.

We address this problem by carefully perturbing the edge costs in the underlying flow graph G_C in each cell C in the quad-tree. Intuitively, this perturbation forces more structured augmenting paths/cycles in the residual graph of G_C, which then ensures that each local update causes only $O(t^2)$ point deletions in total across its children. Although a single point update may still cause many downward insertions, every insertion of a point into a level can be charged either to its original insertion into the input set or to an earlier deletion from that level. Hence, the deletion bound, applied across the $O(1/\epsilon)$ levels, suffices to obtain the desired amortized update-time guarantee.

1.2 RELATED WORK

Beyond the static and dynamic algorithms which we focus on, similar fair clustering problems have been studied in related computational models, such as distributed computing and streaming settings Schmidt et al. (2019); Bera et al. (2022); Wu et al. (2024b); Cohen-Addad et al. (2025). Apart from the group fairness introduced by Chierichetti et al. (2017), several other notions of fairness in clustering have emerged, such as social fairness Ghadiri et al. (2021); Makarychev & Vakilian (2021), individual fairness Jung et al. (2020); Mahabadi & Vakilian (2020); Negahbani & Chakrabarty (2021); Vakilian & Yalçiner (2022), representative fairness Kleindessner et al. (2019); Angelidakis et al. (2022), proportional fairness Chen et al. (2019); Micha & Shah (2020), and a number of other notions Abbasi et al. (2021); Chlamtác et al. (2022); Dai et al. (2022); Dickerson et al. (2023); Hotegni et al. (2023).

2 PRELIMINARIES

We present the main preliminaries here and defer the full details to Section A.

Let R be a set of red points and B a set of blue points in $\mathbb{R}^d$ for dimension $d \geq 2$, and let $P := R \cup B$. We denote by $d(x, y)$ the distance between two points $x, y \in \mathbb{R}^d$. Throughout this paper, we assume that the aspect ratio (i.e., the ratio between the maximum and minimum non-zero pairwise distances) of any point set of size n is bounded by $\text{poly}(n)$. Moreover, we assume that all input points lie within a bounding box of side length Δ, where $\Delta = \text{poly}(n)$.

We now define k-median clustering, set balance, and fair clustering.

Definition 2.1 (k-median)**.** *Given a set of points P, a k-median clustering is a partition of P into k subsets $C_1, ..., C_k$ together with the corresponding centers $S = \{s_1, ..., s_k\} \subset P$, such that the objective function $\sum_{i=1}^{k} \sum_{p \in C_i} d(p, s_i)$ is minimized.*

Definition 2.2 (Balance)**.** *For any $S \subseteq P = R \cup B$, balance$(S) = \min\{\frac{|R \cap S|}{|B \cap S|}, \frac{|B \cap S|}{|R \cap S|}\}$.*

Definition 2.3 (Fair k-median)**.** *Let $r, b > 0$ be parameters such that $r \leq b$, let P be a set of balance at least $\frac{r}{b}$. A fair (r, b)-clustering of P is a clustering of P such that each cluster has balance at least $\frac{r}{b}$. An optimal (r, b)-fair k-median clustering is a (r, b)-fair clustering with minimum k-median cost.*

For the construction of fair clusters, we will use a related notion of fair and fairlet decompositions, which clusters a set into (possibly many) balanced sets.

Definition 2.4 (Fair and fairlet decomposition)**.** *Given parameters $0 < r \leq b$, a* fair *decomposition of a bi-colored point set P is a partition of P into subsets, each having balance at least $\frac{r}{b}$, where each subset is associated with a center in $\mathbb{R}^d$ (which might not be part of the set). For the same input, a* fairlet *decomposition is a fair decomposition in which each subset has size at most $r + b$. The subsets in such a decomposition are referred to as* fairlets*.*

Throughout this paper, we also allow input points to have integer weights. Such instances can be interpreted as allowing multiple co-located copies of a point. We refer to the corresponding decomposition as the *weighted fair* (respectively *fairlet*) *decomposition*.

Finally, we use the following result from Chierichetti et al. (2017). Namely, they show that for any fair decomposition of a point set P, there exists a fairlet decomposition with no greater cost. This result yields a reduction from the fair k-median problem to the problem of finding a good fair decomposition of the input set. We defer the discussion of this reduction to Section B.

3 STATIC ACCELERATION FRAMEWORK

This section is devoted to the acceleration framework from Theorem 1.1 and its consequences. First, we present the formal version of Theorem 1.1 with the exact dependency on dimension d.

Theorem 3.1. *Suppose there exists an algorithm that, given any weighted bi-colored instance consisting of n points in $\mathbb{R}^d$ with total weight W, computes an α-approximate (r, b)-fairlet decomposition in time $\tilde{O}(\text{poly}(n) + W)$. Then, there exists a randomized algorithm that, given an unweighted bi-colored set $P \subseteq \mathbb{R}^d$ of n points, computes an (r, b)-fairlet decomposition of P in time $\tilde{O}(n^{1+d\epsilon})$ with expected approximation ratio $(\alpha d(r + b))^{O(1/\varepsilon)}$.*

Although weighted $(1, t)$-fairlet decomposition algorithms have not been explicitly considered in the literature, it follows as a natural generalization of the unweighted counterpart introduced by Chierichetti et al. (2017). For completeness, we provide the proof of the lemma below in Appendix C. Although we treat this result as a black box in this section, its internal structure will be important for the dynamic algorithm in Section 4.

Lemma 3.2. *Given sets A, B of red and blue points, respectively, of total weight W, there exists an algorithm for weighted t-approximate $(1, t)$-fairlet decomposition with running time $\tilde{O}(\text{poly}(n) + W)$.*

Plugging in the algorithm from Lemma 3.2 into Theorem 3.1, we obtain the following corollary.

Corollary 3.3. *Given sets A, B of red and blue points, respectively, there exists a randomized algorithm for $(1, t)$-fairlet decomposition with expected approximation ratio $O(dt^{O(1/\epsilon)})$ and run time $O(n^{1+d\epsilon})$.*

Finally, using the above result inside the fair k-median framework of Chierichetti et al. (2017) (see Appendix B) and the weighted k-median algorithm of Jiang et al. (2026), we obtain Corollary 1.3, part 1.

Now, we turn to describing the acceleration framework from Theorem 1.1. We give a high level overview of the algorithm, while deferring the detailed description and analysis to Appendix D.

Static Algorithm Description. First, assume algorithm WEIGHTEDFAIRLET is the algorithm satisfying the requirements of Theorem 3.1. Our static algorithm starts by putting all points in a randomly shifted bounding cell of side length 2Δ. The algorithm builds a hierarchy of nested grids, where each cell is partitioned into $n^{d\epsilon}$ sub-cells. We refer to this hierarchy as *quad-tree*, or simply *tree*. Then, we proceed to match points in this tree in a top-down manner, where we denote by P_C the set of points (both red and blue) belonging to cell C that have not been assigned at any level above C. First, if in cell C we have $|P_C| \leq n^{d\epsilon}$, we apply WEIGHTEDFAIRLET directly on set P_C to obtain a fairlet decomposition of set P_C. Otherwise, we proceed as follows.

1. Map points P_C in cell C to the centers of their respective sub-cells, treating red and blue points mapped to the same center as distinct points with their respective multiplicities. This induces a weighted set $\bar{P}_C$, to which we refer to as the *moved* set corresponding to P_C.
2. Invoke procedure WEIGHTEDFAIRLET on set $\bar{P}_C$, which then produces two types of fairlets: those spanning multiple sub-cells and those contained within a single sub-cell of C.
   - For any fairlet spanning multiple sub-cells, we greedily reconstruct a fairlet on its corresponding original points in P_C, while respecting the sub-cell boundaries. The assigned points are then removed.
   - For remaining points in P_C (i.e. those belonging to the second type of fairlets), the algorithm recurses on each non-empty sub-cell of C.

An example illustrating one layer of the recursion is shown in Figure 1.

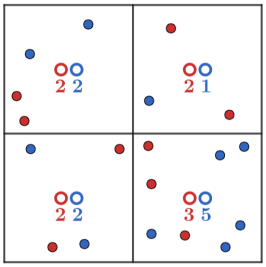
The above procedure always produces a valid fairlet decomposition. Indeed, notice that the point set P_C at each layer of the recursion remains balanced, since it is the union of balanced sets (fairlets) produced in its parent cell. We now bound the cost of the decomposition constructed by our procedure by comparing it with an optimal fairlet decomposition. Fix a fairlet F in this optimal solution and a tree level with sub-cell side length λ. Intuitively, the farther apart the points of F are, the more likely the randomly shifted grid is to separate them on that level, whereas closely grouped points are more likely to remain in the same sub-cell. If F is separated, its moved cost is at least λ. If F is not separated, its points collapse to a single center and its moved cost is zero, and are passed down to be resolved at finer grid levels.

4 DYNAMIC IMPLEMENTATION

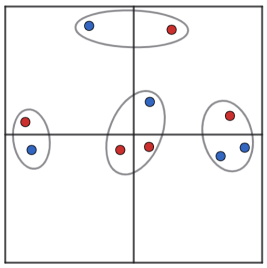
This section is devoted to the dynamic algorithm from Theorem 4.1 and its consequence, Corollary 1.3, part 2. First, we present the full version of Theorem 1.2, with the exact dependency on dimension d.

Theorem 4.1. *For points in $\mathbb{R}^d$ and a fixed integer parameter $t > 1$, there exists a randomized algorithm maintaining a $(dt)^{O(1/\epsilon)}$-approximate $(1, t)$-fair decomposition in amortized update time $\tilde{O}(n^{d\epsilon}t^{O(1/\epsilon)})$ under point insertions and deletions. If the current point set admits no $(1, t)$-fair decomposition, the algorithm instead reports* NULL.

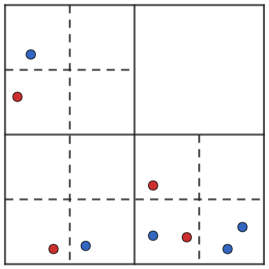
Corollary 1.3, part 2 is then obtained through combining the above result with the dynamic k-median algorithm of Bhattacharya et al. (2025). Now we turn to describing the algorithm from Theorem 4.1. We describe the algorithm on high level, deferring the detailed description and analysis to Appendix E.



(a) Filled points represent points P_C belonging to cell C, while hollow points represent the set $\bar{P}_C$, i.e. the sub-cell centers to which these points are moved, together with their multiplicities. WEIGHTEDFAIRLET is called on $\bar{P}_C$.



(b) Fairlets spanning multiple sub-cells are fixed in the output and not propagated downards.



(c) The algorithm recurses on three non-empty sub-cells of C.

Figure 1: Procedure RECURSIVEFAIRLET on one layer of the recursion

Dynamic Algorithm. The high level idea behind the dynamic algorithm is to maintain the output of the static algorithm of Section 3 throughout the dynamic updates in a top-down manner, i.e. maintain the fairlet decomposition in each cell of the tree, together with the sets of red and blue points that need to be propagated downward. Intuitively, a point update (insertion or deletion) at a cell C might cause fairlets inside C to be deleted or inserted, resulting in $O(t)$ point updates (i.e. *recourse*) at sub-cells of C per such fairlet. These updates may then trigger recursive updates at lower levels of the tree.

Main challenge: bounding the recourse. One natural approach to achieve the above is to recompute the fairlet decomposition in each affected cell via WEIGHTEDFAIRLET procedure (see Lemma 3.2) in a top-down manner. The challenge in this approach is that recomputation of fairlet decomposition in a single cell can result in many (even poly n) total updates in cells on lower levels, which would then result in large overall update time. Hence, a more sophisticated approach is needed, making sure that the number of changes to the fairlet decomposition (and hence, the recourse) is bounded.

To address this challenge, instead of recomputing the minimum cost solution from scratch, consider how the corresponding weighted fairlet decomposition problems before and after an update differ, and apply only careful structural changes to the previous decomposition. However, to better understand how one update affects the fairlet decomposition in each cell, we need to open the black-box algorithm for WEIGHTEDFAIRLET used in the static algorithm in Section 3. The detailed description of WEIGHTEDFAIRLET procedure can be found in Appendix C.

WEIGHTEDFAIRLET procedure. Before we describe the general procedure for weighted $(1, t)$-fairlet decomposition, notice that an optimal $(1, 1)$-fairlet decomposition is simply a minimum-cost perfect matching between red and blue points. This can be modeled as a min-cost flow in the following standard way. Specifically, each blue point supplies one unit, each red point demands one unit, while the cost of a red-blue pair is the distance between its endpoints. For weighted points, these supplies and demands simply become their multiplicities. Finally, for weighted $(1, t)$-fairlet decomposition, we relax the matching requirement by allowing each point p to have between $w(p)$ and $tw(p)$ incident neighbors of the opposite color. To achieve this, we split each point into two vertices joined by a zero-cost edge of capacity $tw(p)$.

However, obtaining the fairlet decomposition is more involved than in the case of $(1, 1)$-fairlet decomposition. Specifically, the optimal flow does not correspond to the optimal fairlet decomposition, but an optimal solution to a related problem of *minimum cost t-bounded edge cover*. A t-bounded edge cover is a subset E_t of red-blue edges of G such that $w(p) \leq \deg_{E_t}(p) \leq t \cdot w(p)$ for all copies of the original red and blue points p. Finally, after pruning redundant edges, it is not hard to see that this edge cover induces a t-approximate $(1, t)$-fair decomposition of the input set.

We now give a high-level description of POINT-INSERTION and POINT-DELETION, the point insertion and deletion procedures at a cell, respectively. The detailed description and analysis of these procedures can be found in Section E.

Point insertion in a cell. We now describe the process of a red point p insertion into a specific cell C. The construction is symmetric with respect to blue point insertions. We maintain the flow network instance G_C associated with C. Recall that G_C is the flow network corresponding to the $(1, t)$-fairlet decomposition of set $\bar{P}_C$, the moved set corresponding to set P_C.

Assume p falls in sub-cell C_i of C and let the corresponding center in G_C be represented by vertex r_{C_i}. Procedure POINT-INSERTION consists of two parts:

1. updating the corresponding minimum-cost flow solution, and
2. updating the underlying fair decomposition.

For the first part, an insertion of point p first increases the capacity of the edge connecting two copies representing r_{C_i} by t in G_C, handled through t unit increments and at most t residual cycle augmentations. Then, it also increases the demand of r_{C_i} and the supply of the auxiliary source of G_C by one, handled by one shortest-path augmentation in the residual graph of G_C. The following lemma shows the correctness of the procedure.

Lemma 4.2. *Let G_C be the flow network associated with point set $\bar{P}_C$ in cell C, and let f be a minimum cost flow in graph G_C. Consider an insertion of red point p into P_C, and let G'_C be the flow network corresponding to the updated set $\bar{P}_C$. A minimum cost flow f' in G'_C can be obtained by performing at most $t + 1$ flow augmentations to f.*

After obtaining the flow solution, we can derive the corresponding t-bounded edge cover of P_C. However, this determines only the number of points of each color from each sub-cell that are assigned in cell C, but not which points are assigned. The second part of POINT-INSERTION procedure then greedily selects which points are propagated to the children cells of C, while a $(1, t)$-fair decomposition is constructed on the remaining points in C.

Deletions of points from a cell are handled in a similar manner. Intuitively, at every edge capacity update, we cancel the flow along the corresponding cycle in residual graph of G, while the supply/demand update cancels the flow along the corresponding path in this graph.

Updating the tree. Insertion (respectively deletion) of a single point in the tree then starts by inserting (resp. deleting) the point to the root cell via POINT-INSERTION (resp. POINT-DELETION). The second part of the procedure computes which points from the children sub-cells are inserted or deleted, and the algorithm recurses on the affected child cells.

Perturbing the underlying graph. Although the above updates are structured, they are unfortunately not enough to solve our main problem of minimizing the solution recourse after each update. Hence, we slightly modify the underlying flow networks G_C for all cells C in the tree in the following way. Let λ be the side length of the child cells of C. We increase the cost of each red-blue edge by λ/n^3 when its endpoints represent the same sub-cell (i.e. when its cost is zero), and by $2\lambda/n^3$ otherwise, while all other edge costs remain unchanged. Intuitively, this change aims to keep the corresponding t-bounded edge cover stable, in the sense that the number of edges connecting the collocated red and blue points that are deleted is small. In other words, the following key claim bounds the number of point deletions in sub-cells a single update in a cell can produce.

Claim 4.3. *Let R_0^-, B_0^- and R_0^+, B_0^+ be the number of red and blue points pushed down for all sub-cells of the updated cell before and after an update is processed. Then* $\sum \max\{R_0^-[i] - R_0^+[i], 0\} + \max\{B_0^-[i] - B_0^+[i], 0\} = O(t^2)$.

Table 1: Datasets and sliding-window parameters.

dataset	size	d	window size (w)	step length	k	sensitive attribute	t
Bank	4,522	3	3,000	5	150	marital	2
Adult	48,842	6	5,000	50	250	gender	3
Diabetes	101,763	3	10,000	50	500	gender	2

Although the above claim bounds only the number of deletions incurring after a single point update, it will be enough to obtain the run time guarantees stated in Theorem 4.1. Moreover, this perturbation does not significantly increase the total cost of the fair decomposition. Specifically, the cost increases only by a factor of t at each level, resulting in a total increase of at most $(dt)^{O(1/\epsilon)}$ across all $O(1/\epsilon)$ levels of the tree.

Handling Unbalanced Sets in the Dynamic Stream. Notice that the above update procedures assume that the input set remains (r, b)-balanced after each update. We now describe how to obtain the algorithm stated in Theorem 4.1, in particular, how to handle data streams in which the input set may become unbalanced. We defer the detailed description to Appendix F.

Starting from an empty input set P, we report NULL until P first becomes balanced. At that point, we invoke the static algorithm from Section 3. If P subsequently becomes unbalanced, we report NULL and maintain two queues L_1 and L_2. The former contains red point insertions and blue point deletions, while the latter contains all remaining updates. We continue updating these two queues until P becomes balanced again, reporting NULL in the meantime. Once P is balanced again, we process the queued updates in a greedy fashion, in an order that keeps P balanced at all times.

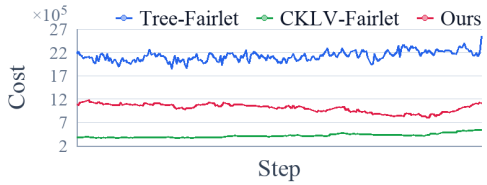
5 Experiments

We validate our new *dynamic* algorithm in our experiments, focusing on evaluating its running time and accuracy performance, compared with two baselines on three datasets. Overall, our algorithm runs ˜10x faster while maintaining comparable accuracy as in the baselines.

Baselines. We are not aware of dynamic baselines for fair clustering, hence we employ slow yet accurate static fairlet decomposition algorithms (Chierichetti et al., 2017; Backurs et al., 2019) as baselines, which are re-run after every update. We denote the baseline of Chierichetti et al. (2017) as "CKLV-Fairlet" and the tree-embedding based method from Backurs et al. (2019) as "Tree-Fairlet". For all baselines and our algorithm, once the fairlet is maintained, we employ the same downstream dynamic algorithm for (vanilla) k-median, which is based on a dynamic tree embedding method as in Goranci et al. (2026), to maintain the final solution.

Datasets and Experiment Setup. We employ three real-world datasets: Bank (Moro et al., 2014), Adult (Becker & Kohavi, 1996), and Diabetes (Clore et al., 2014), all of which have also been widely used in previous studies on (fair) clustering (Chierichetti et al., 2017; Backurs et al., 2019; Cohen-Addad et al., 2025). For each dataset, we extract numerical features to construct a vector in $\mathbb{R}^d$ for each record, and we also use a binary sensitive attribute. Since these datasets do not contain temporal update information, we simulate dynamic insertions and deletions using a sliding-window over the data stream (with a window size w and a sliding step length > 1). For each dataset, we set the number of centers to $k = w/20$ and compute the global balance(P), i.e., the minority-to-majority ratio induced by the binary sensitive attribute. The parameter t for the $(1, t)$-fair decomposition uses this balance. The experiment parameters are summarized in Table 1.

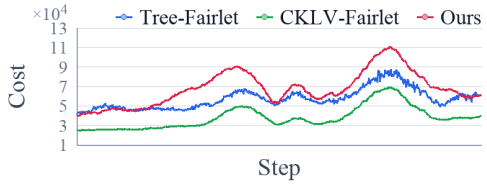
Implementation Details. All baselines and our algorithm are implemented in Python. On each update, after the fairlet decomposition is maintained, we run a tree-embedding based dynamic k-median algorithm outlined in Goranci et al. (2026) (where we use a simple randomly-shifted quadtree for the tree embedding (Indyk, 2004)) as a downstream algorithm, to dynamically maintain the final clustering solution. We use the published research code for Tree-Fairlet, and we implement CKLV-Fairlet with the help of a min-cost flow solver in OR-Tools (Perron & Furnon, 2026). For



(a) Bank

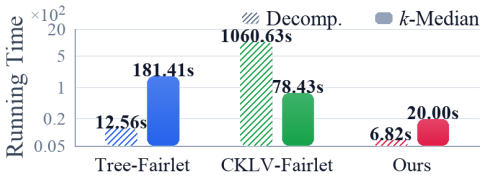


(b) Adult

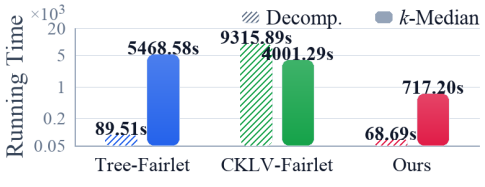


(c) Diabetes

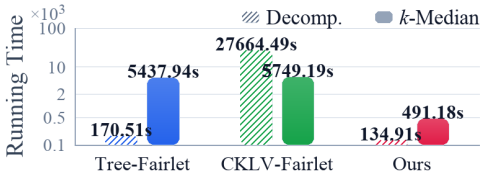
Figure 2: Cost evaluation.



(a) Bank



(b) Adult



(c) Diabetes

Figure 3: Running time evaluation (total over all sliding windows).

our dynamic fairlet decomposition, we mostly follow the algorithms described in Section 4, with the following key details. a) To compute the single-source shortest path in the residual graph, we transform the graph so that the weights are nonnegative using a trick as in e.g., Edmonds & Karp (1972), and then employ Dijkstra's algorithm. However, it could be that after some update new negative edges emerge, then we re-compute the transformed graph using the Bellman-Ford algorithm. These shortest-path optimizations generate the same output as the original algorithm and do not change its worst-case running-time bound, thus preserving its theoretical guarantees. b) In the nested-grid construction, we choose the parameter $\epsilon = 0.1$ to balance the granularity of the subdivision of grid cells and the error. All experiments are run on a MacBook Air 15.3 with an Apple M3 chip (8 cores, 2.22 GHz), 16GB RAM, and macOS 14.4.1 (23E224).

Results. We report the clustering cost over the sliding windows in Figure 2. This is done by reporting the sum of the fairlet decomposition cost, which accounts for assigning each original point to the medoid of its fairlet, and the (weighted) k-median cost, which accounts for clustering the fairlet representatives. This total cost upper bounds the cost of the clustering on the original points. On Adult and Bank, our method consistently achieves much lower cost than Tree-Fairlet, while CKLV-Fairlet gives the lowest cost overall, which is expected since it runs a very heavy yet accurate min-cost flow solver. On Diabetes, our method has a slightly higher cost than the two baselines on some part of the stream, but is generally comparable. Figure 3 reports the total running time over all sliding windows for fairlet decomposition and dynamic k-median maintenance. It can be observed that the downstream dynamic k-median part (solid bars in the figure) has varying running time, even though we use the same dynamic k-median algorithm for all three methods. The main difference is that, different fairlet decompositions introduce different sequences of updates to the downstream, and it is our dynamic fairlet decomposition that changes only slightly after each insertion or deletion of a data point, that results in small updates to the downstream algorithm and substantially reducing its total running time. This leads to ˜10x faster than both baselines. In conclusion, our algorithm achieves significantly shorter running time while achieving comparable clustering cost.

REFERENCES

Mohsen Abbasi, Aditya Bhaskara, and Suresh Venkatasubramanian. Fair clustering via equitable group representations. In *FAccT*, pp. 504–514. ACM, 2021.

Pankaj K. Agarwal and Kasturi R. Varadarajan. A near-linear constant-factor approximation for euclidean bipartite matching? In Jack Snoeyink and Jean-Daniel Boissonnat (eds.), *Proceedings of the 20th ACM Symposium on Computational Geometry, Brooklyn, New York, USA, June 8-11, 2004*, pp. 247–252. ACM, 2004. doi: 10.1145/997817.997856. URL https://doi.org/10.1145/997817.997856.

Muhammad Ali, Piotr Sapiezynski, Miranda Bogen, Aleksandra Korolova, Alan Mislove, and Aaron Rieke. Discrimination through optimization: How facebook's ad delivery can lead to biased outcomes. *Proceedings of the ACM on human-computer interaction*, 3(CSCW):1–30, 2019.

Haris Angelidakis, Adam Kurpisz, Leon Sering, and Rico Zenklusen. Fair and fast k-center clustering for data summarization. In *ICML*, Proceedings of Machine Learning Research, pp. 669–702. PMLR, 2022.

Arturs Backurs, Piotr Indyk, Krzysztof Onak, Baruch Schieber, Ali Vakilian, and Tal Wagner. Scalable fair clustering. In Kamalika Chaudhuri and Ruslan Salakhutdinov (eds.), *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, Proceedings of Machine Learning Research, pp. 405–413. PMLR, 2019. URL http://proceedings.mlr.press/v97/backurs19a.html.

Sayan Bandyapadhyay, Fedor V. Fomin, and Kirill Simonov. On coresets for fair clustering in metric and euclidean spaces and their applications. *J. Comput. Syst. Sci.*, 142:103506, 2024.

MohammadHossein Bateni, Hossein Esfandiari, Hendrik Fichtenberger, Monika Henzinger, Rajesh Jayaram, Vahab Mirrokni, and Andreas Wiese. Optimal fully dynamic *k*-center clustering for adaptive and oblivious adversaries. In *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pp. 2677–2727, 2023.

Barry Becker and Ronny Kohavi. Adult. UCI Machine Learning Repository, 1996. URL https://doi.org/10.24432/C5XW20. [Dataset].

Suman K Bera, Syamantak Das, Sainyam Galhotra, and Sagar Sudhir Kale. Fair k-center clustering in mapreduce and streaming settings. In *Proceedings of the ACM Web Conference 2022*, pp. 1414–1422, 2022.

Suman Kalyan Bera, Deeparnab Chakrabarty, Nicolas Flores, and Maryam Negahbani. Fair algorithms for clustering. In *NeurIPS*, pp. 4955–4966, 2019.

Ioana Oriana Bercea, Martin Groß, Samir Khuller, Aounon Kumar, Clemens Rösner, Daniel R. Schmidt, and Melanie Schmidt. On the cost of essentially fair clusterings. In *APPROX-RANDOM*, LIPIcs, pp. 18:1–18:22. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

Sayan Bhattacharya, Martín Costa, Ermiya Farokhnejad, Shaofeng H-C Jiang, Yaonan Jin, and Jianing Lou. Fully dynamic euclidean k-means. *arXiv preprint arXiv:2507.11256*, 2025.

Vladimir Braverman, Vincent Cohen-Addad, Shaofeng H.-C. Jiang, Robert Krauthgamer, Chris Schwiegelshohn, Mads Bech Toftrup, and Xuan Wu. The power of uniform sampling for coresets. In *FOCS*, pp. 462–473. IEEE, 2022.

Li Chen, Rasmus Kyng, Yang P. Liu, Richard Peng, Maximilian Probst Gutenberg, and Sushant Sachdeva. Maximum flow and minimum-cost flow in almost-linear time. *J. ACM*, 72(3):19:1–19:103, 2025. doi: 10.1145/3728631. URL https://doi.org/10.1145/3728631.

Xingyu Chen, Brandon Fain, Liang Lyu, and Kamesh Munagala. Proportionally fair clustering. In *ICML*, Proceedings of Machine Learning Research, pp. 1032–1041. PMLR, 2019.

Flavio Chierichetti, Ravi Kumar, Silvio Lattanzi, and Sergei Vassilvitskii. Fair clustering through fairlets. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett (eds.), *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pp. 5029–5037, 2017. URL https://proceedings.neurips.cc/paper/2017/hash/978fce5bcc4eccc88ad48ce3914124a2-Abstract.html.

Eden Chlamtác, Yury Makarychev, and Ali Vakilian. Approximating fair clustering with cascaded norm objectives. In *SODA*, pp. 2664–2683. SIAM, 2022.

John Clore, Krzysztof Cios, Jon DeShazo, and Beata Strack. Diabetes 130-US Hospitals for Years 1999-2008. UCI Machine Learning Repository, 2014. URL https://doi.org/10.24432/C5230J. [Dataset].

Vincent Cohen-Addad, Andreas Emil Feldmann, and David Saulpic. Near-linear time approximation schemes for clustering in doubling metrics. *J. ACM*, 68(6):44:1–44:34, 2021.

Vincent Cohen-Addad, Shaofeng H.-C. Jiang, Qiaoyuan Yang, Yubo Zhang, and Samson Zhou. Fair clustering in the sliding window model. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025. URL https://openreview.net/forum?id=VGQugiuCQs.

Emilio Cruciani, Sebastian Forster, Gramoz Goranci, Yasamin Nazari, and Antonis Skarlatos. Dynamic algorithms for k-center on graphs. In *Proceedings of the 2024 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pp. 3441–3462. SIAM, 2024.

Zhen Dai, Yury Makarychev, and Ali Vakilian. Fair representation clustering with several protected classes. In *FAccT*, pp. 814–823. ACM, 2022.

John Dickerson, Seyed Esmaeili, Jamie H Morgenstern, and Claire Jie Zhang. Doubly constrained fair clustering. *Advances in Neural Information Processing Systems*, 36:13267–13293, 2023.

Max Dupre la Tour and David Saulpic. Faster and simpler greedy algorithm for k-median and k-means. In *ICALP*, 2026. To appear.

Jack Edmonds and Richard M. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *J. ACM*, 19(2):248–264, 1972.

Hendrik Fichtenberger, Silvio Lattanzi, Ashkan Norouzi-Fard, and Ola Svensson. Consistent k-clustering for general metrics. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pp. 2660–2678, 2021.

Gabriel Frisch, Jean-Benoist Leger, and Yves Grandvalet. Co-clustering for fair recommendation. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pp. 607–630. Springer, 2021.

Mehrdad Ghadiri, Samira Samadi, and Santosh S. Vempala. Socially fair k-means clustering. In *FAccT*, pp. 438–448. ACM, 2021.

Andrew V. Goldberg and Robert Endre Tarjan. Solving minimum-cost flow problems by successive approximation. In Alfred V. Aho (ed.), *Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, New York, New York, USA*, pp. 7–18. ACM, 1987. doi: 10.1145/28395.28397. URL https://doi.org/10.1145/28395.28397.

Gramoz Goranci, Shaofeng H.-C. Jiang, Peter Kiss, Qihao Kong, Yi Qian, and Eva Szilagyi. Tree embedding in high dimensions: Dynamic and massively parallel. In Kasper Green Larsen and Barna Saha (eds.), *Proceedings of the 2026 Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2026, Vancouver, BC, Canada, January 11-14, 2026*, pp. 1181–1213. SIAM, 2026. doi: 10.1137/1.9781611978971.46. URL https://doi.org/10.1137/1.9781611978971.46.

Monika Henzinger and Sagar Kale. Fully-dynamic coresets. In Fabrizio Grandoni, Grzegorz Herman, and Peter Sanders (eds.), *28th Annual European Symposium on Algorithms, ESA 2020, Pisa, Italy (Virtual Conference), September 7-9, 2020*, volume 173 of *LIPIcs*, pp. 57:1–57:21. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi: 10.4230/LIPICS.ESA.2020.57. URL https://doi.org/10.4230/LIPIcs.ESA.2020.57.

Sèdjro Salomon Hotegni, Sepideh Mahabadi, and Ali Vakilian. Approximation algorithms for fair range clustering. In *ICML*, Proceedings of Machine Learning Research, pp. 13270–13284. PMLR, 2023.

Lingxiao Huang, Shaofeng H.-C. Jiang, and Nisheeth K. Vishnoi. Coresets for clustering with fairness constraints. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett (eds.), *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pp. 7587–7598, 2019. URL https://proceedings.neurips.cc/paper/2019/hash/810dfbbebb17302018ae903e9cb7a483-Abstract.html.

Lingxiao Huang, Jian Li, Pinyan Lu, and Xuan Wu. Coresets for constrained clustering: General assignment constraints and improved size bounds. In *SODA*, pp. 4732–4782. SIAM, 2025.

Piotr Indyk. Algorithms for dynamic geometric problems over data streams. In *STOC*, pp. 373–380. ACM, 2004.

Piotr Indyk. A near linear time constant factor approximation for euclidean bichromatic matching (cost). In Nikhil Bansal, Kirk Pruhs, and Clifford Stein (eds.), *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2007, New Orleans, Louisiana, USA, January 7-9, 2007*, pp. 39–42. SIAM, 2007. URL http://dl.acm.org/citation.cfm?id=1283383.1283388.

Kamal Jain and Vijay V. Vazirani. Approximation algorithms for metric facility location and k-median problems using the primal-dual schema and lagrangian relaxation. *J. ACM*, 48(2):274–296, 2001.

Shaofeng H-C Jiang, Yaonan Jin, Jianing Lou, and Pinyan Lu. Local search for clustering in almost-linear time. In *Proceedings of the 2026 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pp. 5960–5977. SIAM, 2026.

Christopher Jung, Sampath Kannan, and Neil Lutz. Service in your neighborhood: Fairness in center location. In *FORC*, LIPIcs, pp. 5:1–5:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

Matthäus Kleindessner, Pranjal Awasthi, and Jamie Morgenstern. Fair k-center clustering for data summarization. In *International Conference on Machine Learning*, pp. 3448–3457. PMLR, 2019.

Silvio Lattanzi and Sergei Vassilvitskii. Consistent k-clustering. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*, volume 70 of *Proceedings of Machine Learning Research*, pp. 1975–1984. PMLR, 2017.

Sepideh Mahabadi and Ali Vakilian. Individual fairness for k-clustering. In *ICML*, Proceedings of Machine Learning Research, pp. 6586–6596. PMLR, 2020.

Yury Makarychev and Ali Vakilian. Approximation algorithms for socially fair clustering. In *COLT*, Proceedings of Machine Learning Research, pp. 3246–3264. PMLR, 2021.

Ramgopal R. Mettu and C. Greg Plaxton. The online median problem. *SIAM J. Comput.*, 32(3): 816–832, 2003.

Ramgopal R. Mettu and C. Greg Plaxton. Optimal time bounds for approximate clustering. *Mach. Learn.*, 56(1-3):35–60, 2004.

Evi Micha and Nisarg Shah. Proportionally fair clustering revisited. In *ICALP*, LIPIcs, pp. 85:1–85:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

Sérgio Moro, Paulo Rita, and Paulo Cortez. Bank Marketing. UCI Machine Learning Repository, 2014. URL https://doi.org/10.24432/C5K306. [Dataset].

Maryam Negahbani and Deeparnab Chakrabarty. Better algorithms for individually fair k-clustering. In *NeurIPS*, pp. 13340–13351, 2021.

Federico Peiretti and Ruggero G Pensa. Fair associative co-clustering. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pp. 282–300. Springer, 2025.

Laurent Perron and Vincent Furnon. Or-tools, 2026. URL https://developers.google.com/optimization/. v9.15.6755, Apache License 2.0.

Tai Le Quy, Arjun Roy, Gunnar Friege, and Eirini Ntoutsi. Fair-capacitated clustering. In Sharon I-Han Hsiao, Shaghayegh (Sherry) Sahebi, François Bouchet, and Jill-Jênn Vie (eds.), *Proceedings of the 14th International Conference on Educational Data Mining, EDM 2021, virtual, June 29 - July 2, 2021*. International Educational Data Mining Society, 2021.

Melanie Schmidt, Chris Schwiegelshohn, and Christian Sohler. Fair coresets and streaming algorithms for fair k-means. In *International Workshop on Approximation and Online Algorithms*, pp. 232–251. Springer, 2019.

Yukyeong Song, Chenglu Li, Wanli Xing, Shan Li, and Hakeoung Hannah Lee. A fair clustering approach to self-regulated learning behaviors in a virtual learning environment. In *Proceedings of the 14th learning analytics and knowledge conference*, pp. 771–778, 2024.

Till Speicher, Muhammad Ali, Giridhari Venkatadri, Filipe Nunes Ribeiro, George Arvanitakis, Fabrício Benevenuto, Krishna P Gummadi, Patrick Loiseau, and Alan Mislove. Potential for discrimination in online targeted advertising. In *Conference on fairness, accountability and transparency*, pp. 5–19. PMLR, 2018.

Mikkel Thorup. Quick k-median, k-center, and facility location for sparse graphs. *SIAM J. Comput.*, 34(2):405–432, 2004.

Ali Vakilian and Mustafa Yalçiner. Improved approximation algorithms for individually fair clustering. In *AISTATS*, Proceedings of Machine Learning Research, pp. 8758–8779. PMLR, 2022.

Di Wu, Qilong Feng, and Jianxin Wang. Approximation algorithms for fair k-median problem without fairness violation. *Theor. Comput. Sci.*, 985:114332, 2024a.

Xiaoliang Wu, Qilong Feng, Ziyun Huang, Jinhui Xu, and Jianxin Wang. New algorithms for distributed fair k-center clustering: almost accurate as sequential algorithms. In *Proceedings of the 23rd International Conference on Autonomous Agents and Multiagent Systems*, pp. 1938–1946, 2024b.

A Extended Preliminaries

Let R be a set of red points and B a set of blue points in $\mathbb{R}^d$, and let $P := R \cup B$. We denote by $d(x, y)$ the distance between two points $x, y \in \mathbb{R}^d$. Throughout this paper, we assume that the aspect ratio (i.e., the ratio between the maximum and minimum non-zero pairwise distances) of any point set of size n is bounded by $\mathrm{poly}(n)$. Moreover, we assume that all input points lie within a bounding box of side length Δ, where $\Delta = \mathrm{poly}(n)$.

Throughout this paper, we also consider inputs with integer weights. Such instances can be interpreted as allowing multiple co-located copies of a point. In particular, red and blue points may be co-located. For any location that has both red and blue multiplicities, we represent it by two distinct points, one for each color. We now define k-median clustering, set balance, and fair clustering.

Definition A.1 (k-median)**.** *Given a set of points P, a k-median clustering is a partition of P into k subsets $C_1, ..., C_k$ together with the corresponding centers $S = \{s_1, ..., s_k\} \subset P$, such that the objective function $\sum_{i=1}^{k} \sum_{p \in C_i} d(p, s_i)$ is minimized.*

Definition A.2 (Balance)**.** *For any $S \subseteq P = R \cup B$, balance$(S) = \min\{\frac{|R \cap S|}{|B \cap S|}, \frac{|B \cap S|}{|R \cap S|}\}$.*

Definition A.3 (Fair clustering)**.** *Let $r, b > 0$ be parameters such that $r \leq b$, let P be a set of balance at least $\frac{r}{b}$. A fair (r, b)-clustering of P is a clustering of P such that each cluster has balance at least $\frac{r}{b}$. An optimal (r, b)-fair k-median clustering is a (r, b)-fair clustering with minimum k-median cost.*

For the construction of fair clusters, we will use a related notion of fair and fairlet decompositions, which clusters a set into (possibly many) balanced sets.

Definition A.4 (Fairlet decomposition)**.** *Given parameters $0 < r \leq b$, a fair decomposition of a bi-colored point set P is a partition of P into subsets, each having balance at least $\frac{r}{b}$, where each subset is associated with a center (which might not be part of the set). For the same input, a fairlet decomposition is a fair decomposition in which each subset has size at most $r + b$. The subsets in such a decomposition are referred to as* fairlets*.*

Definition A.5 (Cost of a fair decomposition)**.** *Given a fair set of points F with a center c, we define cost$(F) := \sum_{x \in F} d(x, c)$. We define the cost of a decomposition cost$(\mathcal{F}) = \sum_{F \in \mathcal{F}}$ cost(F). Finally, let $\mathrm{opt}(P)$ denote the minimum cost of any (r, b) fair decomposition of P.*

Chierichetti et al. (2017) show that for any fair decomposition of point set P there exists a fairlet decomposition with no greater cost.

Definition A.6 (Weighted Fair Decomposition)**.** *A fair decomposition of a point set P with integer weights $w : P \to \mathbb{Z}_{\geq 0}$ is a fair decomposition of a multi-set of P in which each point $p \in P$ appears $w(p)$ many times. The notion is defined analogously for fairlet decomposition.*

Throughout this paper, we represent a fairlet decomposition $\mathcal{F}$ of a bi-colored point set as a collection of tuples $F = (c_F, R_F, B_F, \mathrm{cost}_F)$, where c_F denotes the center, R_F and B_F denote the sets of red and blue points assigned to the fairlet, and cost_F denotes its cost. In the case of an input P with integer weights $w : P \to \mathbb{Z}_{\geq 0}$, the sets R_F and B_F are multi-sets over P such that each point $p \in P$ appears exactly $w(p)$ times across all fairlets in $\mathcal{F}$.

We describe fair decompositions in a more compact form. Specifically, a fair decomposition is represented as a collection of tuples $F = (c_F, |F|, \mathrm{cost}_F)$, where c_F denotes the center, $|F|$ the size of the set, and cost_F its cost. This representation is more efficient to compute, as it avoids explicitly enumerating point multiplicities in the case of weighted inputs.

Finally, Chierichetti et al. (2017) showed a reduction from the fair k-median problem to a problem of finding a good fair decomposition of the input set. We defer this discussion to Section B.

Finally, we introduce some notation for the minimum cost flow problem.

In the directed minimum-cost flow problem, we are given a directed graph $G = (V, E)$ with edge capacities $c : E \to \mathbb{R}_{\geq 0}$, weights (or costs) $E \to \mathbb{R}$ and vertex demands/supplies $b : V \to \mathbb{R}$ such that $\sum_{v \in V} b(v) = 0$. Our goal is to find a flow $f : E \to \mathbb{R}_{\geq 0}$ satisfying that for all vertices $v \in V : \sum_{(u,v) \in E} f(u, v) - \sum_{(v,u) \in E} f(v, u) = b(v)$ and all edges $e \in E : f(e) \leq c(e)$ (i. e. feasible) minimizing $\sum_{e \in E} w(e) f(e)$.

The residual graph G_r of G with respect to a feasible flow is a graph $G_r = (V, E_r)$, where for each edge $(u, v) \in E$ there are edges $(u, v), (v, u) \in E_r$ with capacities $c(u, v) - f(u, v)$, $f(u, v)$ and costs $w(u, v), -w(u, v)$, respectively. A minimum cost feasible flow f to G satisfies that any cycle in G_r has non-negative cost.

B Fair k-median via Fairlet Decomposition from Chierichetti et al. (2017)

Here, we briefly discuss how a fair k-median solution is obtained from a given fair decomposition, as discussed in Chierichetti et al. (2017).

Definition B.1 (k-Median Instance Associated with a Fair Decomposition)**.** *Given a fair decomposition $\mathcal{F}$, the associated k-median instance is defined on the set of points corresponding to the centers of the fair sets in $\mathcal{F}$, where each center is assigned a weight equal to the size of its corresponding fair set. That is, $(\{c_F\}, w'(c_F))_{F\in\mathcal{F}}$ is a weighted k-median instance where $w'(c_F) = |F|$.*

Chierichetti et al. (2017) show that any solution to the k-median instance associated with a fair decomposition of a bi-colored point set P induces a fair k-median solution for P, where the fair k-median cluster of each point in P is determined by the cluster to which the center of its corresponding fairlet is assigned. The total cost of this solution is at most the sum of the cost of the k-median instance and the cost of the fairlet decomposition. Observe that both fair and fairlet decompositions describe the associated k-median instance. Note that the cost of an optimal fair decomposition lower bounds the cost of an optimal fair k-median solution, as the clusters of a fair k-median solution define a fair decomposition of the same cost.

C Weighted Fair Decomposition

In this section, we briefly present algorithms for weighted t-approximate $(1, t)$-fair and $(1, t)$-fairlet decompositions. In particular, the first part of the result is Lemma 3.2 used in Section 3, while the second part is used in dynamic algorithm in Section 4.

Lemma C.1. *Given sets R, B of red and blue points, respectively, of total weight W, there exist algorithms for weighted t-approximate $(1, t)$-fairlet and $(1, t)$-fair decomposition with running time $\tilde{O}(\mathrm{poly}(n) + W)$ and $O(\mathrm{poly}(n))$, respectively.*

First, we introduce the concept of a *t-bounded edge cover* of a given graph, a generalization of graph matching.

Definition C.2. *A t-bounded edge cover of graph $G = (V, E)$ is a subset of edges $E_t \subseteq E$ satisfying that $1 \leq \deg_{E_t}(v) \leq t$ for all $v \in V$. Assuming vertices of G are associated with integer weights $w : V \to \mathbb{Z}_{\geq 0}$, the same object is a multiset satisfying that $w(v) \leq \deg_{E_t}(v) \leq w(v)t$. A t-bounded edge cover of a bi-colored point set $P = R \cup B$ is an edge cover of the bipartite metric graph G_P. An edge cover is said to be minimal if the removal of any edge would violate the edge cover property.*

Given a bi-colored point set $P = R \cup B$, we show how to compute a t-bounded edge cover of P (Lemma C.3), and then how to extract a $(1, t)$-fair and fairlet decomposition of P from it (Lemma C.4), concluding Lemma C.1.

Lemma C.3. *Given a bi-colored point set $P = R \cup B$ of n points in a metric space with total integer weight W, there exists a deterministic algorithm that computes a minimal minimum-cost t-bounded edge cover of P in time $O(\mathrm{poly}(n, \log t, \log W))$.*

Proof. The algorithm proceeds in three steps. First, given a set P, we construct the corresponding flow problem. Second, we show how to extract a t-bounded edge cover of P from the flow solution. Finally, we prune the edge cover to obtain a minimal one.

Flow Problem Construction. We define the graph $G_f = (V_f, E_f)$ as follows. Let $V_f = \{\beta \cup \rho \cup B \cup R \cup B' \cup R'\}$, where β, ρ represent single vertices (with a slight overload in notation), while sets B, B' and R, R' are sets of size $|B|$ and $|R|$, respectively. Denote $B = \{b_1, \ldots, b_{|B|}\}, R =$

$\{r_1, \ldots, r_{|R|}\}$ and $B' = \{b'_1, \ldots, b'_{|B|}\}$, $R' = \{r'_1, \ldots, r'_{|R|}\}$. The set E_f contains the following directed edges.

- (β, ρ) with capacity ∞ and cost 0
- (β, b_i) with capacity ∞ and cost 0 for all $i \in [|B|]$
- (b_i, b'_i) with capacity $t \cdot w(b_i)$ and cost 0 for all $i \in [|B|]$
- For any edge (b_i, r_j) in G and edge (b'_i, r'_j) with capacity ∞ and cost $d(b_i, r_j)$
- (r'_i, r_i) with capacity $t \cdot w(r_i)$ and cost 0 for all $i \in [|R|]$
- (r_i, ρ) with capacity ∞ and cost 0 for all $i \in [|R|]$

Nodes $b_i \in B$ and $r_i \in R$ have supply $w(b_i)$ and demand $w(r_i)$, respectively. In addition, nodes β and ρ have supply $\sum_{i \in [|B|]} w(b_i)$ and $\sum_{i \in [|R|]} w(R_i)$ respectively. All other nodes have no supply or demand. An example of graph $G_f = (V_f, E_f)$ is shown in Figure 4. Notice that G_f has $O(n^2)$ edges and can be constructed in $O(n^2)$ time. The maximum edge capacity in G_f is nt and edge costs are bounded by $\text{poly}(n)$ (as we assume the aspect ratio of the input is $\text{poly}(n)$). We can get an integral solution f_G to this problem in $O(\text{poly}(n, \log W, \log t))$ time using folklore algorithms (i.e. Goldberg & Tarjan (1987)) (or in $\hat{O}(n^2)$ using state of the art algorithms Chen et al. (2025)).

Edge Cover Extraction. Let $G_t = (R \cup B, E_t)$ be a multi graph where $E(b_i, r_j) = f_G(b'_i, r'_j)$ for all $i \in [|B|], j \in [|R|]$, where $E(b_i, r_j)$ stands for the multiplicity of edge (b_i, r_j) in E_t. We first show that E_t is a t-bounded edge cover of P. We note that the same arguments appear in Chierichetti et al. (2017) for the unweighted case.

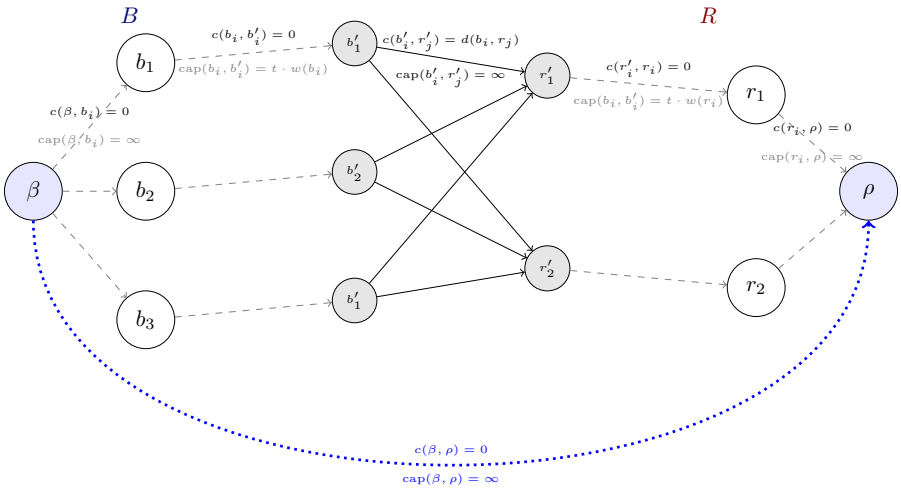


Figure 4: The flow network of Chierichetti et al. (2017)

As only edges running between R' and B' in G_f have non-zero costs and these correspond to edge costs in G_t, it follows that $\text{cost}(f) = \text{cost}(E_t)$. To see that E_t is a t-bounded edge cover observe that in f_g all flow leaving b'_i for any $i \in [|B|]$ must cross edge (b_i, b'_i) in G_f that has capacity $w(b_i)t$. Similarly, for any flow entering r'_j for $j \in [|R|]$ must cross edge (r'_j, r_j) which has capacity $w(r_j)t$. As b_j has supply $w(b_j)$ at least $w(b_j)$ flow is leaving b'_j in f_G. Similarly, as r_j has $w(r_j)$ demand at least $w(r_j)$ flow is entering r'_j in f_G.

It remains to argue that any minimal t-bounded edge cover E^*_t of P defines a feasible flow with the same cost in G_f. Start with some empty flow f^*. Route $E^*_t(b_i, r_j)$ on edges $(b_i, b'_i), (b'_i, r'_j)$

and (r'_j, r_j) in f^* for all $i \in [|B|], j \in [|R|]$. Then route $\deg_{E_t^*}(b_i) - w(b_i)$ flow from β to b_i and $\deg_{E_t^*}(r_j) - w(r_j)$ flow from r_i to ρ in f^*. By definition f^* at this point has cost(E_t^*) and saturates the supplies and demands of $R \cup B$.

Let $w(B) = \sum_{i \in [|B|]} w(b_i), w(R) = \sum_{i \in [|R|]} w(r_i)$. The flow leaving β and entering ρ at this point are $\sum_{i \in [|B|]} \deg_{E_t^*}(b_i) - w(B)$ and $\sum_{i \in [|R|]} \deg_{E_t^*}(r_i) - w(R)$. Note that $\sum_{i \in [|B|]} \deg_{E_t^*}(b_i) = \sum_{i \in [|R_i|]} \deg_{E_t}(r_i) = |E_t^*|$ as E_t^* is bipartite. Hence, β has $w(R) - |E_t^*| + w(B)$ remaining supply and ρ has equal $w(B) - |E_t^*| + w(R)$ remaining demand which we may route along (β, ρ) for free in f^*.

It remains to show that $w(R) - |E_t^*| + w(B)$ is not negative. We claim that the minimality of E_t^* implies that for all edges in E_t^* one endpoint must have degree in E_t^* equaling its weight under w, let's call such nodes light. Indeed, if the contrary would hold the multiplicity of the edge could be reduced without violating that E_t^* is an edge cover. Therefore, all edges of E_t^* are incident on some light node, and by the definition of a light node $w(R) + w(B) \geq |E_t^*|$.

Ensuring Minimality. Finally, we briefly discuss how to prune edges from E_t to obtain a minimal edge cover. Namely, note that if there are 0-cost edges present between red and blue nodes in G_f (i.e. some point has both red and blue weight) then E_t might not be minimal initially. However, we can iterate greedily through the $O(n^2)$ pairs of points in $R \cup B$ and if for any edge both endpoints have degree greater then the edges multiplicity in E_t we may reduce this multiplicity until equality is reached. This does not change the cost of E_t.

□

Lemma C.4. *There exists a deterministic algorithm which given an α-approximate minimal minimum-cost t-bounded edge cover of an integer weighted bi-colored point set P of n points with total weight W returns an αt approximate $(1, t)$-fairlet and $(1, t)$-fair decomposition of P in times $O(n^2)$ and $O(|W|)$, respectively.*

Proof. Let E_t be the given t-bounded edge cover. We first describe the algorithm for extracting a $(1, t)$-fairlet decomposition and then the algorithm for extracting a $(1, t)$-fair decomposition.

Weighted Fairlet Decomposition. We now show how to obtain a fairlet decomposition of P of cost cost(E_t) in time $O(|E_t|) = O(W)$ greedily. Recall that a fairlet decomposition is a union of tuples consisting of the center, red points, blue points and cost of a fairlet. The algorithm first iterates through the set of heavy (i.e. not light with respect to E_t, w) points. For each heavy point it creates fairlets centered at that point of maximum size as long as it can, greedily and arbitrarily. Once the edges of E_t and weights w are updated to reflect the removal of the points of these fairlets the weight of all points must correspond to their degree. The algorithm then creates $(1, 1)$-fairlets based on the remaining edges greedily. The algorithm is given in Algorithm 1.

Weighted Fair Decomposition. The following algorithm uses a similar logic but returns a fair decomposition of the input points in $O(n^2)$ time. Recall that a fair decomposition is described by a union of tuples containing the centers, sizes and costs of the corresponding fair sets. The algorithm first creates fair sets centered around heavy points using all edges in E_t incident on them. Then for any edge (p, p') where neither endpoint is heavy it creates a $(E_t(p, p'), E_t(p, p'))$ fair set of the two endpoints centered arbitrarily at one endpoint. The algorithm is given in Algorithm 2.

Algorithm 2: WEIGHTEDFAIR($P = (R \cup B), w, E_t$)

$P_H \leftarrow \{p \in P \mid w(p) < \deg_{E_t}(p)\}, \mathcal{F} \leftarrow \emptyset$
for $p \in P_H$ **do**
  $\mathcal{F} \leftarrow \mathcal{F} \cup (p, \deg_{E_t}(p) + w(p), \sum_{p' \in N_{E_t}(p)} d(p, p') \cdot E_t(p, p'))$
for $(p, p') \in E_t \mid p, p' \notin P_H$ **do**
  $\mathcal{F} \leftarrow \mathcal{F} \cup (p, 2E_t(p, p'), d(p, p') \cdot E_t(p, p'))$
Return $\mathcal{F}$

Algorithm 1: WEIGHTEDFAIRLET($P = (R \cup B), w, E_t$)

```
P_H ← {p ∈ P | w(p) < deg_{E_t}(p)}, F ← ∅
for p ∈ P_H do
    /* W.L.O.G. Assume p ∈ R                                   */
    while w(p) < deg_{E_t}(p) do
        w(p) ← w(p) − 1
        i ← 1
        cost_F ← 0
        F_B ← ∅
        while w(p) < deg_{E_t}(p) ∧ i ≤ t do
            p' ← arbitrary point in N_{E_t}(p)
            /* where N_{E_t}(p) is the neighborhood of p in E_t.  */
            F_B ← F_B ∪ {p'}
            cost_F ← cost_F + d(p, p')
            E_t ← E_t \ (p, p')
            i ← i + 1
        F ← F ∪ (p, {p}, F_B, cost_F)
for (r, b) ∈ E_t | r ∈ R do
    F ← F ∪ (r, {r}, {b}, d(r, b))
Return F
```

Bounding the Cost. Both of these decomposition approaches return a fair decomposition of cost equaling that of E_t. We will argue that cost(E_t) is at most t times the cost of a $(1, t)$-fair decomposition of P. Note that by Chierichetti et al. (2017) we know that there exists a fairlet decomposition of P with cost equaling the cost of an optimal fair decomposition. Let this $(1, t)$ fairlet decomposition be $\mathcal{F}^*$, and let F be one arbitrary fairlet of $\mathcal{F}^*$. First, observe that if F contains both more than 1 red and 1 blue point, then it can be decomposed into smaller fairlets of the same cost.

Assume $\mathcal{F}^*$ consists of fairlets containing exactly one red or blue point. By adding edges from this unique point of the fairlets to their other points, we obtain a t-bounded edge cover of P. Without loss of generality, assume that some fairlet $F \in \mathcal{F}^*$ contains a single red point r, blue points $b_1, \ldots, b_k$ and center c. By the triangle inequality we have that:

$$\sum_{i\in[k]} d(r, b_i) \leq \sum_{i\in[k]} d(r, c) + d(c, b_i) \leq t(d(r, c) + \sum_{i\in[k]} d(c, b_i)) = t \cdot \text{cost}(F)$$

Therefore, the edges of a minimum cost t-star decomposition have cost at most t-times of the minimum cost fair decomposition. This implies that if E_t is an α-approximate t-bounded edge cover, our algorithms return αt approximate $(1, t)$-fairlet and $(1, t)$-fair decompositions, respectively.

□

Finally, by combining Lemmas C.3 and C.4, we obtain Lemma C.1.

D OUR STATIC ALGORITHM

We first give details of the static algorithm from Section 3, and then prove Theorem 3.1.

Static Algorithm Description. First, we describe a preprocessing step. Obtain point set P' by randomly shifting the points of P as follows. Suppose $P \subseteq [0, \Delta]^d$. Then, draw a random shift vector $r \sim \text{Uniform}([0, \Delta]^d)$. Let $P' = \{p + r \mid p \in P\}$. Let $\mathcal{C}^1$ refer to the $(2\Delta)^d$ box containing the input points. Now we invoke RECURSIVEFAIRLET($\mathcal{C}^1, P', \emptyset$), described in Algorithm 3.

Algorithm 3: RECURSIVEFAIRLET($\mathcal{C}, P = R \cup B, \mathcal{F}_{\text{out}}$)

```
/* Base Case                                                    */
if |P| ≤ n^{2dε} then
    F_out ← F_out ∪ WEIGHTEDFAIRLET(P)
/* Grid Construction                                            */
Let C_1, ··· , C_m be non-empty sub-grid cells obtained by partitioning C into a n^ε × .... × n^ε grid.
Let p_i be the center of C_i
Define red and blue points p_i^R, p_i^B at location p_i with weights w(p_i^R) = |C_i ∩ R|, w(p_i^B) = |C_i ∩ B|
/* Coarse Weighted Fairlet Decomposition                        */
Let F_coarse ← WEIGHTEDFAIRLET({p_1^R, p_1^B, ... }).
for F = (c, R_F, B_F, cost_F) ∈ F_coarse do
    if R_F ∪ B_F contains points from at least two C_i's then
        R'_F ← ∅, B'_F ← ∅
        for C_i s.t. C_i ∩ (R_F ∪ B_F) ≠ ∅ do
            Move |R_F ∩ C_i| arbitrary points from R ∩ C_i into R'_F
            Move |B_F ∩ C_i| arbitrary points from B ∩ C_i into B'_F
        F_out ← F_out ∪ (c, R'_F, B'_F, Σ_{p∈R'_F∪B'_F} d(c, p))
/* Recursive Step                                               */
for C_i s.t. C_i ∩ P ≠ ∅ do
    F_out ← F_out ∪ RECURSIVEFAIRLET(C_i, P ∩ C_i, ∅)
return F_out
```

D.1 PROOF OF THEOREM 3.1

Running Time Analysis. First, we bound the depth of the recursion. Between consecutive layers, the grid side length decreases by a factor of n^ϵ. The recursion stops once a cell contains at most n^ϵ points. Since the aspect ratio and enclosing box are $\text{poly}(n)$, the number of levels is $L = O(1/\epsilon)$.

Now consider the run time of WEIGHTEDFAIRLET procedure across these levels. Each cell has $n^{\epsilon d}$ subcells. Every non-empty cell $\mathcal{C}$ triggers a call to WEIGHTEDFAIRLET, which runs in time $\tilde{O}(n^{\epsilon d} + |P \cap \mathcal{C}|)$. All other operations per cell take time $O(|P \cap \mathcal{C}| + n^{\epsilon d})$. Each point appears in exactly one cell per level, implying at most $O(n/\epsilon)$ non-empty cells overall. Thus, the $n^{\epsilon d}$ term in WEIGHTEDFAIRLET sums to $\tilde{O}(n^{1+\epsilon d}/\epsilon)$, while the $|P \cap \mathcal{C}|$ terms sum to $O(n/\epsilon)$. The total running time is therefore $\tilde{O}(n^{1+\epsilon d}/\epsilon)$.

The remainder of this section proves the approximation ratio of the static algorithm.

D.2 APPROXIMATION RATIO OF THEOREM 3.1

Before we turn to analyzing the expected approximation ratio of the algorithm we will first prove the following core lemma concerning the stability of (r, b)-fairlet decompositions. Lemma D.1 essentially states the following: if the input points are contained in some cell with side length Δ and they undergo some x arbitrary updates, the cost of an optimal solution might change by at most $O(x\sqrt{d}\Delta\,\text{poly}(r+b))$. The exact polynomial factor of $(r+b)$ in the lemma is not important for the purposes of our analysis, as it will be absorbed by the $(r+b)^{O(1/\epsilon)}$ notation in Theorem 1.1.

Lemma D.1. *Let* $\text{opt}(P)$ *stand for the cost of an optimal* (r, b)*-fairlet decomposition of point set* P*. Let* P *be a set of bi-colored points in* $(0, \Delta)^d$ *such that balance*$(P) \geq r/b$*. Let* $P^- \subseteq P$ *and* $P^+ \subseteq (0, \Delta)^d$ *such that balance*$(P \setminus P^-)$*, balance*$(P \cup P^+) \geq r/b$*. Then*

$$\text{opt}(P \setminus P^-) \leq \text{opt}(P) + \sqrt{d}(r+b)^4\Delta|P^-|$$

and

$$\text{opt}(P \cup P^+) \leq \text{opt}(P) + \sqrt{d}(r+b)^4\Delta|P^+|.$$

Proof. Let P be a point set with balance at least r/b, and suppose we wish to extend it by a point p^+ such that the balance of $P \cup \{p^+\}$ remains at least r/b, where $r < b$. Fix a minimum-cost (r, b)-fairlet decomposition $\mathcal{F}^*$ of P, and assume without loss of generality that p^+ is red.

Let $(\mathcal{F}^*)'$ denote a smallest subset of $\mathcal{F}^*$ such that

$$\frac{b_1 + \cdots + b_k}{r_1 + \cdots + r_k + 1} \geq \frac{r}{b},$$

where r_i and b_i denote the numbers of red and blue points, respectively, in the i-th fairlet. Equivalently,

$$(b_1 + \cdots + b_k) \cdot b \geq (r_1 + \cdots + 1) \cdot r.$$

Such a subset exists since $\mathrm{balance}(P \cup \{p^+\}) \geq r/b$. We first claim that for all $i \in [k]$, it must hold that $b_i \cdot b > r_i \cdot r$. Otherwise, as we know the i-th fairlet is balanced and $b_i \cdot b = r_i \cdot r$. Removing the i-th fairlet would preserve the inequality, contradicting the minimality of $(\mathcal{F}^*)'$. Moreover, the difference $b_i \cdot b - r_i \cdot r$ is a positive integer and as the union of the fairlets in $(\mathcal{F})'$ must have balance at least r/b, we know that $b \sum b_i \geq r \sum r_i$. It follows that $k = O(r + b)$.

To construct a decomposition of $P \cup \{p^+\}$, we take $\mathcal{F}^*$, merge all fairlets in $(\mathcal{F}^*)'$ into a single set, and add p^+ to this set, designating p^+ as its center. This operation increases the cost by at most $2\sqrt{d}(r+b)^2\Delta$. The resulting decomposition is fair, and therefore there exists a fairlet decomposition of $P \cup \{p^+\}$ with no greater cost.

If instead a point is removed from P, we may delete the fairlet containing it in $\mathcal{F}^*$. This can only decrease the cost. We then reinsert the remaining points using the above argument. Consequently, the cost of the optimal decomposition changes by at most $2\sqrt{d}(r + b)^3\Delta \leq \sqrt{d}(r + b)^4\Delta$ under a single update.

It remains to argue what happens if multiple points are inserted into or deleted from P at the same time. As deletion can be reduced to insertions in the manner described above, we will only consider the case where $P^+ \subseteq \Delta^d$ is inserted into the input. Note that we may assume that $|P| \geq r + b$ as otherwise any fair decomposition of $P \cup P^+$ has cost at most $2\Delta\sqrt{d}(|P| + |P^+|) \leq \sqrt{d}(r + b)^2\Delta|P^+|$.

It is sufficient to show that if $\mathrm{balance}(P) \geq r/b$ and $\mathrm{balance}(P \cup P^+) \geq r/b$ then there exists an ordering $p_1^+, \ldots, p_{|P^+|}^+$ of P^+ such that $\mathrm{balance}(P \cup (\cup_{j \leq i} p_j^+)) \geq r/b$, that is we can add the points of P^+ to P one by one such that the balance of the input is always at least r/b and our previous argument suffices.

Without loss of generality, assume that P has at least as many red as blue points. Then we may set p_1^+ to be an arbitrary blue point of P^+ if it exists, otherwise an arbitrary red point and we must have that $\mathrm{balance}(P \cup p_1^+) \geq 1$ (as $|P| \geq (r + b)$). We then can recurse on $P' = P \cup p_1^+$ and $(P^+)' = P^+ \setminus p_1^+$.

The only case we didn't consider is when $r = b = 1$. In this case, a fairlet decomposition of the input is just a matching between the input points. For the balance properties to hold, it must be the case that P^+ has the same number of red and blue points. We can then extend $\mathcal{F}^*$ with an arbitrary matching of P^+ and its cost may only increase by $2\sqrt{d}|P^+|\Delta$.

□

To analyze the algorithm, fix some optimum cost fairlet decomposition of the input $\mathcal{F}^*$ and let $E_{\mathcal{F}^*}$ stand for the edges of this fairlet decomposition (i.e. segments connecting the points of the fairlets to their centers). Let the cells of layer i be denoted by $\mathcal{C}_j^i$, where the first layer contains a single cell, the bounding box of the input, $(2\Delta)^d$. Note that each cell $\mathcal{C}_j^i$ contains $n^{2\epsilon}$ sub-cells of the next layer if the algorithm recursed on its sub-cells, i.e. more than n^ϵ points were decomposed in it.

Let ℓ_i stand for the cell-side length of cell $\mathcal{C}_j^i$, that is $\ell_i = 2\Delta/n^{\epsilon \cdot (i-1)}$. Let $P[\mathcal{C}_j^i]_{\text{Alg}}$ stand for the set of points which were passed down to cell $\mathcal{C}_j^i$ by the algorithm, that is the set of points function RECURSIVEFAIRLET was called with in line 15, when considering cell $\mathcal{C}_j^i$. Let $\mathcal{F}_{\text{out}}[\mathcal{C}_j^i]$ stand for the set of fairlets the output $\mathcal{F}_{\text{out}}$ was extended with when processing cell $\mathcal{C}_j^i$, that is either in line 2 or 13.

Let $P[\mathcal{C}_j^i]^-_{\text{Alg}} \subseteq P$ stand for the subset of points of $P \cap \mathcal{C}_j^i$ the algorithm has placed into fairlets containing points outside of $\mathcal{C}_j^i$. Observe that by construction $P \cap \mathcal{C}_j^i = P[\mathcal{C}_j^i]_{\text{Alg}} \cup P[\mathcal{C}_j^i]^-_{\text{Alg}}$ and $P[\mathcal{C}_j^i]_{\text{Alg}} \cap P[\mathcal{C}_j^i]^-_{\text{Alg}} = \emptyset$. Let $P[\mathcal{C}_j^i]^-_{\text{opt}}$ stand for the set of points of $P \cap \mathcal{C}_j^i$ which $\mathcal{F}^*$ places in fairlets which contain a point outside of $\mathcal{C}_j^i$. Let $P[\mathcal{C}_j^i]_{\text{opt}}$ stand for the set of points in $P \cap \mathcal{C}_j^i$ which $\mathcal{F}^*$ places into fairlets containing points exclusively from $\mathcal{C}_j^i$. Observe that $P \cap \mathcal{C}_j^i = P[\mathcal{C}_j^i]^-_{\text{opt}} \cup P[\mathcal{C}_j^i]_{\text{opt}}$ and $P[\mathcal{C}_j^i]^-_{\text{opt}} \cap P[\mathcal{C}_j^i]_{\text{opt}} = \emptyset$.

Finally, let $P[\mathcal{F}_{\text{out}}[\mathcal{C}_j^i]]$ stand for the set of points decomposed when visiting cell $\mathcal{C}_j^i$, that is, the set of points added to the fairlet decomposition in line 14).

Lemma D.2. *For every cell $\mathcal{C}_j^i$, we have that*

$$\mathbb{E}[\mathit{cost}(\mathcal{F}_{\mathit{out}}[C_j^i])] \leq 20d^{3/2}(r+b)^5\alpha\mathbb{E}[(\text{opt}(P[\mathcal{C}_j^i]_{\text{opt}}) + \ell_i(|P[\mathcal{C}_j^i]^-_{\text{opt}}| + |P[\mathcal{C}_j^i]^-_{\mathit{Alg}}|))]$$

and if the recursion doesn't stop in $\mathcal{C}_j^i$, then

$$\mathbb{E}[|P[\mathcal{F}_{\mathit{out}}[\mathcal{C}_j^i]]|] \leq \frac{20d^{3/2}(r+b)^6\alpha}{\ell_{i+1}}\mathbb{E}[(\text{opt}(P[\mathcal{C}_j^i]_{\text{opt}}) + \ell_i(|P[\mathcal{C}_j^i]^-_{\text{opt}}| + |P[\mathcal{C}_j^i]^-_{\mathit{Alg}}|)].$$

Here, the expectation is taken with respect to the random shift.

Proof. First consider the case when the recursion stops in $\mathcal{C}_j^i$. In cell $\mathcal{C}_j^i$, the algorithm finds a fairlet decomposition of the points $P[\mathcal{C}_j^i]_{\text{Alg}}$. It then runs a static α-approximate fairlet decomposition algorithm WEIGHTEDFAIRLET on these points to obtain $\mathcal{F}_{\text{out}}[\mathcal{C}_j^i]$.

We have that $P[\mathcal{C}_j^i]_{\text{opt}} = (P \cap \mathcal{C}_j^i) \setminus P[\mathcal{C}_j^i]^-_{\text{opt}}$ and $P[\mathcal{C}_j^i]_{\text{Alg}} = (P \cap \mathcal{C}_j^i) \setminus P[\mathcal{C}_j^i]^-_{\text{Alg}}$, and hence $|P[\mathcal{C}_j^i]_{\text{opt}} \Delta P[\mathcal{C}_j^i]_{\text{Alg}}| \leq |P[\mathcal{C}_j^i]^-_{\text{opt}}| + |P[\mathcal{C}_j^i]^-_{\text{Alg}}|$, where Δ stands for the symmetric difference of two sets. By Lemma D.1, this implies that $\text{opt}(P[C_j^i]_{\text{Alg}}) \leq \text{opt}(P[\mathcal{C}_j^i]_{\text{opt}}) + \ell_i(|P[\mathcal{C}_j^i]^-_{\text{opt}}| + |P[\mathcal{C}_j^i]^-_{\text{Alg}}|)(r+b)^4$ (deterministically), proving the lemma.

Now we consider the case when the algorithm recourses on the sub-cells of $\mathcal{C}_j^i$. In this more complex scenario, it looks at the sub-cells of $\mathcal{C}_j^i$ on layer $i+1$ and for each sub-cell, it pushes all points to their sub-cell centers. It then finds an α-approximate decomposition of these shifted points. Finally, for any fairlet found this way which crosses sub-cell boundaries, it constructs a fairlet with arbitrary points from the respective sub-cells and these fairlets define $\mathcal{F}_{\text{out}}[\mathcal{C}_j^i]$.

Let $P[\mathcal{C}_j^i]'_{\text{Alg}}$ and $P[\mathcal{C}_j^i]'_{\text{opt}}$ stand for subset of points $P[\mathcal{C}_j^i]_{\text{Alg}}$ and $P[\mathcal{C}_j^i]_{\text{opt}}$, after the shift to their sub-cell centers, respectively. We will first argue that shifting the points of $P[\mathcal{C}_j^i]_{\text{opt}}$ to the sub-grid centers doesn't change the cost of an optimal solution by much.

Claim D.3. *For every cell $\mathcal{C}_j^i$, we have that*

$$\mathbb{E}[\text{opt}(P[\mathcal{C}_j^i]'_{\text{opt}})] \leq 5d(r+b)\mathbb{E}[\text{opt}(P[\mathcal{C}_j^i]_{\text{opt}})].$$

Proof. Let f be a fairlet with points p_f in $\mathcal{F}^*$ and cost $\text{cost}(f)$, and let R be the smallest d-dimensional box containing its points. Then f contributes $\text{cost}(f)$ to $\text{opt}(P[\mathcal{C}_j^i]_{\text{opt}})$ if R completely falls into $\mathcal{C}_j^i$. Under this condition, if R crosses the sub-grid in $\mathcal{C}_j^i$, then it contributes at most $2\ell_{i+1}|p_f| + \text{cost}(f) \leq 2\ell_{i+1}(r+b) + \text{cost}(f)$ to $\text{opt}(P[\mathcal{C}_j^i]'_{\text{opt}})$, otherwise 0, due to the shift of the points to the sub-grid centers.

Let $a_1, ..., a_d$ be the side lengths of R. Note that $a_1, ..., a_d \leq \text{cost}(f)$ and $\text{cost}(f) \geq \max_{j\in[d]} a_j$. If either $\max_j a_j > \ell_{i+1}$, then f must cross the grid on level ℓ_{i+1}. In this case, if f is contained in $\mathcal{C}_j^i$ it contributes $\text{cost}(f)$ to $\text{opt}(P[\mathcal{C}_j^i]_{\text{opt}})$ and $2\sqrt{d}\ell_{i+1}(r+b) + \text{cost}(f) \leq \text{cost}(f)(2\sqrt{d}(r+b))$ to $\text{opt}(P[\mathcal{C}_j^i]'_{\text{opt}})$.

Otherwise, if f is contained in $\mathcal{C}_j^i$, the probability that it is crossing the grid on level ℓ_{i+1} is $1 - \Pi_{j=1}^d(1 - \frac{a_j}{l_{i+1}}) \leq \frac{\sum_{j=1}^d a_j}{l_{i+1}}$ (note that in this case $a_1, ..., a_d \leq \ell_{i+1}$). It is easy to see that $\sum_{j=1}^d a_j \leq$

$\sqrt{d}\text{cost}(f)$. That is, f contributes $\sqrt{d}\text{cost}(f)$ to $\text{opt}(P[\mathcal{C}_j^i]_{\text{opt}})$ and is expected to contribute at most $\text{cost}(f) + 2\sqrt{d}\ell_{i+1}(r+b) \cdot 2\sqrt{d}\text{cost}(f)/\ell_{i+1} = 5d(r+b)\text{cost}(f)$ to $\text{opt}(P[\mathcal{C}_j^i]'_{\text{opt}})$.

□

The algorithm computes an α-approximate fairlet decomposition $\mathcal{F}_{P[\mathcal{C}_j^i]'_{\text{Alg}}}$ of $P[\mathcal{C}_j^i]'_{\text{Alg}}$ using WEIGHTEDFAIRLET. The actual fairlets of $\mathcal{F}_{\text{out}}[\mathcal{C}_j^i]$ are constructed by looking at the fairlets $\mathcal{F}^{>0}_{P[\mathcal{C}_j^i]'_{\text{Alg}}}$ of $\mathcal{F}_{P[\mathcal{C}_j^i]'_{\text{Alg}}}$ crossing the sub-grid of $\mathcal{C}_j^i$ and picking arbitrary points from $P[\mathcal{C}_j^i]_{\text{Alg}}$ to represent them from the appropriate sub-cells. Hence, $\mathcal{F}_{\text{out}}[\mathcal{C}_j^i]$ corresponds exactly to the fairlet decomposition $\mathcal{F}^{>0}_{P[\mathcal{C}_j^i]'_{\text{Alg}}}$ with the difference that the former is obtained by shifting the points of the latter within sub-cells of $\mathcal{C}_j^i$.

Claim D.4. *For the fairlet decomposition corresponding to cell $\mathcal{C}_j^i$, we have that*

$$\textit{cost}(\mathcal{F}_{\textit{out}}[\mathcal{C}_j^i]) \leq 4\sqrt{d}(r+b)\textit{cost}(\mathcal{F}^{>0}_{P[\mathcal{C}_j^i]'_{\textit{Alg}}}).$$

Proof. In shifts within sub-cells of $\mathcal{C}_j^i$, any fairlet of $\mathcal{F}^{>0}_{P[\mathcal{C}_j^i]'_{\text{Alg}}}$ might have its cost stretched by at most $4\sqrt{d}(r+b)\ell_{i+1}$ as each of these fairlet contains at most $r+b$ points. However, each fairlet of $\mathcal{F}^{>0}_{P[\mathcal{C}_j^i]'_{\text{Alg}}}$ must contain an edge crossing the sub-grid and hence have cost at least ℓ_{i+1}. This implies $\text{cost}(\mathcal{F}_{\text{out}}[\mathcal{C}_j^i]) \leq 4\sqrt{d}(r+b)\text{cost}(\mathcal{F}^{>0}_{P[\mathcal{C}_j^i]'_{\text{Alg}}})$.

□

Claim D.5. *For every cell $\mathcal{C}_j^i$, we have that*

$$\text{opt}(P[\mathcal{C}_j^i]'_{Alg}) \leq \text{opt}(P[\mathcal{C}_j^i]'_{\text{opt}}) + \ell_i(|P[\mathcal{C}_j^i]^-_{\text{opt}}| + |P[\mathcal{C}_j^i]^-_{Alg}|)\sqrt{d}(r+b)^4.$$

Proof. First, we have that $P[\mathcal{C}_j^i]_{\text{opt}} = (P \cap \mathcal{C}_j^i) \setminus P[\mathcal{C}_j^i]^-_{\text{opt}}$ and $P[\mathcal{C}_j^i]_{\text{Alg}} = (P \cap \mathcal{C}_j^i) \setminus P[\mathcal{C}_j^i]^-_{\text{Alg}}$. Hence, $|P[\mathcal{C}_j^i]'_{\text{opt}} \Delta P[\mathcal{C}_j^i]'_{\text{Alg}}| \leq |P[\mathcal{C}_j^i]^-_{\text{opt}}| + |P[\mathcal{C}_j^i]^-_{\text{Alg}}|$. By Lemma D.1, this implies that $\text{opt}(P[C_j^i]'_{\text{Alg}}) \leq \text{opt}(P[\mathcal{C}_j^i]'_{\text{opt}}) + \ell_i(|P[\mathcal{C}_j^i]^-_{\text{opt}}| + |P[\mathcal{C}_j^i]^-_{\text{Alg}}|)\sqrt{d}(r+b)^4$, as desired.

□

We are now ready to conclude the first inequality of Lemma D.2.

$$\begin{aligned}
\mathbb{E}[\text{cost}(\mathcal{F}_{\text{out}}[\mathcal{C}_j^i])] &\leq O(\sqrt{d}(r+b))\mathbb{E}[\text{cost}(\mathcal{F}^{>0}_{P[\mathcal{C}_j^i]'_{\text{Alg}}})] && (1)\\
&= O(\sqrt{d}(r+b))\mathbb{E}[\text{cost}(\mathcal{F}_{P[\mathcal{C}_j^i]'_{\text{Alg}}})]\\
&\leq O(\sqrt{d}\alpha(r+b))\mathbb{E}[\text{opt}(P[\mathcal{C}_j^i]'_{\text{Alg}})] && (2)\\
&\leq O(\sqrt{d}\alpha(r+b))\mathbb{E}[\text{opt}(P[\mathcal{C}_j^i]'_{\text{opt}}) + \ell_i(|P[\mathcal{C}_j^i]^-_{\text{opt}}| + |P[\mathcal{C}_j^i]^-_{\text{Alg}}|)(r+b)^4] && (3)\\
&\leq O(\sqrt{d}\alpha(r+b))\mathbb{E}[d(r+b)\,\text{opt}(P[\mathcal{C}_j^i]_{\text{opt}})\\
&\quad + \ell_i(|P[\mathcal{C}_j^i]^-_{\text{opt}}| + |P[\mathcal{C}_j^i]^-_{\text{Alg}}|)\sqrt{d}(r+b)^4] && (4)\\
&\leq O(d^{3/2}\alpha(r+b)^5)\mathbb{E}[\text{opt}(P[\mathcal{C}_j^i]_{\text{opt}}) + \ell_i(|P[\mathcal{C}_j^i]^-_{\text{opt}}| + |P[\mathcal{C}_j^i]^-_{\text{Alg}}|)]
\end{aligned}$$

Inequalities 1, 3 and 4 follow from Claims D.4, D.5 and D.3, respectively. Inequality 2 follows from the fact that the algorithm computes an α-approximate decomposition of $P[\mathcal{C}_j^i]'_{\text{Alg}}$. To conclude the second inequality of the lemma, observe that any fairlet in $\mathcal{F}^{>0}_{P[\mathcal{C}_j^i]'_{\text{Alg}}}$ may contribute at most $r+b$ points to $P[\mathcal{F}_{\text{out}}[\mathcal{C}_j^i]]$ and must have cost at least ℓ_{i+1}.

□

We will sum the variables across all cells for a given level and define $\text{cost}_i(\mathcal{F}_{\text{out}}) = \sum_j \text{cost}(\mathcal{F}_{\text{out}}[C_j^i])$, $\text{cost}_i(\mathcal{F}^*) = \sum_j \text{opt}(P[\mathcal{C}_j^i]_{\text{Alg}})$, and $P[i]^-_{\text{Alg}} = \cup_j P[\mathcal{C}_j^i]^-_{\text{Alg}}$ and $P[i]^-_{\text{opt}} = \cup_j P[\mathcal{C}_j^i]^-_{\text{opt}}$.

Claim D.6. *Let L stand for the depth of the algorithm recursion. Then for all $i \in [L]$, we have that*

$$\mathbb{E}[|P[i]^-_{\text{opt}}|] \le 10\sqrt{d}(r+b)\,\text{opt}(P)/\ell_i.$$

Here, the expectation is taken with respect to the random shift.

Proof. First, notice that a point of P appears in $P[i]^-_{\text{opt}}$ if it belongs to a fairlet in $\mathcal{F}^*$ which has an edge crossing the grid on level i. Any edge of length x in the decomposition is expected to cross the grid on level i at most $10\sqrt{d}x/\ell_i$ times. Each such crossing contributes at most $r+b$ points to $P[i]^-_{\text{opt}}$.

□

Putting Everything Together. We are now ready to upper bound the approximation ratio of the algorithm. We will inductively, starting from $i = 1$, show the following two inequalities:

$$\mathbb{E}[\text{cost}_i(\mathcal{F}_{\text{out}})] \le (400d^2(r+b)^8\alpha)^i\,\text{opt}(P) \tag{5}$$

$$\mathbb{E}[|\cup_{j\le i} P[\mathcal{F}_{\text{out}}[j]]|] \le \frac{(400d^2(r+b)^8\alpha)^i}{\ell_{i+1}}\,\text{opt}(P) \tag{6}$$

Base case. For the case $i = 1$, theorem D.2 states:

$$\begin{aligned}\mathbb{E}[\text{cost}_1(\mathcal{F}_{\text{out}})] &\le 20d^{3/2}(r+b)^5\alpha\mathbb{E}[\text{opt}(P[\mathcal{C}_1^1]_{\text{opt}}) + \ell_1(|P[\mathcal{C}_1^1]^-_{\text{opt}}| + |P[C_1^1]^-_{\text{Alg}}|)]\\ &\le 20d^{3/2}(r+b)^5\alpha\,\text{opt}(P),\end{aligned}$$

as $P[C_1^1]^-_{\text{Alg}} = P[\mathcal{C}_1^1]^-_{\text{opt}} = \emptyset$, proving 5.

For the inequality 6 of the base case, we have

$$\begin{aligned}\mathbb{E}[|\cup_{j\le 1} P[\mathcal{F}_{\text{out}}[j]]|] &= \mathbb{E}[|P[\mathcal{F}_{\text{out}}[1]]|]\\ &\le \frac{20d^{3/2}(r+b)^6\alpha}{\ell_{i+1}}\mathbb{E}[\text{opt}(P[\mathcal{C}_1^1]_{\text{opt}}) + \ell_1(|P[\mathcal{C}_1^1]^-_{\text{opt}}| + |P[C_1^1]^-_{\text{Alg}}|)]\\ &\le \frac{20d^{3/2}(r+b)^6\alpha}{\ell_{i+1}}\,\text{opt}(P).\end{aligned}$$

Inductive step. Now for the case $i > 0$, we have that

$$\begin{aligned}\mathbb{E}[\text{cost}_i(\mathcal{F}_{\text{out}})] &\le 20d^{3/2}(r+b)^5\alpha\mathbb{E}[\text{cost}_i(\mathcal{F}^*) + \ell_i(|P[i]^-_{\text{Alg}}| + |P[i]^-_{\text{opt}}|)]\\ &\le 20d^{3/2}(r+b)^5\alpha(\text{opt}(P) + \frac{10\sqrt{d}(r+b)\,\text{opt}(P)\ell_i}{\ell_i} + 20d^{3/2}(r+b)^5\ell_i|P[i]^-_{\text{opt}}|) \qquad (7)\\ &\le 300d^{3/2}(r+b)^6\alpha\,\text{opt}(P) + 20(r+b)^5\ell_i\mathbb{E}[|\cup_{j<i} P[\mathcal{F}_{\text{out}}[j]]|]\\ &\le 300d^2(r+b)^6\alpha\,\text{opt}(P) + 20(r+b)^5\frac{\ell_i(400d^2(r+b)^8\alpha)^{i-1}}{\ell_i}\,\text{opt}(P) \qquad (8)\\ &\le (400d^2(r+b)^8\alpha)^i\,\text{opt}(P)\end{aligned}$$

eq. (7) follows from theorem D.6, while eq. (8) follows from the inductive hypothesis, proving inequality 5.

Similarly, for inequality 6, we have

$$\begin{aligned}\mathbb{E}[|\cup_{j\le i} P[\mathcal{F}_{\text{out}}[j]]] &= \mathbb{E}[|P[\mathcal{F}_{\text{out}}[i]|] + \mathbb{E}[|\cup_{j\le i-1} P[\mathcal{F}_{\text{out}}[j]]|] \\ &\le \frac{20d^{3/2}(r+b)^6\alpha}{\ell_{i+1}}\mathbb{E}[\text{cost}_i(\mathcal{F}^*) + \ell_i(|P[i]^-_{\text{opt}}| + |P[i]^-_{\text{Alg}}|)] \\ &\quad + \frac{(400d^2(r+b)^8\alpha)^{i-1}}{\ell_i}\,\text{opt}(P) \\ &\le \frac{300d^2(r+b)^7\alpha}{\ell_{i+1}}\,\text{opt}(P) + \frac{20d^{3/2}(r+b)^5(400d^2(r+b)^8\alpha)^{i-1}}{\ell_{i+1}}\,\text{opt}(P) \\ &\quad + \frac{(400d^2(r+b)^8\alpha)^{i-1}}{\ell_i}\,\text{opt}(P) \\ &\le \frac{(400d^2(r+b)^8\alpha)^{i}}{\ell_{i+1}}\,\text{opt}(P).\end{aligned}$$

Finally,

$$\begin{aligned}\mathbb{E}[\text{cost}(\mathcal{F}_{\text{out}})] &= \sum_{i\in[L]}\mathbb{E}[\text{cost}_i(\mathcal{F}_{\text{out}})] \\ &\le \sum_{i\in[L]}(400d^2(r+b)^8\alpha)^i\,\text{opt}(P) = (\alpha d^2(r+b))^{O(1/\epsilon)}\,\text{opt}(P),\end{aligned}$$

concluding the induction.

Finally, the following corollary is used to upper bound the approximation ratio of our algorithm, proving the correctness of Theorem 3.1.

Corollary D.7. *Modify the algorithm such that in any cell where it recurses, the set of points decomposed in that cell are shifted to their respective sub-cell centers. The decomposition output this way is $(\alpha d^2(r+b))^{O(1/\epsilon)}$-approximate in expectation with respect to the original locations of the input points.*

Proof. To prove the corollary, note that on layer i in expectation we decompose at most $(\alpha d^2(r+b))^{O(1/\epsilon)}\,\text{opt}(P)/\ell_{i+1}$ points per the second inequality of the inductive argument and these shifts may increase the assignment cost of any point by an additive $O(\sqrt{d}\ell_{i+1})$ term. □

Hence, the static algorithm returns a $O(\alpha d(r+b))^O(1/\epsilon)$-approximate fairlet decomposition, as desired. This concludes the proof of Theorem 3.1.

E Our Dynamic Algorithm

In this section, we first describe the update procedures of the dynamic algorithm in Section E.1, and then prove Theorem 4.1 in Section E.2. Here, we assume that the dynamic set has balance at least r/b at all times. In Section F, we show how to modify the algorithm to handle cases in which the balance of the input set falls below r/b.

E.1 Update Procedures

Recall the main idea of the dynamic algorithm is to maintain the flow networks corresponding to each cell C of the tree, where the edges in each corresponding flow network (as defined in Appendix C) are slightly perturbed as follows. We increase the cost of every non-zero cost edge running between B', R' (i.e. edges corresponding to the pairs of nodes of G representing different sub-cells) by $2\lambda/n^3$, where λ denotes the side length of the child cells of C. We increase the cost of remaining $B' - R'$ edges by λ/n^3.

The update procedures Point-Insertion and Point-Deletion inside a single cell consist of two parts:

1. updating the corresponding minimum-cost flow solution
2. updating the underlying fair decomposition

For the first part, insertions are handled via POINT-INSERTION-CELL procedure, while deletions are handled via POINT-DELETION-CELL procedure, while the second part is the same for both types of updates. We start by describing the update procedures for the first part in detail.

POINT-INSERTION-CELL procedure. We now describe the process of a red point p insertion into a specific cell. The construction is symmetric with respect to blue point insertions. Assume p falls in cell i the red points of which are represented by r_i in G. We first update the underlying minimum-cost flow solution, and then update the underlying fairlet decomposition. The algorithm is given in Algorithm 4.

At a high level, we increase the capacity of the edge (r'_i, r_i) in both G and its residual graph G_r by 1. If the edge was saturated by the flow before the update, we compute a minimum-cost simple cycle in G_r that contains the edge (r'_i, r_i). If the cycle has negative cost, we augment the flow along it. This process is repeated t times. Next, we increase the supply at β and the demand at r_i by 1. To update the flow accordingly, we compute a minimum-cost simple path in G_r from β to r_i and augment the flow along this path.

Algorithm 4: POINT-INSERTION-CELL(r_i)

```
/* Updating capacities                                              */
for i = 1 to t do
    Increase the capacity of (r'_i, r_i) by 1 in G, update graphs G_r
    if edge (r'_i, r_i) was saturated by f_G in G then
        Π ← minimum-cost simple cycle in G_r through (r'_i, r_i)
        if cost of Π is negative then
            Augment the flow f_G along Π by 1 unit and update G_r accordingly
/* Updating demands/supplies                                        */
Increase the supply and demand of β and r_i respectively by 1 in G
Π ← Minimum cost simple β − r_i path in G_r
Augment the flow f_G along Π by 1 unit and update G_r accordingly
Update G_t according to the changes in f_G
```

POINT-DELETION-CELL procedure. Deletions are handled in a similar manner to insertions, although in reverse. Assume that a red point in the cell represented by r_i in G is to be deleted. Firstly, we decrease the supply and demand of β and r_i respectively by 1. To update the flow we find a minimum-cost simple path from r_i to β in the residual graph G_r and augment f_g along it.

We then repeat the following t times. If the edge (r'_i, r_i) in f_G is now saturated, we find a minimum cost cycle in G_r crossing (r_i, r'_i) (backwards) and augment along f_g it. We then decrease the capacity of the edge (r'_i, r_i) by 1. The algorithm is given in Algorithm 5.

Algorithm 5: POINT-DELETION-CELL(r_i)

```
/* Updating demands/supplies                                        */
Decrease the supply and demand of β and r_i respectively by 1 in G
Π ← Minimum cost simple r_i − β path in G_r
Augment the flow f_G along Π by 1 unit and update G_r accordingly
/* Updating capacities                                              */
for i = 1 to t do
    if Edge (r'_i, r_i) is saturated by f_g then
        Π ← minimum-cost simple cycle in G_r through (r_i, r'_i)
        Augment the flow f_G along Π by 1 unit and update G_r accordingly
    Decrease the capacity of (r'_i, r_i) by 1 in G, and update graphs G_r accordingly
Update G_t according to the changes in f_G
```

Algorithm 6: EnumeratePropagatedPoints(G_t, m)

$R_0, B_0 \leftarrow$ list containing 0-s for all cells C_i
for r_i, b_i *nodes in* G_t *representing the red and blue points in cell* C_i **do**
  if $\deg_{E_t}(r_i) = m(r_i) \wedge \deg_{E_t}(b_i) = m(b_i)$ **then**
    $R_0[i] = E_t(r_i, b_i)$
    $B_0[i] = E_t(r_i, b_i)$
  else
    W.L.O.G assume r_i is heavy i.e. $\deg_{E_t}(r_i) > m(r_i)$
    /* $k_{(1,1)}$ stands for the number of $(1,1)$-fairlets centered at r_i */
    $k_{(1,1)} \leftarrow$ largest integer such that $t(m(r_i) - k_{(1,1)}) \geq \deg_{E_t}(r_i) - k_{(1,1)}$
    if $t(m(r_i) - k_{(1,1)}) > \deg_{E_t}(r_i) - k_{(1,1)}$ **then**
      SpecialFairlet = True
      $t' \leftarrow t - (\deg_{E_t}(r_i) - k_{(1,1)} - t(m(r_i) - k_{(1,1)})$
    else
      SpecialFairlet = False
    if $E_t(r_i, b_i) \leq k_{(1,1)}$ **then**
      $R_0[i] \leftarrow E_t(r_i, b_i)$
      $B_0[i] \leftarrow E_t(r_i, b_i)$
    else if *SpecialFairlet = True* **then**
      if $E_t(r_i, b_i) \leq k_{(1,1)} + t'$ **then**
        if $E_t(r_i, b_i) = k_{(1,1)} + t'$ **then**
          $R_0[i] \leftarrow k_{(1,1)} + 1$
          $B_0[i] \leftarrow k_{(1,1)} + t'$
        else
          $R_0[i] \leftarrow k_{(1,1)}$
          $B_0[i] \leftarrow k_{(1,1)}$
      else
        $R_0[i] \leftarrow k_{(1,1)} + 1 + \lfloor (E_t(r_i, b_i) - k_{(1,1)} - t')/t \rfloor$
        $B_0[i] \leftarrow k_{(1,1)} + t' + \lfloor (E_t(r_i, b_i) - k_{(1,1)} - t')/t \rfloor \cdot t$
    else
      $R_0[i] \leftarrow k_{(1,1)} + \lfloor (E_t(r_i, b_i) - k_{(1,1)})/t \rfloor$
      $R_0[i] \leftarrow k_{(1,1)} + \lfloor (E_t(r_i, b_i) - k_{(1,1)})/t \rfloor \cdot t$
  Return R_i, B_i

Updating the fair decomposition. Updating the fair decomposition consists of two parts. First, EnumeratePropagatedPoints procedure decides exactly how many points are propagated downwards, while WeightedFairlet and WeightedFair procedures produce either a fairlet or a fair decomposition of the remaining points inside the cell.

We start by describing EnumeratePropagatedPoints from Algorithm 6. Each node r_i representing red points in G_r has a special neighbor b_i such that $d(r_i, b_i) = 0$, that is, the node representing the blue points in the same cell. For any zero cost edge (r_i, b_i), if both endpoints are light, then the cell sends down exactly $E_t(r_i, b_i)$ red and blue points in $(1, 1)$ fairlets, where $E_t(r_i, b_i)$ corresponds to the multiplicity of edge (r_i, b_i) in G_t.

The more complicated case is when one endpoint of such an edge is heavy, say r_i. In this case, we need to find a representation of a fairlet decomposition of the edges incident on r_i in G_t. In the decomposition there will be $m(r_i)$ fairlets centered at v_i. We will construct them as follows. First, we try to create as many $(1, 1)$-fairlets as we can. Then, we may create a single $(1, t')$ fairlet for some $t' < t$, and finally, the remaining fairlets will be $(1, t)$-fairlets.

Note that these parameters (the number of $(1, 1)$-fairlets, whether there exists a special $(1, t')$ fairlet, the value of t' and the number of $(1, t)$-fairlets) are just a function of $m(v_i)$ and $\deg_{E_t}(v_i)$. Once these parameters have been calculated, we try to create as many zero cost fairlets among them as possible using the $E_t(r_i, b_i)$ zero cost edges between $r_i - b_i$.

Once we decided which points are being propagated downward, we can compute a fair decomposition of the remaining points in the cell. In cells which contain at most n^ϵ points, the algorithm uses the WEIGHTEDFAIRLET (see Algorithm 1) procedure to obtain a fairlet decomposition after any change to their input points. For other cells, after having calculated the number of points to be pushed down in each sub-cell (R_0, B_0), the algorithm pretends as if all the remaining points are located in their respective sub-cell centers and uses the WEIGHTEDFAIR (see Algorithm 2) procedure to obtain a fair decomposition of them.

E.2 PROOF OF THEOREM 4.1

Correctness. We first show that the perturbation of distances between points in P, as introduced in Section 4, does not significantly affect the quality of the obtained t-bounded edge cover.

Claim E.1. *Let P be a set of weighted bi-colored points with total weight n placed on some grid with side length λ in $\mathbb{R}^d$. Define a shifted notion of distance for the points of P where $d'(p, p')$ takes an arbitrary value in $[d(p, p') + \lambda/n^3, d(p, p') + 2\lambda/n^3]$. Then a t-bounded edge cover of P which is of minimum cost with respect to d' is a $(1 + 1/n)$-approximate minimal t-bounded edge cover of P with respect to the distance metric d.*

Proof. Minimality follows from the fact that under d' no pairwise distance is 0 in P. Let E_t^d and $E_t^{d'}$ stand for minimal minimum cost t-bounded edge covers of P with respect to costs d and d' respectively. If $\text{cost}(E_t^d) = 0$ then the cost of E_t^d under d' is at most $|E_t^d|2\lambda/n^3 \geq \text{cost}(E_t^{d'})$. This means $E_t^{d'}$ can't contain edges connecting separate points of the grid and it has cost 0 with respect to d. Otherwise, $\text{cost}(E_t^d)$ is at least λ with respect to d. Its cost with respect to d' is at most additively $|E_t^d|2\lambda/n^3 < \lambda/n$ larger, hence $\text{cost}(E_t^d)(1 + 1/n) \geq \text{cost}(E_t^{d'})$.

□

Now we prove Lemma 4.2, i.e. we prove that the procedure POINT-INSERTION-CELL correctly maintains the minimum cost flow solution in graph G.

Proof of Lemma 4.2. Note that after the processing of an update to a cell in section 4, the updated capacity and demand/supply values reflect the state of G if in presence of the inserted point. Therefore, we only need to argue that our flow augmentation steps maintain optimality of f_G throughout the update procedures POINT-INSERTION-CELL (Algorithm 4). This can be concluded by the following two folklore lemmas.

For the insertion procedure POINT-INSERTION-CELL, we have the following two lemmas.

Lemma E.2. *Let G be a directed min-cost flow instance with integer capacities, supplies and demands and f_0 be an optimal flow for this instance. Assume G' differs from G only in the capacity of 1 edge e which has 1 more capacity in G' then G. Then starting from f_0 if we augment along a minimum-cost simple cycle containing e in the residual graph defined by G' with respect to f_0 we obtain a minimum cost solution to G'.*

Proof. If f_0 is not optimal in G' in the residual graph of G' with respect to f_0 must contain a negative cost cycle C. This cycle must contain e, as if it weren't it would have existed in the residual graph of G with respect to f_0 which contradicts f_0-optimality.

Every cycle can be decomposed into simple cycles. Hence, we can decompose C into simple cycles, one of which contains e. All other cycles generated this way must have non-negative cost due to the optimality of f_0 in G. We can augment f_0 in G' along this cycle. We will claim that the resulting flow f' is optimal in G' by contradiction.

If this wasn't the case then the residual graph of G' with respect to f' would contain a negative cost cycle C'. By combining C and C' we obtain a circulation in the residual graph of G' with respect to f_0 with cost smaller than that of C. This circulation may contain only 1 negative cost cycle incident on e due to the optimality of f_0 in G, which contradicts the minimality of the cost of C.

□

Lemma E.3. *Let G be a directed min-cost flow instance with integer capacities, supplies and demands and f_0 be an optimal flow solution for this instance. Assume G' differs from G only in the demand and supply of a pair of nodes u, v where u and v have one more demand and supply respectively in G' then in G. Then if we augment f_0 along a simple minimum cost path from v to u in the residual graph of G' with respect to f_0 we obtain an optimal flow of G'.*

Proof. Let f' be an optimal solution to the problem defined by G'. The flow $\Delta f = f' - f_0$ must send 1 unit of flow from u to v and have zero net flow across all other nodes. This implies that Δf can be decomposed into a simple u, v path P and a circulation C. As f_0 and f' satisfy the capacity constraints of G and G' the flow Δf is a valid flow in the residual graph of G' with respect to f_0.

Due to the optimality of f_0 in G no cycle in the circulation C can have negative cost. This implies that we may obtain f' from f_0 through augmenting f_0 along this path P. Hence, we may obtain an optimal solution to flow problem defined by G' starting from f_0 if we augment along a minimum cost simple u, v path in the residual graph of G' with respect to f_0, as such a path has cost at most that of P.

□

Lemmas E.2 and E.3 complete the proof of Lemma 4.2.

□

Now we present the analogue of Lemma 4.2 for the deletion procedure POINT-DELETION-CELL.

Lemma E.4. *Let G_C be the flow network associated with point set $\bar{P}_C$ in cell C, and let f be a minimum cost flow in graph G_C. Consider the deletion of red point p into P_C, and let G'_C be the flow network corresponding to the updated set $\bar{P}_C$. A minimum cost flow f' in G'_C can be obtained by performing at most $t + 1$ flow cancellations to f.*

Proof. We proceed in a similar manner to the proof of Lemma 4.2. Namely, the updated capacity and demand/supply values are maintained correctly under the deletion of a point. We now prove that the flow cancellation steps maintain the optimality of the flow.

Lemma E.5. *Let G be a directed min-cost flow instance with integer capacities, supplies and demands and f_0 be an optimal flow for this instance. Assume G' differs from G only in the capacity of 1 edge e which has 1 less capacity in G' then G. If f_0 does not saturate e, then f_0 is a minimum cost solution also to G'. If flow f_0 saturates e, then starting from f_0 if we cancel the flow along a minimum-cost simple cycle containing e in the residual graph defined by G' with respect to f_0, we obtain a minimum cost solution to G'.*

Proof. Suppose first that f_0 does not saturate e. Then it is east to see that G' cannot contain a negative cost cycle in the residual graph, since the same cycle would appear in the residual graph of G. Hence, f_0 is optimal for G'.

Now suppose that f_0 saturates e, and that the resulting flow after augmentation is not optimal in G'. Then there exists a negative cost cycle C in the residual graph of G' that contains e', the backwards edge of e, and hence a simple cycle of negative cost containing e'. However, since f_0 saturates e in G, e' also exists in G', and hence we could augment f_0 in G along C, contradicting the optimality of f_0.

□

Lemma E.6. *Let G be a directed min-cost flow instance with integer capacities, supplies and demands and f_0 be an optimal flow solution for this instance. Assume G' differs from G only in the demand and supply of a pair of nodes u, v where u and v have one less demand and supply respectively in G' then in G. Then if we cancel flow f_0 along a simple minimum cost path Π from v to u in the residual graph of G' with respect to f_0 we obtain an optimal flow of G'.*

Proof. Suppose that the resulting flow f_1 is G' is not optimal. Consider an optimal flow f' in G', and define the flow $\Delta f = f' - f_0$. Similarly as before, Δf consists of a path from u to v and some circulation C. Since Δf is also a valid flow in the residual graph of G w.r.t. f_0, this means that there

either exists a negative cost cycle in this graph, a contradicting the optimality of f_0, or that there is a path from u to v of cost smaller than Π, a contradiction about the optimality of Π.

□

Lemmas E.5 and E.6 complete the proof of Lemma E.4.

□

Hence, in each cell of the tree, our dynamic algorithm maintains the minimum-cost flow of our static algorithm from Section 3 by Lemmas 4.2 and E.4. Since the algorithm computes the $(1, t)$-fair decomposition of points assigned in each cell of the tree, the dynamic algorithm correctly maintains a $(dt)^{O(1/\epsilon)}$-approximate $(1, t)$-fair decomposition of the input set. This concludes the correctness part of Theorem 4.1.

Runtime. Now we turn to proving the runtime guarantee of Theorem 4.1. We first show that the metric perturbation used in the dynamic algorithm gives us the following insight into the structure of the augmenting paths found by the procedures POINT-INSERTION-CELL and POINT-DELETION-CELL.

Claim E.7. *Let E_t^- and E_t^+ stand for the edges deleted from and inserted into E_t due to the processing of an update at the visited node. Then $E_t^- \cup E_t^+$ can be decomposed into $O(t)$ paths $\Pi_1, \ldots, \Pi_k$, such that, if any edge in $E_t^- \cap \Pi_i$ is of length d/n^3, then it must be the starting or ending edge of Π_i, for all i.*

Proof. Consider a minimum-cost simple path Π found in the residual graph G_r at any point throughout updating the flow solution f_G. As Π is simple, whenever this path enters the node set $B \cup R$ either before and after, it must visit nodes β or ρ or terminate. Therefore, it may enter the node set $B \cup R$ at most $O(1)$ times.

Between two entry points to this node set, it might cross back and forth between nodes of $R' \cup B'$. When we augment this path on the flow solution f_G, if it contains an edge from blue to red or red to blue direction, it introduces an edge to E_t^+ or E_t^- respectively. Therefore, the exercising of this path to f_G adds $O(1)$ paths to $E_t^+ \cup E_t^-$. Assume Π_{E_t} is one of these paths Π that contributes to $E_t^+ \cup E_t^-$, and let (r'_i, b'_j) be an internal edge of this path (which appears in E_t^-). We claim that the cost of (r'_i, b'_j) can't be λ/n^3 in G_r.

Let the prior and next edges of this path be (b', r'_i) and (b'_j, r'). Exchange the sub-path $(b', r'_i), (r'_i, b'_j), (b'_j, r')$ in Π to (b', r') to obtain path Π'. We claim that Π' is of smaller cost than Π implying a contradiction. To see this let c_{G_r} stand for the cost of an edge in G_r and let $d(\cdot, \cdot)$ stand for the distance matrix of the underlying points in $\mathbb{R}^d$. Then,

$$\begin{aligned} c_{G_r}(b', r'_i) - c_{G_r}(r'_i, b'_j) + c_{G_r}(b'_j, r') &= d(b, r_i) + \frac{2\lambda}{n^3} - \frac{\lambda}{n^3} + d(b_j, r) + \frac{2\lambda}{n^3} \\ &\geq d(b, r) + \frac{3\lambda}{n^3} \\ &> c_{G_r}(b', r'), \end{aligned}$$

where the first inequality follows from the triangle inequality.

□

Now we are ready to bound the number of point deletions in sub-cells a single update in a cell can produce. Namely, we prove Claim 4.3.

Proof of Claim 4.3. Consider an update to some cell of the quad-tree. By theorem E.7, the updates to E_t consist of $O(t)$ paths alternating between deletions and insertions, and each such path may only contain 2 deleted edges representing 0 distance. In addition, the set of points present in one sub-cell might also change by 1. For a particular cell i the value of $R_0[i], B_0[i]$ corresponds to the number of its points present in 0-cost fairlets. Let r_i, b_i stand for the nodes representing cell i in G_t.

Values $R_0[i], B_0[i]$ are simply functions of the number of red and blue points in cell i and $E_t(r_i, b_i)$. Furthermore, both $R_0[i]$ and $B_0[i]$ are monotonically increasing with respect to $E_t(r_i, b_i)$ by construction of Algorithm 6. Hence, if $E_t(r_i, b_i)$ did not decrease and the set of points in cell i did not change then $R_0^-[i] \leq R_0^+[i]$ and $B_0^-[i] \leq B_0^+[i]$. This must hold for all but $O(t)$ cells. Similarly, by the construction of Algorithm 6 a single change in $m(r_i), m(b_i)$ or a unit reduction $E_t(r_i, b_i)$ might decrease $R_0[i], B_0[i]$ by at most t. So in total for all cells $R_0[i], B_0[i]$ values might decrease by $O(t^2)$.

□

We are finally ready to bound the update time of the algorithm.

For a given cell, updating the flow network G and the associated solution f_G requires time polynomial with respect to the number of its sub-cells, that is $O(tn^{d\epsilon})$. In cells where the algorithm doesn't recurse a single initialization of the algorithm of section C is required running in similar time. The calculation of R_0, B_0 values is a greedy process which can similarly be done in $O(n^{d\epsilon})$ time. Finding which points are to be deleted from or added to cells of lower layers can greedily be done in time proportional to the number of affected points.

Let ℓ_i^+ and ℓ_i^- be the number of times points ever are added to and deleted from layer i. Let layer 1 be the bounding box of the input and the bottom layer be $L = O(1/\epsilon)$. Assume the algorithm processes T updates, hence $\ell_1^+ \leq T$. For all levels $\ell_i^- \leq \ell_i^+$ and $\ell_i^+ \leq \ell_i^- + T$, as every point has to be deleted from a layer before it can be added again.

By Claim 4.3, an update to layer i might prompt ct^2 deletions on level $i+1$ for some $c > 0$. So $\ell_{i+1}^- \leq ct^2(\ell_i^+ + \ell_i^-)$. Solving the recurrence we get that $\ell_i^+, \ell_i^- \leq (2ct^2+1)^i T$, hence the sum of all point updates across all layers is at most $t^{O(1/\epsilon)}T$. As $O(tn^{d\epsilon})$ time is spent updating a cell, the amortized update time is $\tilde{O}(n^{d\epsilon}t^{O(1/\epsilon)})$, concluding the proof of Theorem 4.1.

F Handling Unbalanced Sets in the Dynamic Stream

In this section, we show how to handle dynamic streams even if they contain sets for which the balance of the current set falls below the desired fraction $\frac{r}{b}$. The algorithm is described in Algorithm 7

At the beginning, we initialize $\mathcal{D} = \emptyset$ and $P = \emptyset$, and $P' = \emptyset$, a subset of P of points we have matched so far. During the update sequence, we also maintain two queues of updates that we have not performed so far: L_1 that contains red point insertions and blue point deletions, and L_2 that contains red point deletions and blue point insertions. After each update to P, we check whether the current set P is balanced. If not, we output NULL, as the fair decomposition of P does not exist. If yes, we run the static algorithm from Section D.1 on P and update $\mathcal{D}$ and P'.

After this, during the data stream, it might happen that set $P := R \cup B$ becomes unbalanced again. In that case, we first output NULL, and then update one of the queues L_1 and L_2. When the set P becomes balanced again, we perform all $|L_1| + |L_2|$ updates on P' in a greedy fashion. Namely, we perform the updates from L_1 and L_2 in such order that, with each update, the set P' remains balanced, as described in Algorithm 7. One case that requires attention is when the current update $update_1$ from $L_1 \cup L_2$ requires deletion of point p that has not yet been added to P'. In that case, we simply skip this operation, and delete the opposite update from $L_1 \cup L_2$.

Finally, it is easy to see that this modification to the dynamic algorithm affects neither the approximation ratio of the decomposition nor the amortized running time of the algorithm. Namely, after T updates to the dynamic set, the algorithm performs at most T regular updates, as described in Section E.

Algorithm 7: GENERALUPDATE($P, update$)

```
Update P
if P = ∅ then
    D ← ∅ , Print NULL
if D = ∅ and P is unbalanced then
    Print NULL
if D = ∅ and P is balanced then
    Update D via static algorithm from Section 3
    P' ← P
else
    if P is unbalanced then
        Print NULL
        Update L_1 or L_2 with update
    else
        while L_1 ≠ ∅ or L_2 ≠ ∅ do
            if (|P' ∩ A| < |P' ∩ B| and L_1 ≠ ∅) or L_1 = ∅ then
                update_1 ← L_1.pop()
            else
                update_1 ← L_2.pop()
            if update_1 = deletion of p and p ∉ P' then
                Remove (insertion of p) from L_1 ∪ L_2
            else
                Update D via UPDATE(P', update_1)
```