Fair Center Clustering in Sliding Windows

Matteo Ceccarello
matteo.ceccarello@unipd.it
University of Padova, Italy

Andrea Pietracaprina
andrea.pietracaprina@unipd.it
University of Padova, Italy

Geppino Pucci
geppino.pucci@unipd.it
University of Padova, Italy

Francesco Visonà
francesco.visona.3@studenti.unipd.it
University of Padova, Italy

Abstract

The k-center problem requires the selection of k points (centers) from a given metric pointset W so to minimize the maximum distance of any point of W from the closest center. This paper focuses on a fair variant of the problem, known as *fair center*, where each input point belongs to some category and each category may contribute a limited number of points to the center set. We present the first space-efficient streaming algorithm for fair center in general metrics, under the sliding window model. At any time t, the algorithm is able to provide a solution for the current window whose quality is almost as good as the one guaranteed by the best, polynomial-time sequential algorithms run on the entire window, and exhibits space and time requirements independent of the window size. Our theoretical results are backed by an extensive set of experiments on both real-world and synthetic datasets, which provide evidence of the significantly better performance/quality tradeoffs attained by our algorithm with respect to the those achievable by running the state-of-the-art sequential baselines on the entire window.

Keywords

Clustering, k-center, Algorithmic Fairness, Streaming, Doubling dimensions, Matroid constraints

1 Introduction

Clustering is a fundamental primitive for machine learning and data mining, with applications in many domains [19]. One popular variant is k-center clustering, which has also been intensely studied in the realm of facility location [31]. Given a set of points W from a metric space and an integer $k \leq |W|$, the k-center problem requires to select k points (dubbed *centers*) from W, which minimize the maximum distance of any point of W from its closest center. The centers induce immediately a partition of W into k *clusters*, one per center, where each point is assigned to the cluster associated to its closest center. When the distance between points models (dis)similarity, each center can then be regarded as a suitable representative for all the points in the corresponding cluster. Indeed, efficient clustering approaches have been traditionally used for two main purposes, one being the partitioning of the dataset into groups of similar data points for unsupervised classification, the other being the selection of a small number of representative points (the centers of the clusters) that are good descriptors of the entire dataset. In particular, k-center has been often employed as a summarization primitive to extract succinct coresets from large datasets, where computationally expensive analyses can then be performed (e.g., see [4, 6] and references therein).

EDBT '26, Tampere (Finland)
© 2025 Copyright held by the owner/author(s). Published on OpenProceedings.org under ISBN 978-3-89318-103-2, series ISSN 2367-2005. Distribution of this paper is permitted under the terms of the Creative Commons license CC-by-nc-nd 4.0.

Applying mainstream clustering algorithms to process data associated to user communities may often generate biased results, potentially causing discriminatory effects which reinforce social inequalities in target applications [9, 13]. In data summarization, unfairness may come in the form of a selection of representatives that do not reflect the demographics of the entire population. Kay et al. [23] found that Google Images queries for professions return a selection of images that often do not reflect the actual gender balance of the professions. For instance the actual percentage of women among *bartenders* ($\approx$ 60%), *technical writers* ($\approx$ 55%), *computer programmers* ($\approx$ 22%) and *bus drivers* ($\approx$ 45%) is very different from the percentage of women in the corresponding image results (respectively $\approx$ 23%, $\approx$ 35%, $\approx$ 16%, and $\approx$ 18%).

In order not to discriminate some groups of individuals with respect to some specific, sensible attribute (e.g., ethnicity, gender, or political views), when clustering is used for summarization purposes, it is of utmost importance that the centers provide a *fair* representation of the population with respect to this attribute. To this end, in *fair center clustering*, we label each point with a *color*, representing the attribute value for the group which the corresponding individual belongs to, and we impose an upper limit on the number of returned centers from each color, so that no color, hence no group, is over-represented in the output. We remark that blindly ignoring sensible attributes does not imply a fair solution [13]: in fact, group membership information can leak from other features of the points, or the features themselves might be discriminative towards some group.

Scalable versions of this fair k-center variant have recently been studied in distributed and insertion-only streaming settings [6]. In this work, we devise and experiment with the first efficient streaming algorithm for fair center clustering under the more challenging, heavily studied sliding-windows model [12], where the sought solution at any time must refer only to a fixed-size window W of the most recent data, disregarding older elements of the stream. This model is applied in scenarios where the recent data is more relevant than historical data, either because current trends are of interest or because regulations like the General Data Protection Regulation (GDPR) require to retain user information only for a limited amount of time [14, Recital (31) and Article 5.1(e)].

1.1 Related work

The k-center problem, being a fundamental primitive in many data analysis workflows, has been studied extensively in the past decades. For brevity, in this section we give an account of those results that relate most closely to our work.

In the sliding window model, [11, 28] provide approximation algorithms for the unrestricted k-center problem without fairness constraints. The variant of k-center with outliers in the sliding window model has been studied in [29]. A relevant generalization of the k-center problem is the *matroid center* problem, where the

10.48786/edbt.2026.17

centers of the clustering are required to be an independent set of a given matroid. The seminal work of [8] provided the first 3-approximation sequential algorithm for this problem, albeit featuring a high computational complexity. The matroid center problem has also been studied in [6, 22, 30] in the insertion-only streaming and in the fully-dynamic settings yielding in all cases a $(3+\varepsilon)$ approximation and also allowing for the presence of outliers.

The fair center problem studied in this paper can be seen as a specialization of the matroid center problem for the so-called *partition matroid*, built on points belonging to different categories and whose independent sets are those containing no more than a fixed upper limit of points per category. In [24] the authors provide a sequential $(3 \cdot 2^{\ell-1} - 1)$-approximation algorithm for fair center (where ℓ is the number of categories), with a runtime linear both in k and in the number of points. Later, [20] improved the approximation factor to 3 while retaining the same time complexity. [2] studies the fair clustering problem in the presence of outliers, providing a sequential, randomized bicriteria approximation algorithm. A further specialization of the problem, where each center is required to represent at least a given fraction of the input points, in addition to respecting the fairness constraints, has been studied by [3]. In the insertion-only streaming setting, the fair center problem has been considered in [10, 15, 26]. These papers all yield 3 approximations for the problem, with improvements in the working space requirements over the general matroid center algorithms discussed before. To the best of our knowledge, ours is the first work to address the fair center problem in the more challenging sliding window setting.

We wish to stress that, although widely studied, the notion of fairness in center selection adopted in this paper is not the only possible one. For instance, [21, 27] consider *individually fair* clustering, where each point is required to have at least one center among its n/k nearest neighbors. [18] also considers this problem in the presence of outliers. Also, observe that all the approaches discussed so far refer to enforcing fairness conditions on the centers, and are thus tightly associated to the notion of clustering as a summarization primitive. There is a very prolific line of research which regards fairness as a set of balancing constraints on the *elements* of the clusters rather than on their centers (see [1, 5] and references therein). For a comprehensive survey on different formulations of fair clustering and on their downstream applications, we refer the interested reader to [9].

Finally, it is worth mentioning that the same notion of fairness studied in this paper has been intensely investigated in the realm of *diversity maximization* [4, 25, 32–35], a problem which can be envisioned as the dual counterpart of clustering.

1.2 Our contribution

We present the first sliding window algorithm for fair center clustering in general metric spaces, which, at any time t, is able to provide an accurate solution for the current window, and requires space and time independent of the window size. More specifically, let S denote a potentially infinite stream of points from a metric space, and let $n > 0$ be the target window size. Each point of S is associated with one of ℓ colors and fairness is modeled by requiring that any feasible solution to the problem contains at most k_i points of color i, for every $1 \leq i \leq \ell$. Let $\varepsilon \in (0, 1)$ be a fixed accuracy parameter, and let Δ be the aspect ratio of S (i.e., the ratio between maximum and minimum pairwise distance). Also, define $k = \sum_{i=1}^{\ell} k_i$ and let α denote the best approximation ratio guaranteed by a polynomial-time sequential algorithm for fair center (currently $\alpha = 3$ [20]). Our main contributions are the following (detailed statements are found in Section 3).

- An algorithm that at any time t returns an $(\alpha+\varepsilon)$-approximate solution to fair center for the current window W_t, requiring working space $m = O\left(k^2 \log \Delta (c/\varepsilon)^{D_{W_t}}\right)$, for a fixed constant c, where D_{W_t} is the doubling dimension of W_t (formally defined in Section 2), which generalizes the notion of Euclidean dimensionality to general metrics. The algorithm's update (resp., query) time to handle a new arrival (resp., to compute a solution) is $O(m)$ (resp., $O(km)$), thus *both independent of the window size* $|W_t|$ (for low-dimensional streams).
- A modification of the above algorithm that at any time t is able to return a $O(1)$-approximate solution to fair center for W_t, requiring working space $m = O\left(k^2 \log \Delta/\varepsilon\right)$, update time $O(m)$ and query time $O(km)$. Thus, in this algorithm *the exponential dependency on the doubling dimension D_{W_t} has been removed*, at the expense of a weaker, but still constant, approximation ratio.
- Extensive experimental evidence on real datasets that the above algorithms indeed provide solutions of quality comparable to the best sequential algorithms run on the entire window, *using only a fraction of the space and being orders of magnitude faster.*

To the best of our knowledge, ours are the first accurate, space and time efficient sliding window algorithms for fair center clustering. Our algorithms build upon the coreset-based strategy used in [11, 28] for the unconstrained k-center problem, but introduce non trivial, crucial modifications in the coreset construction to ensure that accurate fair solutions can be extracted from the coreset.

We remark that although the performance of our most accurate algorithm is expressed in terms of the doubling dimension D_{W_t}, the algorithm does not require the knowledge of this parameter to run, which is very desirable in practice, since the doubling dimension is hard to estimate. Similarly, the aspect ratio Δ of the current window need not be provided as an explicit input to our algorithms, but it can be inferred by maintaining good estimates of the minimum and maximum pairwise distances of the points in W_t (hence, of Δ) without worsening its theoretical and practical performance.

At the very end of Section 3, after our new algorithm and its analysis have been presented in full, we provide a detailed account of the main differences of our approach with respect to previous work.

Organization of the paper. The rest of the paper is structured as follows. Section 2 defines the problem, the computational model and a number of basic notions. Section 3 presents the algorithm (Subsection 3.1) and its analysis (Subsection 3.2). Section 4 reports on the experimental results. Section 5 closes the paper with some concluding remarks. For ease of reading, Table 1 provides a comprehensive list of all the notations used in the paper.

2 Preliminaries

Problem definition. Consider a metric space $\mathcal{X}$ equipped with a distance function $d(\cdot,\cdot)$. For $x \in \mathcal{X}$ and $W \subseteq \mathcal{X}$, let $d(x, W) = \min\{d(x, w) : w \in W\}$ denote the minimum distance of x from a point in W. For a given $W \subseteq \mathcal{X}$ and a natural k, the classic *k-center* clustering problem [16] requires to find a subset $C \subseteq W$ of size at most k minimizing the *radius* $r_C(W) = \max_{p \in W} d(p, C)$.

Table 1: Notations used in this paper

$(\mathcal{X}, d(\cdot,\cdot))$	metric space
$\mathcal{M} = (\mathcal{X}, I)$	matroid on $\mathcal{X}$
$S \subseteq \mathcal{X}$	stream
W_t	window, at time t, of the last n points of S
D_{W_t}	doubling dimension of W_t
$r_C(W)$	radius of C w.r.t. window W
ℓ	number of colors
k_i	max num of points of color i in an independent set
$t(p)$	arrival time of point p
$TTL(p)$	Time-To-Live of point p
d_{min}, d_{max}	min/max distance between points of S
Δ	aspect ratio d_{max}/d_{min}
Γ	set of guesses on the optimal radius
$AV_{\gamma,t}, A_{\gamma,t}$	sets of v-/c-attractors for guess γ at time t
$RV_{\gamma,t}, R_{\gamma,t}$	sets of v-/c-representatives for guess γ at time t
$\psi_\gamma(p)$	v-attractor for point p for guess γ
$\phi_\gamma(p)$	c-attractor for point p for guess γ
$repV_{\gamma,t}(a)$	v-representative of v-attractor $a \in AV_{\gamma,t}$
$repsC_{\gamma,t}(a)$	c-representatives of c-attractor $a \in A_{\gamma,t}$

Observe that any set of centers C defines a natural partition of the points of W into *clusters*, by assigning each point of W to its closest center in C (with ties broken arbitrarily).

In this paper, we study the following variant of the k-center problem, dubbed *fair center*. Assume that each point in $\mathcal{X}$ is associated with one of a finite set of ℓ categories (dubbed *colors* in the following). For a given $W \subseteq \mathcal{X}$ and positive integers $k_1, k_2, \ldots, k_\ell$, a solution to the fair center problem is a set $C \subseteq W$ of centers minimizing $r_C(W)$, under the additional constraint that C contains at most k_i centers of the i-th color, for $1 \leq i \leq \ell$. We denote with OPT_W the radius of an optimal fair center solution for W.

The above fairness constraint can be envisaged as a special case of the more general class of *matroid constraints*, that have been studied in the context of clustering starting from [8]. Given a ground set $\mathcal{X}$, recall that a matroid $\mathcal{M}$ on $\mathcal{X}$ is a pair $\mathcal{M} = (\mathcal{X}, I)$, where $I \subseteq 2^{\mathcal{X}}$ is a family of *independent sets* featuring the following two properties: (a) *downward closure* (if $P \in I$ and $P' \subseteq P$ then $P' \in I$); and (b) *augmentation property* (if $P, Q \in I$ and $|P| > |Q|$ then $\exists x \in P \setminus Q$ such that $Q \cup \{x\} \in I$). An independent set is *maximal* if it is not a proper subset of any other independent set. Observe that as a consequence of the augmentation property, every maximal independent set in a matroid $\mathcal{M}$ has the same cardinality, which is denoted with rank (M). Furthermore, any subset $W \subseteq \mathcal{X}$ induces a (sub)matroid (W, I'), with $I' = \{Y \cap W : Y \in I\}$. We say that a subset $Y \subseteq W$ is a maximal independent set w.r.t. W, if $Y \in I'$ is a maximal independent set of this submatroid.

Given a matroid $\mathcal{M} = (\mathcal{X}, I)$ and a set $W \subseteq \mathcal{X}$, the *matroid center* problem seeks an *independent set* $C \in I$ minimizing $r_C(W)$. For fixed $k_1, k_2, \ldots, k_\ell$ and $W \subseteq \mathcal{X}$, the constraint to be imposed on the solution of fair center can be seen as a matroid constraint with respect to the so-called *partition matroid* of rank $k = \sum_{i=1}^{\ell} k_i$, where the family of independent sets contains all subsets of $\mathcal{X}$ with at most k_i points of each color i, for $1 \leq i \leq \ell$. This implies that any algorithm for the matroid center problem can be immediately specialized to solve the fair center problem.

Doubling metric spaces. Given $W \subseteq \mathcal{X}$, a point $x \in W$ and a real value $r > 0$, the *ball* of radius r centered in x, denoted by $B(x, r) \subseteq W$, is the set $\{p \in W : d(x, p) \leq r\}$. The *doubling dimension* of W is the minimum value D such that, for all $x \in W$, $B(x, r)$ is contained in the union of at most 2^D balls of radius $r/2$. The concept of doubling dimension generalizes the notion of dimensionality of Euclidean spaces and has been used in a wide variety of applications (see [17, 28] and references therein).

Sliding windows model. A *stream* S is a potentially infinite ordered sequence of points from some (metric) space $\mathcal{X}$. At each (discrete) time step $t = 1, 2, \ldots$, a new point p arrives, and we denote with $t(p)$ its arrival time. Given an integer n and a time t, the *window* $W_t \subseteq \mathcal{X}$ at time t of size n is the (multi-)set of the last n points of the stream S. Solving a problem on the sliding windows model entails maintaining data structures that can be queried at any time $t > 0$ to return the solution of the problem for the instance W_t. In the sliding window model, the key performance metrics are (a) the amount of *space* used to store the data structures; (b) the *update time* required to handle the arrival of a point p at time t; and (c) the *query time*, that is the time to extract the solution for window W_t from the data structures.

3 Fair-center for sliding windows

Consider a stream S of colored points from a metric space $\mathcal{X}$ with distance function $d(\cdot,\cdot)$, and let $n > 0$ be the target window size. We let $\mathcal{M} = (\mathcal{X}, I)$ be a partition matroid defined on $\mathcal{X}$, whose independent sets are subsets containing at most $\leq k_i$ points of each color i, for all $1 \leq i \leq \ell$, and whose rank is $k = \sum_{i=1}^{\ell} k_i$. In this section, we present our algorithm that, at any time $t > 0$, is able to return an accurate solution to fair center for the current window W_t of size n. The algorithm is described in Subsection 3.1 and its accuracy, as well as its time and space requirements, are analyzed in Subsection 3.2.

3.1 Algorithm

The algorithm builds upon the one presented in [28] for the unconstrained k-center problem in sliding windows, but it introduces crucial modifications which allow to handle the fairness constraint. Also, a significant simplification of the employed data structures affords more elegant and intuitive correctness and performance analyses with respect to all previous approaches for unconstrained k-center [11, 28, 29].

At any time step t, for a point $p \in S$ with $t(p) \leq t$, its *Time-To-Live* (TTL), denoted as $\text{TTL}(p)$, is the number of remaining steps $\geq t$, in which p will be part of the current window, namely $\text{TTL}(p) = \max\{0, n-(t-t(p))\}$, where n is the size of the window. We say that p is *active* at any time when $\text{TTL}(p) > 0$. $\text{TTL}(p)$ decreases at every time step, and we say that p *expires* at the time when $\text{TTL}(p)$ becomes 0 (i.e., at time $t(p) + n$). Let $d_{\min}$ and $d_{\max}$ be, respectively, the minimum and maximum pairwise distance between points of S, and define $\Delta = d_{\max}/d_{\min}$, which we refer to as the *aspect ratio* of S. For a fixed parameter $\beta > 0$, we define the following set of guesses for the optimal radius (which, being a distance between two points, clearly falls in $[d_{\min}, d_{\max}]$):

$$\Gamma = \left\{(1+\beta)^i : \lfloor \log_{1+\beta} d_{\min} \rfloor \leq i \leq \lceil \log_{1+\beta} d_{\max} \rceil\right\}^1.$$

[1] For ease of presentation, we assume that $d_{\min}$ and $d_{\max}$ are known to the algorithm. However, we remark that the same techniques introduced in [28] can be employed to provide estimates of these quantities and to make Γ adaptive to the aspect ratio of the current window, rather than of the entire stream.

In broad terms, the algorithm maintains, for each guess $\gamma \in \Gamma$, suitable sets of active points whose overall size is independent of n. At any time t, from these points it will be possible to identify a guess providing a tight lower bound on the optimal radius, and, based on this guess, to extract a small coreset which embodies a provably good approximate solution to fair center for W_t. A query will then compute such a solution by running the best sequential algorithm available for fair center on the coreset.

More specifically, for each $\gamma \in \Gamma$, the algorithm maintains two families of active points: *validation points* and *coreset points* In turn, each of these two families consists of two (not necessarily disjoint) sets: namely, AV_γ (v-*attractors*), and RV_γ (v-*representatives*), for validation points; and A_γ (c-*attractors*), and R_γ (c-*representatives*), for coreset points. Intuitively, *attractors* are used by the update algorithm to ensure that all the points of the window are well covered, i.e., each of them has a sufficiently close attractor, while *representatives* include the most recent, relevant points for each attractor. The reason for distinguishing between *validation* and *coreset* points, for each radius guess γ, is that the former are employed to establish whether γ is a good approximation to the optimal radius for unconstrained k-center (which, in turns, is a lower bound to the optimal radius for the fair variant), while the latter provide a set of candidate centers from which a solution to fair k-center clustering can be constructed. Because of these different roles, colors will be relevant only for the selection of the coreset points.

These sets are updated after the arrival of each new point of the stream. Set AV_γ contains at most $k+1$ points at pairwise distance $\geq 2\gamma$. Each v-attractor $v \in AV_\gamma$ is paired with a single v-representative denoted as $repV_\gamma(v)$, which is a recent point of the stream at distance at most 2γ from v, and it is included in RV_γ. We remark that there may be v-representatives that are not paired with any v-attractor. Indeed, when a v-attractor $v \in AV_\gamma$ expires, its v-representative $repV_\gamma(v)$ is not further updated, but it remains in RV_γ until it expires or it is expunged by a suitable clean-up procedure.

A key difference with the algorithm in [28] is in the choice of coreset points, which must now be empowered to account for the fairness constraint. Let $\delta \in (0, 4]$ be a fixed, user-defined precision parameter that is closely related to the resulting approximation quality of the algorithm (see Theorem 3). The set A_γ contains points at pairwise distance $\geq \delta\gamma/2$, where the lower bound is factor $\delta/4$ less than the least distance 2γ between v-attractors in AV_γ. (Observe that only values $\delta \leq 4$ make sense.) There is no fixed upper bound on its size, which, however, will be conveniently bounded by the analysis. Upon arrival, a new point p is either *attracted* by a conveniently chosen point in A_γ at distance $\leq \delta\gamma/2$, if any, or is added to A_γ (and attracts itself). For each $a \in A_\gamma$, the algorithm maintains a *set* of c-representatives $repsC_\gamma(a)$, which is a maximal independent set of active points attracted by a with the longest remaining lifespan, and is included in R_γ. For $i \in [1, \ell]$, we let $repsC^i_\gamma(v) \subseteq repsC_\gamma(v)$ denote the (at most k_i) points of $repsC_\gamma(v)$ of color i. After a c-attractor $a \in A_\gamma$ expires, its set of c-representatives $repsC_\gamma(a)$ is not further updated, but each of its elements remains in R_γ until it expires or it is expunged by the aforementioned clean-up procedure.

Clearly, all of the above sets of validation and coreset points evolve with time and, whenever we need to refer to one such set at a specific time t, we will add t as a second subscript.

Recall that $k = \sum_{i=1}^{\ell} k_i$. We say that a guess $\gamma \in \Gamma$ is *valid* at time t, if $|AV_{\gamma,t}| \leq k$. It is easy to see that if $\gamma \in \Gamma$ is not valid, (i.e., $|AV_{\gamma,t}| \geq k+1$), then γ is a lower bound to the optimal radius for unconstrained k-center on W_t, hence to the optimal radius for fair center on W_t, which can only be as large. The main purpose of validation points is to keep track of current valid guesses, which, as explained above, may be identified by referring to the unconstrained version of k-center, and thus are handled by the algorithm irrespective of their color. Instead, coreset points provide, for each valid guess, a reasonably sized subset of the window points, so that any other window point is "well represented" by a nearby coreset point of the same color, which ensures that an accurate fair solution on the coreset points is also an accurate solution for the entire window. Thus, running a sequential algorithm solely on the coreset points of a valid guess will yield a close approximation of the solution obtainable by running the algorithm on the entire window, while significantly reducing time and memory usage. For each guess, the number of coreset points will depend on k, on the precision parameter δ, and, most importantly, on the doubling dimension D_{W_t} of the current window. However, we remark that while k and δ are parameters that must be given in input, the algorithm does not require the knowledge of the doubling dimension D_{W_t}, which will be used only in the analysis.

In what follows, we describe in more details the operations performed at time step t to handle the arrival of a new point p and, if required, to compute a solution to fair center for the current window W_t.

3.1.1 Update procedure. The arrival of a new point p is handled by procedure Update(p) (Algorithm 1). For every guess $\gamma \in \Gamma$, the data structures are updated as follows. If p is at distance greater than 2γ from any other v-attractor, then p is added to AV_γ, and to RV_γ as representative of itself, otherwise it becomes the new representative of some (arbitrary) v-attractor v with $d(v, p) \leq 2\gamma$, whose previous representative is discarded. In the former case, a procedure Cleanup(p, γ) (Algorithm 2) is invoked to reduce the size of the various sets as follows. If $|AV_\gamma| = k+2$ the v-attractor with minimum TTL is removed from AV_γ. After this, if $|AV_\gamma| = k+1$ all c-attractors, v-representatives, and c-representatives with TTL less than the minimum TTL t_{min} of a v-attractor are removed from A_γ. These removals are justified by the fact that if $|AV_\gamma| = k+1$, then AV_γ acts as a certificate that γ is not a valid guess until time $t + t_{min}$, hence, for that guess, there is no need to keep points that expire earlier than that. For every point p, we denote with $\psi_\gamma(p)$ the v-attractor v such that $p = repV_\gamma(v)$ after the execution of Update(p). We remark that $\psi_\gamma(p)$ is fixed once and forall when p arrives. In fact, $\psi_\gamma(p)$ is not explicitly stored in the data structures, but is solely needed for the analysis. Coreset points are updated as follows. If p is at distance greater than $\delta\gamma/2$ from any other c-attractor, then p is added to A_γ. Otherwise, suppose that p is of color i. Then p is attracted by a and added to the set $repsC_\gamma(a)$, where a is chosen as the c-attractor at distance at most $\delta\gamma/2$ from p with minimum $|repsC^i_\gamma(a)|$. In case $|repsC^i_\gamma(a)| > k_i$ (i.e., $|repsC^i_\gamma(a)| = k_i + 1$), the representative in $|repsC^i_\gamma(a)|$ with minimum TTL is removed from the set. Observe that, unlike AV_γ, the size of A_γ can grow larger than $k+2$. The maximum size of these sets, however, will be conveniently upper bounded by the analysis. For every point p, we denote with $\phi_\gamma(p)$ the c-attractor such that $p \in repsC_\gamma(\phi_\gamma(p))$, after the execution of Update(p). Again, $\phi_\gamma(p)$ is fixed once and forall when p arrives, and it is not explicitly stored in the data structures, since it is solely needed for the analysis.

Algorithm 1: UPDATE(p)

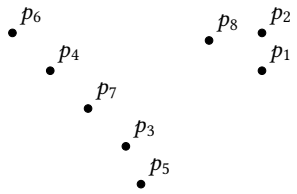
$\delta \leftarrow$ user-defined precision parameter in $(0, 4]$
$i \leftarrow$ color of p
$x \leftarrow$ point expired when p arrives
foreach $\gamma \in \Gamma$ **do**
  Remove x from any set $AV_\gamma, A_\gamma, RV_\gamma, R_\gamma$ containing it;
  /* Identify which attractors p can be a representative for */
1 $EV \leftarrow \{v \in AV_\gamma : d(p, v) \leq 2\gamma\}$;
2 $E \leftarrow \{v \in A_\gamma : d(p, v) \leq \delta\gamma/2\}$;
  /* Assign p to a v-attractor $\psi_\gamma(p)$ */
3 **if** $EV = \emptyset$ **then**
4 $AV_\gamma \leftarrow AV_\gamma \cup \{p\}$;
5 $\psi_\gamma(p) \leftarrow p$;
6 $repV_\gamma(\psi_\gamma(p)) \leftarrow p$;
7 $RV_\gamma \leftarrow RV_\gamma \cup \{repV_\gamma(\psi_\gamma(p))\}$;
8 CLEANUP(p, γ);
9 **else**
10 $\psi_\gamma(p) \leftarrow$ arbitrary element of EV;
11 $repV_\gamma(\psi_\gamma(p)) \leftarrow p$
  /* Assign p to a c-attractor $\phi_\gamma(p)$ */
12 **if** $E = \emptyset$ **then**
13 $A_\gamma \leftarrow A_\gamma \cup \{p\}$;
14 $\phi_\gamma(p) \leftarrow p$; $repsC_\gamma(\phi_\gamma(p)) \leftarrow \{p\}$;
15 $R_\gamma \leftarrow R_\gamma \cup repsC_\gamma(\phi_\gamma(p))$;
16 **else**
17 $\phi_\gamma(p) \leftarrow \arg\min_{a \in E} |repsC^i_\gamma(a)|$;
  /* Update $repsC^i_\gamma(\phi_\gamma(p))$ */
18 $repsC^i_\gamma(\phi_\gamma(p)) \leftarrow repsC^i_\gamma(\phi_\gamma(p)) \cup \{p\}$;
19 **if** $|repsC^i_\gamma(\phi_\gamma(p))| > k_i$ **then**
20 $o_{rem} \leftarrow \arg\min_{o \in repsC^i_\gamma(\phi_\gamma(p))} TTL(o)$;
21 Remove o_{rem} from $repsC_\gamma(\phi_\gamma(p))$ in R_γ;

Algorithm 2: CLEANUP(p, γ)

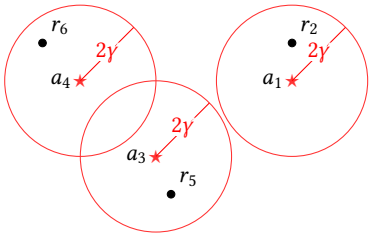
if $|AV_\gamma| = k + 2$ **then** ▷ remove the oldest v-attractor
  $v_{old} \leftarrow \arg\min_{v \in AV_\gamma} TTL(v)$;
  $AV_\gamma \leftarrow AV_\gamma \setminus \{v_{old}\}$;
if $|AV_\gamma| = k + 1$ **then** ▷ remove unnneeded points
  $t_{min} \leftarrow \min_{v \in AV_\gamma} TTL(v)$;
  Remove all q with $TTL(q) < t_{min}$ from A_γ, RV_γ, and R_γ ;

In what follows, we give two examples which illustrate how the validation and coreset points are maintained by the update procedure upon the arrival of stream points.

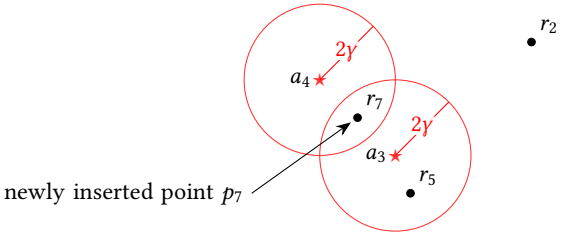
Example 1. The following example considers the setting with $k = 3$ and a window of size $w = 6$ points, focusing on the validation sets AV_γ and RV_γ, for some guess γ. The next figure reports all the points that will be used throughout the example, with the subscript denoting the time instant t at which they arrive in the stream. In this example we ignore the color of each point, as this information is not used to construct the validation sets.



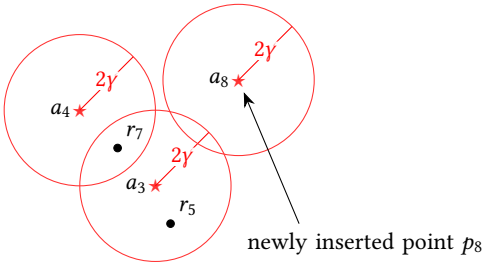
Consider the situation depicted in the following figure at the end of time $t = 6$, where the points $p_1, p_2, \ldots, p_6$ have been relabeled as v-attractors (a_ℓ) or v-representatives (r_ℓ) depending on the role assigned to them by the algorithm.



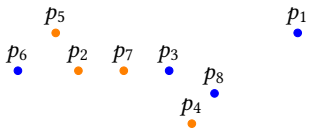
At time $t = 7$ point p_7 enters the window, while point a_1 expires and is removed from the attractor set AV_γ. However, a_1's v-representative r_2 is *not* removed from the set of v-representatives RV_γ. As for the newly inserted p_7, it can be assigned as v-representative to either a_4 or a_3. Suppose that it is assigned to a_4 as its v-representative, replacing r_6 in this role (thus, we rename it r_7). Thus, after the update at time $t = 7$ the situation is the following:



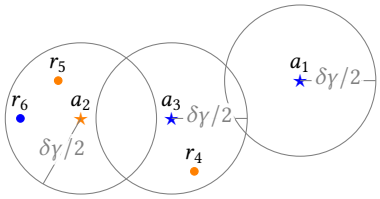
Now, at time $t = 8$ point p_8 enters the window and r_2 expires. Since p_8 is not within the ball of radius 2γ around any v-attractor, then it becomes a v-attractor itself (hence, we relabel it as a_8), and it becomes its own representative as well:



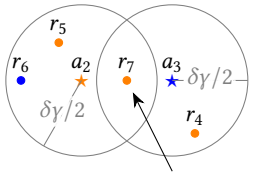
Example 2. We now give an example for the update of the sets of coreset points, $A_\gamma$and R_γ, using a stream of eight points (different from the one of the previous example) featuring two colors, $i = 1$ and $i = 2$. We impose fairness constraints $k_1 = 1$ and $k_2 = 2$, hence $k = 3$, and window length $w = 6$. The points used in the example, together with their colors, are shown in the following figure, where the subscript denotes the time at which the point enters the stream.



The situation at the end of time $t = 6$ is depicted in the following figure, with the c-attractor set $A_\gamma = \{a_1, a_2, a_3\}$. Namely, c-attractor a_1 is the representative of itself, of color $i = 1$, while a_2 has two c-representatives, including itself, of color $i = 2$ and one of color $i = 1$. Finally, c-attractor a_3 has one c-representative (itself) of color $i = 1$ and one c-representative of color $i = 2$.

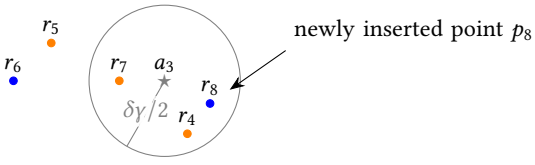


Time $t = 7$ sees the expiration of point a_1 and the insertion of point p_7 of color $i = 2$, which is within distance $\delta\gamma/2$ from both a_2 and a_3. Since a_2 already has $k_2 = 2$ c-representatives of color $i = 2$, then p_7 is assigned as a c-representative for c-attractor a_3, giving the following situation:



newly inserted point p_7

At time $t = 8$, point p_8 of color $i = 1$ enters the window and point a_2 expires. Since p_8 is at distance at most $\delta\gamma/2$ from a_3 it should be assigned as one of a_3's c-representatives. However, at this point a_3 would have $2 > k_1$ c-representatives of color $i = 1$. Therefore a_3, the oldest among the c-representatives of a_3 of color $i = 1$, is removed from set R_γ. Note, however, that being also a c-attractor that is yet to expire, a_3 is not removed from A_γ. Being no longer part of the c-representatives, a_3 is pictured in gray in the following picture, since its color is now immaterial to the algorithm:



In the above examples, for simplicity we have depicted a scenario where the cleanup procedure (which amounts to a simple purge of the oldest attractors and representatives in the data structures when γ is an invalid guess) does not perform any action.

3.1.2 Query procedure. At any time t, to obtain a solution to the fair center problem for W_t, procedure Query() is invoked (Algorithm 3). First, a valid guess γ is identified such that a k-center clustering of radius $\leq 2\gamma$ for the points in RV_γ is found, and for any smaller guess $\gamma' \in \Gamma$ with $\gamma' < \gamma$, there are $k + 1$ points in either $AV_{\gamma'}$ or $RV_{\gamma'}$ with pairwise distance $> 2\gamma'$. As it will be shown in the analysis, this ensures that the coreset points in R_γ contain a good solution to fair center for W_t, which is then computed by invoking the best sequential fair center algorithm $\mathcal{A}$.

Algorithm 3: Query()

```
for increasing values of γ ∈ Γ such that |AV_γ| ≤ k do
    C ← ∅;
    foreach q ∈ RV_γ do
        if C = ∅ ∨ d(q, C) > 2γ then C ← C ∪ {q} ;
        if |C| > k then Break and move to the next guess ;
    if |C| ≤ k then
        return A(R_γ) ;      /* A = sequential fair center algorithm */
```

3.2 Analysis

The following lemma shows that at any time t, the points RV_γ (resp., R_γ) are within distance 4γ (resp., $\delta\gamma$) from all points of the entire W_t, when γ is valid, or of a suitable suffix of W_t, otherwise. (The lemma is similar to [28, Lemma 1] but the proof given below is completely new and applies to the simpler version of the update procedure devised in this paper.)

Lemma 1. *For every $\gamma \in \Gamma$ and $t > 0$, the following properties hold after the execution of the Update procedure for the point arrived at time t:*

(1) If $|AV_{\gamma,t}| \leq k$, then $\forall q \in W_t$ we have:
  (a) $d(q, RV_{\gamma,t}) \leq 4\gamma$;
  (b) $d(q, R_{\gamma,t}) \leq \delta\gamma$;
(2) If $|AV_{\gamma,t}| > k$, then $\forall q \in W_t$ s.t. $t(q) \geq \min_{v \in AV_{\gamma,t}} t(v)$ we have:
  (a) $d(q, RV_{\gamma,t}) \leq 4\gamma$
  (b) $d(q, R_{\gamma,t}) \leq \delta\gamma$;

Proof. For every $\gamma \in \Gamma$ and $t > 0$, we say that a point $q \in W_t$ is *relevant for γ and t* if $|AV_{\gamma,t}| \leq k$ or $|AV_{\gamma,t}| > k$ and $t(q) \geq \min_{v \in AV_{\gamma,t}} t(v)$. Clearly, the proof concerns only relevant points. Also, recall that for every point p of the stream, in Update(p) we set $\psi_\gamma(p)$ (resp., $\phi_\gamma(p)$) to its assigned v-attractor (resp., c-attractor). Clearly, $d(p, \psi_\gamma(p)) \leq 2\gamma$, and $d(p, \phi_\gamma(p)) \leq \delta\gamma/2$. (In fact, the values $\psi_\gamma(p)$ and $\phi_\gamma(p)$ are not explicitly stored in the data structures, as they are just needed for the analysis.)

Let us focus on an arbitrary guess γ. We will argue that, at any time $t > 0$, for every point q which is relevant for γ and t, there exists a point $x \in RV_{\gamma,t}$ and a point $y \in R_{\gamma,t}$ such that $\psi_\gamma(x) = \psi_\gamma(q)$ and $\phi_\gamma(y) = \phi_\gamma(q)$. The lemma will follow immediately since $\psi_\gamma(x) = \psi_\gamma(q)$ implies $d(q, x) \leq 4\gamma$, and $\phi_\gamma(y) = \phi_\gamma(q)$ implies $d(q, y) \leq \delta\gamma$.

Let us first concentrate on validation points. If q is the point arrived at time t (i.e., $t(q) = t$) then it is immediate to see from the pseudocode that q enters $RV_{\gamma,t}$, hence, it suffices to choose $x = q$. Otherwise, it is immediate to argue that since q is relevant at time t, it has also been relevant at all times t', with $t(q) \leq t' < t$, which implies that q has never been removed from $RV_{\gamma,t'}$ during some invocation of Cleanup, which only removes non-relevant points. Consequently, since q entered $RV_{\gamma,t(q)}$, either q is still in $RV_{\gamma,t}$ (and we set $x = q$), or q was eliminated from $RV_{\gamma,}$ due to the arrival of some point x_1 at time $t(x_1) > t(q)$. Observe that x_1 is also relevant. If $x_1 \in RV_{\gamma,t}$, we set $x = x_1$, otherwise, we apply the same reasoning to x_1. Iterating the argument, we determine a sequence of relevant points $x_1, x_2, \ldots, x_r$, with $t(q) < t(x_1) < \ldots < t(x_r)$ and $\psi(q) = \psi(x_1) = \ldots = \psi(x_r)$, with $x_r \in RV_{\gamma,t}$, and we set $x = x_r$.

The argument to show that there exists $y \in R_{\gamma,t}$ such that $\phi_\gamma(y) = \phi_\gamma(q)$ is virtually identical, and is just sketched for completeness. As argued above, since q is relevant for γ and t, it was never eliminated during some invocation of Cleanup. Thus, since $q \in R_{\gamma,t(q)}$, either $q \in R_{\gamma,t}$ (and we set $y = q$) or we can identify a sequence of relevant points $y_1, y_2, \ldots, y_r$ of the same color as q, with $t(q) < t(y_1) < \ldots < t(y_s)$ and $\phi(q) = \phi(y_1) = \ldots = \phi(y_s)$, with $y_s \in R_{\gamma,t}$, and we set $y = y_s$. □

The next lemma (proof omitted) is a straightforward adaptation of [6, Lemma 3], and provides an important technical tool, which will be needed to show that the coreset extracted in Query() contains a good solution to fair center for the current window.

Lemma 2. *Let W_t be the window of points at time step t, and let I_{W_t} be the family of independent sets of the partition matroid defined on W_t (i.e., the feasible solutions to fair center for W_t). Let $Q \subseteq W_t$ ba a coreset that satisfies the following conditions:*

- **(C1)** $d(p, Q) \leq \delta\gamma, \quad \forall p \in W_t$
- **(C2)** *For each independent set $X \in I_{W_t}$ there exists an injective mapping $\pi_X : X \to Q$ such that:*
  - $\{\pi_X(x) : x \in X\} \subseteq Q$ *is an independent set w.r.t. Q*
  - *for each $x \in X$, $d(x, \pi_X(x)) \leq \delta\gamma$*

Then:

- **(P1)** *There exists a solution $S \subseteq Q$ to fair center for W_t of radius $\leq \mathrm{OPT}_{W_t} + 2\delta\gamma$*
- **(P2)** *Every solution to fair center for Q of radius r is also a solution of cost at most $r + \delta\gamma$ for W_t.*

We are now ready to present the main novel ingredient of the analysis of our algorithm. Recall that for any point p of the stream and every guess γ, Update(p) assigns p to a c-attractor $\phi_\gamma(p) \in A_\gamma$, adding p to $repsC_{\gamma,t}(\phi_\gamma(p))$. Note that $\phi_\gamma(p) = p$ if p was added to A_γ. Also, for any point a which at time $t(a)$ was added to A_γ, and for any $t \geq t(a)$, we define

$$W_{\gamma,t}(a) = \{x \in W_t \ : \ \phi_\gamma(x) = a\}.$$

Observe that from time $t(a)$ until the time when a is expunged from A_γ, $W_{\gamma,t}(a)$ grows with the arrival of each new point c-attracted by a. After a leaves A_γ, $W_{\gamma,t}(a)$ progressively looses points as they expire. Also, for $t \geq t(a)$, the set $repsC_{\gamma,t}(a)$ is updated at each arrival of a new point attracted by a, until a is expunged from A_γ. After that happens, the set $repsC_{\gamma,t}(a)$ is not further updated with new points, while its elements will then be gradually expunged from R_γ as they expire. We have:

Lemma 3. *For any time t and valid guess $\gamma \in \Gamma$, the following holds. Let $a \in S$ be a point which arrived at $t(a) \leq t$ and, upon arrival, was added to A_γ. Then $W_{\gamma,t}(a) \cap R_{\gamma,t}$ is a maximal independent set w.r.t. $W_{\gamma,t}(a)$.*

Proof. It is immediate to see that at each time $t \geq t(a)$ for which $a \in A_{\gamma,t}$, the set $repsC_{\gamma,t}(a) = W_{\gamma,t}(a) \cap R_{\gamma,t}$ is always a maximal independent set w.r.t. $W_{\gamma,t}(a)$, since a point $x \in W_{\gamma,t}(a)$ is always inserted into $repsC_{\gamma,t(x)}(a)$, and may cause the expunction from the set of a single point of the same color i_x (the oldest such point), only when there are already k_{i_x} points of color i_x present in $repsC_{\gamma,t(x)-1}(a)$. Let $\hat{t} > t(a)$ be the time when a is expunged from $A_{\gamma,\hat{t}}$. As already observed, for every $t \geq \hat{t}$, $W_{\gamma,t}(a)$ does not acquire new points and shrinks due to the natural expiration of the points c-attracted by a. If points in $R_{\gamma,t}$ were removed only on their expiration, then the maximality of $W_{\gamma,t}(a) \cap R_{\gamma,t}$ w.r.t. $W_{\gamma,t}(a)$ would immediately follow by the fact that, as long as a is in $A_{\gamma,t}$, for each color i, $repsC_{\gamma,t}(a)$ maintains the most recently arrived points of that color. However, whenever γ becomes an invalid guess, extra points from $R_{\gamma,t}$ may be expunged (Line 2 of procedure Cleanup). However, these expunged points, being older than the oldest point in $AV_{\gamma,t}$, will all have expired by the time step $t' > t$ (if any) when $|AV_{\gamma,t'}|$ drops below $k+1$, and thus γ becomes again a valid guess, so their premature elimination will not affect the maximality of $W_{\gamma,t'} \cap R_{\gamma,t'}$. □

The following theorem establishes the quality of the solution returned by Procedure Query.

Theorem 3. *Let α be the approximation ratio featured by the sequential fair center algorithm $\mathcal{A}$ used in Procedure Query(). For fixed $\varepsilon \in (0, 1)$ and $\beta > 0$, by setting*

$$\delta = \frac{\varepsilon}{(1+\beta)(1+2\alpha)}$$

we have that if Procedure Query is run at time t, then the returned solution is an $(\alpha + \varepsilon)$-approximate solution to fair center for the current window W_t.

Proof. Let $\hat{\gamma}$ be the guess such that the solution returned by Query is computed by running $\mathcal{A}$ on the coreset $Q = R_{\hat{\gamma}}$. We first show that conditions (C1) and (C2) of Lemma 2 hold for Q. First observe that by construction $\hat{\gamma}$ is chosen so that $|AV_{\hat{\gamma},t}| \leq k$, therefore condition (C1) holds by virtue of Property 1.(b) of Lemma 1. To show that (C2) also holds, consider any independent set X w.r.t. W_t. We construct the required injective mapping $\pi_X : X \to Q$ incrementally, one point at a time. Let $X = \{x_j \ : \ 1 \leq j \leq |X|\}$. Suppose that we have fixed the mapping for the first $h \geq 0$ elements of X, and assume, inductively, that

$$Y(h) = \{\pi_X(x_j) \ : \ 1 \leq j \leq h\} \cup \{x_j \ : \ h < j \leq |X|\}$$

is an independent set w.r.t. W_t of size $|X|$, and such that, for $1 \leq j \leq h$, $\pi_X(x_j) \in Q$ and $d(x_j, \pi_X(x_j)) \leq \delta\gamma$. We now show how to extend the mapping to index $h+1$. We distinguish between two cases. If $x_{h+1} \in Q$, then we simply set $\pi_X(x_{h+1}) = x_{h+1}$. Since $Y(h+1) = Y(h)$ and $d(x_{h+1}, \pi_X(x_{h+1})) = 0 \leq \delta\gamma$, the mapping is correctly extended to index $h+1$. Conversely, if $x_{h+1} \notin Q$, observe that $x_{h+1} \in W_{\hat{\gamma},t}(a)$, where $a = \phi(x_{h+1})$. Since $\hat{\gamma}$ is a valid guess, by Lemma 3 it follows that $Z = W_{\hat{\gamma},t}(a) \cap Q$ is a maximal independent set w.r.t. $W_{\hat{\gamma},t}(a)$. Then, it will always be possible to set $\pi_X(x_{h+1}) = y_{h+1}$, where y_{h+1} is a point of Z of the same color as x_{h+1}, and such that $y_{h+1} \neq \pi_X(x_j)$, for $1 \leq j \leq h$, or otherwise the number of points in $X \cap W_{\hat{\gamma},t}(a)$ of the same color as x_{h+1} would have to be larger than those in Z, contradicting the maximality of the latter subset w.r.t. $W_{\hat{\gamma},t}(a)$. By the properties of the partition matroid, it follows that $Y(h+1)$ is still an independent set w.r.t. W_t. Also, since both x_{h+1} and $\pi_X(x_{h+1}) = y_{h+1}$ belong to $W_{\hat{\gamma},t}(a)$, we have that $d(x_{h+1}, \pi_X(x_{h+1})) \leq d(x_{h+1}, a) + d(a, \pi_X(x_{h+1}) \leq \delta\gamma$. Therefore, given that, by construction, $\pi_X(x_{h+1}) \in Q$, the mapping is correctly extended to index $h+1$. By iterating the argument up to index $|X|$, we have that $Y(|X|) \subseteq Q$, and we conclude that condition (C2) holds for $Q \subseteq W_t$. Therefore, by Lemma 2, since (C1) and (C2) hold for Q, Properties (P1) and (P2) also hold. Property (P1) implies that the optimal solution to fair center for Q has radius $\leq \mathrm{OPT}_{W_t} + 2\delta\hat{\gamma}$. Therefore, when invoked on Q, algorithm $\mathcal{A}$ will return a solution of radius $\leq \alpha(\mathrm{OPT}_{W_t} + 2\delta\hat{\gamma})$ for Q, which, by Property (P2), is also a solution of radius $\leq \alpha\mathrm{OPT}_{W_t} + (1 + 2\alpha)\delta\hat{\gamma}$ for W_t. Now, let r_k^* be the radius of the optimal solution to unconstrained k-center for W_t. It is easy to see that $\hat{\gamma} \leq (1+\beta)r_k^*$. Indeed, the inequality is trivial if $\hat{\gamma} = d_{\min}$, while, otherwise, it follows from the fact that Procedure

Query discarded guess $\gamma = \hat{\gamma}/(1+\beta)$ because $k+1$ points of W_t at pairwise distance 2γ exist and, clearly, two of these points must be closest to the same center of the optimal solution of unconstrained k-center for W_t of radius r_k^*. Since $r_k^* \leq \text{OPT}_{W_t}$, we have that $\hat{\gamma} \leq (1+\beta)\text{OPT}_{W_t}$. The above discussion and the choice of δ immediately imply that Query will return a solution whose radius, with respect to the entire window W_t, is at most

$$\begin{aligned}&\alpha\text{OPT}_{W_t} + (1+2\alpha)\delta\hat{\gamma} \leq \\ &\quad \leq \alpha\text{OPT}_{W_t} + (1+2\alpha)\frac{\varepsilon}{(1+\beta)(1+2\alpha)}(1+\beta)\text{OPT}_{W_t} \\ &\quad = (\alpha+\varepsilon)\text{OPT}_{W_t}. \quad \square\end{aligned}$$

By using the algorithm of [20] as algorithm $\mathcal{A}$ in Query, we obtain:

Corollary 1. *For fixed $\varepsilon \in (0,1)$, at any time $t > 0$, procedure Query can be used to compute a $(3+\varepsilon)$-approximate solution to fair center for window W_t.*

The following theorem bounds the size of the working memory required by our algorithm.

Theorem 4. *Under the same parameter configuration of Theorem 3, at any time t during the processing of stream S, the sets stored in in the working memory contain*

$$O\left(k^2 \frac{\log \Delta}{\log(1+\beta)} \left(\frac{c}{\varepsilon}\right)^{D_{W_t}}\right)$$

points overall, where $c = 32(1+\beta)(1+2\alpha)$, D_{W_t} is the doubling dimension of the current window W_t and Δ is the aspect ratio of S.

Proof. Consider a time t and a guess $\gamma \in \Gamma$. The following facts can be proved through the same arguments used in the proof of [28, Theorem 2] for the case of unconstrained k-center on a window of doubling dimension D_{W_t}.

Fact 1. *At each time t, $|AV_\gamma| \leq k+1$ and $RV_\gamma \leq 2(k+1)$.*

Fact 2. *A_γ contains at most $k' = 2(k+1)(32/\delta)^{D_{W_t}}$ points.*

It remains to upper bound the size of R_γ. Clearly, by Fact 2, there can be altogether at most $k \cdot k'$ points in R_γ contained in the representative sets $repsC_\gamma$ of points currently in A_γ. We are then left to bound the number of points contained in representative sets of points not in A_γ at the current time. Call O_γ the union of such sets. Let a_i, with $i = 1, 2, \ldots$ be an enumeration of all points that upon arrival have been added to A_γ. For every $i \geq 1$ it must hold that a_i has expired or has been expunged from A_γ by the time $a_{i+k'+1}$ enters A_γ, or otherwise A_γ would have size greater than k' at time $t(a_{i+k'+1})$, which would contradict Fact 2. Consequently, any point x added to $repsC_\gamma(a_i)$ must have arrived while a_i was still in A_γ, hence before $a_{i+k'+1}$, thus $\text{TTL}(x) < \text{TTL}(a_{i+k'+1})$ at any time when they are both active. Now, consider the current time, and let a_j be the point which was most recently removed from A_γ. By the above property, any point x that belonged to $repsC_\gamma(a_\ell)$, with $\ell \leq j-(k'+1)$, arrived prior to a_j, hence cannot be in memory at time t. Hence, O_γ can only comprise points that belonged to $repsC_\gamma(a_\ell)$, with $j-(k'+1) < \ell \leq j$, which immediately implies that $|O_\gamma| \leq k \cdot k'$. The theorem follows by considering that in Theorem 3 we fixed $\delta = \varepsilon/((1+\beta)(1+2\alpha))$, and that the number of guesses $|\Gamma| = O(\log \Delta/\log(1+\beta))$. $\square$

For what concerns running time, it can be easily argued that Procedure Update requires time linear in the aggregate size of the stored sets, while the Procedure Query first executes $O(\log \Delta/\log(1+\beta))$ attempts at discovering suitable unconstrained k-clusterings on sets of $O(k)$ points, each requiring $O(k^2)$ time, and then invokes the sequential fair center algorithm $\mathcal{A}$ on R_γ, for some guess $\gamma \in \Gamma$. The above discussion immediately implies the following theorem.

Theorem 5. *Under the same parameter configuration of Theorem 4, Procedure Update runs in time*

$$O\left(k^2 \frac{\log \Delta}{\log(1+\beta)} \left(\frac{c}{\varepsilon}\right)^{D_{W_t}}\right)$$

while procedure Query runs in time

$$O\left(k^2 \frac{\log \Delta}{\log(1+\beta)} + T_{\mathcal{A}}\left(k^2 \left(\frac{c}{\varepsilon}\right)^{D_{W_t}}\right)\right)$$

where $T_{\mathcal{A}}(m)$ is the running time of sequential algorithm $\mathcal{A}$ on an instance of size m.

It is important to remark that for windows of constant doubling dimension, both space and time requirements of our algorithm are independent of the window size n.

The space and time requirements of our algorithm can be made independent of the dimensionality of the stream, at the expense of a worsening in the approximation guarantee. We have:

Corollary 2. *Procedures* Update, Cleanup *and* Query *can be modified so to yield, at any time t, a $(31+O(\varepsilon))$-approximate solution, storing $O\left(k^2 \log \Delta/\varepsilon\right)$ points in the working memory, and requiring $O\left(k^2 \log \Delta/\varepsilon\right)$ update and $O\left(k^2 \log \Delta/\varepsilon + T_{\mathcal{A}}(k^2)\right)$ query time.*

Proof. We recast our algorithm by eliminating the two sets of coreset points and all operations related to their maintenance. Also, rather than a simple point, we make sure to maintain in $repV_\gamma(v)$ the most recent maximal independent set of points attracted by v. Procedures Update and Cleanup are modified accordingly. Finally, Query computes the final clustering by selecting the right guess γ as before and then applying algorithm $\mathcal{A}$ to RV_γ rather than R_γ. The bounds on time and space are immediately obtained by noticing that the size of the set RV_γ is no more than a factor k larger than its size in the original algorithm. The bound on approximation can be obtained by setting $\beta = \varepsilon$ and by a straightforward adaptation of the analysis of the original algorithm for the case $\delta = 4$, a value for which the set of v-points on which $\mathcal{A}$ is invoked coincides with the old set R_γ. $\square$

Main differences with previous work. The algorithms presented above, together with their analysis, introduce the following novelties with respect to the work in [11, 28]:

- The involved data structures (maintaining validation and coreset points) have been simplified through the removal of the *orphan* point category, [11, 28], which has been absorbed within the sets of representatives.
- Our new approach maintains, as c-representatives for each c-attractor a, the most recent maximal independent set of all window points attracted by a, rather than a single representative. This is instrumental to guarantee that, for a suitable guess γ, the set of c-representatives contains a good solution to the constrained problem on the current window.
- The proof of Lemma 1 has been entirely restructured over the respective one in [11, 28], featuring a novel, shorter and more intuitive argument.

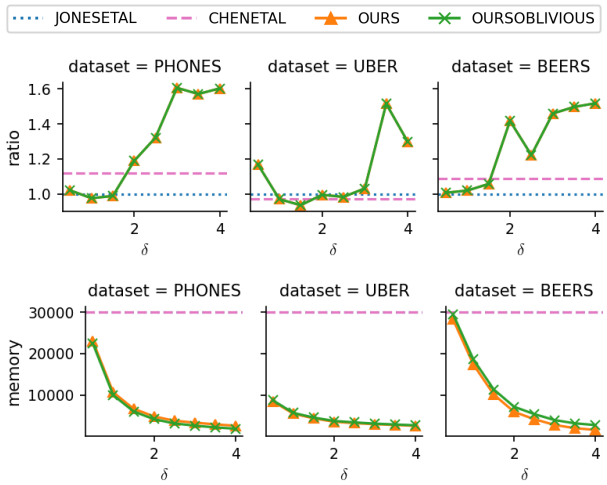


Figure 1: Approximation ratio (top) and memory (bottom, in number of points) for varying δ, with $k = 30$.

- With the exception of the technical Lemma 2, the remainder of the analysis is completely novel.

4 Experiments

We evaluate experimentally our coreset-based algorithm against state of the art baselines, aiming at answering the following questions:

- How does the coreset size impact the performance? (§ 4.1)
- How do algorithms behave on different window lengths? (§ 4.2)
- What is the influence of the data dimensionality? (§ 4.3)

Below, we provide details of the experimental setup, namely, a description of the datasets, the implemented algorithms and baselines, and the performance metrics used in our comparisons.

Datasets. We consider three real-world datasets.

- PHONES[2] is a dataset of sensor data from smartphones: each of the 13 062 475 points represents the position of a phone in three dimensions, and is labeled with one of $\ell = 7$ categories corresponding to user actions: stand, sit, walk, bike, stairs up, stairs down, null. The aspect ratio of the entire dataset is $\approx 6.4 \cdot 10^5$;
- BEERS[3] collects reviews for 1 586 615 beers. We use the review timestamp to dictate the streaming order. Each data point features 6 dimensions, and is associated with one out of $\ell = 27$ beer styles[4] that we use as categories. The aspect ratio is ≈ 17.88;
- UBER[5] contains over 4.5 million Uber pickups in New York City from April to September 2014. Each data point has 2 dimensions (pickup location), and is associated to one out of $\ell = 8$ TLC base companies. The aspect ratio is $\approx 1.67 \cdot 10^4$.

Besides these real-world datasets, some experiments are run on suitably crafted synthetic data, which will be specified later.

[2]UCI repository: https://doi.org/10.24432/C5689X
[3]https://www.kaggle.com/datasets/thedevastator/1-5-million-beer-reviews-from-beer-advocate
[4]We conflate similar beer styles, see the data preprocessing code for details.
[5]https://github.com/fivethirtyeight/uber-tlc-foil-response

Algorithms. As baseline algorithms we consider the matroid center algorithm by [8] (ChenEtAl), and the fair center algorithm by [20] (JonesEtAl). Note that both are sequential algorithms for the problem. In the sliding window setting, upon a query, we run the baseline algorithms on *all* points of the current window. Thus, in principle, these provide the most accurate solutions, at the expense of high space/time complexity. Algorithm Ours is the implementation of our algorithm with knowledge of the minimum and maximum pairwise distances, $d_{\min}$ and $d_{\max}$, between points of the stream, and invokes baseline $\mathcal{A}$ = JonesEtAl in Query only on the points of the selected coreset. Since the aspect ratio of a stream is often unknown, we also implemented OursOblivious, where running estimates of $d_{\min}$ and $d_{\max}$ *for the current window* are obtained by means of the techniques of [28], based on a sliding-window diameter-estimation algorithm. In OursOblivious, the update and query procedures consider only the guesses that are within the current $[d_{\min}, d_{\max}]$ interval. For both Ours and OursOblivious, the progression of the guess values is defined fixing $\beta = 2$. In fact, a set of experiments (omitted for brevity) have shown that varying this parameter has no significant impact on the results. The precision parameter δ varies in the set $\delta \in \{0.5, 1, 1.5, 2, 2.5, 3, 3.5, 4\}$, with the understanding that lower values result in finer-grained (i.e. larger) coresets, while $\delta = 4$ is equivalent to using a coreset comparable in size to the validation set (i.e., the one yielding the result of Corollary 2). This set of values for δ indeed allows spanning a very wide range of coreset sizes. For the window size, we consider values between 10 000 and 500 000 points. The cardinality constraints on the ℓ colors are set so that, for every i, k_i is set to be proportional to the number of points of color i in the entire dataset. Most of the experiments have been run with $k = \sum_{i=1}^{\ell} k_i = 30$, so that, for any of the used datasets, if the proportionality of the colors is balanced, there is at least one center per color. Also, we performed experiments with a larger number of clusters $k = 100$. We set a timeout of 24 hours on each execution.

The algorithms are implemented using Java 1.8, with the code publicly available[6]. The experiments have been carried out on a machine with 500Gb of RAM, equipped with a 32-core AMD EPYC 7282 processor.

Performance metrics. We consider four performance indicators: (1) the number of points maintained in memory by the algorithms; (2) the running time of procedure Update; (3) the running time of procedure Query; (4) the approximation *ratio*, namely the ratio between the obtained radius and the best radius ever found by ChenEtAl or JonesEtAl when run on all points of the window.

All the metrics are computed as averages over 200 consecutive sliding windows of the stream.

4.1 Dependency on the coreset size

A first set of experiments focuses on the influence of the coreset size on the performance of our algorithms. We fix the window size to 30 000 points, and obtain varying coreset sizes by varying δ between 0.5 and 4. The top graphs of Figure 1 show, for all datasets and all algorithms, how the solution quality, assessed through the approximation ratio defined above, changes with δ. We observe that Ours and OursOblivious find solutions of comparable quality, as do ChenEtAl and JonesEtAl (observe that the performance of these two latter algorithms is clearly independent of δ, since they are always executed on the entire

[6]https://github.com/aidaLabDEI/streaming-fair-center-clustering

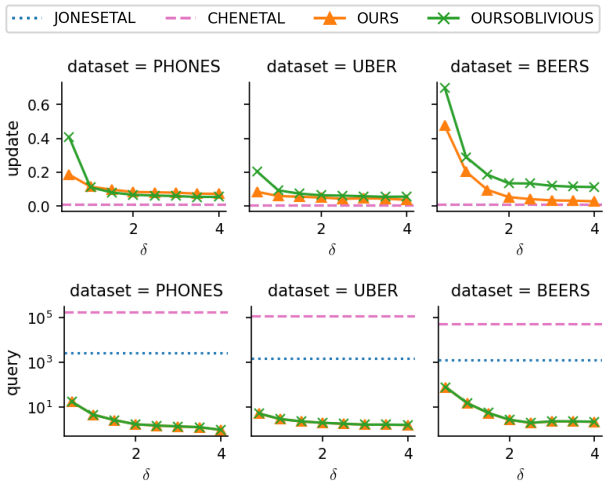


Figure 2: Running time in milliseconds of update (top) and query (bottom, logarithmic scale) for varying δ, with $k = 30$.

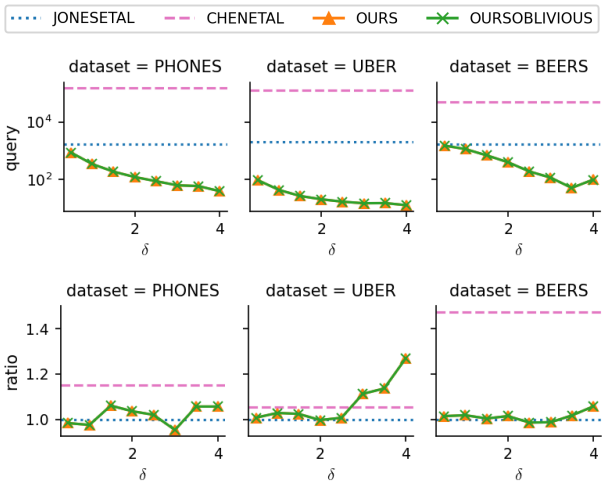


Figure 3: Running time of query in milliseconds (top, log scale) and approximation ratio (bottom) for different values of δ with $k = 100$.

window). When the coreset is smallest (i.e., $\delta = 4$), clearly our algorithms find worse solutions, but always within a factor 2 from those returned by the baselines; as the coreset gets larger (i.e., for smaller δ), then the solutions become of quality comparable (and, surprisingly, sometimes better) to the one computed by the sequential baselines.

The bottom graphs of Figure 1 compare the memory usage of the algorithms. As expected, for all values of δ, our algorithm uses less memory than the baselines, which maintain the entire window in memory, with gains becoming substantially more marked as δ grows. This behavior provides experimental evidence of the interesting memory-accuracy tradeoff featured by our algorithms, and is in accordance with the theoretical analysis. We also note that OursOblivious and Ours require comparable memory and running time. Therefore, we may conclude that being oblivious to the aspect ratio of the window (and thus requiring to keep updated estimates of $d_{\min}$ and $d_{\max}$) does not have a significant impact on the performance or on the quality of the returned clustering.

The top and bottom graphs of Figure 2 report the runtime performance, in milliseconds, for the update and query procedures, respectively. Clearly, the sequential baselines feature next-to-zero update time, simply because they store the entire window, thus, at each time step, there is nothing to do other than adding one point and removing one to maintain the current window. Nevertheless, both Ours and OursOblivious feature reasonable update times, not exceeding 0.15 milliseconds, which is just a tiny fraction of the cost of a query. It is interesting to observe that despite the overhead incurred by OursOblivious to estimate $d_{\min}$ and $d_{\max}$ for the current window, the advantage of having fewer guesses makes it always faster than Ours. In both cases, using larger values of δ (i.e., smaller coresets) leads to faster update times, as expected. However, it is on query times where the difference with the baselines is most striking: by confining the execution of the expensive baselines only to a small coreset of window points, Ours and OursOblivious are able to find a solution for every window *up to two orders of magnitude faster* than JonesEtAl, which is in turn two orders of magnitude faster than ChenEtAl. Even with the smallest values of δ, which afford comparable accuracy to the sequential baselines, the gains are between one and two orders of magnitude!

Figure 3 (top) reports the query running time for $k = 100$ for different values of δ and a window size of 30 000 points. We observe that our algorithm is consistently faster than the baselines even at these larger values of k. For $\delta = 0.5$ (i.e. fine grained coresets) the performance of our algorithm on BEERS matches that of the JonesEtAl baseline, due to the high number of categories in this dataset: using a coarser coreset at $\delta = 3.5$ allows to gain over an order of magnitude in performance, with a loss of less than 2% in terms of radius as shown in Figure 3 (bottom). Furthermore, note that on UBER our algorithm attains up to two orders of magnitude gains with respect to the JonesEtAl baseline, by virtue of the low dimensionality of the dataset that our algorithm is able to exploit.

4.2 Dependency on the window size

The next set of experiments analyzes the behavior of the algorithms for varying window sizes. For Ours and OursOblivious, we consider the most accurate and slowest setting in the benchmark, thus fixing $\delta = 0.5$. (For the sake of conciseness, we omit plots for the update time and the approximation ratio, which are consistent with the findings of the previous section). Figure 4 (top) reports on the memory usage. Clearly, the memory used by the sequential baselines increases linearly with the window size. In contrast, the memory used by both versions of our algorithm, after an initial increase, stabilizes to a value *independent of the window size*. This difference is reflected in the query time performance (Figure 4, bottom). As observed in the previous section, the time gain when employing coresets compared to the use of the sequential baselines on the entire window is already of orders of magnitude for relatively small windows, and, as shown by the plots in the figure, it keeps increasing very steeply with the window size. In fact, for windows larger than 30 000 points, the execution of the experiments relative to ChenEtAl timed out, requiring about 400 seconds per window. Similarly, those relative to JonesEtAl timed out for windows larger than 200 000 points.

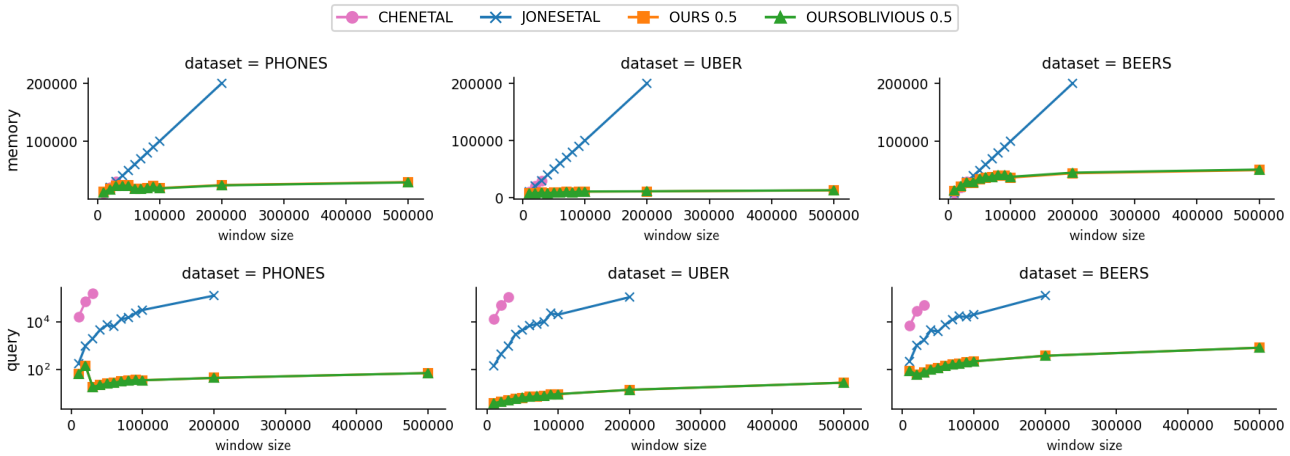


Figure 4: Memory (top, in number of points) and running time of query (bottom, in milliseconds) at varying window sizes. The scale of query time is logarithmic.

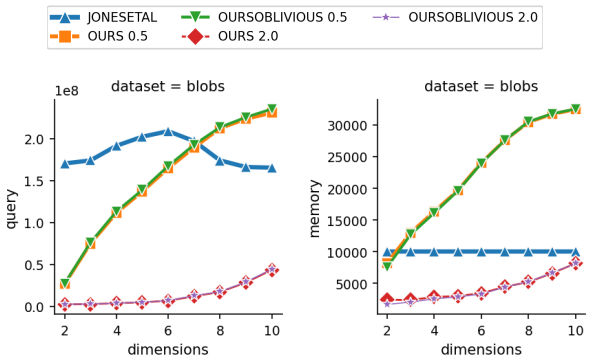


Figure 5: Query time (left, in milliseconds) and memory (right, in number of points) with respect to the dimensionality, on the `blobs` datasets.

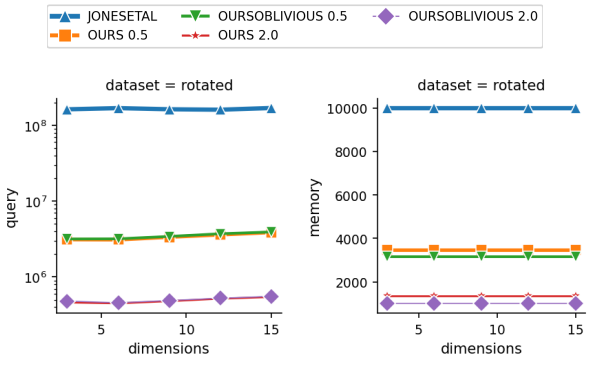


Figure 6: Query time and memory (resp. left and right) with respect to the dimensionality on the `rotated` datasets. The scale of query time is logarithmic.

4.3 Dependency on the dimensionality

This latter set of experiments studies the influence of the data dimensionality on the performance of our algorithm. To this end we generated two families of synthetic datasets: `blobs`, used to assess how the memory and query times of our algorithm are influenced by the dimensionality; and `rotated`, used to verify that these costs are related to the *actual* dimensionality of the dataset, rather then the number of coordinates of each point. The `blobs` datasets have 1 000 000 points: each dataset is a mixture of 21 multivariate d-dimensional Gaussians, for $2 \leq d \leq 10$. The covariance matrix is $\Sigma = I_d\sigma^2$ with $\sigma = 2$, and each point is assigned a random color out of 7, ensuring an even distribution. We set $k_i = 3$, for $1 \leq i \leq 7$, and the window size to 10 000. The `rotated` datasets are derived from `PHONES` (which has 3 dimensions) by adding zero-filled dimensions to the original data, followed by a rigid random rotation of the entire extended dataset. With this procedure we generate datasets with up to 15 nonzero coordinates, but by construction, all data still lie on a 3-dimensional subspace.

In these experiments, we use only JonesEtAl as baseline and, for our algorithm, we consider only two settings: $\delta = 0.5$, which yields larger coresets but higher accuracy; and $\delta = 2$, which yields smaller coresets and lower accuracy (that is still comparable, however, to the one attained by the sequential baseline). For conciseness, we omit the plots of the update time and the approximation ratio, which are coherent with previous findings.

Figure 5 reports the query time (left) and memory (right) on the `blobs` datasets. As expected, the performance of sequential baseline JonesEtAl is insensitive to the dimensionality, while both query time and memory usage of our algorithm grow with the dimensionality, the growth being much steeper in the larger coreset setting ($\delta = 0.5$), as suggested by the theoretical bounds. We remark that our algorithm uses less memory than the sequential baseline for $\delta = 2$, even for higher dimensions.

Figure 6 reports results on the `rotated` datasets. Note that in this case, where the data lies in a 3-dimensional subspace, the query time and memory for our algorithm are independent of the number of dimensions of the dataset. This confirms that our algorithm's performance depends on the *actual* dimensionality of the dataset, rather than on the sheer number of coordinates of the vectors associated to the points.

5 Conclusions

This paper presents the first sliding-window algorithm to enforce fairness in center selection, under the k-center objective. The

algorithm works for general metrics, stores a number of points independent of the size of the window, and provides approximation guarantees comparable to those of the best sequential algorithm applied to all the points of the window. Its space and time requirements are analyzed in terms of the dimensionality and of the aspect ratio of the input stream, although these values, which are difficult to estimate from the data, do not have to be known to the algorithm. A variant of the algorithm affording a dimensionality-independent analysis, while still achieving $O(1)$ approximation, is also provided. Among the many avenues for future work, we wish to mention the extension of our algorithms to the robust variant of fair center, tolerating a fixed number of outliers, possibly fairly chosen among the most distant points w.r.t. a given solution. We believe that good approximations for robust fair center in sliding windows may be attained by building on previous work for robust unconstrained k-center, matroid and fair center in the literature [2, 6, 7, 29].

Acknowledgments

This work was supported, in part, by MUR of Italy, under Projects PRIN 2022TS4Y3N (EXPAND: scalable algorithms for EXPloratory Analyses of heterogeneous and dynamic Networked Data), and PNRR CN00000013 (National Centre for HPC, Big Data and Quantum Computing).

Artifacts

The code implementing the algorithms evaluated in this paper is available at a public Github repository (https://github.com/aidaLabDEI/streaming-fair-center-clustering), together with detailed instructions on how to run the code and obtain the datasets used in the experimental evaluation.

References

[1] S.S. Abraham, D. Padmanabhan, and S.S. Sundaram. 2020. Fairness in Clustering with Multiple Sensitive Attributes. In *EDBT*. 287–298.
[2] D. Amagata. 2024. Fair k-center Clustering with Outliers. In *AISTATS (PMLR, Vol. 238)*. 10–18.
[3] H. Angelidakis, A. Kurpisz, L. Sering, and R. Zenklusen. 2022. Fair and Fast k-Center Clustering for Data Summarization. In *ICML (PMLR, Vol. 162)*. 669–702.
[4] M. Ceccarello, A. Pietracaprina, and G. Pucci. 2020. A General Coreset-Based Approach to Diversity Maximization under Matroid Constraints. *ACM-TKDD* 5, 14 (2020), 60:1–60:27.
[5] M. Ceccarello, A. Pietracaprina, and G. Pucci. 2024. Fast and Accurate Fair k-Center Clustering in Doubling Metrics. In *Web Conference*. ACM, United States, 756–767.
[6] M. Ceccarello, A. Pietracaprina, G. Pucci, and F. Soldà. 2023. Scalable and space-efficient Robust Matroid Center algorithms. *J. Big Data* 10, 1 (2023), 49.
[7] D. Chakrabarty and M. Negahbani. 2019. Generalized Center Problems with Outliers. *ACM Trans. on Algorithms* 15, 3 (2019), 41:1–41:14.
[8] D.Z. Chen, J. Li, H. Liang, and H. Wang. 2016. Matroid and Knapsack Center Problems. *Algorithmica* 75, 1 (2016), 27–52.
[9] A. Chhabra, K. Masalkovaite, and P. Mohapatra. 2021. An Overview of Fairness in Clustering. *IEEE Access* 9 (2021), 130698–130720.
[10] A. Chiplunkar, S. Kale, and S.N. Ramamoorthy. 2020. How to Solve Fair k-Center in Massive Data Models. In *ICML (PMLR, Vol. 119)*. 1877–1886.
[11] V. Cohen-Addad, C. Schwiegelshohn, and C. Sohler. 2016. Diameter and k-Center in Sliding Windows. In *ICALP (LIPIcs, Vol. 55)*. 19:1–19:12.
[12] M. Datar and R. Motwani. 2016. The Sliding-Window Computation Model and Results. In *Data Stream Management - Processing High-Speed Data Streams*. 149–165.
[13] C. Dwork, M. Hardt, T. Pitassi, O. Reingold, and R.S. Zemel. 2012. Fairness through awareness. In *ITCS*. ACM, New York, NY, USA, 214–226.
[14] European Parliament. [n. d.]. Regulation 2016/679. https://web.archive.org/web/20250729120038/https://eur-lex.europa.eu/legal-content/EN/TXT/HTML/?uri=CELEX:32016R0679&from=EN
[15] J. Gan, M. J. Golin, Z. Yang, and Y. Zhang. 2023. Fair k-Center: a Coreset Approach in Low Dimensions. *CoRR* abs/2302.09911 (2023).
[16] T.F. Gonzalez. 1985. Clustering to Minimize the Maximum Intercluster Distance. *Theoretical Computer Science* 38 (1985), 293–306.
[17] L-A. Gottlieb, A. Kontorovich, and R. Krauthgamer. 2014. Efficient Classification for Metric Data. *IEEE Trans. Information Theory* 60, 9 (2014), 5750–5759.
[18] L. Han, D. Xu, Y. Xu, and P. Yang. 2023. Approximation algorithms for the individually fair k-center with outliers. *J. Glob. Optim.* 87, 2 (2023), 603–618.
[19] C. Hennig, M. Meila, F. Murtagh, and R. Rocci. 2015. *Handbook of cluster analysis*. CRC Press, United States.
[20] M. Jones, H.L. Nguyen, and T.D. Nguyen. 2020. Fair k-Centers via Maximum Matching. In *ICML (PMLR, Vol. 119)*. 4940–4949.
[21] C. Jung, S. Kannan, and N. Lutz. 2019. A Center in Your Neighborhood: Fairness in Facility Location. *CoRR* abs/1908.09041 (2019).
[22] S. Kale. 2019. Small Space Stream Summary for Matroid Center. In *APPROX-RANDOM (LIPIcs, Vol. 145)*. 20:1–20:22.
[23] M. Kay, C. Matuszek, and S.A. Munson. 2015. Unequal Representation and Gender Stereotypes in Image Search Results for Occupations. In *CHI*. ACM, 3819–3828.
[24] M. Kleindessner, P. Awasthi, and J. Morgenstern. 2019. Fair k-Center Clustering for Data Summarization. In *ICML (PMLR, Vol. 97)*. 3448–3457.
[25] Y. Kurkure, M. Shamo, J. Wiseman, S. Galhotra, and S. Sintos. 2024. Faster Algorithms for Fair Max-Min Diversification in R^d. *Proc. ACM Manag. Data* 2, 3 (2024), 137.
[26] Z. Lin, L. Guo, and C. Jia. 2024. Streaming Fair k-Center Clustering over Massive Dataset with Performance Guarantee. In *PAKDD (3) (LNCS, Vol. 14647)*. Springer, Singapore, 105–117.
[27] S. Mahabadi and A. Vakilian. 2020. Individual Fairness for k-Clustering. In *ICML (PMLR, Vol. 119)*. 6586–6596.
[28] P. Pellizzoni, A. Pietracaprina, and G. Pucci. 2022. Adaptive k-center and diameter estimation in sliding windows. *Int. J. Data Sci. Anal.* 14, 2 (2022), 155–173.
[29] P. Pellizzoni, A. Pietracaprina, and G. Pucci. 2022. k-Center Clustering with Outliers in Sliding Windows. *Algorithms* 15, 2 (2022), 52.
[30] P. Pellizzoni, A. Pietracaprina, and G. Pucci. 2025. Fully Dynamic Clustering and Diversity Maximization in Doubling Metrics. *ACM Trans. Knowl. Discov. Data* 19, 4 (2025), 1–45.
[31] L.V. Snyder. 2011. Introduction to facility location. In *Wiley Enciclopedia of Operations Research and Management Science*. Wiley, United States.
[32] J. Stoyanovich, K. Yang, and H.V. Jagadish. 2018. Online Set Selection with Fairness and Diversity Constraints. In *EDBT*. 241–252.
[33] Y. Wang, F. Fabbri, and M. Mathioudakis. 2022. Streaming Algorithms for Diversity Maximization with Fairness Constraints. In *ICDE*. IEEE, 41–53.
[34] Y. Wang, F. Fabbri, M. Mathioudakis, and J. Li. 2023. Fair Max–Min Diversity Maximization in Streaming and Sliding-Window Models. *Entropy* 25, 7 (2023). doi:10.3390/e25071066
[35] Y. Wang, M. Mathioudakis, J. Li, and F. Fabbri. 2023. Max-Min Diversification with Fairness Constraints: Exact and Approximation Algorithms. In *SDM*. SIAM, 91–99.