Fair and Fast k-Center Clustering for Data Summarization

Haris Angelidakis [*1] **Adam Kurpisz** [*2] **Leon Sering** [*2] **Rico Zenklusen** [*2]

Abstract

We consider two key issues faced by many clustering methods when used for data summarization, namely (a) an unfair representation of "demographic groups" and (b) distorted summarizations, where data points in the summary represent subsets of the original data of vastly different sizes. Previous work made important steps towards handling separately each of these two issues in the context of the fundamental k-Center clustering objective through the study of fast algorithms for natural models that address them.

We show that it is possible to effectively address both (a) and (b) simultaneously by presenting a clustering procedure that works for a canonical combined model and

(i) is fast, both in theory and practice,
(ii) exhibits a worst-case constant-factor guarantee, and
(iii) gives promising computational results showing that there can be significant benefits in addressing both issues together instead of sequentially.

1. Introduction

The size of the digital universe in 2020 was estimated to be dozens of zettabytes of data, the vast majority of which were generated within the last few years (see Statista and Forbes estimates (Holst, 2021; Marr, 2018) for additional information). Moreover, the International Data Corporation (IDC) (2019) report predicts that by the end of 2025 nearly 80 zettabytes will be generated every year by IoT devices, with an average of more than 4 devices per human being.

This massive amount of data generated goes hand in hand with the need for algorithmic techniques to process and analyze them. One particularly useful application of machine learning algorithms for analyzing massive data sets is data summarization. Given a large data set, the goal of data summarization is to extract a small subset of samples that represents well the whole data set. Performing further analysis on a smaller data set is much easier and in that sense data summarization algorithms act as an interface between the digital universe and human users, e.g., in search engines.

A crucial challenge faced when representing a large data set by a smaller subset is the proper handling of fairness considerations. We focus on two central aspects of obtaining a fair representation of the underlying data, namely

(a) that sensitive attributes are properly reflected in the summary (think about a fair representation of races or other demographics in the summary), and

(b) different data points in the summary should not represent subsets of the original data of vastly different sizes, thus leading to a distorted summary.

There are numerous applications where both aspects are relevant, including fair voting system design (Celis, Huang, and Vishnoi, 2018) and single-cell RNA sequencing (Do, Elbassioni, and Canzar, 2020).

Regarding (a), it has been repeatedly observed that even though data summarization procedures are generally not biased by design, the fact that many of them are oblivious to sensitive attributes can quickly lead to a strong bias by coincidence. Significant measures have been taken to counteract the bias in society, e.g., the concept of *disparate impact* introduced in 1971 in the U.S. Supreme Court: Griggs v. Duke Power Co. (see for example (Feldman, Friedler, Moeller, Scheidegger, and Venkatasubramanian, 2015)). Nevertheless, the rapidly developing field of learning algorithms still faces many challenges in this regard. As an example, it was observed that in 2019, 85% of supermarket cashier job advertisements on Facebook were targeted to women, while taxi driver positions were proposed to black men to an extent of 74% (Ali, Sapiezynski, Bogen, Korolova, Mislove, and Rieke, 2019). More emphatic examples include bias in the crime-related search results when keywords more likely associated with black people are used (Sweeney, 2013). Various examples showing bias using gender, age, and race have been studied (see, e.g., (Tambe, Cappelli, and Yakubovich, 2019)). Finally, news feed personalization often amplifies existing beliefs of users due to a severe underrepresentation

[*]Equal contribution [1]CoW Protocol [2]Department of Mathematics, ETH Zurich, Zurich, Switzerland. Correspondence to: Adam Kurpisz <adam.kurpisz@ifor.math.ethz.ch>.

Proceedings of the 39^{th} International Conference on Machine Learning, Baltimore, Maryland, USA, PMLR 162, 2022.

of opposing views. This phenomenon, which is linked to the popularized notions of *filter bubbles* and *echo chambers*, can lead to unintended confirmation bias that increases political and social polarization on an unprecedented scale (see (Barberá, Jost, Nagler, Tucker, and Bonneau, 2015; Barberá, 2020; Kelly, 2021) and references therein).

One of the most effective and heavily studied classes of algorithms for data summarization is clustering. Even though clustering approaches have been developed to address the issue of sensitive attributes, they are often prone to return highly distorted summarizations. The goal of this work is to address this gap for one of the most heavily studied clustering variants, namely the k-Center problem. Moreover, we are interested in very fast algorithms that can scale to large data sets and with provably good worst-case behavior.

Before formally stating our results, we provide relevant details on the k-Center problem and on prior work on variants addressing the two above-mentioned fairness aspects separately. This leads to a canonical model capturing both aspects, which is the one we consider.

1.1. Towards Private and Representative k-Center

In the k-Center problem, we are given a finite metric space (X, d) and we need to choose k points in X, called *centers* or *representatives*, with the goal to minimize the maximum dissimilarity (measured as a distance) between any data point of X and its closest center. Efficient (and tight) 2-approximation algorithms for the k-Center problem have been developed during the last few decades (Gonzalez, 1985; Hochbaum and Shmoys, 1986); however, applied to data sets that include sensitive attributes, they tend to compute biased solutions. Chierichetti, Kumar, Lattanzi, and Vassilvitskii (2017) aimed at addressing this issue by adding a fairness constraint to k-Center, where every point in a data set is assigned one of two colors and the goal is to compute a k-Center clustering such that the ratio between colors in every cluster matches the global ratio. The model attracted significant attention stimulating further algorithmic advances; Backurs, Indyk, Onak, Schieber, Vakilian, and Wagner (2019) gave a nearly linear time algorithm for the problem and Bera, Chakrabarty, Flores, and Negahbani (2019) generalized the setting to multiple colors. A notion of fairness that better fits the nature of data summarization was studied by Kleindessner, Awasthi, and Morgenstern (2019). In this model, which we call *Representative k-Center* (Rep-kC), the input is a finite metric space with colored points and the goal is to construct a k-Center clustering that contains a certain number of centers/representatives from each color class. Kleindessner, Awasthi, and Morgenstern (2019) provide a $(3 \cdot 2^{\gamma-1} - 1)$-approximation algorithm, where γ is the number of colors, with a running time that is linear in the number of data points $n := |X|$. A slower algorithm with a tight 3-approximation guarantee can be constructed by reducing the problem to the Matroid Center problem, which admits a 3-approximation (Chen, Li, Liang, and Wang, 2016). Finally Jones, Nguyen, and Nguyen (2020) recently provided a 3-approximation algorithm for this problem with running time $O(nk)$.

A solution of Representative k-Center indeed succeeds in providing a fair number of representatives from each sensitive group, leading to a diverse set of centers. Unfortunately, prior procedures for Representative k-Center are prone to return summarizations where data points in the summary represent subsets of the original data of vastly different sizes (see also our discussion in Section 5), thus risking to create a distorted picture of the original data set.

One natural approach towards neutralizing such distortions in k-Center solutions was studied by Aggarwal, Panigrahy, Feder, Thomas, Kenthapadi, Khuller, and Zhu (2010), and more recently by Rösner and Schmidt (2018). In this model, which we call the *Private k-Center problem*,[1] every selected center has to represent at least a given amount of data. The private setting has gained significant attention due to its broad spectrum of possible applications. In particular, the terminology *private* or *privacy-preserving* stems from applications where one seeks to obtain a summary that does not reveal information about specific points of the underlying data. Rösner and Schmidt (2018) presented an elegant way to incorporate privacy aspects in several k-Center variants. Unfortunately, their approach does not seem to extend to the representative setting. (And, moreover, the running times obtained through their approach, though polynomial, are significantly higher than the "gold standard" of a running time dependence (nearly) linear in $|X|$.)

In this work, we address the aforementioned limitations by presenting a fast algorithm with hard theoretical guarantees for the natural combination of the above-mentioned models of Representative k-Center and Private k-Center, which we call *Private Representative k-Center* (Priv-Rep-kC). Hence, this is a generalization of the settings studied by Kleindessner, Awasthi, and Morgenstern (2019), Jones, Nguyen, and Nguyen (2020), and Rösner and Schmidt (2018), addressing both the sensitive attribute issue and the issue of summary distortion due to centers/representatives corresponding to clusters of vastly different sizes by combining well-known canonical models that address these issues separately. Formally, Priv-Rep-kC is defined as follows.

Definition 1.1 (Priv-Rep-kC)**.** Given is a finite metric space (X, d) with a partition $\{X_1, \dots, X_\gamma\}$ of X, two integers $k, L \in \mathbb{Z}_{\geq 0}$, and numbers $a_i, b_i \in \{0, \dots, |X|\}$ with

[1] Different names have been used for the problem previously. In particular, it was called (k, r)-*Center* in (Aggarwal, Panigrahy, Feder, Thomas, Kenthapadi, Khuller, and Zhu, 2010) and *Privacy-Preserving k-Center* in (Rösner and Schmidt, 2018).

$a_i \leq b_i$ for $i \in [\gamma]$. A solution corresponds to a set of centers $C \subseteq X$ and an assignment $\phi : X \to C$ such that

(1) $|C| \leq k$,

(2) $a_i \leq |C \cap X_i| \leq b_i$ for every $i \in [\gamma]$, and

(3) $|\phi^{-1}(c)| \geq L$ for every $c \in C$.

The goal is to minimize its *radius*: $\max_{x \in X} d(x, \phi(x))$.

Note that a PRIV-REP-KC instance is feasible if there are enough centers to fulfill the lower bounds in (2) (i.e., $\sum_{i=1}^{\gamma} a_i \leq k$ and $|X_i| \geq a_i$ for $i \in [\gamma]$), and enough points to fulfill the privacy constraints (3) ($L \cdot \sum_{i=1}^{\gamma} a_i \leq |X|$). As these conditions are easy to check, we always assume that the PRIV-REP-KC instances we consider are feasible.

1.2. Our results

Our main result is an algorithm for PRIV-REP-KC with three main features/insights:

(i) it is fast and scales well to large problem sets, as its running time depends only linearly in $|X|$,

(ii) it has a constant-factor worst-case guarantee, and

(iii) computational results show that there can be significant benefits in handling representativeness and privacy together instead of addressing them sequentially.

The following theorem formalizes our theoretical contributions, which are complemented in Section 5 by computational results.

Theorem 1.2. *There is a* 15*-approximation algorithm for* PRIV-REP-KC *that runs in time* $O(nk^2 + k^5)$*, where* n *is the number of points and* k *is the upper bound on the number of centers that are allowed to be opened.*

In particular, this shows that constant-factor approximations can be achieved even when requiring k-Center solutions to be both representative and private. As already k-Center is well-known to be APX-hard, constant-factor guarantees are arguably the best one can hope for (unless P = NP). Moreover, the actual performance of our algorithm as observed in our computational results is significantly stronger than the theoretical worst-case guarantee stated in Theorem 1.2.

1.3. Basic terminology and notation

A *k-Center* instance is a triple (X, d, k) where (X, d) is a finite metric space and $k \in \mathbb{Z}_{\geq 1}$. Adding a privacy bound $L \in \mathbb{Z}_{\geq 1}$ leads to a Private k-Center instance (X, d, k, L). A *clustering* (C, ϕ) of a k-Center instance consists of at most k centers $C \subseteq X$ and a map $\phi : X \to C$, also called *assignment*. Its *radius* is $\max_{x \in X} d(x, \phi(x))$. We call a clustering an *r-clustering* for $r \in \mathbb{R}_{\geq 0}$ if the radius of the clustering is at most r. The clustering is *L-private* (or simply *private*) if $|\phi^{-1}(c)| \geq L \ \forall c \in C$. In a PRIV-REP-KC instance, we call a set of centers $C \subseteq X$ *representative* if they fulfill condition (2) in Definition 1.1. Throughout the text, n will always denote the size $|X|$ of X.

2. Overview of our approach

In order to obtain a private clustering, we compute in a first step a small family of *backbones*. A backbone describes how many centers can be opened in certain areas of the metric space while still being able to obtain a private clustering with a good radius. In a second step, we compute for each backbone the best (in a well-defined way) representative set of centers that corresponds to that backbone; we say that this is a *realization of the backbone*. One of these candidate sets will lead to the clustering we return. Figure 1 provides a sketch of this idea.

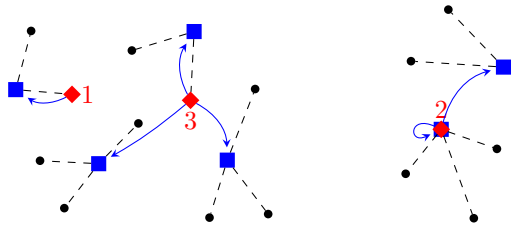


Figure 1. Example of a Private k-Center problem (for some $k \geq 6$) with $L = 3$. The backbone is a set of points Π (red diamonds) with positive integers η_π for $\pi \in \Pi$ (in red). For each $\pi \in \Pi$, we open up to η_π centers (blue rectangles) close to π (highlighted by blue arrows). We call this a *realization*. We construct backbones that guarantee that any centers opened that way admit a private clustering with small radius (highlighted by the dashed lines).

Going through backbones allows us to ignore privacy constraints in the second step, leading to a simpler subproblem that we can solve optimally. The theoretical insight justifying this approach is that we show how to quickly compute a small family of backbones such that one will always lead to a good (constant-factor) PRIV-REP-KC clustering.

We now formally define backbones and realizations thereof, and briefly describe the building blocks of our approach. Later sections expand on the details of these building blocks and provide formal proofs of their guarantees.

Definition 2.1 (backbone)**.** Let (X, d, k) be a k-Center instance. A *backbone* of (X, d, k) is a tuple (Π, η) where $\Pi \subseteq X$ with $|\Pi| \leq k$ and $\eta \in \mathbb{Z}_{\geq 1}^{\Pi}$. Moreover, (Π, η) is a *(ρ, L)-backbone* for $\rho \in \mathbb{R}_{\geq 0}, L \in \mathbb{Z}_{\geq 1}$ if there is a ρ-clustering with centers Π that, for each $\pi \in \Pi$, assigns at least $\eta_\pi \cdot L$ points to π.

One way to think about a (ρ, L)-backbone (Π, η) is that if we open, for each $\pi \in \Pi$, up to η_π many centers close to π, then we have guarantees that there are enough points close

to these centers to obtain a private clustering of small radius. The notion of a Δ-realization defined below formalizes and quantifies this view. (The parameter $\Delta \in \mathbb{R}_{\geq 0}$ measures how far centers are opened from the backbone centers Π.)

Definition 2.2 (Δ-realization of a backbone)**.** Let (X, d, k) be a k-Center instance, (Π, η) a backbone of it, and let $\Delta \in \mathbb{R}_{\geq 0}$. A set $C \subseteq X$ is a Δ-realization of (Π, η) if $|C| \leq k$ and there is a map $\psi : C \to \Pi$ such that

(i) $d(c, \psi(c)) \leq \Delta$ for each $c \in C$, and

(ii) $|\psi^{-1}(\pi)| \in \{1, \dots, \eta_\pi\}$ for each $\pi \in \Pi$.

The definitions of backbones and realizations imply an upper bound on the radius necessary to obtain a private clustering with centers given by a realization.

Observation 2.3. Let (X, d, k, L) be a Private k-Center instance and $C \subseteq X$ a Δ-realization of a (ρ, L)-backbone of (X, d) with $|C| \leq k$. Then, (X, d, k, L) admits a private $(\rho + \Delta)$-clustering with centers C.

Thus, realizations of backbones come with privacy guarantees. Still, to obtain good clusterings for PRIV-REP-KC through a backbone-based approach, the following computational questions remain:

- How to obtain a (ρ, L)-backbone with small ρ for which there is a representative Δ-realization with small Δ?
- Given such a backbone, how to compute a representative Δ-realization for Δ as small as possible?
- Given representative centers, how to compute fast a private r-clustering with these centers and smallest r?

A key challenge in this work is to obtain very fast algorithms that resolve these questions. We answer the first question in a strong form as described in the statement below.

Theorem 2.4. *Let (X, d, k, L) be a Private k-Center instance. One can construct in $O(nk^2)$ time at most k^2 many backbones such that, for any L-private r-clustering (C, ϕ), there is a $(7r, L)$-backbone in the computed group for which C is an $8r$-realization.*

Note that Theorem 2.4 applies to *any* L-private clustering and not just representative ones. This generality allows for applying our approach to Private k-Center problems with additional constraints on the centers beyond PRIV-REP-KC.

The second question is about solving the following problem.

Definition 2.5 (MIN-REP-REALIZATION)**.** Given a PRIV-REP-KC instance and backbone (Π, η), the MIN-REP-REALIZATION problem asks to find a Δ-realization that is representative of smallest possible Δ.

MIN-REP-REALIZATION can be solved through flow techniques analogous to (Jones, Nguyen, and Nguyen, 2020). However, as we need to solve this problem for a family of up to k^2 backbones, we identify and exploit synergies between the subproblems to obtain a sufficiently fast procedure leading to the overall running time of $O(nk^2 + k^5)$.

Algorithm 1 summarizes (in a slightly simplified form) the steps of our algorithm.

Algorithm 1 Simplified version of our PRIV-REP-KC algorithm.

1. Create $q \leq k^2$ backbones $(\Pi_1, \eta_1), \dots, (\Pi_q, \eta_q)$ fulfilling the conditions of Theorem 2.4.
2. **for** $i = 1, \dots, q$ **do**:
   - Solve MIN-REP-REALIZATION for backbone (Π_i, η_i) to get a Δ_i-realization C_i.
   - Compute private clustering with centers C_i of smallest possible radius r_i.
3. **return** the clustering with smallest r_i.

To obtain the claimed running time, we will not perform the second step in the for-loop for each C_i, but use prior computed parameters of the backbones and realization to identify a single C_i for which we perform the second step. As we will see, this can be done in a way that maintains the theoretical worst-case approximation guarantee of 15 that we get for Algorithm 1, as shown below.

Theorem 2.6. *Algorithm 1 is a* 15*-approximation for* PRIV-REP-KC.

Proof. Let $\mathrm{OPT} \subseteq X$ be the set of centers of an optimal PRIV-REP-KC solution and r^* its radius. By Theorem 2.4, there is one backbone (Π, η) among the at most k^2 many backbones we compute such that (i) (Π, η) is a $(7r^*, L)$-backbone, and (ii) OPT is an $8r^*$-realization of (Π, η). Hence, by solving MIN-REP-REALIZATION for (Π, η), we obtain an $8r^*$-realization C of (Π, η) (because OPT is an $8r^*$-realization and we get a Δ-realization for the smallest possible Δ). Observation 2.3 now implies that there is a private $15r^*$-clustering with centers C. Since we compute the private clustering with centers C and the smallest radius, we find a $15r^*$-clustering, as desired. □

As mentioned, the theoretical worst-case bound is very conservative in view of our computational results.

Organization of remaining sections. To compute backbones as claimed in Theorem 2.4, we first construct certain L-private clusterings. This relies on a fast private clustering procedure, discussed in Section 3, which we also reuse later. Section 4 shows how to obtain backbones as claimed in Theorem 2.4 from these clusterings. Our computational results are presented in Section 5. Missing proofs and further details can be found in the appendix. Appendix A provides an overview of the appendices and pointers where to find the missing proofs.

3. Obtaining private clusterings fast

A crucial component of our fast procedure that computes the desired backbones is the ability to solve quickly problems related to private clustering. An elementary problem we need to solve, which highlights some hurdles faced in this context and employed techniques, is the following.

Definition 3.1 (MIN-PRIV-RADIUS)**.** Let (X, d, k, L) be a Private k-Center instance and $C \subseteq X$ with $|C| \leq k$. The task is to find a private r-clustering with centers C and smallest r.

While it is not too difficult to find a polynomial-time algorithm for MIN-PRIV-RADIUS, the challenge lies in obtaining a running time linear in n. We obtain the following.

Lemma 3.2. *An optimal solution to* MIN-PRIV-RADIUS *can be computed in $O(nk^2)$ time.*

To illustrate some of the ideas used, first consider the following arguably easier problem: Decide, for a given radius r, whether a set of centers C admits a private r-clustering. This boils down to finding a way to assign each $x \in X$ to a center in $C \cap B(x, r)$, where $B(x, r) \coloneqq \{y \in X \colon d(y, x) \leq r\}$ is the ball around x of radius r, such that at least L points are assigned to each center. One way to solve this problem that nicely extends to generalized versions used later, is through the following two steps. We first define a maximum flow problem that computes an assignment where each center $c \in C$ gets assigned exactly L points (if this is possible), and then we complete the assignment by assigning the unassigned points to their closest center. The second step is straightforward, and the first one is readily modeled as a maximum cardinality assignment problem where up to L points can be assigned to each center and $x \in X$ can be assigned to $c \in C$ if and only if $d(x, c) \leq r$. Given that the second step respects the radius r, a private r-clustering with centers C exists if and only if this assignment problem admits an assignment of cardinality $|C| \cdot L$. This is a special case of Maximum Cardinality b-Matching, which can be cast as a maximum flow problem and solved through standard max-flow algorithms. However, as the corresponding flow graph can have up to $\Theta(kn)$ edges and $\Theta(n)$ nodes, obtaining the desired $O(nk^2)$ runtime is not immediate. Nevertheless, an $O(nk^2)$ runtime can be achieved through a carefully designed and implemented augmenting path procedure (see Appendix B), which even extends to a more general problem that we need to solve, as discussed later.

The above discussion about computing private r-clusterings with centers C naturally leads to the following approach for MIN-PRIV-RADIUS. We can construct the above-described flow problem, parameterized by r, stepwise from smaller to larger r until the first r is obtained for which the maximum flow value is $|C| \cdot L$. A key observation is that the only change in the flow problem when r increases is that more edges get introduced. This allows for reusing a flow computed for a smaller r as a starting flow for a larger r. However, there remains a hurdle towards obtaining a running time linear in n. Namely, a canonical realization of this idea requires to sort the $\Theta(nk)$ (in the worst case) point-center distances $d(x, c)$ for $x \in X$ and $c \in C$ to be able to introduce corresponding edges in the assignment problem in the right order; even just this step would introduce an additional $\log n$ factor. We show how this can be avoided by not sorting all point-center distances, but doing a more coarse bucket-wise sorting based on a repeated application of a linear-time selection algorithm.

Remarks on running time. A natural approach for PRIV-REP-KC is to first compute a representative clustering, for example by using the current state-of-the-art algorithm of Jones, Nguyen, and Nguyen (2020), to obtain a set of centers C and then solve MIN-PRIV-RADIUS with centers C to obtain a private clustering. (In our computational results, we will compare our algorithm against this natural benchmark.) Any such procedure already inherits a running time dependency of nk^2 from MIN-PRIV-RADIUS. For small k, this scales like the running time of our $O(nk^2 + k^5)$ PRIV-REP-KC procedure. Hence, making solutions private seems to be a natural bottleneck when designing algorithms for PRIV-REP-KC. Also, solving MIN-PRIV-RADIUS for centers coming from a representative clustering algorithm like the one of Jones, Nguyen, and Nguyen (2020), voids its performance guarantee and can lead to PRIV-REP-KC clusterings with arbitrarily bad radii.

3.1. Private clusterings

This section focused so far on finding private clusterings assuming that the centers were given. To obtain a good set of centers for our backbones we rely on a method of Gonzalez (1985), which is the centerpiece of a classic 2-approximation for k-Center. Given a k-Center instance (X, d, k), it computes in $O(nk)$ time a chain $C_1 \subseteq \ldots \subseteq C_k \subseteq X$ of center candidates as described in Algorithm 2. For k-Center, one can show that the final set C_k is a set of centers leading to a 2-approximation (Gonzalez, 1985). However, for our purposes, we retain the full chain $(C_1, \ldots, C_k)$. For brevity, we call any set C_i a *Gonzalez prefix*, as it is a prefix of the centers $c_1, \ldots, c_k$.

Algorithm 2 Gonzalez' algorithm.

1. $C_0 = \emptyset$.
2. **for** $i = 1, \ldots, k$ **do**:
   - Determine $c_i \in \arg\max_{x \in X} \{d(x, C_{i-1})\}$.
   - $C_i \coloneqq C_{i-1} \cup \{c_i\}$.
3. **return** $(C_1, \ldots, C_k)$.

Interestingly, as later discussed in Section 4, the Gonzalez prefixes can be used as backbone centers that, if equipped with well-chosen η-vectors, lead to a family of backbones as claimed in Theorem 2.4. Before moving to these backbones, we show how Gonzalez prefixes can be used to obtain fast private clusterings by presenting a very fast and tight 2-approximation for Private k-Center, the most elementary private variant of k-Center. This allows us to present an instructive and simpler version of reasonings that are central in obtaining guarantees on backbones introduced later.

Lemma 3.3. *Algorithm 3 is a 2-approximation for Private k-Center that can be implemented to run in $O(nk^2)$ time.*

Algorithm 3 2-approximation for Private k-Center

1. Compute Gonzalez prefixes $(C_1, \dots, C_k)$.
2. Solve MIN-PRIV-RADIUS for each C_i to obtain a clustering (C_i, ϕ_i).
3. **return** clustering (C_j, ϕ_j) of smallest radius.

A straightforward implementation of Algorithm 3 that invokes Lemma 3.2 to solve MIN-PRIV-RADIUS independently for each C_i would lead to a running time of $O(nk^3)$. As we discuss in Appendix B, one can exploit that $C_1, \dots, C_k$ form a chain, which allows for warm-starting our maximum flow algorithm when dealing with the centers C_{i+1} by using the flow computed for C_i. For this to work, we crucially exploit that the minimum radius r_i needed to assign at least L points to each center of C_i is non-decreasing in the index i. This is needed for the warm-start because it makes sure that the flow computed for C_i does not use point-center distances exceeding r_{i+1}.

We highlight that our $O(nk^2)$ running time for Private k-Center is significantly faster than prior techniques. In particular, it is the first constant-factor approximation (and the factor 2 is best possible unless $\mathtt{P} = \mathtt{NP}$) whose running time dependence on n is linear. In comparison, the running time dependence on n of a prior 2-approximation by Aggarwal, Panigrahy, Feder, Thomas, Kenthapadi, Khuller, and Zhu (2010) is at least quadratic. Similarly, a prior very general approach of Rösner and Schmidt (2018) to add privacy constraints to different variants of k-center problems has a superlinear running time dependence on n (and, unfortunately, also does not extend to PRIV-REP-KC).

Before continuing with the construction of the backbones, we prove the approximation guarantee of Algorithm 3.

Proof. (Algorithm 3 is a 2-approximation for Private k-Center.) Let $\mathrm{OPT} \subseteq X$ be an optimal set of centers for the given Private k-Center instance with optimal radius r^*. Consider the Gonzalez prefixes $C_1, \dots, C_k$, and, for $i \in [k]$, let r_i be the smallest radius such that $d(x, C_i) \leq r_i$ for each $x \in X$. In words, r_i is the smallest radius for which a k-Center clustering with centers C_i exists. We have $r_k \leq 2r^*$ because, as mentioned, the Gonzalez prefix C_k gives a 2-approximation for the k-Center problem, and the optimal radius for the k-Center problem is no larger than r^*. Let $\ell \in [k]$ be the lowest index satisfying $r_\ell \leq 2r^*$. Because Gonzalez' algorithm picks farthest points as new centers, we have $d(c, c') > 2r^*$ for all distinct $c, c' \in C_\ell$. We complete the proof by showing that there is a private $2r^*$-clustering with centers C_ℓ. For each $c \in C_\ell$ there is a center $q \in \mathrm{OPT} \cap B(c, r^*)$ because each point is no more than r^* away from the closest point in OPT. Hence, all (at least L many) points that the optimal clustering assigns to q are within $B(q, r^*) \subseteq B(c, 2r^*)$, and can thus be assigned to c in a $2r^*$-clustering with centers C_ℓ. This shows that a radius of $2r^*$ allows for assigning at least L points to each center of C_ℓ. Assigning all remaining unassigned points to the closest center in C_ℓ leads to a private $2r^*$-clustering. □

4. Constructing good backbones

The previous section showed a fast procedure for Private k-Center, which does not impose additional constraints on the centers to be opened. Unfortunately, when the centers need to fulfill further constraints, such a simple procedure fails. We illustrate this on an instructive special case of PRIV-REP-KC, where we consider a Private k-Center instance (X, d, k, L) together with a lower bound $a \in \mathbb{Z}_{\geq 0}$ on the number of centers that need to be opened. Hence, the family of feasible sets of centers is $\{C \subseteq X \colon a \leq |C| \leq k\}$. Consider the example instance shown in Figure 2.

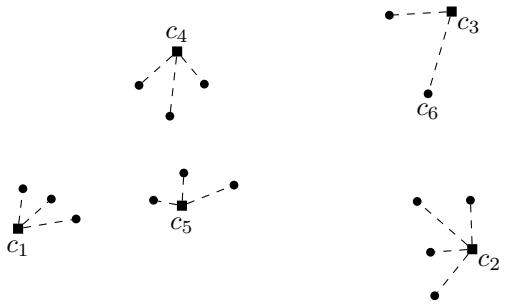


Figure 2. Private 6-Center example in the plane with $L = 3$. Exactly 6 centers must be opened ($a = 6$). The first six centers found by Gonzalez' algorithm are $c_1, \dots, c_6$. The dashed lines show a private clustering with centers $C_5 \coloneqq \{c_1, \dots, c_5\}$ of smallest radius. Though this solution opens too few centers, we construct a backbone from it that leads to a good clustering with 6 centers.

Note that opening all 6 centers leads to a large private radius because only 8 points are on the right part of the instance where the 3 centers c_2, c_3, and c_6 got opened. Hence, to obtain privacy, at least one point from the left part needs to be assigned to a far-away center (c_2, c_3, or c_6) on the right.

By increasing the gap between left and right points, this can lead to an arbitrarily large radius.

We overcome this by starting with a private clustering (C, ϕ) of smallest radius with centers $C := C_5$, as highlighted in Figure 2 (this corresponds to solving MIN-PRIV-RADIUS for the centers C as discussed in Section 3). This clustering has a good radius but opens one fewer center than required. If there was a center $c \in C$ with at least $2L$ points assigned to it (i.e., $|\phi^{-1}(c)| \geq 2L$), then we could open an additional center $c' \in \phi^{-1}(c)$ and reassign L points from center c to c', without increasing the radius by much. However, in general this may not be the case, as in our example in Figure 2.

Instead of opening a further center right away, we first transfer/reassign points from some centers to close-by centers with the goal to obtain a center with at least $2L$ points assigned to it. We achieve this with an idea inspired by a technique introduced in (An, Bhaskara, Chekuri, Gupta, Madan, and Svensson, 2015) for Capacitated k-Center. Concretely, for a private clustering (C, ϕ), we first set a *transfer threshold* $\tau \in \mathbb{R}_{\geq 0}$. We then consider a minimum spanning forest (C, F) in the graph (C, E_τ) with $E_\tau := \{\{c, c'\} \mid d(c, c') \leq \tau\}$, where the length of an edge $\{c, c'\}$ is $d(c, c')$. We call (C, F) the *transfer forest*. We fix an arbitrary root in each connected component of the transfer forest. We then go through the centers C in each component from leaves to root. (Formally, we need that whenever we consider a vertex, all its descendants have already been considered.) When considering a center $c \in C$, say with n_c many points assigned to it, we transfer $n_c - L \cdot \lfloor n_c / L \rfloor$ many of these points to the neighbor $\overline{c}$ of c in (C, F) that is closer to the root. We only transfer points from c to $\overline{c}$ that have originally be assigned to c by ϕ, and not points that got assigned to c through a prior transfer. This is always possible because each center has at least L points originally assigned to it, as (C, ϕ) is a private clustering and the number of points to transfer is strictly below L. (Even though this is not necessary for the theoretical guarantees, we transfer the ones closest to $\overline{c}$.) See Figure 3 for an example.

Let $\widetilde{\phi} : X \to C$ be the new assignment. We now define a backbone (Π, η) with centers $\Pi = C$ and, for $c \in C$, we set $\eta_c = \lfloor |\widetilde{\phi}^{-1}(c)| / L \rfloor$. Note that in the example in Figure 3, we can now find, starting from this backbone, a Δ-realization with the required 6 centers and small radius. We call a backbone obtained from (C, ϕ) a *τ-aggregation* of (C, ϕ).

The following is the key structural result in obtaining Theorem 2.4. It shows that τ-aggregations of Gonzalez prefixes lead to good backbones.

Theorem 4.1. *Let (X, d, k, L) be a Private k-Center instance and $(\overline{C}, \overline{\phi})$ a private r-clustering. Then there exists a Gonzalez prefix C and transfer threshold $\tau \in \mathbb{R}_{\geq 0}$ such that the following holds. For any private clustering (C, ϕ) with centers C and smallest radius and any backbone (Π, η) that is a τ-aggregation of (C, ϕ), we have*

(i) (Π, η) is a $(7r, L)$-backbone, and

(ii) $\overline{C}$ is a $8r$-realization of (Π, η).

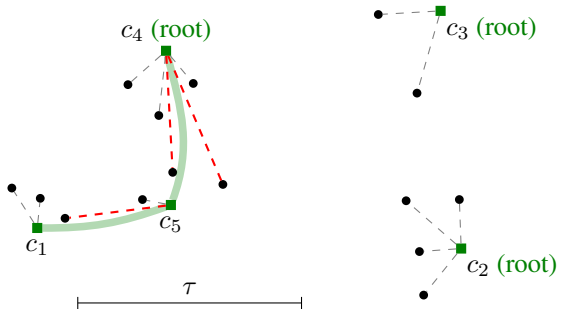


Figure 3. Example of how transfers are done for the instance in Figure 2 with transfer threshold τ as indicated. The transfer forest is shown in green. It has three connected components: $\{c_1, c_4, c_5\}$, $\{c_2\}$, and $\{c_3\}$. In the only non-trivial component, i.e., $\{c_1, c_4, c_5\}$, we chose c_4 as (arbitrary) root. The dashed red lines highlight points that got transferred. These transfers lead to a backbone with centers C and all η-values being 1 except for c_4, which has an η-value of 2.

We need to consider different Gonzalez prefixes to obtain a backbone with a number of centers that is a good fit for (C, ϕ). Too many backbone centers can make it expensive to obtain privacy, as any realization of the backbone needs to open at least one center per backbone center, whereas too few backbone centers may require a very large radius to cover all points.

To prove Theorem 4.1, we show that, for one Gonzalez prefix C, there is a well-chosen (and not too large) transfer threshold τ such that, for any center $\overline{c} \in \overline{C}$, all points in $B(\overline{c}, r)$ get assigned (by the clustering (C, ϕ)) to centers in the same connected component of the transfer forest. This ensures that each connected component of the transfer forest has enough total η-value to open one center for each center $\overline{c} \in \overline{C}$ from whose r-ball it got at least one point. Whereas this shows that each connected component of the transfer forest has *globally* enough points, we still have to show that things also work out *locally*; in other words, that point (ii) of Theorem 4.1 holds. This can be analyzed separately for each connected components of the transfer forest and boils down to showing that, for any subset of $\overline{C}$-centers with points assigned to the same connected component of the transfer forest, there is enough total η-value close-by to achieve an $8r$-realization of these centers with the considered backbone. This reduces to proving the existence of a certain bipartite matching, which we show by using Hall's Theorem.

To ensure that only k^2 backbones are needed to obtain the guarantees of Theorem 2.4, we observe that for each Gon-

zalez prefix C at most k values of τ need to be considered; indeed, we only need to consider values of τ leading to different transfer forests. This results in no more than k relevant values for τ as transfer forests are forests encountered in Kruskal's algorithm when finding a minimum spanning tree in the complete graph with vertices C and distances given by d. Hence, instead of considering τ explicitly, we consider all forests encountered in Kruskal's algorithm and use those as transfer forests; see Appendix C for more details.

5. Computational results

We now validate our algorithm empirically with a twofold goal. First, we motivate the fairness considerations in this paper by showing that the state-of-the-art algorithm for the representative k-Center by Jones, Nguyen, and Nguyen (2020) is prone to return solutions of vastly different sizes of clusters. Second, we show that these solutions cannot always easily be modified to strong private clusterings by simply solving MIN-PRIV-RADIUS on their centers to obtain a private clustering. Solutions obtained this way, which sequentially resolve the two fairness issues, are compared with solutions computed with our algorithm, which treats both fairness aspects simultaneously.

Data sets. We use the following real data sets:

- The *Adult* data set (Kohavi and Becker (1996)) contains records about individuals extracted from the 1994 US census (Kohavi, 1996). The dataset has 32561 records. Following the experiments in (Jones et al., 2020) we chose the numeric attributes *age*, *fnlwgt*, *education-num*, and *hours-per-week* to represent points in $\mathbb{R}^4$ (distances in all datasets are Euclidean). *Race*, which takes 5 different values, is chosen as sensitive attribute (hence, we have 5 colors). In the clustering context, the dataset was also used in (Chierichetti et al., 2017; Bera et al., 2019; Backurs et al., 2019; Harb & Lam, 2020; Esmaeili et al., 2020; Halabi et al., 2020).

- The *Diabetes* data set (Strack, DeShazo, Gennings, Olmo, Ventura, Cios, and Clore (1996)) includes information from 130 US hospitals over 10 years about patients suffering from diabetes. It has 101763 records. We chose numeric attributes *age* and *time-in-hospital* to represent points in $\mathbb{R}^2$ and *gender* [male/female] as the sensitive attribute. This dataset has been used previously in clustering context in (Chierichetti et al., 2017; Chen et al., 2019; Backurs et al., 2019).

- The *Query* data set (Anagnostopoulos (2019)) includes synthetic range and radius query workloads derived from Gaussian distributions over the real data set that reports crimes in Chicago.[2] We chose the *count* version that consists of 10000 points. To represent points in the space, we chose attributes *X-coordinate* and *Y-coordinate*. *Count* is the sensitive attribute, which takes 7 different values.

- The *Electric* data set (Hebrail and Berard (2012)) includes 2049280 noncorrupted measurements gathered in a house located in Sceaux (7km of Paris, France) between December 2006 and November 2010. To represent points, we selected attributes *Global_active_power* and *Global_reactive_power*. The sensitive attribute is was chosen to be *Sub_metering_3*, which takes 32 different values.

Algorithms. We run four different algorithms:

- JNN algorithm (Jones, Nguyen, and Nguyen, 2020). It is the state-of-the-art procedure both in running time and performance guarantee for Representative k-Center. We use the *Alg 2-Seq* variant of the algorithm.

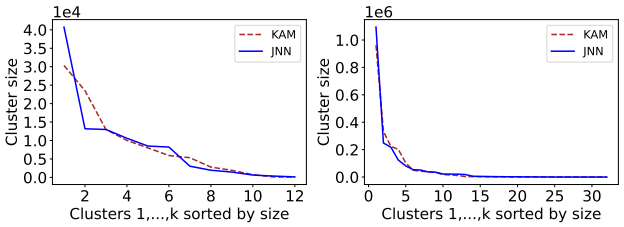
- KAM algorithm (Kleindessner, Awasthi, and Morgenstern, 2019) for Representative k-Center, used in (Jones, Nguyen, and Nguyen, 2020) as a baseline.

- JNN+PRIV. First representative centers C are computed via JNN. Then MIN-PRIV-RADIUS is solved with centers C to obtain the final clustering.

- Our algorithm that solves PRIV-REP-KC by treating both fairness aspects simultaneously.

Results. Because JNN does not allow for setting a lower bound on the number of centers for each color, and in fact always opens the upper bound of centers for each color, we set $a_i = b_i$ for all $i \in [\gamma]$. The required number of centers per group is set proportional to the size of the group.

We first run JNN and KAM on the datasets *Diabetes* and *Electric* for $k = 12$ and $k = 32$, respectively. The sizes of the computed clusters are highlighted in Figure 4.



(a) Diabetes: $k = 12$ (b) Electric: $k = 32$

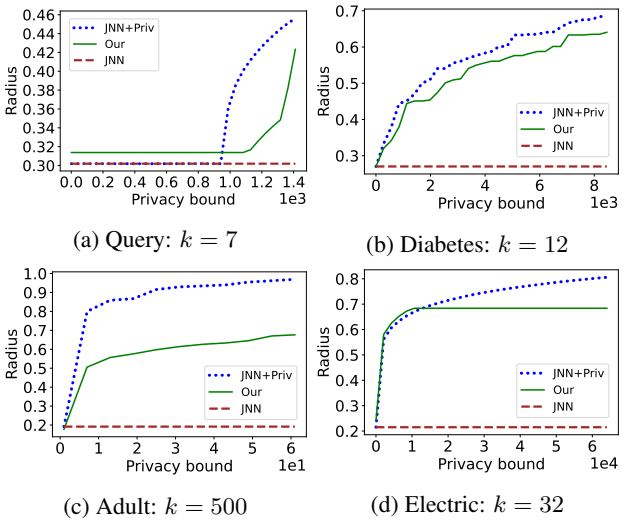
Figure 4. Cluster sizes computed with JNN and KAM for datasets Diabetes and Electric for sample values of k and L.

Observe that cluster sizes computed by JNN and KAM are similar and can vary strongly in their sizes. For the Diabetes

[2] https://data.cityofchicago.org/Public-Safety/Crimes-2001-to-Present/ijzp-q8t2

data set (Figure 4a), around 40% of points are assigned to a single center. The situation is even more extreme for the Electric data set (Figure 4b). Here, one center represents more than 53% of points, and the 16 smallest clusters cover together less than 0.7% of the data.

Computing non-distorted solutions, where clusters have similar sizes is a key challenges in data summarization. We thus compare in our second experiment the JNN+PRIV algorithm, which is based on the state-of-the-art JNN algorithm for REP-KC problem, to our algorithm on all four datasets. The results are shown in Figure 5; the x-axis represents the used privacy bound L, which spans the full range from 1 up to $\lfloor n/k \rfloor$ in regular increments.



(a) Query: $k = 7$ (b) Diabetes: $k = 12$

(c) Adult: $k = 500$ (d) Electric: $k = 32$

Figure 5. Comparison of JNN+PRIV and our algorithm. The value obtained by JNN (which returns non-private clusterings) is shown for reference.

We observe gains in terms of the radius of up to 30% between our algorithm and JNN+PRIV. Moreover, our algorithm outperforms JNN+PRIV on the whole range of privacy bounds for the datasets *Diabetes* and *Adult*. In the other cases, it returns solutions of quality comparable to JNN+PRIV.

We note that our algorithm has comparable worst-case and practical running time compared to JNN+PRIV as the bottleneck of both procedures is satisfying the privacy constraints (see Section 3 for more details).

To prove the scalability of our algorithm, some experiments were run on large datasets of more than 10^6 records. On regular notebook hardware equipped with Intel(R) Core(TM) i7-10510U CPU @ 1.80GHz (8 cores) and 16 GB of RAM, our algorithm for the *Diabetes* instance with more than 10^5 records for $k = 12$ runs below 1 second and for the *Electric* dataset with more than $2 \cdot 10^6$ records and $k = 32$ the algorithm terminates within 2 minutes.

Acknowledgments

The first author would like to thank Stefan Canzar for suggesting to investigate the Representative k-Center problem, which served as the starting point for this work.

This project received funding from Swiss National Science Foundation grants 200021_184622, PZ00P2_174117 and the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 817750).

References

Aggarwal, G., Panigrahy, R., Feder, T., Thomas, D., Kenthapadi, K., Khuller, S., and Zhu, A. Achieving anonymity via clustering. *ACM Trans. Algorithms*, 6(3):49:1–49:19, 2010.

Ali, M., Sapiezynski, P., Bogen, M., Korolova, A., Mislove, A., and Rieke, A. Discrimination through Optimization: How Facebook's Ad Delivery Can Lead to Biased Outcomes. *Proceedings of the ACM on Human Computer Interaction (HCI)*, 3(CSCW), 2019.

An, H.-C., Bhaskara, A., Chekuri, C., Gupta, S., Madan, V., and Svensson, O. Centrality of trees for capacitated k-center. *Mathematical Programming, Series B*, 154:29–53, 2015.

Anagnostopoulos, C. Query analytics workloads dataset data set. UCI Machine Learning Repository, 2019. https://archive.ics.uci.edu/ml/datasets/Query+Analytics+Workloads+Dataset.

Backurs, A., Indyk, P., Onak, K., Schieber, B., Vakilian, A., and Wagner, T. Scalable Fair Clustering. In *Proceedings of the 36th International Conference on Machine Learning (ICML)*, pp. 405–413, 2019.

Barberá, P. *Social Media and Democracy*, chapter Social Media, Echo Chambers, and Political Polarization, pp. 34–55. Cambridge University Press, 2020.

Barberá, P., Jost, J. T., Nagler, J., Tucker, J. A., and Bonneau, R. Tweeting From Left to Right: Is Online Political Communication More Than an Echo Chamber? *Psychological Science*, 26(10):1531–1542, 2015.

Bera, S. K., Chakrabarty, D., Flores, N., and Negahbani, M. Fair Algorithms for Clustering. In *Proceedings of the 33rd Annual Conference on Neural Information Processing Systems (NeurIPS)*, pp. 4955–4966, 2019.

Celis, L. E., Huang, L., and Vishnoi, N. K. Multiwinner Voting with Fairness Constraints. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 144–151, 2018.

Chen, D. Z., Li, J., Liang, H., and Wang, H. Matroid and Knapsack Center Problems. *Algorithmica*, 75(1):27–52, 2016.

Chen, X., Fain, B., Lyu, L., and Munagala, K. Proportionally Fair Clustering. In *Proceedings of the 36th International Conference on Machine Learning (ICML)*, pp. 1032–1041, 2019.

Chierichetti, F., Kumar, R., Lattanzi, S., and Vassilvitskii, S. Fair Clustering Through Fairlets. In *Proceedings of the 31st Annual Conference on Neural Information Processing Systems (NIPS)*, pp. 5029–5037, 2017.

Do, V. H., Elbassioni, K., and Canzar, S. Sphetcher: Spherical Thresholding Improves Sketching of Single-Cell Transcriptomic Heterogeneity. *iScience*, 23(6):101126, 2020. ISSN 2589-0042.

Esmaeili, S. A., Brubach, B., Tsepenekas, L., and Dickerson, J. Probabilistic Fair Clustering. In *Proceedings of the 34th Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2020.

Feldman, M., Friedler, S. A., Moeller, J., Scheidegger, C., and Venkatasubramanian, S. Certifying and Removing Disparate Impact. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 259–268, 2015.

Gonzalez, T. F. Clustering to Minimize the Maximum Intercluster Distance. *Theoretical Computer Science*, 38: 293–306, 1985.

Halabi, M. E., Mitrovic, S., Norouzi-Fard, A., Tardos, J., and Tarnawski, J. Fairness in Streaming Submodular Maximization: Algorithms and Hardness. In *Proceedings of the 34th Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2020.

Harb, E. and Lam, H. S. KFC: A Scalable Approximation Algorithm for k-Center Fair Clustering. In *Proceedings of the 34th Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2020.

Hebrail, G. and Berard, A. Individual household electric power consumption data set. UCI Machine Learning Repository, 2012. https://archive.ics.uci.edu/ml/datasets/individual+household+electric+power+consumption.

Hochbaum, D. S. and Shmoys, D. B. A unified approach to approximation algorithms for bottleneck problems. *Journal of the ACM*, 33(3):533–550, 1986.

Holst, A. Volume of data/information created, captured, copied, and consumed worldwide from 2010 to 2025, 2021. Statista report.

International Data Corporation (IDC). The Growth in Connected IoT Devices Is Expected to Generate 79.4zb of Data in 2025, According to a New IDC Forecast, 2019. https://www.idc.com/getdoc.jsp?containerId=prUS45213219.

Jones, M., Nguyen, H., and Nguyen, T. Fair k-Centers via Maximum Matching. In *Proceedings of the 37th International Conference on Machine Learning (ICML)*, pp. 4940–4949, 2020.

Kelly, M. Political polarization and its echo chambers: Surprising new, cross-disciplinary perspectives from princeton. Princeton News, dec 2021. https://www.princeton.edu/news/2021/12/09/political-polarization-and-its-echo-chambers-surprising-new-cross-disciplinary.

Kleinberg, J. and Tardos, E. *Algorithm Design*. Addison Wesley, 2006.

Kleindessner, M., Awasthi, P., and Morgenstern, J. Fair k-Center Clustering for Data Summarization. In *Proceedings of the 36th International Conference on Machine Learning (ICML)*, pp. 3448–3457, 2019.

Kohavi, R. Scaling up the Accuracy of Naive-Bayes Classifiers: A Decision-Tree Hybrid. 1996.

Kohavi, R. and Becker, B. Adult data set. UCI Machine Learning Repository, 1996. https://archive.ics.uci.edu/ml/datasets/adult.

Marr, B. How Much Data Do We Create Every Day? The Mind-Blowing Stats Everyone Should Read, 2018. https://www.forbes.com/sites/bernardmarr/2018/05/21/how-much-data-do-we-create-every-day-the-mind-blowing-stats-everyone-should-read/?sh=45f0654460ba.

Peters, O. R. L. Pattern-defeating Quicksort. *CoRR*, abs/2106.05123, 2021. URL https://arxiv.org/abs/2106.05123.

Rösner, C. and Schmidt, M. Privacy Preserving Clustering with Constraints. In *Proceedings of the 45th International Colloquium on Automata, Languages, and Programming (ICALP)*, pp. 96:1–96:14, 2018.

Strack, B., DeShazo, J. P., Gennings, C., Olmo, J. L., Ventura, S., Cios, K. J., and Clore, J. N. Diabetes 130-us hospitals for years 1999-2008 data set. UCI Machine Learning Repository, 1996. https://archive.ics.uci.edu/ml/datasets/diabetes+130-us+hospitals+for+years+1999-2008.

Sweeney, L. Discrimination in Online Ad Delivery. *Communications of the ACM*, 56(5):44–54, 2013.

Tambe, P., Cappelli, P., and Yakubovich, V. Artificial Intelligence in Human Resources Management: Challenges and a Path Forward. *California Management Review*, 61 (4):15–42, 2019.

A. Overview of appendix

In this section, we give an overview of how the rest of the appendix is structured.

In Appendix B we expand on the material of Section 3, and present an algorithm that computes for a given ordered set of centers C the optimal private assignments for all prefixes of C in time $O(nk^2)$. In particular, this proves Lemma 3.2 (the formal proof is given at the end of Appendix B).

Appendix C corresponds to the material of Section 4; more precisely, we present a procedure that generates $\binom{k}{2}$ many backbones.

Then, in Appendix D we present an algorithm that optimally solves the MIN-REP-REALIZATION problem (see Definition 2.5) in time $O(nk + k^4)$. Even better we can solve the MIN-REP-REALIZATION problem for a whole batch of backbones in that time, as long as they share the same set of centers. This enables us to compute the best realization (fulfilling the representative condition) of the $\binom{k}{2}$-many backbones in a total running time of $O(nk^2 + k^5)$, as they can be grouped into k such batches.

We continue with Appendix E, where we select among the $\binom{k}{2}$-many realizations one that satisfies the performance guarantee and has also in practice a private assignment with small radius.

Finally, in Appendix F we put everything together and present the full algorithm for solving PRIV-REP-KC and prove the total running time of $O(nk^2 + k^5)$.

Afterwards, in Appendix G, we give a formal proof of the performance guarantee of our algorithm, including the formal proof of Theorem 2.4 and Theorem 4.1. Together with the proven running times this provides the proof for the main theorem (Theorem 1.2).

Throughout the rest of these appendices, we use the following notation. (X, d) denotes a finite metric space, k denotes the upper bound on the number of centers we are allowed to open, and L denotes the privacy lower bound. For simplicity, we also write *private* instead of *L-private*. For the sake of convenience, we assume that $k \leq n/L$, since there exists no L-private assignments if k is larger than n/L; thus, if we are given a k such that $k > n/L$, then we simply redefine its value and set $k := \lfloor n/L \rfloor$. Similarly, we assume that $k \leq \sum_{\ell \in \gamma} b_\ell$, as this is a natural upper bound on the number of centers that can be opened. We also use the notion of a partial assignment $\phi \colon X \to C$, i.e., an assignment of points to centers for which the mapping ϕ might not be defined for all points $x \in X$ yet; in such a context, we say that a center $c \in C$ is *private* if there are already at least L points assigned to it, i.e., $\left|\phi^{-1}(c)\right| \geq L$.

B. Full analysis of results of Section 3: Obtaining private clusterings fast

In this section, we elaborate on the material of Section 3, and in particular, we give a formal proof of the running time stated in Lemma 3.3. We remind the reader here that a proof of the approximation factor of 2 was already given in Section 3. It is straightforward to adapt the ideas presented here in order to obtain an algorithm for MIN-PRIV-RADIUS that runs in time $O(nk^2)$, and thus prove Lemma 3.2, and so we do not explicitly do so.

We will mainly focus on step (2) of Algorithm 3 and give a fast implementation of it. As a reminder, we are given a finite metric space (X, d), a number $L \in \mathbb{N}_{\geq 0}$ (privacy bound) and a set of centers $C_k := \{ c_1, c_2, \ldots, c_k \} \subseteq X$, for some $k \in \mathbb{N}_{\geq 1}$, and the goal is to solve MIN-PRIV-RADIUS for each C_i, $i \in [k]$, where $C_i = \{c_1, \ldots, c_i\}$. Throughout, we often use the notation $C := C_k$.

Our goal is to develop an algorithm that implements the above in total time $O(nk^2)$. More precisely, for each $i \in [k]$, we compute an assignment $\phi_i \colon X \to C_i$ that satisfies $\left|\phi_i^{-1}(c)\right| \geq L$ for every $c \in C_i$, and its radius $\max_{x \in X} d(x, \phi_i(x))$ is minimum over all possible private assignments. As noted, simply achieving polynomial running time is a much easier task, but our goal here is to respect the $O(nk^2)$ running time bound.

Overview of algorithm. At a high level, we process the sets C_i, $i \in [k]$, in increasing index order, and for each C_i, we create a flow network in such a way so that a maximum flow computation gives a (partial) private assignment. In order to find the minimum radius for which such an assignment exists, we iteratively add edges (that directly correspond to distances between centers and points) to the flow network, in increasing order of length, and check whether there is an augmenting path in the network. In case there is, we increase the flow. Once the maximum flow reaches a certain value, then we move to the next set C_{i+1} while maintaining the current state of the flow network, so as to avoid recomputations. In other words, we

build on the network constructed for C_i in order to compute a private assignment for the set C_{i+1}. The reason we can do this is that the optimal radius for a set C_i is always smaller or equal than the optimal radius for C_{i+1}, since $C_i \subset C_{i+1}$.

In order to ensure that we achieve the desired running time, we need to be careful not to explicitly sort the edges with respect to their length. We avoid this by observing that we can partition the edges into buckets such that an edge in a bucket with a smaller index corresponds to a distance that is smaller than an edge in a bucket with a larger index, and we can then process these buckets, one by one, to increase the maximum flow (i.e., instead of adding edges in increasing length order, one by one, we add the edges of a bucket all together). Once we find a bucket that allows the maximum flow to reach the desired value, we then focus only on this bucket, which contains a smaller number of edges, and identify the edge corresponding to the optimal distance.

We start by describing all the steps of the algorithm in Appendices B.1 to B.4, we then present the whole algorithm in Appendix B.5 and finally we prove that the running time is $O(nk^2)$ in Appendix B.6.

B.1. The flow network

The key idea of the algorithm is to sequentially solve maximum flow problems in a iteratively growing graph. In other words, we consider each set C_i, in increasing index order, and construct a flow network corresponding to computing a private assignment based on the flow network constructed for the set C_{i-1}. Moreover, we avoid recomputing the maximum flow from scratch, as we can use the already computed maximum flow for the previous networks in order to speed up the computations.

Every flow network that we consider, denoted as $N_{C_i}(r)$, is parameterized by the set C_i, $i \in [k]$, and a candidate radius $r \geq 0$. In particular, we have a node for every center in C_i, and we also have a separate copy of the whole point set X; note that this implies that there are two nodes corresponding to a center $c \in C_i$, one as part of the copy of C_i and one as part of the copy of X. Moreover, there is a source node s that is connected to each center in $c \in C_i$ with an arc (s, c) of capacity L. Each center $c \in C_i$ is connected to every point $x \in X$ with an arc (c, x) of capacity 1, if and only if $d(c, x) \leq r$. Finally, every point $x \in X$ is connected to the sink t with an arc (x, t) of capacity 1. Let $\mathcal{R} \coloneqq \{(c, x) : c \in C_k, x \in X\}$; we have $|\mathcal{R}| = nk$. Note that each network $N_{C_i}(r)$ contains a subsets of the arcs in $\mathcal{R}$. It will be convenient to associate each edge $e = (c, x) \in \mathcal{R}$ with the distance $d(c, x)$, and so from now on we call $d(e)$ the distance label of $e \in \mathcal{R}$.

For a set C_i, the high level idea is to start with $N_{C_i}(0)$ and gradually insert each arc $(c, x) \in \mathcal{R}$ with $c \in C_i$, one by one, in increasing order of the distance labels. Since s and t are disconnected the maximum s-t flow value in $N_{C_i}(0)$ is 0. We now insert the relevant edges in $\mathcal{R}$ one by one in increasing order of the distance label $d(e)$, while maintaining a maximum s-t flow. As soon as the maximum flow value reaches $i \cdot L$, we stop. The maximum flow f_i can now be transformed to an optimal (i.e., of minimum radius) private partial assignment $\phi_i \colon X' \to C$ by setting $X' \coloneqq \{\, x \in X : \ f_i(x, t) = 1 \,\}$ and $\phi_i(x) \coloneqq c$ for every $x \in X'$, where $c \in C_i$ is the unique center such that $f_i(c, x) = 1$. By construction, and since the flow value is $i \cdot L$, it is easy to see that the privacy condition is satisfied, and in particular, we have $|\phi_i^{-1}(c)| = L$ for every $c \in C_i$. Moreover, the radius of this partial assignment is minimum, as we processed the edges in $\mathcal{R}$ in increasing order of their distance label, and prior to the addition of the last edge that gave a flow of value $i \cdot L$, the flow value was strictly smaller, and so there could not have been any private partial assignment with smaller radius. In order to obtain an optimal private full assignment one can then assign all unassigned points to their nearest center in C_i; it is not hard to see that the radius of the resulting assignment is the optimal radius for the MIN-PRIV-RADIUS problem with input C_i.

The above already describes an algorithm that finds the optimal private assignment for C_i; however, its running time contains at least a factor $\Omega(n \log n)$ due to the sorting of the edges in $\mathcal{R}$. On top of that, implemented naively, such a procedure would need to be executed k times, one for each set C_i, and so the running time would not respect the $O(nk^2)$ bound.

To circumvent these barriers, we will avoid sorting all the distances, and moreover, we will deal with all the different sets C_i "simultaneously". For that, we process the sets $C_1, \ldots, C_k$ in increasing index order. We start with C_1 and the network $N_{C_1}(0)$, and consider the relevant edges in $\mathcal{R}$, one by one, in increasing order of distance labels. Each time we add an edge, we check whether there is an augmenting path that can lead to an increase of the maximum flow value, and if so, we increase the flow. We stop this process as soon as the maximum flow value equals L, which corresponds to an optimal partial assignment for C_1, as explained above. We then turn to the set C_2. A key observation is that if r_i is the smallest radius for which $N_{C_i}(r_i)$ has a maximum flow of value $i \cdot L$, then $r_{i-1} \leq r_i$ for every $i = 2, \ldots, k$. In particular $r_2 \geq r_1$, which means that all the edges that were added in $N_{C_1}(r_1)$ will be part of the flow network $N_{C_2}(r_2)$. Thus, once we move to C_2, we can continue adding edges and there will never be the need to remove any edges that were added when processing

a set with smaller index. So, we continue in a similar fashion and stop as soon as the maximum flow value equals $2L$, then continue with C_3, and so on. Finally, to obtain a worst-case running time that is linear in n, we observe that we can implement the algorithm in a way so that a total sorting of the edges in $\mathcal{R}$ is not needed. Instead, we only need to partially sort them and place them in buckets. In the following sections, we discuss the implementation details and prove how they lead to the desired $O(nk^2)$ running time.

B.2. Putting edges into buckets

As noted above, given a set C_i, we would ideally like to process the relevant edges of $\mathcal{R}$ in increasing order of distance labels. However, sorting them would violate our desired running time guarantee. For that reason, we instead partition all the edges of $\mathcal{R}$ into buckets $R_1, R_2, \ldots, R_m$, for some $m \leq k^3$ such that the following is satisfied: within each bucket the edges are not sorted but the distance-labels of all edges in bucket R_i are smaller than the distance-labels of all edges in bucket R_j for every $i < j$. This can be done in a worst-case running-time that is linear in n. Note that there might be edges with identical distance label. For those we define a total ordering by using a tie-break according to the center-index and some (arbitrary but fixed) ordering of X.

Lemma B.1. *There is an algorithm that partitions $\mathcal{R}$ into $m \leq k^3$ buckets $R_1, \ldots, R_m$, each containing at most $\lceil 2n/k^2 \rceil$ edges, such that $d(e_1) \leq d(e_2)$ for every $e_1 \in R_i$, $e_2 \in R_j$ with $1 \leq i < j \leq m$. Moreover, its running time is $O(nk \log k)$.*

Proof. We utilize a Selection algorithm with $O(n)$ running time. We use the Selection algorithm to first compute the median edge in $\mathcal{R}$ w.r.t. the distance label (with tie-breaking), and using the median, we obtain a partition of $\mathcal{R}$ into two sets, one containing all the edges smaller than the median and the median itself, and the other containing all the edges that are larger than the median; note that each resulting set has a size of at most $\lceil nk/2 \rceil$.. This can be done in time at most cnk, where $c > 0$ is a universal constant determined by the Selection algorithm. Next, we compute the median edge in each of the two subsets to obtain a more refined partition of $\mathcal{R}$ into four sets; this again takes total time at most cnk. Continuing this for i levels, we end up with 2^i buckets, each containing at most $\lceil nk/2^i \rceil$ edges, and the total running time is at most $i \cdot cnk$. Thus, by setting $i = \lfloor \log k^3 \rfloor$, we end up with $2^i \leq 2^{\log k^3} = k^3$ buckets, each of size at most $\lceil nk/2^i \rceil \leq \lceil nk/2^{\log k^3 - 1} \rceil \leq \lceil 2n/k^2 \rceil$ and a total running time of $O(nk \log k)$. □

Note that this algorithm benefits a lot from parallel computation, which makes it very fast even for very large number of edges. In the actual implementation we use the function `select_nth_unstable` (version 1.49.0) of the standard library of Rust, which is based on the quickselect portion of the pattern-defeating quicksort by Peters (2021).

B.3. Computing maximum flows

In order to maintain a maximum flow in the iteratively growing flow network, we need two subroutines: `addEdge` and `removeEdge`. The first one is responsible for adding one new edge in the flow network and checking for the existence of an augmenting path. The latter is needed when we have identified the bucket that contains the desired edge length; more details will be given in Appendix B.4. As both of these subroutines perform a flow augmentation, we have a third (internal) procedure called `augmentFlow`. Furthermore, to efficiently find an augmenting path within the residual network we need some data structures and additional procedures to maintain them, which we now present.

B.3.1. Data structures

In order to achieve a running time of $O(nk^2)$ we use the following data structures:

- `edge_present`$[\cdot][\cdot]$: a 2-dimensional Boolean array of size $k \times n$. Each entry `edge_present`$[c][x]$, for $c \in C$ and $x \in X$, indicates whether the edge (c, x) is currently present in the flow network or not.
- `max_flow`: an integer denoting the maximum flow value in the current network.
- `center_of`$[\cdot]$: a 1-dimensional array of n points. Each entry `center_of`$[x]$, for $x \in X$ is the center which x is currently assigned to. For example, if `center_of`$[x] = c$, this means that the point x is currently assigned to center c. If a point x is currently unassigned, then we will denote this with `center_of`$[x] =$ `None`.
- `number_of_covered_points`$[\cdot]$: a 1-dimensional array of k integers. Each entry `number_of_covered_points`$[c]$, for $c \in C$, contains the number of points that are currently assigned to center c.

- `unassigned`$[\cdot]$: 1-dimensional array of size k. Each entry `unassigned`$[c]$, for $c \in C$, is a queue of points, which contains all points that are currently unassigned and could be assigned to center c.
- `reassign`$[\cdot][\cdot]$: 2-dimensional array of $k \times k$ queues of points. Each queue `reassign`$[c][c']$, for $c \neq c' \in C$, contains a list of the points that are currently assigned to the center c but could also be assigned to the center c'.
- `next_to_non_private`$[\cdot]$: a 1-dimensional array of k centers. Each entry `next_to_non_private`$[c]$, $c \in C$, indicates whether there is a path in the residual network from c to a center c' that is not private yet, i.e., that has not yet been assigned L points. If this is the case, then `next_to_non_private`$[c]$ is the center that leads closer to such a non-private center, if no such path exists the value is `None`. For the case that c itself is non-private it points to itself.

Clearly, `center_of` and `number_of_covered_points` encode an s-t-flow if they are consistent. More precisely, if `center_of`$[x] = z \in C$, then $f_{(z,x)} = 1$ and $f_{(x,t)} = 1$. If `center_of`$[x] =$ `None`, we have $f_{(c,x)} = 0$ for any $c \in C$ and also $f_{(x,t)} = 0$. The flow value of each edge (s, c), where s is the source and $c \in C$ is given by `number_of_covered_points`$[c]$.

The data structures `unassigned`, `reassign` and `next_to_non_private` are important to perform a fast flow augmentation, which is equivalent to finding an s-t-path in the residual network. Here, we are looking for such a path in reverse order from t to s. Whenever `unassigned`$[c]$ is non-empty there is an unassigned point $x \in X$ that could be assigned to c, i.e., the path (c, x, t) is present in the residual network. In the case that c is not private yet (i.e., `number_of_covered_points`$[c] < L$) we have found an augmenting path, namely (s, c, x, t). But in the case that c is already private, (s, c) is not in the residual network, but there might an augmenting path from some other non-private center c^* to c. For this, first note that there is an augmenting path of length 2 from center c' to center c if and only if `reassign`$[c][c']$ is non-empty. Namely, the path (c', x, c) with $x \in$ `reassign`$[c][c']$. In other words, `reassign` defines a graph H on the set of centers C and in order to find an augmenting path from non-private c^* to c, we must find a path in H from c to c^*. This can be done by performing a breadth first search (BFS) in H, starting at c. This would not only decide whether we can reach a non-private center from c but it also determines a path (if there is any) by pointing to a neighbored center in the right direction. As H often only changes slightly during the execution of our algorithm, we do not recompute a BFS in each step (which would take $O(k^2)$), but instead maintain and adjust these path-pointers, which is exactly the information stored in `next_to_non_private`.

B.3.2. Procedures

The procedure `rebuildReachability` rebuilts `next_to_non_private` by starting with the non-private centers and doing a BFS in H. It is presented in Algorithm 4 and has a worst case running-time of $O(i^2) \subseteq O(k^2)$ as proven in Lemma B.2.

Algorithm 4 `rebuildReachability`

Input: index i of current center prefix C_i

```
next_to_non_private[c] ← None for all c ∈ C_i // reset
// do backwards BFS in H starting at private centers:
Q ← newQueue()
for c ∈ C_i do
    if number_of_covered_points[c] < L then
        Q.push(c)
        next_to_non_private[c] ← c // center c is private
    end
end
while Q ≠ ∅ do
    c ← Q.pop()
    for c' ∈ C_i do
        if reassign[c][c'] ≠ ∅ and next_to_non_private[c'] = None then
            Q.push(c')
            next_to_non_private[c'] = c
        end
    end
end
```

Lemma B.2. *The procedure* `rebuildReachability` *has a worst-case running time of $O(A \cdot i)$, where $A \leq i$ is the number of centers $c \in C_i$ that are pushed to the queue Q at some point in time, and which is equal to the number of centers $c \in C_i$ for which* `next_to_non_private[c]` *changes from* `None` *to some center.*

Proof. Clearly, each center can only be pushed once onto the queue, as this happens only to centers for which `next_to_non_private[c]` $=$ `None` but it is set to a center right after. This means $A \leq i$, and hence, the running time of $O(i)$ for the setup and $O(A \cdot i)$ for the breadth first search (line 10 to 18) implies a total running time of $O(A \cdot i)$. □

The procedure that searches for an augmenting path and increases the maximum flow by 1 whenever possible is described in Algorithm 5. The main idea behind it is to check whether s and t are connected in the residual network by searching for a center c_j which has an unassigned point x that could be assigned to it (hence, path (c_j, x, t) is present in the residual network) and that has a path to a non-private center in H (which is equivalent to the existence of an s-c_j-path in the residual network).

Algorithm 5 `augmentFlow`

Input: index i of current center prefix C_i

```
1  j ← 1
2  while j ≤ i do
3  |  if unassigned[c_j] ≠ ∅ and next_to_non_private[c_j] ≠ None then
4  |  |  break
5  |  else
6  |  |  j ← j + 1
7  |  end
8  end
9  if j = i + 1 then
10 |  return // there is no augmenting path
11 end
12 // augmenting path exists
13 c ← c_j
14 x ← unassigned[c].pop()
15 center_of[x] = c
16 number_of_covered_points[c] = number_of_covered_points[c] + 1
17 max_flow ← max_flow + 1
18 while number_of_covered_points[c] ≥ L do
19 |  c' ← next_to_non_private[c] // c' is closer to a non-private center
20 |  x ← reassign[c][c'].pop()
21 |  center_of[x] = c'
22 |  number_of_covered_points[c] = number_of_covered_points[c] − 1
23 |  number_of_covered_points[c'] = number_of_covered_points[c'] + 1
24 |  reassign[c'][c].push(x)
25 |  c ← c'
26 end
27 rebuildReachability(i)
```

Lemma B.3. *The procedure* `augmentFlow` *runs in time of $O(i)$ if* `max_flow` *is not increased and in time $O(i^2)$ if* `max_flow` *is increased by* 1.

Proof. Clearly, if there is no augmenting path the algorithm stops after iterating once over $[i]$. If there is an augmenting path the invocation of `rebuildReachability` is the bottle-neck with a running-time of $O(i^2)$; see Lemma B.2. □

The pseudo-code in Algorithm 6 describes how an edge $(c, x) \in \mathcal{R}$ is added to the flow network and how the data structures are updated in order to describe a maximum flow in the iteratively increased network. Note that if x was assigned before only a single edges has been added to the graph H, namely (c', c). Hence, if c' could previously not reach a private-center

but c could (and still can), now also c' can reach a private-center. Hence, `next_to_non_private` needs to be updated by setting `next_to_non_private`$[c'] = c$. As other centers might reach c' in H even more centers might reach a non-private center now. For that reason the algorithm performs a BFS (as described in line 10 to 18 of Algorithm 4) to identify those centers and update `next_to_non_private` accordingly. For the running-time analysis it is important to note that only `None` values are set to some centers, but never the other way round.

Algorithm 6 `addEdge`

Input: edge $e = (c, x)$, index i of current center prefix C_i

```
edge_present[c][x] = True
if center_of[x] = None then
    unassigned[c].push(x)
else
    c' ← center_of[x] // point x is currently assigned to center d
    reassign[c'][c].push(x)
    if next_to_non_private[c'] = None and next_to_non_private[c] ≠ None then
        continue BFS of rebuildReachability by setting Q = { c' } and executing line 10 to 18 of Algorithm 4.
        // takes O(k) for each center c that is set from None to something.
    end
end
augmentFlow(i)
```

Lemma B.4. *The procedure* `addEdge` *in Algorithm 6 whose input is $e \in \mathcal{R}$ has a worst-case running time of $O(i^2)$ if the flow value is increased by 1 during the* `augmentFlow` *invocation. Otherwise the running time is given by $O(A_e \cdot i)$, where $A_e \leq i$ is the number of centers, for which* `next_to_non_private` *is set from* `None` *to something during the BFS of* `rebuildReachability`.

Proof. This follows immediately from Lemma B.3 and the argument in the proof of Lemma B.2. □

In Algorithm 7 we present the procedure for removing an edge $(c, x) \in \mathcal{R}$ from the network. If this edge was not flow carrying then the maximum flow does not change. In the case that $f_{(c,x)} = 1$ we first remove the flow along the path (s, c, x, t) but it might be the case that afterwards there is an augmenting path not using this edge (this is determined in `augmentFlow`). Either way, as c might change from private to non-private the data structure `next_to_non_private` must be rebuilt. Note that the removal of x from `unassigned` and `reassign` can be done in constant time, as we only implicitly remove them by setting `edge_present`$[c][x] =$ `False`. (Still we kept the remove-commands in the pseudo-code to emphasize that x should be considered as not present in the queues anymore.)

Algorithm 7 `removeEdge`

Input: edge $e = (c, x)$, index i of current center prefix C_i

```
edge_present[c][x] = False
if center_of[x] = None then
    unassigned[c].remove(x)
else if center_of[x] ≠ c then
    reassign[center_of[x], c].remove(x)
    rebuildReachability(i)
else
    // in this case e was flow carrying
    max_flow ← max_flow − 1
    center_of[x] = None
    rebuildReachability(i)
    augmentFlow(i)
end
```

Lemma B.5. *The procedure* `removeEdge` *in Algorithm 7 has a worst-case running time of $O(i^2)$.*

Proof. This follows immediately from Lemmas B.2 and B.3. □

B.4. Settle a prefix

Consider a fixed center prefix C_i. As the goal is to find the minimum radius of a partial private assignment for C_i, it is important that we identify the correct edge $e \in \mathcal{R}$ for which the maximum flow value is equal to $i \cdot L$ in the network $N_{C_i}(d(e))$ containing $E \coloneqq \{\, e' \in \mathcal{R} \mid d(e') \leq d(e) \,\}$, while it is strictly smaller than $i \cdot L$ in the network $N_{C_i}(d(e)) \setminus \{e\}$.

Since we cannot afford to sort all edges in $\mathcal{R}$ we might have added edges e' with $d(e') > d(e)$ to the network before we add e itself. To solve this problem we only check after adding all edges of a bucket R to the network, whether the maximum flow value has reached $i \cdot L$. If not, we continue with the next bucket, otherwise we know that the correct edge can be found within R. In order to find it, we perform a binary search on the edges of R. We say that prefix C_i gets *settled* in the bucket R. More precisely, we first split the bucket into two sub-buckets R^- and R^+, such that all edges in R^- are smaller than all edges in R^+. This can be achieved in time $O(|R|)$ by using a linear-time Selection algorithm. Next, we remove all edges from R^+ and check again if the maximum flow value is equal to $i \cdot L$. If yes, we do this procedure recursively on R^-, as the correct edge must be contained in there. If not, the edges must be in R^+, hence, we do a recursion on these edges (note that in that recursion, instead of removing the bigger half, we have to add the smaller half). The exact procedure is given in Algorithm 8 and Algorithm 9.

Algorithm 8 `settlePrefix`

Input: index i of prefix C_i to be settled, current bucket R
Output: partial assignment ϕ_i (with centers C_i) and radius r_i

```
e ← searchForEdge(i, R, True)
r_i ← d(e)
φ_i ← center_of.copy()
// assignment and radius have been found; now remove current bucket from network:
for e = (c, x) ∈ R do
    if edge_present[c][x] = True then
        removeEdge(e')
    end
end
```

Lemma B.6. *The worst-case running time of the procedure* `settlePrefix` *is given by* $O(|R| \cdot i^2 + n)$.

Proof. The invocation of `searchForEdge` takes $O(|R| \cdot i^2)$ time as half of the edges are added or removed (each takes $O(i^2)$ time by Lemmas B.4 and B.5) and the bucket size gets halved in each recursion step. This leads to a running time of $O(i^2 \cdot (|R|/2 + |R|/4 + \cdots + 1)) = O(i^2 \cdot |R|)$. The same running time is needed for removing all present edges of the bucket after the radius has been found. Finally the copying of `center_of` takes $O(n)$ time, as this is the size of this array. □

B.5. Making all prefixes private

Now we are ready to present the complete algorithm for computing a private assignment for all prefixes of a given set of centers. For this, one additional data structure is needed, namely an array of queues `pending`$[\cdots]$, where `pending[i]` stores the edges that were already considered for adding them to the network for the prefix C_i, $i \in [k]$, but were withheld as this prefix C_i was not considered yet. The main procedure can be described as follows. After creating all edges and putting them into buckets, the algorithm iterates over all buckets with counter j and over all center prefixes with counter i. This two iterations are done in parallel. It adds the edges bucket-wise (meaning that the flow value is only checked after each edges of the bucket is added) to the network and checks whether the maximum flow value has reached $i \cdot L$. If so, the correct bucket has been found and the current center prefix needs to be settled in there. If not, the next bucket is added to the network. After a prefix has been settled all edges of the current bucket are removed and the next prefix is considered $i \leftarrow i + 1$. First all pending arcs in `pending[i]` are added to the network and then the process is continued by adding the current bucket.

B.6. Running time and correctness

Lemma B.7. *The procedure* `makePrefixesPrivate` *in Algorithm 10 has a worst-case running time of* $O(nk^2)$.

Algorithm 9 `searchForEdge`

Input: index i of current center prefix C_i, current bucket R, boolean `bucket_present` indicating if edges of R are present in the flow network
Output: edge e

```
1  if |R| = 1 then
2      e ← R.pop()
3      if bucket_present then
4          addEdge(e)
5      end
6      return e // this is the edge we are looking for as max_flow = i · L
7  end
8  R⁻, R⁺ ← splitAtMedian(R) // takes O(|R|)-time
9  if bucket_present then
10     for e′ ∈ R⁺ do
11         RemoveEdge(e′)
12     end
13 else
14     for e′ ∈ R⁻ do
15         addEdge(e)
16     end
17 end
18 if max_flow ≥ i · L then
19     e ← searchForEdge(i, R⁻, True)
20 else
21     e ← searchForEdge(i, R⁺, False)
22 end
23 R ← (R⁻, R⁺) // update bucket to be "more sorted" for later use
24 return e
```

Proof. Creating the edges takes $O(nk)$ time, and by B.1, putting them into buckets can be done in $O(nk \log k)$. To analyze the main loop we divide the buckets into two categories, the buckets $\mathcal{B}_S$ in which a prefix is settled in (more precisely, $\mathcal{B}_S$ contains every R for which `settlePrefix`$[i, R]$ is invoked at least once), and the rest of the buckets $\mathcal{B}_N := \mathcal{B} \setminus \mathcal{B}_S$.

Since `settlePrefix` is invoked at most k times, $\mathcal{B}_S$ contains at most k buckets, which in total contain at most $O(n/k)$ edges. Each of these edges might be added to the network in line 11 or line 17 multiple times, as they might be removed during the `settlePrefix` procedure. Nonetheless, for each `settlePrefix` invocation at most $O(n/k^2)$ such edges are removed (since this is the bucket size), which means that the number of `addEdge` invocations by edges in buckets contained in $\mathcal{B}_S$ is still bounded by $O(n/k)$ (note that we consider the binary search within `settlePrefix` separately). By Lemma B.4 `addEdge` has a worst case running time of $O(k^2)$, which gives a combined bound on the running time for adding these edges of $O(n/k \cdot k^2) = O(nk)$. Furthermore, `settlePrefix` is invoked k times, each with a running time $O(n/k^2 \cdot k^2 + n) = O(n)$ (using B.6 with $|R| = O(n/k^2)$ and $i = k$). This gives a combined running time of $O(nk)$.

For edges in buckets contained in $\mathcal{B}_N$, we make a further distinction:

$$\mathcal{R}_+ := \left\{ e \in \bigcup_{R \in \mathcal{B}_N} R \;\middle|\; \texttt{addEdge}[e] \text{ increases } \texttt{max\_flow} \text{ by } 1 \right\},$$

$$\mathcal{R}_= := \left\{ e \in \bigcup_{R \in \mathcal{B}_N} R \;\middle|\; \texttt{addEdge}[e] \text{ does not increase } \texttt{max\_flow} \right\}.$$

Since `max_flow` is bounded by $k \cdot L$ and none of these edges are ever removed from the network, it holds that $|\mathcal{R}_+| \leq k \cdot L \leq n$. Hence with Lemma B.4, the total running time for adding these edges to the network is bounded by $O(nk^2)$.

There could be $O(nk)$ edges in $\mathcal{R}_=$ but none of them increases the maximum flow value. We will show that the amortized

Algorithm 10 `makePrefixesPrivate`

Input: metric space (X, d), privacy bound L, ordered centers $C = (c_1, \dots, c_k)$
Output: for $i \in [k]$: assignments $\phi_i : X \to \{ c_1, \dots, c_i \}$ with radius r_i satisfying the privacy-condition.

$\mathcal{R} \leftarrow \{$ edge $e = (c, x)$ for all $c \in C$ and $x \in X \}$ // an edge in the flow network.
$R_1, \dots, R_m \leftarrow$ `putIntoBuckets`$(\mathcal{R}, d)$ // at most k^3 buckets of size at most $2\lceil n/k^2 \rceil$
$i \leftarrow 1$ // index of the center prefix C_i
$j \leftarrow 1$ // index of the current bucket
`pending`[c] $\leftarrow$ `newQueue`() for all $c \in C, b \in [m]$ // withheld edges
while $i \leq k$ **do**
  // this is the main while-loop that deals with each prefix C_i
  // first, we process edges from previous buckets that were withheld
  while `pending`[i] $\neq \emptyset$ **do**
    $e :=$ `pending`[i].`pop`()
    `addEdge`(e)
  end
  // then we add edges bucket-wise until `max_flow` $= i \cdot L$
  while `max_flow` $< i \cdot L$ **do**
    for $e = (c, x) \in R_j$ **do**
      if $c \in C_i$ **then**
        `addEdge`(e) // e is added to flow network, a new `max_flow` is computed
      else
        `pending`[c].`push`(e) // e is postponed as c is a center not considered yet
      end
    end
    $j \leftarrow j + 1$
  end
  $j \leftarrow j - 1$ // at this point, we have identified that R_j settles prefix C_i
  $\phi_i, r_i \leftarrow$ `settlePrefix`(i, R_j)
  // current bucket is completely removed from network; restart current bucket
  $i \leftarrow i + 1$
end
// assign all unassigned points to their nearest centers:
for $i = 1, \dots, k$ **do**
  for $x \in X$ **do**
    if $\phi_i(x) =$ `None` **then**
      $\phi_i(x) \leftarrow \arg\min_{c \in C_i} d(c, x)$
      $r_i \leftarrow \max \{ r_i, d(\phi_i(x), x) \}$
    end
  end
end
return $(\phi_1, r_1), \dots, (\phi_k, r_k)$

running time for adding these edges is also bounded by $O(nk^2)$. By Lemma B.4 the running time of adding an edge to the network is bounded by $O(A_e \cdot k)$, where $A_e \leq k$ is the number of centers for which `next_to_non_private` is set from `None` to something. But the values of `next_to_non_private` are only ever reset to `None` when `rebuildReachability` is invoked, which only happens when edges are removed in `removeEdge` or when the `max_flow` value is increased by one in `augmentFlow`. When considering the sequence of edges in $\mathcal{R}_=$ this happens at most $O(n + k)$-times, namely when `max_flow` is incremented or when a prefix is settled in a bucket. In other words, $\sum_{e \in \mathcal{R}_=} A_e \leq n \cdot k$, which gives a total running time for adding edges in $\mathcal{R}_=$ of $O(k \cdot \sum_{e \in \mathcal{R}_=} A_e) = O(nk^2)$.

Finally, assigning the remaining points to their nearest center takes $O(nk^2)$ time. In total `makePrefixesPrivate` has a worst-case running time of $O(nk^2)$. □

Lemma B.8. *The procedure* `makePrefixesPrivate` *in Algorithm 10 computes, for a given set of ordered centers C, the optimal private assignments for all prefixes of C.*

Proof. Suppose that is not true. Then there is a prefix C_i for which there is another private assignment ϕ' with smaller radius r'. Let $c \in C_i$ and $x \in X$ be the center-point pair with $\phi_i(x) = c$ and $d(c, x) = r$ (if there are multiple, we choose the one where c has the highest index or in case of a tie, we choose an x that is maximal with respect to the fixed ordering on X). If x was assigned to c during the filling up in line 33 that would mean that c is the closest center to x, which would lead to a contradiction as $d(\phi'(x), x) \geq d(c, x) = r > r'$. Hence, x has to be assigned to c during the maximum flow computation. At the moment the prefix C_i was settled, edge $e = (c, x)$ had to be the last edge that was added to the network. More precisely, all other edges e' from C_i to X with $d(e') \leq r$ were also present in the flow network. In particular, all edges of the form $e' = (\phi'(x), x)$ as $d(e') \leq r' < r$. But modeling ϕ' as a flow (by taking only the L closest points for each center in C_i) this would lead to an s-t-flow of value $i \cdot L$ but without using the edge (c, x). But this is a contradiction, as the edge e was necessary for achieving a maximum flow value of $i \cdot L$ in the `settlePrefix` procedure. □

The proof of Lemma 3.2 now follows immediately:

Proof of Lemma 3.2. The algorithm Algorithm 10 solves MIN-PRIV-RADIUS by Lemma B.8 (just pick any ordering of the centers) and has a running time of $O(nk^2)$ as proven in Lemma B.7. □

B.7. Making a single set of centers private

The final algorithm is going to choose a final set of centers that satisfy the representative condition. In order to provide the best possible private assignment with it, we use the procedure as described above but we only care for the private assignment of the full set. For convenience reasons, we include the `makePrivate` algorithm (see Algorithm 11). It is clear that the running time of `makePrivate` equals the running time of `makePrefixesPrivate`, namely $O(nk^2)$; see Lemma B.7.

Algorithm 11 `makePrivate`

Input: metric space (X, d), privacy bound L, a set of centers C
Output: private assignment $\phi_i : X \to C$ with minimal radius r.

```
i ← |C|
c_1, ..., c_i ← any order of centers in C
(φ_1, r_1), ..., (φ_i, r_i) ← makePrefixesPrivate((X, d), L, (c_1, ..., c_i))
return (φ_i, r_i)
```

C. Full analysis of results of Section 4: Constructing good backbones

Suppose we are given a set of centers Π with $|\Pi| \leq k$ together with a private assignment $\phi\colon X \to \Pi$, such that Π does not satisfy the representative condition; as a reminder, the representative condition asks that $a_i \leq |\Pi \cap X_i| \leq b_i$ for every $i \in [\gamma]$. In order to be able to find a representative set of centers C, which admits a private assignment of small radius, we first need to find backbones with good Δ-realization for the MIN-REP-REALIZATION problem (we call these *representative Δ-realizations*).

Throughout this section we only consider a single set of centers Π and all backbones will have the form (Π, η) for some $\eta \in \mathbb{Z}_{\geq 0}^{\Pi}$. For our final algorithm the procedure presented in the following will be applied to all Gonzalez' prefixes $C_1, \ldots, C_k$ providing a total of at most $k(k+1)/2$-many backbones.

We consider special backbones that are created from the private assignment ϕ (or later of assignments that are close to ϕ), which we call fixed-aggregations (or τ-aggregations, respectively). They are defined as follows.

Definition C.1 (fixed-aggregation)**.** For a given set of centers Π with assignment $\phi\colon X \to \Pi$ we call the vector $\eta \in \mathbb{Z}_{\geq 0}^{\Pi}$ with

$$\eta_\pi = \left\lfloor \frac{|\phi^{-1}(\pi)|}{L} \right\rfloor \text{ for all } \pi \in \Pi$$

the *fixed-aggregation* of ϕ of total value $\sum_{\pi \in \Pi} \eta_\pi$.

Note that such an aggregation might not be sufficient for a representative Δ-realization. To see this, suppose that the value of the fixed-aggregation is smaller than $\sum_{\ell\in[\gamma]} a_\ell$. In such a case we cannot open enough centers to satisfy the representative condition. But even if there exists a representative Δ-realization for the backbone, the Δ might be unnecessarily large.

To obtain better backbones we allow points of some cluster $\phi^{-1}(\pi)$ to be transferred to a nearby cluster $\phi^{-1}(\pi')$. To specify what *nearby* mean, we define a neighborhood relation between clusters.

Definition C.2 (transfer threshold and neighborhood-graph)**.** For a given *transfer threshold* (or just *threshold*) $\tau \in \mathbb{R}_{\geq 0}$, we call two centers π and π' *τ-neighbored* if $d(\pi,\pi') \leq \tau$. This defines an undirected *τ-neighborhood-graph* $G_\tau \coloneqq (\Pi, \{\,\{\,\pi,\pi'\,\} \mid d(\pi,\pi') \leq \tau\,\})$.

For a given threshold τ the algorithm allows points of some cluster, to be transferred to centers of some of the neighbored clusters. This gives us additional flexibility for the backbones, as for large τ all clusters would be neighbored to each other. Still, the smaller this threshold is, the better the final bound on the radius will be. We extend the definition of a fixed-aggregation to a τ-aggregation by allowing points to be transferred to some neighbored cluster.

Definition C.3 (τ-aggregation)**.** For a given set of centers Π with assignment $\phi\colon X \to \Pi$ we call the vector $\eta \in \mathbb{Z}^{\Pi}_{\geq 0}$ a *τ-aggregation* if there exists an assignment $\phi'\colon X \to \Pi$ such that $\phi(x)$ and $\phi'(x)$ are τ-neighbored for all $x \in X$ and η is a fixed-aggregation of Π with ϕ'

We also call a backbone (Π,η) a τ-aggregation if there exists a assignment ϕ such that η is a τ-aggregation for Π and ϕ.

We are interested in τ-aggregations with large entries, as this leads to more flexibility for the representative Δ-realization.

For a given threshold τ, let $\Pi_1, \Pi_2, \ldots, \Pi_j$ be partition of Π given by the connected components of the neighborhood graph G_τ. Since no point can be transferred to a center of a different connected component, the value any τ-aggregation is upper bounded by

$$U_\tau \coloneqq \sum_{\ell=1}^{j} \left\lfloor \frac{\left|\phi^{-1}(\Pi_\ell)\right|}{L} \right\rfloor, \qquad \text{where} \quad \phi^{-1}(\Pi_\ell) \coloneqq \bigcup_{\pi\in\Pi_\ell} \phi^{-1}(\pi).$$

Fortunately, we can always achieve this maximal value.

Lemma C.4. *For every transfer threshold τ, there exists a τ-aggregation η^τ of value U_τ.*

Proof. We consider each connected component Π_ℓ for $\ell \in [j]$ individually. For each $\ell \in [j]$, let $X_\ell \coloneqq \{\, x \in X \mid \phi(x) \in \Pi_\ell \,\}$ be the set of all points with centers in this connected component. We show that there is an assignment $\phi_\ell\colon X_\ell \to \Pi_\ell$ such that $\phi(x)$ and $\phi_\ell(x)$ are τ-neighbored for all $x \in X_\ell$ and the fixed-aggregation of ϕ_ℓ has a total value of $\left\lfloor \left|\phi^{-1}(\Pi_\ell)\right| / L \right\rfloor$.

To see this, we pick any spanning tree T of the connected component of C_ℓ and declare any center $\pi^* \in \Pi_\ell$ as the *root* (which we now picture as being on top). Starting at the leaves (at the bottom) of T, we transfer points up, always closer to the root. More precisely, at a center π we consider the number of points n_π that are currently assigned to π (note that this might be larger than the original size of $\left|\phi^{-1}(\pi)\right|$ as some points might have already been transferred into this cluster). We keep $L \cdot \lfloor n_\pi/L \rfloor$ points in the cluster and transfer the remaining $n_\pi - L \cdot \lfloor n_\pi/L \rfloor$ points to the cluster corresponding to the parent node in T. Note that we never move more than $L-1$ points away from each cluster. Since each original cluster had at least L points (ϕ is a private assignment) we never transfer a point more than once. After this procedure each center except for π^* has exactly a multiple of L many points assigned to it. Since no point was "lost on the way", the fixed-aggregation η of this new assignment ϕ_ℓ has a total value of $\left\lfloor \left|\phi^{-1}(\Pi_\ell)\right| / L \right\rfloor$.

Doing this individually for each connected component of G_τ leads to a τ-aggregation of total value U_τ. □

Overall, we would like to keep the threshold τ as small as possible, such that points are not farther than needed. If we consider all pair-wise distances between centers in Π, i.e., $D \coloneqq \{\, d(\pi,\pi') \mid \pi,\pi' \in \Pi \,\}$, the neighborhood graph G_τ would be the same for each $\tau \in [\delta_1,\delta_2]$, for each pair of consecutive distances $\delta_1,\delta_2 \in D$. Naively, we could try all values in D as threshold, which would lead to $O(|\Pi|^2)$ many values. But by the definition of U_τ it becomes clear that for two thresholds τ_1 and τ_2 the values U_{τ_1} and U_{τ_2} only differ if the connected component of G_{τ_1} and G_{τ_2} differ. As these connected components merge monotonically for increasing thresholds (for $\tau = 0$ all clusters are isolated, for big τ everything is connected) there are only $|\Pi|$-many thresholds for different connected components. When considering Kruskal's algorithm for computing a minimum spanning tree, it becomes apparent that these relevant threshold values are precisely the distances that appear in a

minimum spanning tree (MST) of the complete graph $G = (\Pi, \binom{\Pi}{2})$ with edges weights given by $d(\pi, \pi')$. (In addition we consider also $\tau = 0$ as a threshold.)

Computational-wise, such MST provides an additional benefit. Instead of recomputing for every τ some spanning tree in each connected component of G_τ for determining the τ-aggregation, it is also possible to compute the MST T once at the beginning and add the edges of the MST one by one (in increasing order according to d) to the initially empty graph $G = (\Pi, \emptyset)$.

This leads to the following procedure `createBackbones` given in Algorithm 12, which computes for a given set of center Π, $|\Pi|$-many backbones. It iterates through all thresholds τ given by a MST T in the complete centers graph $(C, \binom{C}{2})$ and computes in the procedure `computeTauAggregation` (see Algorithm 13) a τ-aggregation η^τ of value U_τ.

Algorithm 12 `createBackbones`

Input: metric space (X, d), privacy-bound L, set of centers Π, private assignment $\phi\colon X \to \Pi$
Output: $|\Pi|$-many backbones (Π, η^j) together with transfer thresholds τ^j for $j = 0, 1, \ldots, |\Pi| - 1$

$i \leftarrow |\Pi|$
$n_\pi \leftarrow \left|\phi^{-1}(\pi)\right|$ for all $\pi \in \Pi$ `// cluster sizes`
$T \leftarrow$ `computeMinimumSpanningTree`$(G = (\Pi, \binom{\Pi}{2}), d)$ `// e.g. Prim's algorithm.`
$e_1, e_2, \ldots, e_{i-1} \leftarrow$ sort edges in T according to increasing d-value
$\tau_0 \leftarrow 0$
$F_0 \leftarrow \emptyset$
for $j = 1, 2, \ldots, i-1$ **do**
  $\tau_j \leftarrow d(e_j)$
  $F_j \leftarrow \{ e_1, \ldots, e_j \}$
end

for $j = 0, 1, \ldots, i-1$ **do**
  $\eta^j \leftarrow$ `computeTauAggregation`$((X, d), L, \Pi, (n_\pi)_{\pi\in\Pi}, \tau^j, F_j)$
end
return $((\Pi, \eta^j), \tau^j, F^j)_{j=0}^{i-1}$

The subroutine `computeTauAggregation` described in Algorithm 13 simply executes the algorithm that is described in the proof of Lemma C.4 to compute a τ-aggregation of the best possible value U_τ. As the transfer forest (i.e., the spanning tree for each connected component) and the cluster sizes are provided as input, this can be done in $O(|C|)$ time as proven in Lemma C.5.

Algorithm 13 `computeTauAggregation`

Input: metric space (X, d), privacy-bound L, set of centers Π, cluster sizes $(n_\pi)_{\pi\in\Pi}$, transfer threshold τ, transfer forest F
Output: τ-aggregation $\eta \in \mathbb{Z}_{\geq 0}^{\Pi}$

$i \leftarrow |\Pi|$
$\pi_1, \ldots, \pi_i \leftarrow$ topological sort Π w.r.t. forest F `// can be computed by a BFS in` $O(|\Pi|)$ `time`
for $j = 1, \ldots, i$ **do**
  $\eta_{\pi_j} \leftarrow \lfloor n_{\pi_j} / L \rfloor$
  if π_j *has parent in* F **then**
    $\pi' \leftarrow$ parent of π_j in F
    $n_{\pi'} \leftarrow n_{\pi'} + (n_{\pi_j} - L \cdot \lfloor n_{\pi_j} / L \rfloor)$ `// remaining points could be transferred to parent`
  end
end
return $(\eta_\pi)_{\pi\in\Pi}$

Lemma C.5. `computeTauAggregation` *has a running time of* $O(|\Pi|)$.

Proof. The algorithm first determines a topological ordering of the centers according to the transfer forest F, which can be done by a breadth-first-search (BFS) in linear time. Then we compute the η_π values in this order and always transfer the

remainder of n_π/L to the parent node. As each of these actions can be done in constant time, this leads to an overall linear running time. □

With this we can prove the total running time for creating the backbones.

Lemma C.6. `createBackbones` *has a running time of* $O(|\Pi|^2)$.

Proof. Prim's algorithm runs in $O(|\Pi|^2)$ and sorting takes $O(|\Pi| \log |\Pi|)$ time. By Lemma C.5 `computeTauAggregation` has a running time of $O(|C|)$ and is invoked $|\Pi|$-times, leading to a running time of $O(|\Pi|^2)$. □

D. Solving MIN-REP-REALIZATION optimally

In this section we will show how to obtain for a given backbone (Π, η) the optimal Δ-realization C that satisfies the representative condition

$$a_j \leq |C \cap X_j| \leq b_j \quad \text{for all } j \in [\gamma].$$

Such a realization is called *representative-Δ-realization* and the mapping $\psi\colon C \to \Pi$ is called *origin mapping*.

In order to achieve the claimed running time, we do not consider every of the $k(k+1)/2$-many backbones individually, but use some synergy between backbones that share the same set of centers. Therefore, the input for our procedure in the final algorithm will be a set of i backbones $(\Pi, \eta^j)_{j=0}^{i-1}$ (with $i = |\Pi|$) as they were created by `createBackbones` (Algorithm 12). But in general the algorithm works for any set of backbones as long as they all share the same set of centers. We call such set a *batch* of backbones.

The overall idea is to use a flow computation in a network with demands and capacities on the edges (lower and upper bounds), to determine a representative-Δ-realization with minimal Δ. The flow problem itself is solved individually for each of the backbones, but the main part of the underlying network only needs to be created once (see Algorithm 14), which saves a lot of time.

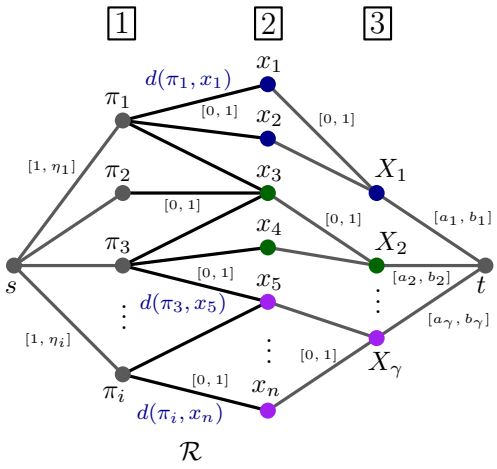
For a fixed backbone (Π, η) the acyclic flow network G_η with demands and capacities is structured in three node-layers (in addition to a source and sink); see Figure 6a. The first layer consists of one node for each center in Π. There is an edge from the source to each of these nodes with bound interval $[1, \eta_j]$ (i.e., these edges have a demand of 1 and a capacity of η_j). The capacities of these edges are the only thing that differs for the individual backbones.

The second layer consists of all points X, and similar to the `makePrefixesPrivate` procedure there is an edge (π_ℓ, x) from each $\pi_\ell \in \Pi$ to each point $x \in X$ with a bound interval $[0, 1]$ and a distance label $d(e) \coloneqq d(\pi_\ell, x)$. We denote the set of all these edges $\mathcal{R}$.

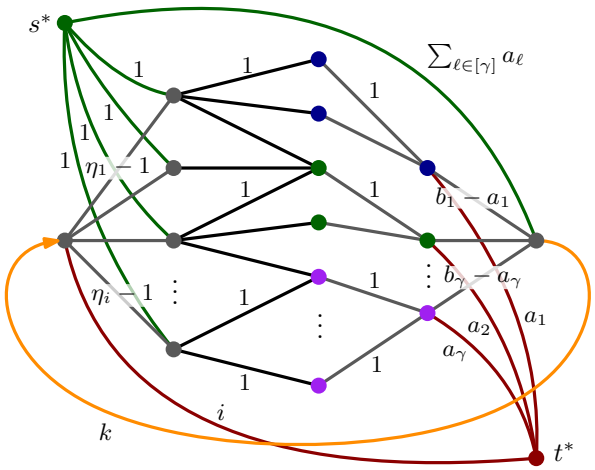
The third layer consists of a node for each color class X_j, $j \in [\gamma]$. Each point x is only connected to the color class it is contained in and the bound interval is again $[0, 1]$. Finally, each color class X_j is connected to the sink by an edge with bound interval $[a_j, b_j]$.

For a fixed backbone (Π, η) the algorithm itself now works as follows. It starts with the network, where none of the labeled edges $\mathcal{R}$ between the first and the second layer are present. Clearly, there is no feasible s-t-flow that satisfies the lower capacities. Now, the algorithm adds the edges one by one in increasing order according to the distance-labels. In each step it checks whether there is a feasible s-t-flow of value at most k or not. The task of finding a feasible s-t-flow can be transformed into a maximum flow computation in an auxiliary network H_η via a super-source-super-sink-construction as depicted in Figure 6. Hence a feasible s-t-flow of value at most k in the original network G_η exists if and only if the maximum flow in the auxiliary network H_η has value $\sum_{\ell \in [\gamma]} a_\ell + i$. More details on this equivalence can be found in Section 7.7 in the textbook of Kleinberg and Tardos (2006). The algorithm stops as soon as a feasible flow in G_η is found (i.e., as soon as the maximum flow in H_η reaches value $\sum_{\ell \in [\gamma]} a_\ell + i$).

The new centers C of the representative-Δ-realization are exactly the points $x \in X$, which have a flow-throughput in the second layer. The origin mapping $\psi\colon C \to \Pi$, which stores the old cluster center of the new centers, are given by $\psi(c) \coloneqq \pi$, where π is the uniquely determined center with $f_{(\pi,c)} = 1$. These new centers C are as close as possible to the original centers, as we added the edges in $\mathcal{R}$ in increasing order. Furthermore, by the demands and capacities of edges between the



(a) The flow network G_η with demand and capacities. All edges are orientated from left to right. The edges between layer 1 and 2 are added one by once in increasing d-label ordering. The goal is to find a feasible s-t-flow of value at most k.



(b) The auxiliary network H_η that turns the task of finding a feasible flow into a maximum flow problem. The labels indicate the capacity and all edges (except for the edge (t, s)) are orientated from left to right. There is a maximum flow of value $i + \sum_{\ell\in[\gamma]} a_\ell$ if and only if there is a feasible s-t-flow of value at most k in the original network G_η.

Figure 6. The flow networks to find a representative-Δ-realization for a backbone (Π, η) with minimal Δ.

third layer and the sink, C satisfy the representative constraints. The distance of the edge added last to the network denotes the value of Δ. Formally, this is shown in the proof of Lemma D.3.

Finally, in order to achieve a worst case running time of $O(nk + k^4)$, we apply additional tweaks to the algorithm.

First of all, the second layer does not need to contain all possible points X, as we can open at most k centers. More precisely, for each original cluster center π we only need to consider the k closest points $Y_c \subseteq X$ that would satisfy the representative condition, namely, $Y_\pi \cap X_j \in [a_j, b_j]$ for all $j \in [\gamma]$ and $|Y_\pi| \leq k$. We call Y_π the candidates of π and only consider the edges (π, y) for $y \in Y_\pi$. As these are at most k, the full network has at most $O(k \cdot |\Pi|)$-many nodes and also that many edges. Note that, even though some points of Y_π might be later assigned to another $\pi' \in \Pi$ as there might be overlap between the candidate sets Y_π and $Y_{\pi'}$, still no point in $X \setminus Y_\pi$ would even be used as new center of π, as the representativeness must be satisfied by all new centers combined (independently of their origin). The is shown in detail in the proof of Lemma D.3.

Secondly, as mentioned above, these candidate sets (and therefore the edges set $\mathcal{R}$) are always the same for all backbones with the same set of centers Π. Hence, they need to be computed only once, which will be happen in the procedure `prepareEdges`; see Algorithm 14.

Lemma D.1. *The running time of* `prepareEdges` *is* $O(|X| \cdot |\Pi|)$.

Proof. For each center in Π we go through all color-classes (which form a partition of X) and execute the select algorithm twice to get the a_j (b_j, respectively) closest points to the center. Using a partition-based selection-algorithm, for example the quick-select algorithm in combination with the median-of-medians-algorithm for pivoting, this can be done in linear time, i.e., in $O(|X_j|)$. Hence, all color-classes combined give a running time of $O(|X|)$, which results in the total running time of $O(|X| \cdot |\Pi|)$. □

The flow computation for a fixed backbone takes place in `computeRepresentativeDeltaRealization`, which is described in Algorithm 15.

Lemma D.2. *If* $|\Pi| \leq k$ *the running time of* `computeRepresentativeDeltaRealization` *is* $O(k^3)$.

Proof. The network and flow initialization takes $O(k^2)$ time as this is an upper bound on the number of nodes and edges. As $|\mathcal{R}| \in O(k^2)$ the first for-loop iterates at most that many times. Finding an augmenting path can take up to $O(k^2)$ time

Algorithm 14 `prepareEdges`

Input: metric space (X, d), color classes $X_1 \dot\cup X_2 \dot\cup \ldots \dot\cup X_\gamma = X$, lower and upper bounds (a_j, b_j) for $j \in [\gamma]$, total upper bound k, set of centers Π
Output: edges $\mathcal{R}$

for $\pi \in \Pi$ **do**
  $Y_\pi \leftarrow \emptyset$ `// candidate set`
  for $j = 1, \ldots, \gamma$ **do**
    $A_j \leftarrow a_j$ points from X_j that are closest to π `// use selection-algorithm to do this in` $O(|X_j|)$
    $B_j \leftarrow b_j$ points from X_j that are closest to π
    $Y_\pi \leftarrow Y_\pi \cup A_j$
  end
  `// fill up candidates with closest remaining points (never more than` b_j `of` X_j`)`
  $Z \leftarrow k - \sum_{j\in[\gamma]} a_j$ points from $\bigcup_{j\in[\gamma]}(B_j \setminus A_j)$ that are closest to π
  $Y_\pi \leftarrow Y_\pi \cup Z$
  $\mathcal{R}_\pi \leftarrow \{\, (\pi, y) \mid y \in Y_\pi \,\}$
end
return $\bigcup_{\pi\in\Pi} \mathcal{R}_\pi$

Algorithm 15 `computeRepresentativeDeltaRealization`

Input: color classes $X_1 \dot\cup X_2 \dot\cup \ldots \dot\cup X_\gamma = X$, lower and upper bounds (a_j, b_j) for $j \in [\gamma]$, total upper bound k, sorted edges $\mathcal{R}$ with d-labels, backbone (Π, η)
Output: distance Δ, representative-Δ-realization C with origin mapping $\psi\colon C \to \Pi$

$i \leftarrow |\Pi|$
prepare auxiliary network H_η without any edges between the first and second layer; see Figure 6b.
initialize maximum flow f in H_η: it has value $|f| = \min\{\, i, \sum_{\ell\in[\gamma]} a_\ell \,\}$; all flow goes along (s^*, t, s, t^*)
for $e \in \mathcal{R}$ *in increasing* d*-value order* **do**
  add e to H_η
  $\Delta \leftarrow d(e)$
  if *there exists an augmenting* s^**-*t^**-path* P *in the residual network* **then**
    augment P in f (flow value $|f|$ is incremented by 1)
  end
  if $|f| = i + \sum_{\ell\in[\gamma]} a_\ell$ **then**
    break
  end
end
$C \leftarrow \emptyset$
for $e = (\pi, y) \in \mathcal{R}$ **do**
  if $f_e = 1$ **then**
    $C \leftarrow C \cup \{\, y \,\}$
    $\psi(y) \coloneqq \pi$
  end
end
return Δ, C, ψ

as this can be done by executing a BFS, which has a running time of $O(|V| + |E|) = O(k^2)$. Similar to the argument in the proof of Lemma B.7 it is possible to continue the BFS of the previous iteration as long as no augmenting path has been found. To do this we store which nodes are reachable from s^* in the residual network. Whenever an additional edge (π, x) is added to the network, we can continue the BFS from x if π was reachable from s^* but x was not. Overall the number of BFS executions is upper bounded by the final maximum flow value $|f|$ which is equal to $i + \sum_{\ell \in [\gamma]} a_\ell \in O(k)$. Hence the total running time is bounded by $O(k^3)$. □

Lemma D.3. *The procedure* `computeRepresentativeDeltaRealization` *computes a representative-Δ-realization for the provided backbone (Π, η) that has the smallest Δ possible. The produced origin mapping $\psi \colon C \to \Pi$ satisfy the conditions of Definition 2.2.*

Proof. Since f is a maximum flow in H_η of value $i + \sum_{\ell \in [\gamma]} a_\ell$, it corresponds to a feasible flow g in G_η of value at most k; see (Kleinberg & Tardos, 2006) for details. Note that g equals f on the edges within $\mathcal{R}$ and these value determine which candidates are chosen to be in C. As the throughput of g at each node π of the first layer is in $\{1, \dots, \eta_\pi\}$, each set $\{ e \in \mathcal{R} \mid f_e = 1 \text{ and } \texttt{tail}(e) = \pi \}$ contains that many elements yielding that $\left|\psi^{-1}(\pi)\right| \in \{1, \dots, \eta_\pi\}$.

All edges $e \in \mathcal{R}$ between the first and second layer of H_η (and therefore also of G_η) satisfy $d(e) \leq \Delta$. Hence, $d(c, \psi(c)) \leq \Delta$.

Furthermore, the throughput of g at each node X_ℓ in the third layer are in $\{a_\ell, \dots, b_\ell\}$ for all $\ell \in [\gamma]$, which proves that C satisfy the representative requirement. As g has a flow value of at most k, it holds that $|C| \leq k$.

Finally, to see that the resulting representative-Δ-realization minimizes Δ, suppose there would be another representative-Δ'-realization C' with origin mapping $\psi' \colon C' \to \Pi$ but with $\Delta' < \Delta$. We choose C', ψ' to be point-wise as close as possible to the original centers Π. More formally, we assume that there is no $c \in C'$ and $x \in X$ such that $d(x, \psi'(c)) < d(c, \psi'(c))$ and $\tilde{C} \coloneqq (C' \setminus \{c'\}) \cup \{x\}$ is a representative-Δ'-realization. (If that would be the case consider $\tilde{C}$ instead of C'.)

If all edges $(\psi'(c'), c')$ for $c' \in C'$ would be present in $\mathcal{R}$, then this leads immediately to a contradiction, as this solution would correspond to a feasible flow in G_η, and hence, also a flow f' in H_η with flow value $i + \sum_{\ell \in [\gamma]} a_\ell$. As f' would not use any edges $e \in \mathcal{R}$ with $d(e) = \Delta$, this yields a contradiction to the fact that f is at all times a maximum flow in H_η.

Therefore, the only possibility is that there is a $c' \in C'$ and some $\pi \in \Pi$, such that $(\pi, c') \notin \mathcal{R}$, or in other words, c' was not chosen to be a candidate for π (i.e., $c' \notin Y_\pi$) during the `prepareEdges` procedure. Let ℓ be the color-class of c' and let $m = |X_\ell \cap C|$ the number of new centers of that color.

If $m = a_\ell$ this leads to a contradiction as Y_π contains the a_ℓ points within X_ℓ that are closest to π. If c' is not one of them, there is a closer $y \in Y_\pi$ that was not chosen by other centers and $d(\pi, y) < d(\pi, c')$ contradicting the way we choose C'.

If $m > a_\ell$, the new center c' could be replaced by any other candidate. As Y_π contains the k points that are closest to π (never more than b_q for any color class $q \in [\gamma]$), we again get a contradiction if c' is not one of them, as we can find a closer candidate $y \in Y_\pi$. □

Finally, we can combine `prepareEdges` and `computeRepresentativeDeltaRealization` to obtain a procedure that realizes a batch of backbones (sharing the same set of center Π) in $O(nk + k^4)$ time.

Algorithm 16 `realizeBatchOfBackbones`

Input: metric space (X, d), color classes $X_1 \dot\cup X_2 \dot\cup \dots \dot\cup X_\gamma = X$, lower and upper bounds (a_ℓ, b_ℓ) for $\ell \in [\gamma]$, total upper bound k, batch of backbones $(\Pi, \eta^1), (\Pi, \eta^2), \dots, (\Pi, \eta^i)$

Output: distances Δ^j and sequence of representative-Δ^j-realizations C^j with origin mapping ψ^j for $j = 1, \dots, i$

```
R ← prepareEdges((X, d), (X_ℓ)_{ℓ∈[γ]}, (a_ℓ, b_ℓ)_{ℓ∈[γ]}, k, Π)
sort R in increasing order according to d(e)    // takes O(k^2 log k) time
for j = 1, ..., i do
    Δ^j, C^j, ψ^j ← computeRepresentativeDeltaRealization((X_ℓ)_{ℓ∈[γ]}, (a_ℓ, b_ℓ)_{ℓ∈[γ]}, k, R, (Π, η^j))
end
return (Δ^j, C^j, ψ^j)_{j∈[i]}
```

Lemma D.4. *If the number j of provides backbones satisfies $j \leq k$ and it holds that $|\Pi| \leq k$, then the running time of* `realizeBatchOfBackbones` *is given by $O(nk + k^4)$, where $n = |X|$ is the total number of points.*

Proof. By Lemma D.1 `prepareEdges` takes $O(n \cdot k)$ time and sorting all edges (their number is bounded by k^2) can be done in a running time of $O(k^2 \log k)$. By Lemma D.2 invoking `computeRepresentativeDeltaRealization` at most k times take $O(k^4)$ time. Together this results in a running time of $O(nk + k^4)$. □

E. Selecting the final set of centers.

At this point we described how to create $q := k(k+1)/2$ backbones and computing a representative-Δ-realization for all of them providing q candidates of representative centers. It remains to select the best of them. Ideally, we would like to compute for all these candidates the best private assignment and then simply choose the set of centers with the minimal radius. Unfortunately, these candidates do not form a chain in general, which means that we would need to compute a private assignment for each of them individually, which would lead to a total running time of $O(q \cdot nk^2) = O(nk^4)$. To achieve a better running time we use the procedure `selectFinalCenters` in Algorithm 17 instead.

Algorithm 17 `selectFinalCenters`

Input: metric space (X, d), privacy-bound L,
for each batch $i \in [k]$: set of centers Π_i, private assignments $\phi_i \colon X \to \Pi$, transfer thresholds $(\tau_i^j)_{j\in[i]}$, transfer forests $(F_i^j)_{j\in[i]}$, realization distances $(\Delta_i^j)_{j\in[i]}$, representative-Δ_i^j-realizations $(C_i^j)_{j\in[i]}$, origin mapping $(\psi_i^j)_{j\in[i]}$
Output: final set of centers C^*,

for $i = 1, \dots, k$ **do**
  $j_i^* \leftarrow \arg\min_{j\in[i]} \left(\tau_i^j + \Delta_i^j\right)$
  $\bar{\phi}_i, \bar{r}_i \leftarrow$ `privateAssignmentHeuristic`$((X,d), L, \Pi_i, \phi_i, \tau_i^{j_i^*}, F_i^{j_i^*}, C_i^{j_i^*}, \psi_i^{j_i^*})$
end
$i^* \leftarrow \arg\min_{i\in[k]} \bar{r}_i$
return $C_{i^*}^{j_{i^*}^*}$

As a first step, we select only one backbone for each of the backbone-batches (which share the same Gonzalez prefix $\Pi = C_i$), namely the one that minimizes the transfer threshold τ plus the realization distance Δ. Even though it might be the case that the selected backbone will not have the best private-radius in the end (among all backbones of this batch), it is the one that gives the best performance guarantee; see Appendix G.1. We end up with k candidates (given by $C_i^{j_i^*}$), one for each batch i. Still we cannot afford to compute the best private assignment for each of them. That is why we run a heuristic for determining a private assignment, instead. `privateAssignmentHeuristic` as it is described below runs in $O(nk)$ time, and most importantly, will be good enough to satisfy the performance guarantee. After computing a private assignment using `privateAssignmentHeuristic` for of the remaining k candidates, we choose the candidate that has the assignment with the best radius.

The `privateAssignmentHeuristic` is given in Algorithm 18.

Algorithm 18 `privateAssignmentHeuristic`

Input: metric space (X, d), privacy-bound L, set of centers Π, private assignment $\phi \colon X \to \Pi$, transfer threshold $\tau \geq 0$, transfer forest F, representative-realization (new centers) C, origin mapping $\psi \colon C \to \Pi$
Output: new assignment $\bar{\phi} \colon X \to C$ with radius $\bar{r}$

$i \leftarrow |\Pi|$
$\phi' \leftarrow$ `transferPoints`$((X,d), L, \Pi, \phi, \tau, F)$
for $\pi \in \Pi$ **do**
  $X_\pi \leftarrow \phi'^{-1}(\pi)$
  $C_\pi \leftarrow \psi^{-1}(\pi)$
  $\bar{\phi}_\pi \leftarrow$ `reassignToNewCenters`$((X_\pi, d), C_\pi, L)$
end
$\bar{\phi} \leftarrow \bigcup_{\pi\in\Pi} \bar{\phi}_\pi$
$\bar{r} \leftarrow \max_{x\in X} d(x, \bar{\phi}(x))$
return $\bar{\phi}, \bar{r}$

This procedure takes the original private assignment $\phi_i \colon X \to \Pi$ and first executes the transferring of points in the transfer forest $F^i_{j^*_i}$. More precisely, the procedure `transferPoints` in Algorithm 19 does the same as `computeTauAggregation` in Algorithm 13 except it really transfers points to the new centers, instead of only transferring the numerical remainder. Since each point will be moved at most once this can be done in $O(|X|)$ time (see Lemma E.1).

Afterwards, the points within each cluster X_π should be reassigned to the new centers of the cluster C_π (according to the representative-Δ-realization C), such that each new center $c \in C_\pi$ has at least L points assigned to it. Surely, this is possible as C is a realization of the backbone (more precisely the τ-aggregation) (Π, η), where $\eta_\pi = \lfloor |X_\pi| / L \rfloor$.

Algorithm 19 `transferPoints`

Input: metric space (X, d), privacy-bound L, set of centers Π, private assignment $\phi \colon X \to \Pi$, transfer threshold $\tau \geq 0$, transfer forest F
Output: new assignment $\phi' \colon X \to \Pi$

$i \leftarrow |\Pi|$
$n_\pi \leftarrow \left|\phi^{-1}(\pi)\right|$ for all $\pi \in \Pi$
$\phi' \leftarrow$ empty partial assignment $(X \to \Pi)$
$\pi_1, \dots, \pi_i \leftarrow$ topological sort Π w.r.t. the forest F. `// can be computed by a BFS in` $O(|\Pi|)$ `time`
for $j = 1, \dots, i$ **do**
  $\eta_{\pi_j} \leftarrow \lfloor n_c / L \rfloor$
  if π_j *has parent in* F **then**
    $\pi' \leftarrow$ parent of π_j in F
    $r \leftarrow (n_{\pi_j} - L \cdot \lfloor n_{\pi_j} / L \rfloor)$ `// remainder`
    $n_{c'} \leftarrow n_{c'} + r$ `// remaining points could be transferred to parent`
    choose $X' \subseteq \phi^{-1}(\pi_j)$ with $|X'| = r$ that are closest to π'
    $\phi'(x) \coloneqq \pi'$ for all $x \in X'$ `// transfer` r `many points to the parent cluster`
  end
  $\phi'(x) = \pi_j$ for all $x \in \phi^{-1}(\pi_j) \setminus X'$ `// all non-transferred points stay in the cluster`
end
return ϕ'

There are many options to implement the heuristic `reassignToNewCenters`. In our implementation we do the following: Consider a cluster (after transferring) $X_\pi \coloneqq \phi'^{-1}(\pi)$ with new centers $C_\pi \coloneqq \psi^{-1}(\pi)$. Ideally, we would start with the point $x \in X_\pi$ that maximizes $d(x, C_\pi)$ and assign them first to its nearest center $\arg\min_{c \in C_\pi} d(x, c)$. As we cannot afford the time to sort all distances between X_π and C_π, we actually only partially sort them into buckets of size L (again use the linear-time selection algorithm; cf. Appendix B.2). As soon as a center $c \in C_\pi$ covers L points, we continue the process but ignoring c (since assigning more points to them might lead to not enough points for the other centers). If all centers are satisfied (they cover L points) we assign the remaining point greedily to their nearest center in C_π.

Such a heuristic would lead to a running time of $O(|X_\pi| \cdot |C_\pi|)$ for each cluster $\pi \in \Pi$, which sums up to a worst-case running time of $O(|X| \cdot |C|)$ in total. But also easier heuristic, e.g., iterating over all centers $c \in C_\pi$ and assign any L unassigned points of X_π to them, afterwards assign the remaining points to any center in C_π, would suffice for the performance guarantee.

Lemma E.1. `transferPoints` *has a running time of* $O(|X|)$.

Proof. Determining the cluster sizes takes $O(|X|)$ in total. The topological sorting via BFS can be done in $O(|\Pi|)$ time.

Choosing X' out of $\phi^{-1}(\phi_j)$ (i.e., the r closest points to π') can be done in $O(\phi^{-1}(\phi_j))$ by using a partition-based selection-algorithm with linear running time (e.g., quick-select with median-of-medians for pivoting). And reassigning the points (defining ϕ') also takes $O(\phi^{-1}(\phi_j))$ time. Hence, summing up over all cluster gives us a running time of $O(|X|)$, which is also the running time of the whole procedure. □

Lemma E.2. *For* $|\Pi| \leq |C|$ `privateAssignmentHeuristic` *has a running time of* $O(|X| \cdot |C|)$.

Proof. By Lemma E.1 `transferPoints` needs $O(|X|)$ time. Our implementation of `reassignToNewCenters` takes $O(|X_\pi| \cdot |C_\pi|)$ times. Since $(X_\pi)_{\pi \in \Pi}$ form a partition of $\mathcal{X}$ and similarly $(C_\pi)_{\pi \in \Pi}$ for a partition of C. all $|\Pi|$-many

invocation of `reassignToNewCenters` sums up to a running time bounded by $O(|X| \cdot |C|)$ □

Lemma E.3. `selectFinalCenters` *has a running time of* $O(nk^2)$ *if* $|\Pi_i| \leq k$ *and* $\left|C_j^i\right| \leq k$ *for all* $i \in [k]$ *and* $j \in [i]$.

Proof. Note that we consider k batches and batch i consists of $i \leq k$ entries. Hence, by Lemma E.2 the running time of each of the k `privateAssignmentHeuristic` invocations is $O(nk)$ yielding a total running time of $O(nk^2)$. □

F. Complete algorithm for PRIV-REP-KC

All building blocks have been defined and studied. It is time for the final algorithm.

Algorithm 20 PRIV-REP-KC-algorithm

Input: metric space (X, d), upper bound k, privacy bound L, color classes $X_1 \dot\cup X_2 \dot\cup \ldots \dot\cup X_\gamma = X$, color-specific lower and upper bounds (a_j, b_j) for $j \in [\gamma]$
Output: radius r^*, representative centers C^*, private assignment ϕ^*

$(c_1, \ldots, c_k) \leftarrow$ `gonzalez`$((X, d), k)$

$(\phi_1, r_1), \ldots, (\phi_k, r_k) \leftarrow$ `makePrefixesPrivate`$((X, d), L, (c_1, \ldots, c_k))$

for $i = 1, \ldots, k$ **do**
  $((\Pi_i, \eta_i^j), \tau_i^j, F_i^j)_{j \in [i]} \leftarrow$ `createBackbones`$((X, d), L, \{c_1, \ldots, c_i\}, \phi_i)$
end

for $i = 1, \ldots, k$ **do**
  $(\Delta_i^j, C_i^j, \psi_i^j)_{j \in [i]} \leftarrow$ `realizeBatchOfBackbones`$((X, d), (X_\ell)_{\ell \in [\gamma]}, (a_\ell, b_\ell)_{\ell \in [\gamma]}, k, (\Pi_i, \eta_i^j)_{j \in [i]})$
end

$C^* \leftarrow$ `selectFinalCenters`$((X, d), L, (\Pi_i, \phi_i, \tau_i^j, F_i^j, \Delta_i^j, C_i^j, \psi_i^j)_{j \in [i]})$

$\phi^*, r^* \leftarrow$ `makePrivate`$((X, d), L, C^*)$

return r^*, C^*, ϕ^*

Lemma F.1. *Our final algorithm in Algorithm 20 has a running time of* $O(nk^2 + k^5)$*, where* $n := |X|$ *is the number of points in the metric space.*

Proof. Gonzalez' algorithm (see Algorithm 2) runs in $O(nk)$. By Lemma B.7 `makePrefixesPrivate` has a running time of $O(nk^2)$. Each of the k invocation of `createBackbones` take $O(k^2)$, so in total $O(k^3)$ time; see Lemma C.6. For big k the bottle-neck are the k invocations of `realizeBatchOfBackbones`. By Lemma D.4, each of them take $O(nk + k^4)$ time, so in total $O(nk^2 + k^5)$. Our implementation of the `privateAssignmentHeuristic` leads to a running time of $O(nk^2)$ for `selectFinalCenters`; see Lemma E.3. Finally, `makePrivate` (which just uses the more general `makePrefixesPrivate`) takes $O(nk^2)$, as it is proven in Lemma B.7.

As all this steps are processed in sequence the overall running time is $O(nk^2 + k^5)$. □

G. Proof of Theorem 1.2.

In this section, we prove the main result of the paper.

We first argue about the running time. Our algorithm, presented in Algorithm 20 first runs `gonzalez` algorithm that computes a family of k sets of centers in time $O(nk)$, see Section 3.1 for details. Given a family of k sets of centers

we run `makePrefixesPrivate` procedure that for every set of centers in the family, by Lemma B.7, computes a private assignment in the overall running time $O(nk^2)$. Next, for each of k private assignments we obtained in the previous step we run the `createBackbones` procedure, each of which, by Lemma C.6, has the running time of $O(k^2)$. In the next step, one more time, for every Gonzalez prefix we run `realizeBatchOfBackbones` that, by Lemma D.4, has the running time of $O(nk + k^4)$. Next, given k batches of backbones from the previous step, we run `selectFinalCenters` that returns the solution for PRIV-REP-KC, by Lemma E.3, in time $O(nk^2)$. Finally, for representative centers computed in previous steps, we run the `makePrivate` procedure that solves MIN-PRIV-RADIUS problem, to find even better solution. By Lemma 3.2, the procedure runs in time $O(nk^2)$. Thus the total running time of Algorithm 20 is at most $O(nk + nk^2 + k \cdot k^2 + k \cdot (nk + k^4) + nk^2 + nk^2) = O(nk^2 + k^5)$.

Now we prove the approximation guarantee. We start with proving Theorem 2.4.

G.1. Proof of Theorem Theorem 2.4.

Since the running time statement was proved at the beginning of Appendix G, thus in the following, we prove just the approximation guarantee.

To prove the statement, we introduce the following definitions. For every Gonzalez prefix $C_i = \{c_1, \ldots, c_i\}$ for $i \in [k]$ we define $r_i \coloneqq \max_{x \in X} d(C_i, x)$. Moreover for every Gonzalez prefix C_i and a private assignment ϕ_i for $i \in [k]$, computed with the `makePrefixesPrivate` procedure, we define $\bar{r}_i \coloneqq \max_{x \in X} d(\phi_i(x), x)$.

Lemma G.1. *Let (X, d, k, L) be a Private k-Center instance. For any L-private r-clustering (C, ϕ) there exists an index i such that $\bar{r}_i \leq 2r$.*

Proof. First, we note that by Lemma B.8 the `makePrefixesPrivate` optimally solves the MIN-PRIV-RADIUS problem for all Gonzalez prefixes. Thus it can be used as a procedure to solve MIN-PRIV-RADIUS problem in Algorithm 3. Since, by Lemma 3.3, Algorithm 3 is a 2-approximation for the Private k-Center problem, this implies that there exists a Gonzales prefix C_i, computed by Algorithm 3, such that $\bar{r}_i \leq 2r$ for any L-private r-clustering (C, ϕ) for the (X, d, k, L) Private k-Center instance. □

In the main Algorithm 20 proposed in this paper for the PRIV-REP-KC problem, after computing private assignments with the procedure `makePrefixesPrivate`, for every Gonzalez prefix, we run the procedure `createBackbones`. For every Gonzalez prefix, this procedure computes k transfer forests with the corresponding backbones. In the following three lemmas, we prove the main properties of the transfer forest we will use for the proof of Theorem 2.4.

Lemma G.2. *Let (X, d, k, L) be a Private k-Center instance. For any L-private r-clustering (C, ϕ) and transfer forest (C_i, F_τ), for $i \in [k]$, with transfer threshold $\tau \in \mathbb{R}_{\geq 0}$ satisfying $\tau \geq 2\bar{r}_i + r$, it holds that, for any $c \in C$ all the points $\phi_i^{-1}(B(c, r))$ belong to the same connected component in the transfer forest (C_i, F_τ).*

Proof. Consider a center $c \in C$ and a connected component that contains the point $\phi_i^{-1}(c)$. Clearly, we have that $d\left(c, \phi_i^{-1}(c)\right) \leq \bar{r}_i$. We complete the proof by showing that for any other point $x \in B(c, r)$, the point $\phi_i^{-1}(x)$ lies in the same connected component. Indeed, since $d\left(x, \phi_i^{-1}(x)\right) \leq \bar{r}_i$, we have

$$d\left(\phi_i^{-1}(c), \phi_i^{-1}(x)\right) \leq d\left(\phi_i^{-1}(c), c\right) + d\left(c, x\right)) + d\left(x, \phi_i^{-1}(x)\right) \leq 2\bar{r}_i + r \leq \tau \ ,$$

where the first inequality follows by the triangle inequality. Hence, because the distance between $\phi_i^{-1}(c)$ and $\phi_i^{-1}(x)$ is below the transfer threshold, both points must lie in the same connected component of the transfer forest. □

Before we discuss the remaining two properties of transfer forests that we need, we introduce the following notation for the objects that are computed by Algorithm 20. Let (X, d, k, L) be a Private k-Center instance. For a Gonzales prefix C_i, a private assignment ϕ_i, a transfer threshold τ and a transfer forest (C_i, F_τ) with transfer threshold τ, let $\psi_{i,\tau}$ be an assignment computed by the procedure `transferPoints`. Moreover, let $\eta_{i,\tau}$ be the corresponding τ-aggregation computed with the procedure `computeTauAggregation`. The following lemma holds.

Lemma G.3. *Let (X, d, k, L) be a Private k-Center instance. For any transfer forest (C_i, F_τ), for $i \in [k]$, with transfer threshold $\tau \in \mathbb{R}_{\geq 0}$, and $W \subseteq C_i$ such that W is a subset of a connected component in the transfer forest (C_i, F_τ), it holds*

that:

$$\eta_{i,\tau}(W) = \left\lfloor \frac{\left|\psi_{i,\tau}^{-1}(W)\right|}{L} \right\rfloor .$$

Proof. First note that, by construction, for every center $w \in W$ that is not a root of the connected component of the transfer forest (C_i, F_i) that contains W, we have $\eta_{i,\tau}(w) = \frac{\left|\psi_{i,\tau}^{-1}(w)\right|}{L}$. Moreover, for the center $c_{\text{root}} \in C_i$ that is the root of the connected component of the transfer forest (C_i, F_i) that contains W, we have $\eta_{i,\tau}(c_{\text{root}}) = \left\lfloor \frac{\left|\psi_{i,\tau}^{-1}(c_{\text{root}})\right|}{L} \right\rfloor$. This implies as desired

$$\begin{aligned}\left\lfloor \frac{\left|\psi_{i,\tau}^{-1}(W)\right|}{L} \right\rfloor &= \left\lfloor \frac{\left|\psi_{i,\tau}^{-1}(W \setminus \{c_{\text{root}}\})\right|}{L} + \mathbb{1}_{c_{\text{root}} \in W} \frac{\left|\psi_{i,\tau}^{-1}(c_{\text{root}})\right|}{L} \right\rfloor \\ &= \frac{\left|\psi_{i,\tau}^{-1}(W \setminus \{c_{\text{root}}\})\right|}{L} + \left\lfloor \mathbb{1}_{c_{\text{root}} \in W} \frac{\left|\psi_{i,\tau}^{-1}(c_{\text{root}})\right|}{L} \right\rfloor = \eta_{i,\tau}(W) \ ,\end{aligned}$$

where $\mathbb{1}_{c_{\text{root}} \in W}$ is equal to 1 if $c_{\text{root}} \in W$ and 0 otherwise. □

To introduce the last property of transfer forests that we need, we use the notion of a parent of a vertex in the set of centers C_i on which the forest (C_i, F_τ) is spanned, as defined and computed in the procedure `computeTauAggregation`. In words, the parent of a vertex $c \in C_i$ is the neighboring vertex of c closer to the root in the connected component of the transfer forest that contains c. For simplicity, we use the convention that the parent of a root c is the root itself. For a subset $W \subseteq C_i$ we denote the set of parents of vertices in W by $P(W)$.

Lemma G.4. *Let (X, d, k, L) be a Private k-Center instance. For any transfer forest (C_i, F_τ), for $i \in [k]$, with transfer threshold $\tau \in \mathbb{R}_{\geq 0}$, and $W \subseteq C_i$, it holds that $\left|\phi_i^{-1}(W)\right| \leq \left|\psi_{i,\tau}^{-1}(W \cup P(W))\right|$.*

Proof. First, note that for any center $c \in C_i$ the procedure `computeTauAggregation` always transfers strictly less than L points to the parent of c. Thus

$$\left|\psi_{i,\tau}^{-1}(W \cup P(W))\right| = \left|\phi_i^{-1}(W)\right| + \left|\phi_i^{-1}(P(W))\right| - L \cdot |P(W)| \geq \left|\phi_i^{-1}(W)\right| \ ,$$

where the last inequality follows from the fact that ϕ is a private assignment, which implies that $\left|\phi_i^{-1}(P(W))\right| \geq L \cdot |P(W)|$. □

We are ready to prove Theorem 2.4.

Proof of Theorem 2.4. First, note that Algorithm 20 computes for every Gonzalez prefix C_i, for $i \in [k]$, a private assignment ϕ_i with procedure `makePrefixesPrivate`. Then, for every Gonzalez prefix C_i, the procedure `createBackbones` computes up to k different transfer thresholds such that, for each $\tau \in \mathbb{R}_{\geq 0}$, one of these transfer forests is a transfer forest corresponding to the transfer threshold τ. Together with τ-aggregations computed with `computeTauAggregation`, this leads to at most k backbones.

Now, let (X, d, k, L) be a Private k-Center instance, and let (C, ϕ) be an L-private r-clustering. By Lemma G.1 there exists a Gonzalez prefix C_i such that $\bar{r}_i \leq 2r$. Now, for the selected Gonzalez prefix we want to select a backbone. By Lemma G.2, for every $\tau \geq 2\bar{r}_i + r$ it holds that for any $c \in C$ all the points $\phi_i^{-1}(B(c, r))$ belong to the same connected component of the transfer forest (C_i, F_τ) with transfer threshold τ. Let

$$\tau^* := 2\bar{r}_i + r \ \leq 5r \ ,$$

and consider, among the at most k computed transfer forests, the transfer forest (C_i, F_{τ^*}) that corresponds to a transfer threshold of τ^*.

Next we prove that the backbone corresponding to Gonzalez prefix C_i and the transfer threshold τ^* is a $(7r, L)$-backbone. By the definition of the (ρ, L)-backbone, we have to show that there exists a $7r$-clustering with centers in C_i that, for each $c \in C_i$, assigns at least $\eta_{i,\tau^*}(c)$ points to c. To see that this holds, consider a point $x \in X$. We clearly have $x \in \phi^{-1}(c)$

for some $c \in C$. Note that there exists a center $c' \in C_i$ such that $x \in \phi_i^{-1}(c')$ and, by Lemma G.1, $d(x, c') \leq 2r$. Let us make the first attempt to construct the desired clustering with centers in C_i by assigning every point $x \in X$ to $\phi_i(x)$. This is a 2-clustering, however center in C_i might get more or less than η_{i,τ^*} points assigned. The key observation is that the τ^*-aggregation vector η_{i,τ^*} that is created in `computeTauAggregation` outputs the same vector as the `transferPoints` procedure, where the points are really reassigned to new centers. A point $x \in X$ assigned to a center $c' \in C_i$ can be reassigned to its parent $P(\{c'\})$. Note however that, in every step, at most L points from a cluster $\phi_i^{-1}(c')$ are reassigned to the parent of the center c'. Since the initial assignment ϕ_i is L-private, every cluster has at least L points and thus every point is reassigned at most once. Since for every point x we have $d(x, \phi_i(x)) \leq 2r$ and $d(\phi_i(x), P(\{\phi_i(x)\}) \leq \tau^* \leq 5r$, every point in the ψ_{i,τ^*} assignment is at most $7r$ far from its center and thus ψ_{i,τ^*} is a $7r$-clustering with centers in C_i that, for each $c \in C_i$, assigns at least $\eta_{i,\tau^*}(c)$ points to c.

It remains to show that C is an $8r$-realization for the $(7r, L)$-backbone (C_i, η_{i,τ^*}). By the definition of the Δ-realization, we have to show that there exists a map $\psi : C \mapsto C_i$ such that:

1. $d(c, \psi(c)) \leq 8r$ for each $c \in C$, and
2. $|\psi^{-1}(c')| \in \{1, \ldots, \eta_{i,\tau^*}\}$ for each $c' \in C_i$.

We show it by proving that there exists a b-matching in the bipartite graph $G(U \cup V, E)$ constructed as follows. Let U be the set of centers C and let V be the set of centers C_i. We connect a center $c \in C$ to a center $c' \in C_i$ with an edge if and only if $d(c, c') \leq 8r$. To finalize the construction we want every center $c \in C$ to be matched to exactly one center $c' \in C_i$ and we want every center $c' \in C_i$ to be matched to at most $\eta_{i,\tau^*}(c')$ centers in C. Hence this leads to a b-matching problem in G. Note that the existence of a solution to this b-matching in G is equivalent to C being an $8r$-realization of the $(7r, L)$-backbone (C_i, η_{i,τ^*}). We prove the existence of a b-matching in G using the Hall's theorem. More precisely, we show that for every subset of center $S \in C$, it holds that $|S| \leq \eta_{i,\tau^*}(N(S))$, where $N(S)$ is the set of centers in C_i connected with at least one center in S.

To prove that Hall's condition is satisfied, let $W \subseteq C_i$ be defined by

$$W \coloneqq \{c \in C_i : \ \phi_i^{-1}(c) \cap \phi^{-1}(S) \neq \emptyset\} \ .$$

Note that $W \subseteq N(S)$. Thus $|S| \leq \frac{|\phi_i^{-1}(W)|}{L}$ which, by Lemma G.4, is at most $\frac{|\psi_{i,\tau^*}^{-1}(W \cup P(W))|}{L}$. This implies

$$|S| \leq \left\lfloor \frac{|\psi_{i,\tau^*}^{-1}(W \cup P(W))|}{L} \right\rfloor = \eta_{i,\tau^*}(W \cup P(W)) \ ,$$

where the last inequality follows from Lemma G.3. Hence, Hall's condition is satisfied because $W \cup P(W) \subseteq N(S)$, and thus $|S| \leq \eta_{i,\tau^*}(N(S))$. □

We highlight that the above proof also immediately implies Theorem 4.1.

Proof of Theorem 4.1. The theorem immediately follows by observing that the above proof of Theorem 2.4 only considers backbones coming from the computed Gonzales prefixes. □

G.2. Proof of Theorem 1.2.

In this subsection we want to prove that Algorithm 20 computes 15-approximate solution for PRIV-REP-KC. Note that in Theorem 2.6 we proved that the simplified version of it, i.e., Algorithm 1, is a 15-approximation for PRIV-REP-KC. The proof that Algorithm 20 is a 15-approximation follows the same line of reasoning. We show that the changes compared to Algorithm 1 do not impact the approximation guarantee.

Theorem G.5. *Algorithm 20 is a* 15*-approximation for* PRIV-REP-KC.

Proof. Let (C, ϕ) be an optimal PRIV-REP-KC solution with radius r^*. By Theorem 2.4 we know that Algorithm 20 computes $q \leq k^2$ backbones $(\Pi_1, \eta_1), \ldots (\Pi_q, \eta_q)$ one of which is a $(7r^*, L)$-backbone for which C is an $8r^*$-realization.

For every Gonzalez prefix Algorithm 20 runs `realizeBatchOfBackbones`, that for each of at most k backbones runs `computeRepresentativeDeltaRealization`. Procedure `computeRepresentativeDeltaRealization` for every backbone (Π_i, η_i) returns the set of centers C_i that is a Δ_i-realization of (Π_i, η_i). By Lemma D.3 procedure

`computeRepresentativeDeltaRealization` solves the MIN-REP-REALIZATION problem and thus one of the set $C_1, \ldots, C_q$ is an $8r^*$-realization of $(7r^*, L)$-backbone. Note, that similarly to Algorithm 20, by Observation 2.3 after solving MIN-PRIV-RADIUS for all the centers $C_1, \ldots, C_q$ one of the solutions would be a 15-approximate for the PRIV-REP-KC. However, this would not lead to an algorithm with the claimed running time. Instead, Algorithm 20 runs `selectFinalCenters` that for every Gonzalez index i first, among $q_i \leq k$ backbones selects the one that corresponds to the smallest $\Delta_j + \tau_j$, for $j \in [q_i]$. Following the notation in the algorithm for `selectFinalCenters` lets denote this index by j_i^*. Note that since we chose the smallest $\Delta_j + \tau_j$, for $j \in [q_i]$ for each Gonzalez index i, among the selected backbones there still exists one backbone (Π, η) for which the computed set of centers with `computeRepresentativeDeltaRealization` is an $8r^*$-realization of the $(7r^*, L)$-backbone. Then Algorithm 20, for every Gonzalez index $i \in [k]$ runs `privateAssignmentHeuristic`. The procedure for given backbone $(\Pi^{j_i^*}, \eta^{j_i^*})$, transfer forest $F^{j_i^*}$, transfer threshold $\tau^{j_i^*}$, set of centers $C^{j_i^*}$ computed with `computeRepresentativeDeltaRealization` that is $\Delta^{j_i^*}$-realization of the backbone $(\Pi^{j_i^*}, \eta^{j_i^*})$ with the assignment $\psi^{j_i^*} : C^{j_i^*} \mapsto \Pi^{j_i^*}$ and a private assignment $\phi^{j_i^*} : X \mapsto \Pi^{j_i^*}$ finds an assignments $\overline{\phi}_i : X \mapsto \Pi^{j_i^*}$, with radius $\overline{r}_i \coloneqq \max_{x \in X} d(x, \overline{\phi}_i) \leq \tau^{j_i^*} + \Delta^{j_i^*}$. This implies that, for $\ell \coloneqq \arg\min_{i \in [k]} \overline{r}_i$ the set of centers $C^{j_\ell^*}$ is an $8r^*$-realization of the $(7r^*, L)$-backbone and the solution $(C^{j_\ell^*}, \overline{\psi}_\ell)$ is a 15-approximate solution for the PRIV-REP-KC. Note, that for the selected set of centers $C^{j_\ell^*}$ Algorithm 20 runs the procedure `makePrivate`, to find even better assignment than the one given by $\overline{\psi}_\ell$. Thus Algorithm 20 is a 15-approximation for the PRIV-REP-KC. □

The proof of Theorem 1.2 now follows by observing that Algorithm 20 has the claimed guarantees.

Proof of Theorem 1.2. The running time bound of $O(nk^2 + k^5)$ of Algorithm 20 follows from Lemma F.1 and it is a 15-approximation due to Theorem G.5. □