Exact Recovery of Clusters in Finite Metric Spaces Using Oracle Queries

Marco Bressan MARCO.BRESSAN@UNIMI.IT
Dept. of Computer Science, Università degli Studi di Milano, Italy

Nicolò Cesa-Bianchi NICOLO.CESA-BIANCHI@UNIMI.IT
DSRC & Dept. of Computer Science, Università degli Studi di Milano, Italy

Silvio Lattanzi SILVIOL@GOOGLE.COM
Google

Andrea Paudice ANDREA.PAUDICE@UNIMI.IT
Dept. of Computer Science, Università degli Studi di Milano & Istituto Italiano di Tecnologia, Italy

Editors: Mikhail Belkin and Samory Kpotufe

Abstract

We investigate the problem of exact cluster recovery using oracle queries. Previous results show that clusters in Euclidean spaces that are convex and separated with a margin can be reconstructed exactly using only $\mathcal{O}(\log n)$ same-cluster queries, where n is the number of input points. In this work, we study this problem in the more challenging non-convex setting. We introduce a structural characterization of clusters, called (β, γ)-convexity, that can be applied to any finite set of points equipped with a metric (or even a semimetric, as the triangle inequality is not needed). Using (β, γ)-convexity, we can translate natural density properties of clusters (which include, for instance, clusters that are strongly non-convex in $\mathbb{R}^d$) into a graph-theoretic notion of convexity. By exploiting this convexity notion, we design a deterministic algorithm that recovers (β, γ)-convex clusters using $\mathcal{O}(k^2 \log n + k^2 (6/\beta\gamma)^{\mathrm{dens}(X)})$ same-cluster queries, where k is the number of clusters and $\mathrm{dens}(X)$ is the density dimension of the semimetric. We show that an exponential dependence on the density dimension is necessary, and we also show that, if we are allowed to make $\mathcal{O}(k^2 + k \log n)$ additional queries to a "cluster separation" oracle, then we can recover clusters that have different and arbitrary scales, even when the scale of each cluster is unknown.

Keywords: Non-convex clusters, same-cluster queries, geodesic convexity.

1. Introduction

We investigate the problem of exact reconstruction of clusters using oracle queries in the semi-supervised active clustering framework (SSAC) of Ashtiani et al. (2016). In SSAC, we are given n points in a metric space, and the goal is to partition these points into k clusters with the help of an oracle answering queries of the form "do x and y belong to the same cluster?". When the metric is Euclidean, Ashtiani et al. (2016), Bressan et al. (2020) and Bressan et al. (2021) show that exact reconstruction is possible using only $\mathcal{O}(\log n)$ oracle queries, which mirrors the query complexity of efficient active learning. These results heavily rely on the Euclidean geometry of the clusters; in particular, clusters are assumed to be convex (e.g., ellipsoidal) and separable with a margin. These assumptions exclude many natural definitions of "cluster", such as those based on notions of density, or those computed by popular techniques like spectral clustering, linkage clustering, or DBSCAN.

In this work we study exact cluster recovery in metric spaces, or —more generally— finite semimetric spaces (where the triangle inequality is not necessarily satisfied). The use of semimetrics in clustering, and in other machine learning tasks, is motivated by the fact that in many applications domains the notion of distance is strongly non-Euclidean (Gottlieb et al., 2017). Classic examples include the Wasserstein distance in vision, the Levenshtein distance in string matching, the cosine dissimilarity in document analysis, the Pearson dissimilarity in bioinformatics. In all these cases, the notions of convexity and separability with margin are lost, or exist only in certain generalized forms, so the cluster recovery techniques of Ashtiani et al. (2016); Bressan et al. (2020, 2021) do not apply anymore. To fill this gap, we introduce a novel notion of cluster convexity that can be applied to any finite semimetric space and that can be exploited algorithmically.

We start by considering *geodesic convexity* in weighted graphs (Pelayo, 2013), a well-known generalization of Euclidean convexity that has been used, among others, for node classification in graphs Thiessen and Gärtner (2020). Given a weighted graph $\mathcal{G}$, a subset $C \subseteq V(\mathcal{G})$ is said to be geodesically convex if every shortest path between any two vertices of C lies entirely in C. Thus, in a finite semimetric space, we could say that C is a convex cluster if it is geodesically convex in the weighted graph $\mathcal{G}$ encoding the semimetric (with the distances as weights). Unfortunately, this condition is too lax. To see this, take n distinct points on a circle with the Euclidean metric. In $\mathcal{G}$, the shortest path between any two points x, y is always the edge (x, y) itself. Thus, according to this definition, any subset of the n points will be geodesically convex, and so every clustering will be admitted, which means that $\Omega(n)$ queries will be needed to recover the clustering. However, we will show that a variant of this approach gives a suitable notion of convexity, one that yields efficient recovery with only $\mathcal{O}(\log n)$ queries while capturing the density of the clusters.

Our contributions.

We introduce (β, γ)-*convex clusterings*, a novel family of clusterings defined on the weighted graph $\mathcal{G}$ encoding the semimetric on X. For any $\varepsilon > 0$, let $G_X(\varepsilon)$ be the unweighted subgraph of $\mathcal{G}$ obtained by deleting all edges (x, y) with $d(x, y) > \varepsilon$. We say that a clustering is (β, γ)-convex (with $\beta, \gamma \in (0, 1]$) if for some $\varepsilon > 0$ every cluster C satisfies the following three properties. *Connectivity*: the subgraph of $G_X(\varepsilon)$ induced by C is connected. *Local metric margin*: if $x \in C$ and $y \notin C$, then $d(x, y) > \beta\varepsilon$[1]. *Geodesic convexity with margin*: in $G_X(\varepsilon)$, if $x, y \in C$ and the shortest path between x and y has length ℓ, then any simple path between x and y of length at most $\ell(1 + \gamma)$ does not leave C. The smallest ε for which these properties hold is called the *radius* of the clusters. It is important to observe that (β, γ)-convexity includes nontrivial cases. For instance, (β, γ)-convex clusters can be strongly non-convex in $\mathbb{R}^d$, as depicted in Figure 1. Moreover, we can allow the clusters to have different radii $\varepsilon_1, \ldots, \varepsilon_k$ (see below), as depicted in Figure 2. These examples suggest that, in a certain sense, one can view (β, γ)-convexity as a way of translating density into convexity. Moreover, if we drop any one of the three conditions —connectedness, local metric margin, and geodesic convexity with margin— the clusters can become disconnected, too close to one another, or interspersed with other clusters, see Section 3.1.

Our first result shows that (β, γ)-convex clusterings can be recovered efficiently using a small number of same-cluster queries (SCQ for short). More precisely, if $\varepsilon, \beta, \gamma$ are known, and for each

1. Note that, for any clustering C in a finite semimetric space X and for any $\varepsilon > 0$, a β such that the local metric margin condition holds can always be found. In this respect, β defines a hierarchy over clusterings, where large values of β identify clusterings that are easier to learn.

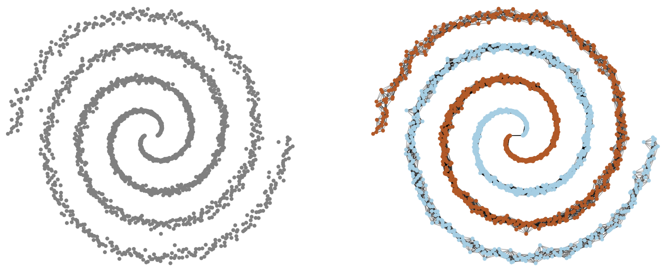
Figure 1: Left: a toy point set. Right: the graph $G_X(\varepsilon)$ and a valid (β, γ)-convex clustering for X (clusters encoded by the color of the points), for any $\beta < \frac{1}{2}$ and $\gamma > 0$.

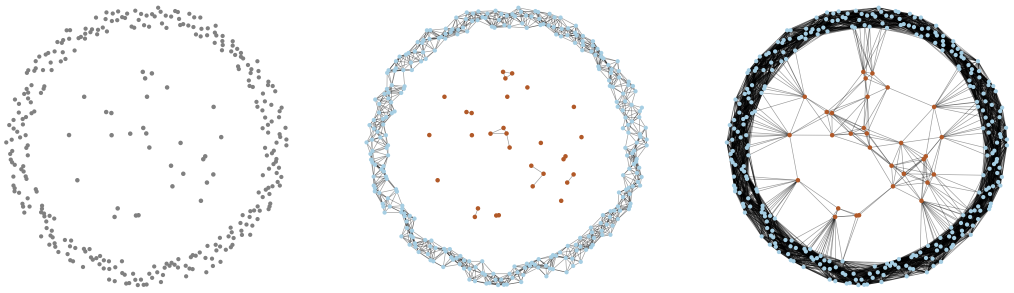
Figure 2: Left: a toy point set. Middle and right: the graphs $G_X(\varepsilon_1)$ and $G_X(\varepsilon_2)$ where $\varepsilon_1 < \varepsilon_2$ are the radii of the "outer" cluster C_1 and the "inner" cluster C_2. The clustering is (β, γ)-convex for $\beta \leq .5$ and $\gamma \leq .1$.

cluster we know an arbitrary initial point, called *seed*,[2] then we can deterministically recover all clusters with $\mathcal{O}\big(k^2 \log n + k^2 ({}^{6}/_{\beta\gamma})^{\mathrm{dens}(X)}\big)$ SCQ queries. Here, $\mathrm{dens}(X)$ is the *density dimension* of X (Gottlieb and Krauthgamer, 2013), a generalization of the doubling dimension that, in metric spaces, is used to bound the size of packings. This dependence of our exponent on $\mathrm{dens}(X)$ is asymptotically optimal, as we prove that, in the worst case, any algorithm needs $\Omega(2^{\mathrm{dens}(X)})$ SCQ queries to recover a (β, γ)-convex clustering. The running time of our algorithm is $\mathcal{O}\big(k^2(n+m)\big)$, where m is the number of edges of $\mathcal{G}$ (i.e., the number of finite distances between the input points). The key step of our algorithm consists in finding a *cluster separator* between each cluster C and the other clusters. To do this, we need to find edges between C and other clusters in the graph $G_X(\varepsilon)$, which requires to carefully exploit the structural properties of (β, γ)-convexity. We note that, interestingly, in the r-dimensional Euclidean setting, the cluster recovery algorithm of (Bressan et al., 2021) makes a number of queries roughly of the order of $\mathcal{O}\big(({}^{1}/_{\gamma})^r \log n\big)$, where γ is the convex margin. This shows that our notion of convexity plays a role similar to that of Euclidean convexity.

Next, we investigate the power of queries. First, we show that, without seed nodes, any algorithm using only the SCQ oracle needs $\Omega(n)$ queries to recover the clusters. To circumvent this lower bound, we add a more powerful query, called SEED. Given a partition of X and the id of a cluster, the

2. We note here that this last assumptions can be dropped if the clusters have roughly similar sizes. In fact, if the gap between the size of the largest and smallest clusters is χ we can obtain the seed w.h.p. by sampling $\tilde{\mathcal{O}}(k\chi)$ nodes and retrieving their cluster membership using $\tilde{O}(k^2\chi)$ same cluster queries

SEED query provides a certificate (i.e., a point of X) that the cluster is cut by the partition, or answers negatively if the cluster is not cut. We show that, if we can use $\mathcal{O}\left(k^2 \log n + k^2 \left(6/\beta\gamma\right)^{\operatorname{dens}(X)}\right)$ SCQ queries plus only $\mathcal{O}(k^2)$ SEED queries, then we can recover clusters with *different* radii $\varepsilon_1, \ldots, \varepsilon_k$, a case that we model by generalizing the (β,γ)-convex definition in a natural way. This allows us to capture clusters with different "scales", as shown in Figure 2. If the radii $\varepsilon_1, \ldots, \varepsilon_k$ are unknown, we show that $\mathcal{O}(k \log n)$ SEED queries are sufficient to learn them in time $\mathcal{O}\big(m\alpha(m,n) + kn \log n\big)$, where α is the inverse of the Ackermann function, and that no algorithm can learn the radii with less than $\Omega(k \log \frac{n}{k})$ queries. Furthermore, if one of β and γ is unknown, we show that we can still learn the clusters by paying a small overhead.

2. Related work

Exact reconstruction of clusters with same-cluster queries was introduced by Ashtiani et al. (2016), who showed how to recover exactly the optimal k-means clustering with $\mathcal{O}(\operatorname{poly}(k) \log n)$ queries when each cluster lies inside a sphere centered in the cluster's centroid and well separated from the spheres of other clusters. Bressan et al. (2020) extend these results to clusters separated by arbitrary ellipsoids with arbitrary centers, and Bressan et al. (2021) to arbitrary convex clusters with margin. Both results assume the standard Euclidean metric.

SEED queries have been used by Hanneke (2009) as "positive example queries", by Balcan and Hanneke (2012) as "conditional class queries", by Beygelzimer et al. (2016); Attenberg and Provost (2010) as "search queries", and, implicitly, also by Tong and Chang (2001); Doyle et al. (2011). Previous works also show that SEED queries are well justified in practice, as noted by Beygelzimer et al. (2016).

As with $O(k)$ SCQ queries one can learn the cluster id of any point (up to a relabeling of the clusters), we could reduce our problem to the problem of classifying the nodes of $G_X(\varepsilon)$. Dasarathy et al. (2015) develop a probabilistic active classification algorithm, called S^2 (*s*hortest-*s*hortest-path), whose label complexity depends on the graph's structure. In particular, the label complexity is linear in the size of the boundary of the cut between nodes with different labels. Unfortunately, even under (β,γ)-convexity, in $G_X(\varepsilon)$ this boundary can have size $\Omega(n)$. Thiessen and Gärtner (2020) show a deterministic algorithm with label complexity proportional to the size of the shortest path cover of the graph (the smallest set of shortest paths that cover all nodes). Again, in $G_X(\varepsilon)$ this cover could have size $\Omega(n)$ even under (β,γ)-convexity. Active classification on unweighted graph has been also studied by Afshani et al. (2007); Cesa-Bianchi et al. (2010); Guillory and Bilmes (2011), but only with approximate reconstruction guarantees (i.e., $\Omega(n)$ queries may be needed for exact recovery).

Mazumdar and Saha (2017) study exact cluster reconstruction with a SCQ oracle on weighted graphs, where weights express similarities. They prove a logarithmic query bound using a Monte-Carlo algorithm and a log-linear bound using a Las-Vegas algorithm. However, similarly to a stochastic block model, they assume that the weights are drawn from some latent distribution that depends on the clustering. Stochastic block models (Zhang et al., 2014; Gadde et al., 2016) and geometric block models (Chien et al., 2020) have been also considered as generative models for active clustering on graphs.

Center-based (Balcan and Long, 2013), density-based (Mai et al., 2013), spectral (Wang and Davidson, 2010; Shamir and Tishby, 2011), and hierarchical (Eriksson et al., 2011; Krishnamurthy et al., 2012) clustering have been also studied in a more restricted active learning setting, where the algorithm has access to the pairwise distances through an oracle.

3. Preliminaries and notation

Our algorithms receive in input a semimetric represented by an undirected weighted graph $\mathcal{G} = (X, \mathcal{E}, d)$, where $d(x,y) > 0$ is the weight[3] of the edge $(x,y) \in \mathcal{E}$ and $|X| = n$. For $\varepsilon > 0$, we let $G_X(\varepsilon)$ be the undirected graph with vertex set X where $x, y \in X$ are connected if and only if $d(x,y) \le \varepsilon$. Given a graph $G = (V, E)$ and two vertices $x, y \in V$, we denote by $d_G(x,y)$ the shortest-path distance between x and y in G and by $\rho(G)$ the number of connected components of G. Given any subset $U \subseteq V$, we use $G[U]$ to denote the subgraph of G induced by U, and we use $\Gamma(U)$ to denote the set of edges having exactly one endpoint in U. An edge $(x,y) \in \Gamma(U)$ is called a *cut edge* of U.

For any $x \in X$ and $r > 0$, the ball of center x and radius r is $B(x,r) = \{y \in X : d(x,y) \le r\}$. For any set $K \subseteq X$, we denote by $\mathcal{M}(K, r)$ the maximum cardinality of any subset A of K such that all distinct $x, y \in A$ satisfy $d(x,y) > r$. Following Gottlieb et al. (2017), we define the *density constant* of X as:[4]

$$\mu(X) = \min \{\mu \in \mathbb{N} : (x \in X) \wedge (r > 0) \Rightarrow \mathcal{M}(B(x,r), {}^{r}/_{2}) \le \mu\} \tag{1}$$

Therefore, in X, any ball of radius r contains at most $\mu(X)$ points at pairwise distance larger than $\frac{r}{2}$. The *density dimension* of X is $\operatorname{dens}(X) = \log_2 \mu(X)$. It is easy to see that, for any ball $B(x,r)$ and for any $\eta \in (0,1)$ we have:

$$\mathcal{M}(B(x,r), \eta r) \le \mu(X)^{\left\lceil \log_2 \frac{1}{\eta} \right\rceil} \le \left({}^{2}/_{\eta}\right)^{\operatorname{dens}(X)} \tag{2}$$

When d is a metric, we have $\operatorname{dbl}(X) \le \operatorname{dens}(X) \le 2 \operatorname{dbl}(X)$ where $\operatorname{dbl}(X)$ is the doubling dimension of X, see Lemma 17 in Appendix A. We denote by $\mathcal{M}^*(\eta)$ the maximum of $\mathcal{M}(B(x,r), \eta r)$ over all $x \in X$ and $r > 0$. The quantity $\mathcal{M}^*(\eta)$ appears in our bounds, with η being a function of β and γ. Note that $\mathcal{M}^*(\eta) \le \left({}^{2}/_{\eta}\right)^{\operatorname{dens}(X)}$, by (2).

A k-clustering of X is a partition $\mathcal{C} = (C_1, \dots, C_k)$ of X. We denote by $\mathcal{C}(x)$ the cluster id of x, so that $x \in C_{\mathcal{C}(x)}$. We now introduce the characterization of the clusterings considered in this work. The following definition is for clusters with identical radii; a more complex version will be needed in the case with different radii, see Section 5.

Definition 1 ((β, γ)-convex clustering) *For any $\beta, \gamma \in (0,1]$, a k-clustering $\mathcal{C} = (C_1, \dots, C_k)$ of X is (β, γ)-convex if $\exists\, \varepsilon > 0$ such that for each $i \in [k]$ the following properties hold:*

1. connectedness*: the subgraph induced by C_i in $G_X(\varepsilon)$ is connected*
2. local metric margin*: for all $x, y \in X$, if $x \in C_i$ and $y \notin C_i$, then $d(x,y) > \beta\varepsilon$*
3. geodesic convexity with margin*: if $x, y \in C_i$, then in $G_X(\varepsilon)$ any simple path between x and y of length at most $(1+\gamma) d_{G_X(\varepsilon)}(x,y)$ lies entirely in C_i*

The smallest value of ε satisfying the three properties is called the *radius* of the clusters.[5]

In this work, we assume the algorithm obtains information about the unknown target clustering $\mathcal{C}$ through *same-cluster queries* (Ashtiani et al., 2016):

SCQ: for any $x, y \in X$, $\text{SCQ}(x,y)$ returns $\mathbb{I}\{\mathcal{C}(x) = \mathcal{C}(y)\}$, where $\mathbb{I}\{\cdot\}$ is the indicator function

3. Our query bounds do not depend on the size of the weights.
4. While (Gottlieb et al., 2017) uses open balls, we use closed balls.
5. Actually, our algorithm of Section 4 works with *any* such ε. We use the minimum to disambiguate the radius.

If only same-cluster queries are available, we assume that, together with X, the recovery algorithm is given a *seed node* $s_i \in C_i$ for each cluster C_i. Seed nodes are collected in a vector $\boldsymbol{s} = (s_1, \dots, s_k)$. Without seed nodes, $\Omega(n)$ same-cluster queries are needed to recover $\mathcal{C}$ (see Section 8)[6].

When the cluster radii are different and/or unknown, or the seed nodes are not available, we show that there are instances where $\Omega(n)$ same cluster queries are needed. To overcome this limitation, we allow the algorithm to use another type of queries, called *seed queries*:

SEED: for any $S \subseteq X$ and $i \in [k]$, $\textsc{seed}(S, i)$ returns an arbitrary point $s_i \in C_i \cap S$, or NIL if $C_i \cap S = \emptyset$.

The SEED query is a kind of separation oracle: given a partition $(S, X \setminus S)$ of X and a cluster label i, the query can be used to check whether C_i is cut by this partition. Indeed, if C_i is cut, then $\textsc{seed}(S, i)$ and $\textsc{seed}(X \setminus S, i)$ return a point of C_i belonging to, respectively, S and $X \setminus S$. Note that these queries are very natural. In a crowdsourcing setting, they correspond to asking the rater to identify an entity (say a picture of a car) within a set (say a set of pictures).

3.1. Necessity of the properties of Definition 1

We give some representative examples of degenerate clusterings resulting from dropping any of the properties of Definition 1. We assume that $k = 2$ and $X \subseteq \mathbb{R}^2$ with d being the Euclidean metric. The examples are depicted in Figure 3 below.

Removing connectivity. Choose any $\beta, \gamma \leq 1$. Let X consist of disjoint subsets, each one with Euclidean diameter $\leq \varepsilon$, and sufficiently far from each other. Label any subsets as C_1 and the rest as C_2. Note that the second and third property of Definition 1 are satisfied.

Removing local metric margin. Choose any $\gamma \leq 1$. Let C_1 be formed by collinear points equally spaced by ε, and the same for C_2, so that two extremal points of C_1 and C_2 are at arbitrarily small distance $\delta < \varepsilon$. Note that the first and third property of Definition 1 are satisfied.

Removing geodesic convexity with margin. Choose any $\beta < \frac{1}{2}$. The set X is formed by collinear points equally spaced by $\frac{\varepsilon}{2}$, with the points of cluster C_1 interleaved with those of cluster C_2. Note that the first and second property of Definition 1 are satisfied, but the third is violated for any $\gamma = \omega(1/n)$: take the two extreme nodes of C_1 and change their shortest path to use a node of C_2.

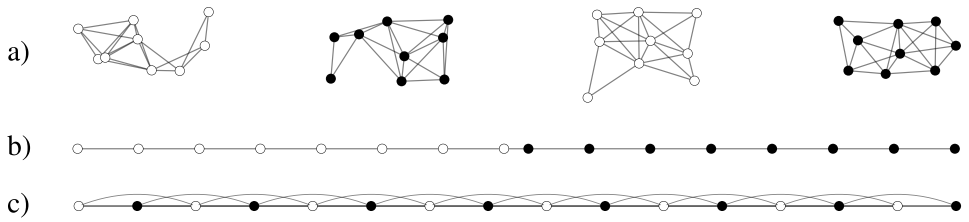

Figure 3: a) a clustering violating connectivity, b) a clustering violating local metric margin, c) a clustering violating geodesic convexity with margin. Empty nodes are in C_1, filled nodes are in C_2. Shown are the edges of $G_X(\varepsilon)$.

6. As we already said, if the sizes of the clusters are "roughly" balanced we can remove this assumption and sample $\widetilde{\mathcal{O}}(k)$ random nodes and use $\widetilde{\mathcal{O}}(k^2)$ same-cluster queries to obtain the seed nodes with high probability.

3.2. Relationship with other clustering notions

As noted, (β,γ)-convexity is meant to capture density-based clusterings produced by popular algorithms such as Single Linkage and DBSCAN. Those clusterings, however, are in general *not* recoverable with less than $\Omega(n)$ queries, since they allow for $\Omega(n)$ ties (points that can be assigned to one of two clusters in an arbitrary way). Therefore, (β,γ)-convexity should be thought of as an *additional* property, to be requested *on top of* existing notions of clustering in order to obtain efficient exact recovery. Here, we give two examples of how existing notions of clustering yield (β,γ)-convexity in particular cases.

The first example is DBSCAN, whose parameters are the connectivity radius r and the density parameter κ. The clustering is defined by looking at $G_X(r)$, clustering together any maximal (sub)tree whose vertices all have degree at least κ, and assigning each remaining vertex to the same cluster as some of its neighbors (if it has a neighbor). Now, suppose that $G_X(r)$ is formed by k connected components, and each one of them is spanned by a tree on vertices that have degree at least κ. Then, by taking $\varepsilon = r$, one can see that each such component is a cluster that is (β,γ)-dense for $\beta \geq 1$ and every $\gamma > 0$.

The second example is that of spherical clusters with margin (Ashtiani et al., 2016). In this case we use the generalized (β,γ)-convexity, see Definition 10. Let $X \subset \mathbb{R}^d$, and suppose that every cluster C_i is contained in some ball $B_i = B(c_i, r_i)$, so that, for some $\delta > 0$, $B(c_i, r_i(1+\delta)) \cap C_j = \emptyset$ for all $j \neq i$. Suppose that each C_i is a set of $(\frac{a}{\delta})^d$ points drawn independently and uniformly from B_i, for some constant $a > 0$. We claim that the resulting clustering is (β,γ)-convex with high probability, for $\beta = 1$ and any $\gamma > 0$. Indeed, $(\frac{a}{\delta})^d$ points draws uniformly at random from B_i are with high probability a $\frac{\delta r_i}{2}$-net for B_i (Vershynin, 2018) when a is sufficiently large. In this case, it is easy to see that C_i is connected in $G_X(\varepsilon_i)$, where $\varepsilon_i = \delta r_i$, thus satisfying condition (1) of Definition 10. Moreover, by assumption, for any $x \in C_i$ and any $y \in C_j$ with $j \neq i$ we have $d(y,x) > \varepsilon_i$, thus satisfying conditions (2) and (3) of Definition 10 for $\beta = 1$ and any $\gamma > 0$.

Paper organization. Section 4 gives the step-by-step construction of our algorithm for recovering (β,γ)-convex clusterings. Section 5 extends (β,γ)-convexity to capture clusters with different radii, and shows how, by introducing SEED queries, our algorithm can be extended to this case. Section 6 shows how we can learn the radii as well as β or γ, using again SEED queries. Section 7 discusses an efficient implementation of our algorithms. Section 8 presents our query complexity lower bounds.

4. Exact recovery of (β,γ)-convex clusters

Throughout this section we assume that $\varepsilon, \beta, \gamma$ are known, and that we know a vector of seed nodes $\boldsymbol{s} = (s_1, \ldots, s_k)$, so that $s_i \in C_i$ for all i. Under these assumptions, our goal is to construct an algorithm, called RecoverSingleCluster, that for any i returns C_i using $\mathcal{O}\big(k \log n + k\,\mathcal{M}^*\big(\frac{\beta\gamma}{2+\gamma}\big)\big)$ SCQ queries and time $\mathcal{O}(k(n+m))$. By running RecoverSingleCluster once for each i, it is immediate to obtain a full cluster recovery algorithm, RecoverClustering, with the following guarantees. Note that $\mathcal{M}^*\big(\frac{\beta\gamma}{2+\gamma}\big) \leq (6/\beta\gamma)^{\mathrm{dens}(X)}$ since $\mathcal{M}^*(\eta) \leq (2/\eta)^{\mathrm{dens}(X)}$ and $\gamma \leq 1$.

Theorem 2 *Suppose $\mathcal{C}$ is (β,γ)-convex (Definition 1). Then* RecoverClustering$(X, \varepsilon, \boldsymbol{s})$ *deterministically returns $\mathcal{C}$ in time $\mathcal{O}(k^2(n+m))$ using $\mathcal{O}\big(k^2 \log n + k^2\,(6/\beta\gamma)^{\mathrm{dens}(X)}\big)$* SCQ *queries.*

The rest of this section describes RecoverSingleCluster and proves Theorem 2, with the exception of the running time bound, which is proven in Section 7. Unless otherwise specified, from now on G stands for $G_X(\varepsilon)$.

4.1. Margin-based separation, and binary search on shortest paths.

We start with a subroutine MBS (Margin-Based Separator) that, given an input set Z and the cluster index i, computes $Z \cap C_i$. The routine uses the local metric margin and is efficient when the metric radius of Z is small.

Algorithm 1: MBS(Z, ε, u)

```
1 $U := \emptyset$
2 for each connected component $H$ of $G_Z(\beta\varepsilon)$ do
3     choose any $x \in V(H)$
4     if SCQ$(x, u) = 1$ then add $V(H)$ to $U$
5 return $U$
```

Lemma 3 *For any $u \in X$ and any $Z \subseteq X$ such that $Z \subseteq B(z, r)$ for some $z \in X$ and $r < \infty$,* MBS(Z, ε, u) *returns $Z \cap C_i$ using $\mathcal{M}^*\left(\frac{\beta\varepsilon}{r}\right)$* SCQ *queries, where $i = \mathcal{C}(u)$.*

Proof Let $G_Z = G_Z(\beta\varepsilon)$. If H is any connected component of G_Z, then $d(x, y) \le \beta\varepsilon$ for all $x, y \in H$. Thus, a repeated application of the local metric margin implies that all nodes of H belong to the same cluster. Therefore, either $V(H) \cap C_i = \emptyset$, or $V(H) \subseteq C_i$. This shows that x is added to U if and only if $x \in C_i$, proving the correctness. For the query complexity, let P be the set of points x queried by the algorithm at line 4. Clearly P is an independent set in G_Z and, thus, $d(x, y) > \beta\varepsilon$ for all distinct $x, y \in P$. By the definition of $\mathcal{M}^*$ this implies that $|P| \le \mathcal{M}^*(\frac{\beta\varepsilon}{r})$. ■

Next, we introduce a condition for finding efficiently a cut edge of C_i. A path $\pi = (x_1, \dots, x_{|\pi|})$ is *C_i-prefixed* if there exists an index $j^* \in \left[|\pi|\right]$ such that $x_j \in C_i$ if and only if $j \in \{1, \dots, j^*\}$.

Lemma 4 *Let $C_i \subseteq R \subseteq X$ such that $G[R]$ is connected. Then, in $G[R]$, any shortest path between any $s_i \in C_i$ and any $s \in R \setminus C_i$ is C_i-prefixed.*

Proof Suppose by contradiction that in $G[R]$ there exists a shortest path π between $s_i \in C_i$ and $s \in R \setminus C_i$ that is not C_i-prefixed. Then some prefix π' of π is a shortest path between $s_i \in C_i$ and $s_i' \in C_i$ that intersects $R \setminus C_i$. Now observe that π' is a shortest path in G, too. This holds because any shortest path between s_i and s_i' in G lies inside $G[C_i]$ by geodesic convexity, and thus inside $G[R]$ as $C_i \subseteq R$. By geodesic convexity this implies that $\pi' \subseteq G[C_i]$, a contradiction. ■

Finally, we observe that, in a C_i-prefixed simple path, a cut edge of C_i can be found via binary search from the endpoints of the path. This yields a subroutine FindCutEdge with the following guarantees (pseudocode in Appendix B):

Observation 1 *Given a simple C_i-prefixed path π,* FindCutEdge(π) *returns the unique cut edge of C_i in π using $\mathcal{O}(\log n)$* SCQ *queries.*

4.2. Cluster separators

We introduce the notion of cluster separator, which is at the heart of our algorithm.

Definition 5 *A partition (S_i, S_j) of X is an (i, j)-separator of X if $S_i \cap C_j = S_j \cap C_i = \emptyset$.*

This is similar to half-spaces in abstract closure systems (Seiffarth et al., 2019), where we would have $C_i \subseteq S_i$ and $C_j \subseteq S_j$. We use the weaker condition $S_i \cap C_j = S_j \cap C_i = \emptyset$ because in some of our algorithms X will be a subset of the input set, in which case $C_j \subseteq X$ might not hold. On the other hand, we will always make sure that $C_i \subseteq X$ holds.

Now, if $C_i \subseteq X$ and (S_i, S_j) is an (i,j)-separator for X, then $C_i \subseteq S_i$ but $C_j \cap S_i = \emptyset$. Thus, if we could compute an (i,j)-separator for all $j \neq i$, then we could compute C_i by a simple set intersection. Unfortunately, it is not clear how to compute (S_i, S_j) for an arbitrary j, even given the seed node s_j. However, as we shall see, we can compute (S_i, S_j) if we know a cut edge (u_i, u_j) between C_i and C_j. The trick is to take each $x \in X$ and look at its distance $d_G(x, u_i)$ from u_i. If $d_G(x, u_i) < \frac{1}{\gamma}$, then we learn whether $u_i \in C_i$ using MBS. If instead $d_G(x, u_i) \geq \frac{1}{\gamma}$, then the comparison $d_G(x, u_i) \leq d_G(x, u_j)$ tells us whether x should be in S_i or in S_j, without using any query. This is implied by geodesic convexity through the following lemma, proven in Appendix B:

Lemma 6 *Let $(u_i, u_j) \in G$ be a cut edge between C_i and C_j. For all $x \in X$ with $\frac{1}{\gamma} \leq d_G(u_i, x) < \infty$, if $d_G(u_i, x) \leq d_G(u_j, x)$ then $x \notin C_j$, and if $d_G(u_i, x) \geq d_G(u_j, x)$ then $x \notin C_i$.*

The intuition is that $d_G(u_i, x) \leq d_G(u_j, x)$ and $x \in C_j$ cannot both hold, because this would violate the geodesic convexity of C_j, and the same holds when i and j are exchanged.

The above discussion leads to ClusterSeparator (Algorithm 2), whose correctness and query cost are proven in Lemma 7. Clearly enough, to use ClusterSeparator we need to compute the cut edge (u_i, u_j), and we show how to do so in the next sections.

```
Algorithm 2: ClusterSeparator($G, u_i, u_j$)
1 $Z := \{x \in V(G) : d_G(u_i, x) < 1/\gamma\}$
2 $Z_i :=$ MBS$(Z, \varepsilon, u_i)$,    $Z_j := Z \setminus Z_i$
3 $U_i := \emptyset$,    $U_j := \emptyset$
4 for each $x \in V(G) \setminus Z$ do
5     if $d_G(x, u_i) \leq d_G(x, u_j)$ then add $x$ to $U_i$ else add $x$ to $U_j$
6 return $(Z_i \cup U_i,\ Z_j \cup U_j)$
```

Lemma 7 *Suppose $(u_i, u_j) \in G$ is a cut edge between C_i and C_j. Then,* ClusterSeparator(G, u_i, u_j) *returns a pair (S_i, S_j) that is an (i,j)-separator of $V(G)$, using $\mathcal{O}(\mathcal{M}^*(\beta\gamma))$* SCQ *queries.*

Proof The query bound follows from Lemma 3 by observing that $Z \subseteq B(u_i, \varepsilon/\gamma)$. For the correctness, we show that (Z_i, Z_j) is an (i,j)-separator of Z and (U_i, U_j) is an (i,j)-separator of $V(G) \setminus Z$. For Z, Lemma 3 guarantees that $Z_i = C_i \cap Z$, and the algorithm sets $Z_j = Z \setminus Z_i$. So $Z_i \cap C_j = Z_j \cap C_i = \emptyset$, and (Z_i, Z_j) is an (i,j)-separator of Z. Consider now any $x \in V(G) \setminus Z$. By definition of Z, we have $d_G(x, u_i) \geq \frac{1}{\gamma}$. Therefore, by Lemma 6, if the algorithm assigns x to U_i then $x \notin C_j$, and if the algorithm assigns x to U_j then $x \notin C_i$. Therefore $U_i \cap C_j = U_j \cap C_i = \emptyset$, and (U_i, U_j) is an (i,j)-separator of $V(G) \setminus Z$. ■

4.3. Recovering a single cluster

We can now describe RecoverSingleCluster (Algorithm 3). The algorithm starts by computing R_i, the set of nodes reachable from s_i in G, and the corresponding induced subgraph $G_i = G[R_i]$. Note that, by the connectedness of the clusters, initially R_i is simply the union of C_i and zero or more other clusters. Now, we search for some seed node $s_h \in \boldsymbol{s}$, such that $s_h \in R_i$ but $s_h \neq s_i$. If no such node is found, then $R_i = C_i$ and we are done. Otherwise, we compute the shortest path π between s_h and s_i in G_i. By Lemma 4, π is C_i-prefixed, and so by Observation 1 we can find a cut edge (u_i, u_j) of C_i with $\mathcal{O}(\log n)$ SCQ queries. With the cut edge (u_i, u_j), we can compute an (i, j)-separator (S_i, S_j) of $X = V(G)$ using ClusterSeparator. Finally, we update G_i to be the connected component of s_i in $G[R_i \cap S_i]$, and R_i to be its node set. By definition of (S_i, S_j), this update removes from G_i all points of C_j, so we have reduced by at least one the number of clusters other than C_i intersected by R_i. After at most $k-1$ of these rounds, we will be left with $R_i = C_i$.

Unfortunately, this process has a problem: we can run out of seeds in R_i. Indeed, by taking $R_i \cap S_i$, we could remove every seed node s_h, even if $R_i \cap S_i$ still contains points of C_h. If this is the case, then, even though $R_i \neq C_i$, RecoverSingleCluster would be stuck, unable to compute a new cut edge to remove some cluster from R_i. One is tempted to ignore the fact that $s_h \notin R_i$, and simply compute the shortest path between s_h and s_i for all $h \neq i$, obtaining a set of different cut edges. This however does not work, as all those shortest paths could use the same cut edge (u_i, u_j).

We bypass this obstacle by exploiting the separators found by RecoverSingleCluster. By carefully analysing the cuts induced by those separators, we devise an algorithm, FindNewSeed, that either finds some new seed $s_h \in R_i \setminus C_i$ or certifies that $R_i = C_i$. FindNewSeed is invoked by RecoverSingleCluster at the beginning of each round, and we describe it in Section 4.4.

Algorithm 3: RecoverSingleCluster$(G, \varepsilon, \boldsymbol{s}, i)$

```
1  $G_i :=$ {the connected component of $s_i$ in $G$},  $R_i := V(G_i)$
2  $\boldsymbol{u} :=$ an empty vector
3  while true do
4      $s_h := \mathrm{FindNewSeed}(G, R_i, \varepsilon, \boldsymbol{s}, \boldsymbol{u}, i)$
5      if $s_h =$ NIL then stop and return $R_i$
6      $\pi(s_i, s_h) :=$ ShortestPath$(G[R_i], s_i, s_h)$
7      $(u_i, u_j) :=$ FindCutEdge$(\pi(s_i, s_h))$
8      add $u_i$ to $\boldsymbol{u}$
9      $(S_i, S_j) :=$ ClusterSeparator$(G, u_i, u_j)$
10     $G_i :=$ {the connected component of $s_i$ in $G[R_i \cap S_i]$},  $R_i := V(G_i)$
```

Lemma 8 RecoverSingleCluster$(G, \varepsilon, \boldsymbol{s}, i)$ *returns* C_i *using* $\mathcal{O}\big(k \log n + k\, \mathcal{M}^*(\frac{\beta\gamma}{2+\gamma})\big)$ SCQ *queries.*

The proof is along the lines of the discussion above, see Appendix B.

4.4. Finding new seed nodes

We describe FindNewSeed, which finds a node $s_h \in R_i \setminus C_i$ if $R_i \setminus C_i \neq \emptyset$, and otherwise detects that $R_i = C_i$ and returns NIL. The key idea behind FindNewSeed is the following: if R_i does not contain any seed $s_h \in \boldsymbol{s}$ other than s_i, then for each $h \neq i$ either $C_h \cap R_i = \emptyset$, or, by the connectedness of $G[C_h]$, the cut $\Gamma(R_i)$ must contain some edge of $G[C_h]$. Therefore, the task boils down to finding such an edge, or deciding that no such edge exists. Clearly, we cannot just check all edges in $\Gamma(R_i)$, as this would require too many queries. Thus, we proceed as follows.

Consider the beginning of a generic iteration of RecoverSingleCluster, and let $\boldsymbol{u}$ be the set of all nodes u_i that appeared in a cut edge used in some previous iteration. First, for every $u \in \boldsymbol{u}$, we consider the set Z of nodes $x \in R_i$ such that $d_G(u,x) < \frac{2}{\gamma}+1$. Then, like we did in ClusterSeparator, we use MBS to recover efficiently the subset $Z \setminus C_i$. If this subset is nonempty, then we just return any $x \in Z \setminus C_i$ and we are done. If after considering every $u \in \boldsymbol{u}$ we have not found a seed, then we turn to the remaining nodes, that is, all nodes x such that $d_G(u,x) \geq \frac{2}{\gamma}+1$ for all $u \in \boldsymbol{u}$. In this case, as a consequence of geodesic convexity with margin we prove the following structural result: if x has an edge $(x,y) \in \Gamma(R_i)$, then $x \notin C_i$. So, if any such x exists, which we can check without making any query, then we can again return x. If both attempts to find $x \in R_i \setminus C_i$ fail, we can show that necessarily $R_i = C_i$.

Lemma 9 below states the guarantees of FindNewSeed. Its proof is found in Appendix B.

Algorithm 4: FindNewSeed($G, R_i, \varepsilon, \boldsymbol{s}, \boldsymbol{u}, i$)

```
1 if $\boldsymbol{s} \cap R_i \neq \{s_i\}$ then return any $s \in \boldsymbol{s} \cap R_i \setminus \{s_i\}$
2 for each $u \in \boldsymbol{u}$ do
3     $Z = \{x \in R_i : d_G(u,x) < 2/\gamma + 1\}$
4     $Z_i :=$ MBS$(Z, \varepsilon, u)$
5     if $Z \setminus Z_i \neq \emptyset$ then return any $x \in Z \setminus Z_i$
6 if $\exists\,(x,y) \in \Gamma(R_i)$ such that $\forall u \in \boldsymbol{u} : d_G(u,x) \geq 2/\gamma + 1$ then return any such $x$
7 else return NIL
```

Lemma 9 *Consider the beginning of any iteration of* RecoverSingleCluster$(G, \varepsilon, \boldsymbol{s}, i)$. *Then, the call to* FindNewSeed$(G, R_i, \varepsilon, \boldsymbol{s}, \boldsymbol{u})$ *returns a point* $x \in R_i \setminus C_i$ *if* $R_i \neq C_i$, *or* NIL *if* $R_i = C_i$, *using at most* $k\,\mathcal{M}^*(\frac{\beta\gamma}{2+\gamma})$ SCQ *queries. Moreover,* FindNewSeed *can be adapted so as to make at most* $k\,\mathcal{M}^*(\frac{\beta\gamma}{2+\gamma})$ SCQ *queries over the entire execution of* RecoverSingleCluster.

5. Extension to clusters with different radii

In this section we generalize the notion of (β,γ)-convexity (Definition 1) so to allow each cluster C_i to have its own radius, denoted by ε_i. Then, by using SEED queries, we will extend our cluster recovery algorithm to this generalized setting. For technical reasons, we need to strengthen geodesic convexity in a hereditary fashion.

Definition 10 (generalized (β,γ)-convex clustering) *For any* $\beta,\gamma \in (0,1]$, *a* k*-clustering* $\mathcal{C} = (C_1,\dots,C_k)$ *of* X *is* (β,γ)*-convex if* $\forall\, i \in [k] : \exists\, \varepsilon_i > 0$ *such that the following properties hold:*

1. connectedness*: the subgraph induced by* C_i *in* $G_X(\varepsilon_i)$ *is connected*
2. local metric margin*: for all* $x,y \in X$*, if* $x \in C_i$ *and* $y \notin C_i$*, then* $d(x,y) > \beta\varepsilon_i$
3. geodesic convexity with margin*: for any* $\varepsilon \leq \varepsilon_i$*, if* $x,y \in C_i$ *and* $d_{G_X(\varepsilon)}(x,y) < \infty$*, then in* $G_X(\varepsilon)$ *any simple path between* x *and* y *of length at most* $(1+\gamma)d_{G_X(\varepsilon)}(x,y)$ *lies entirely in* C_i

In Lemma 20 in the Appendix, we show that if the conditions above are satisfied, then they are satisfied in particular by the smallest ε_i such that C_i is connected in $G_X(\varepsilon_i)$.[7] Therefore, without loss of generality, we assume that each ε_i satisfies this minimality assumption.

We turn to the recovery of $\mathcal{C}$. We show that, with some care, the problem can be reduced to the case of identical radii, at the price of making $\mathcal{O}(k^2)$ SEED queries. This gives an algorithm RecoverClustering2 with the following guarantees, where $\boldsymbol{\varepsilon} = (\varepsilon_1, \dots, \varepsilon_k)$ is the vector of the radii:

Theorem 11 *Suppose $\mathcal{C}$ is (β, γ)-convex (Definition 10). Then,* RecoverClustering2$(X, \boldsymbol{\varepsilon}, \boldsymbol{s})$ *deterministically returns $\mathcal{C}$, has the same runtime as* RecoverClustering$(X, \boldsymbol{\varepsilon}, \boldsymbol{s})$*, and uses the same number of* SCQ *queries as* RecoverClustering$(X, \boldsymbol{\varepsilon}, \boldsymbol{s})$*, plus at most $\mathcal{O}(k^2)$* SEED *queries.*

The basic idea of RecoverClustering2 is to invoke RecoverSingleCluster for each $i \in [k]$, as we did for the case of identical radii, using the graph $G_X(\varepsilon_i)$ when we want to recover C_i. This does not work straight away, however: in $G_X(\varepsilon_i)$, any cluster C_j with $\varepsilon_j < \varepsilon_i$ is by definition not required to satisfy geodesic convexity, which means that RecoverSingleCluster can fail. However, we can show that this approach works if we adopt the following precautions:

1. recover the clusters in nondecreasing order of radius
2. when recovering C_i, restrict $G_X(\varepsilon_i)$ to its connected component containing s_i
3. after recovering C_i, delete it from X.

Note that, for each i, this procedure works on a potentially different graph — thresholded at a different radius, and containing only a subset of the original points. Thus, its correctness may not be obvious. In particular, it may not be obvious that the clustering induced by the sub-instance used at the i-th iteration is (β, γ)-convex, which is necessary for RecoverSingleCluster to work. We show that it is: for every i, at the i-th iteration, the residual clustering is (β, γ)-convex. Thus the input to RecoverSingleCluster satisfies the hypotheses of Lemma 8, and by that lemma, RecoverSingleCluster will return C_i using $\mathcal{O}\big(k \log n + k\, \mathcal{M}^*(\frac{\beta\gamma}{2+\gamma})\big)$ SCQ queries, as desired. In all this, the role of SEED queries is to find one seed node s_h for each cluster in the connected component of $G_X(\varepsilon_i)$ containing s_i, as required by RecoverSingleCluster.

The formal construction of RecoverClustering2 and the proof of Theorem 11 are given in Appendix C.1.

6. Learning the radii and the convexity parameters

In this section we show how to use SEED queries to learn the cluster radii and to deal with the case where one of β or γ is unknown. For learning the radii, we prove:

Theorem 12 *Suppose $\mathcal{C}$ is (β, γ)-convex (Definition 10). Then, the cluster radii $\varepsilon_1, \dots, \varepsilon_k$ can be learned using $\mathcal{O}(k \log n)$* SEED *queries in time $\mathcal{O}(m\, \alpha(m, n) + kn \log n)$, where $\alpha(m, n)$ is the functional inverse of the Ackermann function.*[8]

7. Note that this is different from requiring that $G_X(\varepsilon_i)[C_i]$ is connected; here we are only requiring that any two points of C_i have a connecting path in $G_X(\varepsilon_i)$. Lemma 20 however shows that the smallest ε_i that satisfies one condition is also the smallest ε_i that satisfies the other condition.

8. For all practical purposes, α can be considered constant. For instance, $\alpha(m, m) \le 4$ for all $m \le \frac{1}{8} 2^{2^{2^{65536}}}$.

This result hinges on three observations. First, as said above, ε_i is actually the smallest ε such that all nodes of C_i belong to a single connected component of $G_X(\varepsilon)$. Second, with $\mathcal{O}(1)$ SEED queries, we can test whether C_i belongs to a single connected component of G, for any given graph G, see Claim 1. Third, if T is any minimum spanning tree of $\mathcal{G}$, then the connected components of $G_X(\varepsilon)$ are exactly the connected components of $T(\varepsilon)$, see Claim 2. Our algorithm starts by computing T, which takes time $\mathcal{O}(m\,\alpha(m,n))$ where α is inverse Ackermann, see (Chazelle, 2000). Then, for each cluster C_i, we perform a binary search to find the smallest edge weight ε such that C_i is connected in $T(\varepsilon)$. All details are given in Appendix D. In Section 8, we also prove a nearly-matching lower bound of $\Omega(k \log \frac{n}{k})$ queries.

We conclude by discussing the case where one among β and γ is unknown. Equipped with the SEED queries, we make a series of halving guesses for β or γ, until we detect that the clustering is correct. This yields the following result (proof in Appendix D):

Theorem 13 *Suppose $\mathcal{C}$ is (β,γ)-convex (Definition 10), and let $R = \log(\frac{4}{\beta\gamma})\,\mathrm{dens}(X)$. If only one between β and γ is unknown, then we can recover $\mathcal{C}$ with a multiplicative overhead of $\mathcal{O}(R)$ in both query cost and running time, plus $\mathcal{O}(k^2R)$ SCQ queries and $\mathcal{O}(kR)$ SEED queries. This applies to each one of our algorithms (i.e., with radii that are identical or not, known or unknown).*

7. Bounds on the running time

All our algorithms admit efficient implementations, with a running time linear in the size of $\mathcal{G}$ (or essentially linear, see Section 6). Here, we give a quick overview of these implementations; for a more complete discussion, see Appendix E.

Recall that our input is the weighted graph $\mathcal{G} = (X,\mathcal{E},d)$, where $(u,v) \in \mathcal{E}$ if and only if $d(u,v) < \infty$. Recall also that $n = |X|$ and $m = |\mathcal{E}|$. We assume that d can be accessed in constant time, which can be achieved with a hash map, whose construction takes time $O(m)$ (Fredman et al., 1982). We further assume that, for any graph G and for any $x \in V(G)$, the edges incident to x can be listed in constant time per edge. Under these assumptions, the following basic fact holds:

Observation 2 *Given $\mathcal{G}$, for any ε, the graph $G_X(\varepsilon)$ can be computed in time $\mathcal{O}(n+m)$. This holds in general for thresholding any subgraph of $\mathcal{G}$.*

Using Observation 2, we can easily implement RecoverSingleCluster so that it runs in time $\mathcal{O}(k^2(n+m))$: essentially, for $\mathcal{O}(k^2)$ times the algorithm computes the distances of all nodes of $G_X(\varepsilon)$ from some given node u, which takes time $\mathcal{O}(n+m)$ via breadth-first search. Since RecoverSingleCluster is invoked once per each cluster, this would give a total running time of $\mathcal{O}(k^3(n+m))$ for both RecoverClustering and RecoverClustering2. With some extra effort, however, we show how to adapt RecoverSingleCluster so that it runs $\mathcal{O}(k(n+m))$, by amortizing in particular the cost of its subroutine FindNewSeed. In the end, we obtain a total running time of $\mathcal{O}(k^2(n+m))$ for both our cluster recovery algorithms, RecoverClustering and RecoverClustering2.

8. Lower Bounds

In this section we show that some of our parameters and assumptions are necessary, and in particular: (1) in general, to recover a (β,γ)-convex clustering, any algorithm needs $\Omega(2^{\mathrm{dens}(X)})$ queries; (2) to recover a (β,γ)-convex clustering without initial seed nodes, any algorithm needs $\Omega(n)$ SCQ queries;

(3) to learn the radii of k clusters, any algorithm needs $\mathcal{O}(k \log \frac{n}{k})$ SCQ and/or SEED queries. The full proofs are deferred to Appendix F.

Theorem 14 (Dependence on $\operatorname{dens}(X)$.) *Choose any $\beta, \gamma \in (0,1)$. There is a distribution of (β,γ)-convex 2-clusterings $\mathcal{C}$ (Definition 1 or Definition 10), where $n = |X| = 2^{\operatorname{dens}(X)}$ is arbitrarily large, such that any algorithm (even randomized) needs $\Omega(2^{\operatorname{dens}(X)})$ SCQ and/or SEED queries to recover $\mathcal{C}$ with constant probability. This holds even if $\beta, \gamma, \varepsilon$ are known.*

Theorem 15 (Necessity of seeds.) *Choose any $\beta, \gamma \in (0,1]$. There is a distribution of (β,γ)-convex 2-clusterings $\mathcal{C}$ (Definition 1 or Definition 10), where $X \subseteq \mathbb{R}^2$ with $n = |X|$ arbitrarily large and d the Euclidean distance, such that any algorithm (even randomized) needs $\Omega(n)$ SCQ queries to recover $\mathcal{C}$ with constant probability if no seed nodes are given. This holds even if $\gamma, \alpha, \varepsilon$ are known.*

Theorem 16 (Cost of learning the radii.) *For any $k \geq 2$, and any sufficiently large n, there exists a distribution of $(1/2, 1)$-convex k-clusterings (Definition 1 or Definition 10) on n points such that any algorithm (even randomized) needs $\Omega(k \log \frac{n}{k})$ SEED and/or SCQ queries to learn the radii of all clusters with constant probability.*

Acknowledgments

The authors gratefully acknowledge partial support by the Google Focused Award "Algorithms and Learning for AI" (ALL4AI). Nicolò Cesa-Bianchi is also supported by the MIUR PRIN grant Algorithms, Games, and Digital Markets (ALGADIMAR) and by the EU Horizon 2020 ICT-48 research and innovation action under grant agreement 951847, project ELISE (European Learning and Intelligent Systems Excellence).

References

Peyman Afshani, Ehsan Chiniforooshan, Reza Dorrigiv, Arash Farzan, Mehdi Mirzazadeh, Narges Simjour, and Hamid Zarrabi-Zadeh. On the complexity of finding an unknown cut via vertex queries. In *Proc. of COCOON*, pages 459–469. Springer, 2007.

Hassan Ashtiani, Shrinu Kushagra, and Shai Ben-David. Clustering with same-cluster queries. In *Advances in Neural Information Processing Systems 29*, pages 3216–3224, 2016.

Josh Attenberg and Foster Provost. Why label when you can search? Alternatives to active learning for applying human resources to build classification models under extreme class imbalance. In *Proc. of ACM KDD*, page 423–432, 2010.

Maria Florina Balcan and Steve Hanneke. Robust interactive learning. In *Proc. of COLT*, volume 23, pages 20.1–20.34, 2012.

Maria-Florina Balcan and Phil Long. Active and passive learning of linear separators under log-concave distributions. In *Proc. of COLT*, pages 288–316, 2013.

Alina Beygelzimer, Daniel J Hsu, John Langford, and Chicheng Zhang. Search improves label for active learning. In *Advances in Neural Information Processing Systems*, volume 29, 2016.

Marco Bressan, Nicolò Cesa-Bianchi, Silvio Lattanzi, and Andrea Paudice. Exact recovery of mangled clusters with same-cluster queries. In *Advances in Neural Information Processing Systems 33*, 2020.

Marco Bressan, Nicolò Cesa-Bianchi, Silvio Lattanzi, and Andrea Paudice. On margin-based cluster recovery with oracle queries. *CoRR*, abs/2106.04913, 2021. URL https://arxiv.org/abs/2106.04913.

N Cesa-Bianchi, C Gentile, F Vitale, and G Zappella. Active learning on trees and graphs. In *Proc. of COLT*, pages 320–332, 2010.

Bernard Chazelle. A minimum spanning tree algorithm with inverse-Ackermann type complexity. *J. ACM*, 47(6):1028–1047, November 2000. ISSN 0004-5411. doi: 10.1145/355541.355562.

Eli Chien, Antonia Tulino, and Jaime Llorca. Active learning in the geometric block model. In *Proc. of AAAI*, volume 34, pages 3641–3648, 2020.

Gautam Dasarathy, Robert Nowak, and Xiaojin Zhu. S2: An efficient graph based active learning algorithm with application to nonparametric classification. In *Proc. of COLT*, pages 503–522, 2015.

Scott Doyle, James Monaco, Michael Feldman, John Tomaszewski, and Anant Madabhushi. An active learning based classification strategy for the minority class problem: application to histopathology annotation. *BMC Bioinformatics*, 12(1), 2011.

Brian Eriksson, Gautam Dasarathy, Aarti Singh, and Rob Nowak. Active clustering: Robust and efficient hierarchical clustering using adaptively selected similarities. In *Proc. of AISTATS*, pages 260–268. JMLR Workshop and Conference Proceedings, 2011.

Michael L. Fredman, Janos Komlos, and Endre Szemeredi. Storing a sparse table with O(1) worst case access time. In *Proc. of IEEE SFCS*, page 165–169, 1982.

Akshay Gadde, Eyal En Gad, Salman Avestimehr, and Antonio Ortega. Active learning for community detection in stochastic block models. In *Proc. of IEEE ISIT*, pages 1889–1893. IEEE, 2016.

Lee-Ad Gottlieb and Robert Krauthgamer. Proximity algorithms for nearly doubling spaces. *SIAM Journal on Discrete Mathematics*, 27(4):1759–1769, 2013.

Lee-Ad Gottlieb, Aryeh Kontorovich, and Pinhas Nisnevitch. Nearly optimal classification for semimetrics. *The Journal of Machine Learning Research*, 18(1):1233–1254, 2017.

Andrew Guillory and Jeff Bilmes. Active semi-supervised learning using submodular functions. In *Proc. of UAI*, pages 274–282, 2011.

Steve Hanneke. *Theoretical Foundations of Active Learning*. PhD thesis, Carnegie Mellon University, USA, 2009. AAI3362265.

Akshay Krishnamurthy, Sivaraman Balakrishnan, Min Xu, and Aarti Singh. Efficient active algorithms for hierarchical clustering. In *Proc. of ICML*, pages 267–274, 2012.

Son T Mai, Xiao He, Nina Hubig, Claudia Plant, and Christian Böhm. Active density-based clustering. In *Proc. of IEEE ICDM*, pages 508–517. IEEE, 2013.

Arya Mazumdar and Barna Saha. Query complexity of clustering with side information. In *Advances in Neural Information Processing Systems 30*, pages 4682–4693, 2017.

Ignacio M. Pelayo. *Geodesic convexity in graphs*. Springer-Verlag New York, 2013. ISBN 978-1-4614-8698-5. doi: 10.1007/978-1-4614-8699-2.

Florian Seiffarth, Tamás Horváth, and Stefan Wrobel. Maximal closed set and half-space separations in finite closure systems. In *Proc. of ECML PKDD*, pages 21–37. Springer, 2019.

Ohad Shamir and Naftali Tishby. Spectral clustering on a budget. In *Proc. of AISTATS*, pages 661–669. JMLR Workshop and Conference Proceedings, 2011.

Maximilian Thiessen and Thomas Gärtner. Active learning on graphs with geodesically convex classes. In *Proc. of MLG*, 2020.

Simon Tong and Edward Chang. Support vector machine active learning for image retrieval. In *Proc. of ACM ICM*, page 107–118, 2001.

Roman Vershynin. *High-Dimensional Probability: An Introduction with Applications in Data Science*. Cambridge Series in Statistical and Probabilistic Mathematics. Cambridge University Press, 2018. doi: 10.1017/9781108231596.

Xiang Wang and Ian Davidson. Active spectral clustering. In *Proc. of IEEE ICDM*, pages 561–568. IEEE, 2010.

Pan Zhang, Cristopher Moore, and Lenka Zdeborová. Phase transitions in semisupervised clustering of sparse networks. *Physical Review E*, 90(5):052802, 2014.

Appendix A. Supplementary material for Section 3

Lemma 17 *If X is a metric space, then $\operatorname{dbl}(X) \le \operatorname{dens}(X) \le 2\operatorname{dbl}(X)$ where $\operatorname{dbl}(X)$ and $\operatorname{dens}(X)$ are respectively the doubling dimension and the density dimension of X.*

Proof Recall that $\operatorname{dbl}(X) = \log_2 D(X)$, where $D(X)$ is the doubling constant of X:

$$D(X) = \min\Big\{D \in \mathbb{N} \,:\, (x \in X) \wedge (r > 0) \Rightarrow \mathcal{N}\Big(B(x,r), \frac{r}{2}\Big) \le D\Big\} \tag{3}$$

where $\mathcal{N}(K,\eta)$ is the covering number of K, that is, the smallest number of closed balls of radius η whose union contains K. We recall from Section 3 that $\operatorname{dens}(X) = \log_2 \mu(X)$, where:

$$\mu(X) = \min\Big\{\mu \in \mathbb{N} \,:\, (x \in X) \wedge (r > 0) \Rightarrow \mathcal{M}\Big(B(x,r), \frac{r}{2}\Big) \le \mu\Big\} \tag{4}$$

Now, since we are in a metric space, we have the well-known relationship:

$$\mathcal{M}(K, 2\eta) \le \mathcal{N}(K,\eta) \le \mathcal{M}(K,\eta) \tag{5}$$

On the one hand, $\mathcal{N}(K,\eta) \le \mathcal{M}(K,\eta)$ implies $D(X) \le \mu(X)$, and therefore $\operatorname{dbl}(X) \le \operatorname{dens}(X)$. On the other hand, $\mathcal{M}(K,2\eta) \le \mathcal{N}(K,\eta)$ implies:

$$\begin{aligned}
\mathcal{M}(B(x,r), {}^{r}/_{2}) &\le \mathcal{N}(B(x,r), {}^{r}/_{4}) && (6)\\
&\le \mathcal{N}(B(x,r), {}^{r}/_{2}) \cdot D(X) && (7)\\
&\le D(X) \cdot D(X) && (8)
\end{aligned}$$

Therefore, $\mu(X) \le D(X)^2$, and $\operatorname{dens}(X) \le 2\operatorname{dbl}(X)$. ■

Appendix B. Supplementary material for Section 4

B.1. Pseudocode of FindCutEdge

Algorithm 5: FindCutEdge($\pi(s_i, s_h)$)

```
1 if $|\pi(s_i,s_h)| = 1$ then return $\pi(s_i,s_h)$
2 choose a median node $x \in \pi(s_i,s_h)$
3 if SCQ$(s_i,x) = +1$ then return FindCutEdge($\pi(x,s_h)$)
4 else return FindCutEdge($\pi(s_i,x)$)
```

B.2. Proof of Lemma 6

Lemma 6 *Let $(u_i,u_j) \in G$ be a cut edge between C_i and C_j. For all $x \in X$ with $\frac{1}{\gamma} \le d_G(u_i,x) < \infty$, if $d_G(u_i,x) \le d_G(u_j,x)$ then $x \notin C_j$, and if $d_G(u_i,x) \ge d_G(u_j,x)$ then $x \notin C_i$.*

Proof Suppose $d_G(u_i, x) \leq d_G(u_j, x)$ and $x \in C_j$; we show this leads to a contradiction. Consider the path $\pi = \pi(x, u_i) + (u_i, u_j)$. First, π is a simple path. If this was not the case, then we would have $u_j \in \pi(x, u_i)$; but this would imply $d_G(u_j, x) \leq d_G(u_i, x) - 1$, contradicting our assumption $d_G(u_i, x) \leq d_G(u_j, x)$. Second, we have:

$$\begin{aligned} |\pi| &= d_G(x, u_i) + 1 && && (9)\\ &\leq d_G(x, u_j) + 1 && \text{since } d_G(u_i, x) \leq d_G(u_j, x) && (10)\\ &\leq d_G(x, u_j)\,(1+\gamma) && \text{since } d_G(x, u_j) \geq d_G(x, u_i) \geq \frac{1}{\gamma} && (11)\end{aligned}$$

Thus, π is a simple path between two nodes of C_j, with length at most $(1+\gamma)$ times their distance in G, and containing a node of C_i. This violates the geodesic convexity of C_j. We conclude that $x \notin C_j$. The other case is symmetric. ■

B.3. Proof of Lemma 8

Lemma 8 RecoverSingleCluster$(G, \varepsilon, \boldsymbol{s}, i)$ *returns* C_i *using* $\mathcal{O}\big(k \log n + k\,\mathcal{M}^*(\frac{\beta\gamma}{2+\gamma})\big)$ SCQ *queries.*

Proof First, we prove the correctness. For each $\ell = 1, 2, \ldots$, we denote by R_i^ℓ the set R_i at the beginning of the ℓ-th iteration, and by $\kappa(R_i^\ell)$ the number of distinct clusters that R_i^ℓ intersects. We show that, at the beginning of the ℓ-th iteration, the following invariants hold:

I1. $G[R_i^\ell]$ is connected

I2. $C_i \subseteq R_i^\ell$

I3. $\kappa(R_i^\ell) \leq \kappa(R_i^1) - (\ell - 1)$

Note that I2 and I3 imply that the algorithm terminates by returning C_i (line 5) after at most k iterations. Invariant I1 holds by the construction of R_i^ℓ, so we focus on proving I2 and I3.

Suppose first $\ell = 1$. Then, I2 holds by the assumptions of the lemma, and I3 is trivial. Suppose then I1, I2, I3 hold for some $\ell \geq 1$, and that iteration $\ell + 1$ exists; we show that I2, I3 hold at iteration $\ell + 1$ as well. First, if iteration $\ell + 1$ exists, then $C_i \subsetneq R_i^\ell$. In this case, by Lemma 9, FindNewSeed will return a node $s_h \in R_i^\ell \setminus C_i$. Since $G[R_i^\ell]$ is connected by I1, the shortest path $\pi(s_i, s_h)$ exists in $G[R_i^\ell]$. Because of Lemma 4 applied to $R = R_i^\ell$, such a path is also C_i-prefixed. Therefore, FindCutEdge$(\pi(s_i, s_h))$ returns a cut edge (u_i, u_j) of C_i in $\pi(s_i, s_h)$. At this point the hypotheses of Lemma 7 are satisfied, and therefore the output (S_i, S_j) of ClusterSeparator(G, u_i, u_j) is an (i, j)-separator of $V(G)$. Hence, $C_i \subseteq S_i$ and $C_j \cap S_i = \emptyset$. Therefore, $C_i \subseteq R_i^\ell \cap S_i$. This implies that the connected component of s_i in $G[R_i^\ell \cap S_i]$ still contains C_i. The vertex set of this connected component is precisely $R_i^{\ell+1}$ (line 10). Therefore, I2 holds at iteration $\ell + 1$. Moreover, observe that $u_j \in R_i^\ell$, since $u_j \in \pi(s_i, s_h) \subseteq R_i^\ell$. Therefore $R_i^\ell \cap C_j \neq \emptyset$. However, by construction, $R_i^{\ell+1} \subseteq R_i^\ell \cap S_i$ and $S_i \cap C_j = \emptyset$ since (S_i, S_j) is an (i, j)-separator of $V(G)$. Therefore, $\kappa(R_i^{\ell+1}) \leq \kappa(R_i^\ell) - 1$. Because I3 holds at ℓ, $\kappa(R_i^{\ell+1}) \leq \kappa(R_i^1) - (\ell - 1) - 1 = \kappa(R_i^1) - ((\ell + 1) - 1)$. So, I3 holds at iteration $\ell + 1$, too.

Finally, we bound the number of queries. First, by Lemma 9, FindNewSeed makes at most $k\,\mathcal{M}^*(\frac{\beta\gamma}{2+\gamma})$ SCQ in total across all iterations. Second, FindCutEdge makes $\mathcal{O}(\log n)$ queries at each iteration, see Observation 1. Third, ClusterSeparator makes at most $\mathcal{M}^*(\beta\gamma)$ queries at each iteration, see Lemma 7. Summing the three terms we obtain the claimed bound. ■

B.4. Proof of Lemma 9

Lemma 9 *Consider the beginning of any iteration of* RecoverSingleCluster$(G, \varepsilon, \boldsymbol{s}, i)$*. Then, the call to* FindNewSeed$(G, R_i, \varepsilon, \boldsymbol{s}, \boldsymbol{u})$ *returns a point* $x \in R_i \setminus C_i$ *if* $R_i \neq C_i$*, or* NIL *if* $R_i = C_i$*, using at most* $k\,\mathcal{M}^*(\frac{\beta\gamma}{2+\gamma})$ SCQ *queries. Moreover,* FindNewSeed *can be adapted so as to make at most* $k\,\mathcal{M}^*(\frac{\beta\gamma}{2+\gamma})$ SCQ *queries over the entire execution of* RecoverSingleCluster.

We need to prove two technical results, Lemma 18 and Lemma 19. Then, we will prove Lemma 9.

Lemma 18 *Let* G *be such that* $C_i \subseteq V(G)$*, and let* (S_i, S_j) *be an* (i,j)*-separator for* $V(G)$ *obtained from* ClusterSeparator$(V(G), u_i, u_j)$*. If* $(x, y) \in \Gamma(S_i)$ *and* $d_G(u_i, x) \geq \frac{2}{\gamma} + 1$*, then* $x \notin C_i$*.*

Proof We show that $x \in C_i$ violates the geodesic convexity of C_i. First $(x, y) \in \Gamma(S_i)$ implies $y \in S_j$. Moreover, $d_G(u_i, y) \geq d_G(u_i, x) - 1 > \frac{1}{\gamma}$. Now recall the code of ClusterSeparator. Since $d_G(u_i, y) > \frac{1}{\gamma}$, then $y \notin Z_j$. But $y \in S_j = Z_j \cup U_j$, and therefore $y \in U_j$. This means that ClusterSeparator executed line 5, which happens only if:

$$d_G(u_j, y) < d_G(u_i, y) \tag{12}$$

Now consider the path $\pi = (u_i, u_j) + \pi(u_j, y) + (y, x)$ from u_i to x where $\pi(u_j, y)$ has length $d_G(u_j, y)$. First, we observe that π is a simple path. Suppose indeed by contradiction that π is not simple. Since $\pi(u_j, y)$ is simple (it is a shortest path), and since $u_i \neq x$ (because $d_G(u_i, x) \geq 1$ by assumption), we must have $u_i \in \pi(u_j, y)$ or $x \in \pi(u_j, y)$. If $u_i \in \pi(u_j, y)$, then $d_G(u_j, y) > d_G(u_i, y)$, which contradicts (12). If instead $x \in \pi(u_j, y)$, then $d_G(u_j, y) > d_G(u_j, x)$, which gives:

$$\begin{aligned} d_G(u_j, y) &> d_G(u_j, x) && && (13)\\ &\geq d_G(u_i, x) && \text{since } x \in S_i && (14)\\ &\geq d_G(u_i, y) - 1 && \text{since } (x, y) \in E(G) && (15) \end{aligned}$$

which, since d_G is integral, implies $d_G(u_j, y) \geq d_G(u_i, y)$. This contradicts again (12). Thus, π is a simple path.

Now we show that $|\pi| \leq d_G(u_i, x)\,(1+\gamma)$. From (12), we have:

$$\begin{aligned} d_G(u_j, y) &\leq d_G(u_i, y) - 1 && \text{since } d_G \in \mathbb{N} && (16)\\ &\leq d_G(u_i, x) + d_G(x, y) - 1 && \text{since } (x, y) \in E(G) && (17)\\ &= d_G(u_i, x) && && (18) \end{aligned}$$

And therefore:

$$\begin{aligned} |\pi| &= d_G(u_j, y) + 2 && && (19)\\ &\leq d_G(u_i, x) + 2 && \text{since } d_G(u_j, y) \leq d_G(u_i, x) && (20)\\ &\leq d_G(u_i, x)\,(1+\gamma) && \text{since } d_G(u_i, x) \geq \frac{2}{\gamma} && (21) \end{aligned}$$

Therefore π is a simple path between two nodes of C_i that violates the geodesic convexity of C_i. So $x \notin C_i$, as claimed. ∎

Now recall RecoverSingleCluster$(G, \varepsilon, \boldsymbol{s}, i)$. Let R_i^ℓ be the value of R_i at the beginning of the ℓ-th iteration, and let (S_i^ℓ, S_{j_ℓ}) be the separator computed by ClusterSeparator at the ℓ-th iteration (note: the first cluster is always i, but the second cluster varies with ℓ).

Lemma 19 *If $(x, y) \in \Gamma(R_i^\ell)$, then $(x, y) \in \Gamma(S_i^\tau)$ for some $\tau \in \{1, \dots, \ell - 1\}$.*

Proof Let:

$$\tau = \min \left\{1 \le t \le \ell - 1 \,:\, (x, y) \in \Gamma(R_i^{t+1})\right\} \tag{22}$$

First, we have $x \in S_i^\tau$. Indeed, by construction $R_i^{\tau+1} \subseteq R_i^\tau \cap S_i^\tau$, and we know $x \in R_i^{\tau+1}$. Second, we have $y \notin S_i^\tau$. Suppose indeed by contradiction that $y \in S_i^\tau$. Since $x \in R_i^\tau$, and since $(x, y) \notin \Gamma(R_i^\tau)$, then $y \in R_i^\tau$. Therefore, $y \in S_i^\tau \cap R_i^\tau$. So y would be connected to x in $G[S_i^\tau \cap R_i^\tau]$, and therefore we would have $y \in R_i^{\tau+1}$ as well, by construction of $R_i^{\tau+1}$ as a connected component. But then, $y \in R_i^{\tau+1}$ would imply $(x, y) \notin \Gamma(R_i^{\tau+1})$ which contradicts our hypothesis. Therefore $x \in S_i^\tau$ and $y \notin S_i^\tau$, which implies $(x, y) \in \Gamma(S_i^\tau)$, as claimed. ■

Proof [**of Lemma 9**] The first bound on the number of queries follows by Lemma 3, by observing that $Z \subseteq B(u, \varepsilon(\frac{2}{\gamma} + 1)) = B(u, \varepsilon\frac{2+\gamma}{\gamma})$. Let us now prove the correctness; the claim on the adapted version will follow afterwards.

First, suppose that $R_i = C_i$. Then $\boldsymbol{s} \cap R_i = \{s_i\}$. Moreover, at each iteration of the loop, by Lemma 3 $Z_i = Z$, so $Z \setminus Z_i = \emptyset$. Finally, no edge $(x, y) \in \Gamma(R_i)$ exists such that $d_G(u, x) \ge \frac{2}{\gamma} + 1$ for all $u \in \boldsymbol{u}$. Indeed, if such an edge (x, y) existed, by Lemma 19 we would have $(x, y) \in \Gamma(S_i)$ at some previous round of RecoverSingleCluster, which by Lemma 18 implies $x \notin C_i$ — a contradiction, since $x \in R_i = C_i$. Hence, FindNewSeed reaches the last line and returns NIL.

Suppose now that $R_i \neq C_i$. If $s_h \in R_i$ for some $s_h \neq s_i$, then FindNewSeed returns s_h at line 1. Otherwise, we must have $C_h \cap R_i \neq \emptyset$ but $C_h \nsubseteq R_i$ for some $h \neq i$. By the connectedness of C_h in G, this implies the existence of an edge $(x, y) \in \Gamma(R_i)$ with $x \in C_h$. If any such edge exists with $d_G(u, x) < \frac{2}{\gamma} + 1$ for some $u \in \boldsymbol{u}$, then line 5 will find such an x and return it. Otherwise, any such edge has $d_G(u, x) \ge \frac{2}{\gamma} + 1$ for all $u \in \boldsymbol{u}$. In this case, line 6 will return some x such that $(x, y) \in \Gamma(R_i)$ and $d_G(u, x) \ge \frac{2}{\gamma} + 1$ for all $u \in \boldsymbol{u}$, which is correct since by Lemma 19 and Lemma 18 we have $x \notin C_i$.

To make FindNewSeed use at most $k\,\mathcal{M}^*(\frac{\beta\gamma}{2+\gamma})$ queries over the whole execution of RecoverSingleCluster, we keep track of $Z \setminus Z_i$ in the following way. At each invocation, we compute the set $Z^{(u)} = \{x \in R_i \,:\, d_G(u, x) < \frac{2}{\gamma} + 1\}$, where u is the last node added to $\boldsymbol{u}$. Then we invoke MBS only on $Z^{(u)}$, obtaining $Z_i^{(u)}$, and we then add $Z^{(u)} \setminus Z_i^{(u)}$ to $Z \setminus Z_i$. Finally, we remove from $Z \setminus Z_i$ all points not in R_i. This sequence of operations costs $\mathcal{M}^*(\frac{\beta\gamma}{2+\gamma})$ SCQ queries, and the resulting set $Z \setminus Z_i$ will be exactly the one computed by FindNewSeed above. Hence the behavior of the algorithm is unchanged, but the total number of queries is at most $k\,\mathcal{M}^*(\frac{\beta\gamma}{2+\gamma})$. ■

Appendix C. Supplementary material for Section 5

C.1. Lemma 20

Lemma 20 *Let $\mathcal{C}$ be a (β, γ)-convex k-clustering of X (Definition 10), and for $i \in \{1, \dots, k\}$ let:*

$$\zeta_i = \min\{\zeta : \forall\, x, y \in C \,:\, d_{G_X(\zeta)}(x, y) < \infty\} \tag{23}$$
$$\zeta_i^* = \min\{\zeta : \rho(G_{C_i}(\zeta)) = 1\} \tag{24}$$

Then $\zeta_i = \zeta_i^*$, and all properties of Definition 10 hold when $\varepsilon_i = \zeta_i = \zeta_i^*$.

Proof First, observe that $\zeta_i \le \zeta_i^*$, since $\rho(G_{C_i}(\zeta)) = 1$ implies $\forall\, x, y \in C \,:\, d_{G_X(\zeta)}(x, y) < \infty$, and thus the minimum in (23) is taken over a superset of that of (24). Now, consider the graph $G_X(\zeta_i)$. For any two nodes $x, y \in C_i$, since $\zeta_i \le \zeta_i^*$ and $d_{G_X(\zeta_i)}(x, y) < \infty$, the geodesic convexity implies that any shortest path in $G_X(\zeta_i)$ between x and y lies in C_i. But this means that the subgraph induced by C_i in $G_X(\zeta_i)$ is connected. By definition of ζ_i^* this implies that $\zeta_i^* \le \zeta_i$. We conclude that $\zeta_i = \zeta_i^*$.

For the second claim, we consider each property in turn when $\varepsilon_i = \zeta_i^*$. The connectedness of $G_{C_i}(\zeta_i^*)$ holds by definition of ζ_i^*. Now let ζ be any value such that the three properties hold when $\varepsilon_i = \zeta$. Then $\zeta \ge \zeta_i^*$, because for $\varepsilon_i < \zeta_i^*$ the connectedness fails, by definition of ζ_i^*. This implies that the local metric margin and the geodesic convexity with margin hold for $\varepsilon_i = \zeta_i^*$, since they hold for $\varepsilon_i = \zeta \ge \zeta_i^*$. ■

C.2. Pseudo-code of RecoverClustering2 and proof of Theorem 11

Theorem 11 *Suppose $\mathcal{C}$ is (β, γ)-convex (Definition 10). Then,* RecoverClustering2$(X, \boldsymbol{\varepsilon}, \boldsymbol{s})$ *deterministically returns $\mathcal{C}$, has the same runtime as* RecoverClustering$(X, \boldsymbol{\varepsilon}, \boldsymbol{s})$*, and uses the same number of* SCQ *queries as* RecoverClustering$(X, \boldsymbol{\varepsilon}, \boldsymbol{s})$*, plus at most $\mathcal{O}(k^2)$* SEED *queries.*

To prove the theorem, we need a technical lemma. It guarantees that, if we take the connected component of the cluster with smallest radius ε^* in $G_X(\varepsilon^*)$, then the clustering induced by $\mathcal{C}$ on that subgraph is (β, γ)-convex (Definition 1) with radius ε^*.

Lemma 21 *Let $\mathcal{C}$ be a (β, γ)-convex κ-clustering of X (Definition 10). Let ε^* be the smallest radius of any cluster of $\mathcal{C}$, and let G^* be the connected component of $G_X(\varepsilon^*)$ that contains a cluster with radius ε^*. Finally, let $X^* = V(G^*)$, and let $\mathcal{C}^* = \big(C_1 \cap X^*, \dots, C_\kappa \cap X^*\big)$. Then, $\mathcal{C}^*$ is a (β, γ)-convex clustering of X^* with radius ε^* (Definition 1).*

Proof We verify that the three properties of Definition 1 hold with radius ε^*. This implies that ε^* is also the smallest such value, since the corresponding cluster becomes disconnected in $G^*(\varepsilon)$ for any $\varepsilon < \varepsilon^*$ — and thus ε^* is indeed the radius of the clustering. We define $C_i^* = C_i \cap X^*$ for all $i \in [\kappa]$. Note that the graph on which we verify the properties is $G_{X^*}(\varepsilon^*)$, which is precisely G^* by the maximality of X^* as a connected component. Hence, from now on we write G^* for $G_{X^*}(\varepsilon^*)$.

Connectivity: the subgraph induced by C_i^* in G^* is connected.
Proof. Consider two points $x, y \in C_i^*$; obviously $x, y \in C_i$. Since G^* is connected, $d_{G^*}(x, y) < \infty$. Moreover, $d_{G^*}(x, y) = d_{G_X(\varepsilon^*)}(x, y)$, since by construction G^* is the connected component of

$G_X(\varepsilon^*)$ containing x and y. Thus, $d_{G_X(\varepsilon^*)}(x,y) < \infty$. Moreover, $\varepsilon^* \le \varepsilon_i$ by assumption. Thus, $x, y \in C_i$, and $d_{G_X(\varepsilon^*)}(x,y) < \infty$ with $\varepsilon^* \le \varepsilon_i$. Then, by the geodesic convexity of $\mathcal{C}$ on X (Definition 10), any shortest path π between x and y in $G_X(\varepsilon^*)$ lies entirely in C_i. Since again G^* is the connected component of $G_X(\varepsilon^*)$ containing x, y, then π must lie in X^*. We conclude that $\pi \in C_i \cap X^* = C_i^*$. This holds for any choice of x, y. Therefore, $G^*[C_i^*]$ is connected.

Local metric margin: for all $x, y \in X^*$, if $x \in C_i^*$ and $y \notin C_i^*$, then $d(x,y) > \beta\varepsilon^*$.
Proof. Since $x \in C_i^*$ and $y \notin C_i^*$, then $x \in C_i$ and $y \notin C_i$. By the local metric margin of $\mathcal{C}$, and since $\varepsilon_i \ge \varepsilon^*$, we have $d(x,y) > \beta\varepsilon_i \ge \beta\varepsilon^*$.

Geodesic convexity with margin: if $x, y \in C_i^*$, then in G^* any simple path between x and y of length at most $(1+\gamma)d_{G^*}(x,y)$ lies entirely in C_i^*.
Proof. By the same argument of connectivity, we invoke the geodesic convexity (Definition 10) for $x, y \in C_i$, for $\varepsilon = \varepsilon^* \le \varepsilon_i$. We obtain that $G_X(\varepsilon^*)$ contains no simple path of length at most $(1+\gamma)d_{G_X(\varepsilon^*)}(x,y)$ between x and y that leaves C_i. This implies that no such path exists in G^* as well, since $G^* \subseteq G_X(\varepsilon^*)$. Moreover, since $C_i^* = C_i \cap X^*$, no such path exists in G^* that leaves C_i^* (otherwise it would leave C_i). Recalling that $(1+\gamma)d_{G_X(\varepsilon^*)}(x,y) = (1+\gamma)d_{G^*}(x,y)$, we deduce that in G^* there is no path of length at most $(1+\gamma)d_{G^*}(x,y)$ between x and y that leaves C_i^*. This is the geodesic convexity of C_i^* in G^* (Definition 1). ■

We can now present the algorithm for recovering (β,γ)-convex clusters with different radii.

Algorithm 6: RecoverClustering2($X, \boldsymbol{\varepsilon}, \boldsymbol{s}$)

```
1 assume $\varepsilon_1 \le \ldots \le \varepsilon_k$
2 for $i = 1, \ldots, k$ do
3     $G^*$ := the connected component of $s_i$ in $G_X(\varepsilon_i)$,   $X^* := V(G^*)$
4     for $j = i+1, \ldots, k$ do $s_j^* :=$ SEED$(X^*, j)$
5     $\boldsymbol{s}^* := (s_i, s_{i+1}^*, \ldots, s_k^*)$
6     $\widehat{C_i}$ := RecoverSingleCluster$(G^*, \varepsilon_i, \boldsymbol{s}^*, i)$
7     output $\widehat{C_i}$
8     $X := X \setminus \widehat{C_i}$
```

Proof [of Theorem 11] For each $i = 1, \ldots, k$ let X_i be the value of X at the beginning of the i-th iteration of RecoverClustering2($X, \boldsymbol{\varepsilon}, \boldsymbol{s}$). We show that the following three invariants holds:

1. $X_i = C_i \cup \ldots \cup C_k$
2. $(C_i, \ldots, C_k)$ is a (β,γ)-convex clustering of X_i (Definition 10)
3. if $i > 1$ then the algorithm has output $C_1, \ldots, C_{i-1}$ so far

When $i = 1$ we have $X_i = X$, and all invariants clearly hold. Now assume that they hold at the beginning of the i-th iteration for some $i \ge 1$. We will show that the algorithm sets $\widehat{C_i} = C_i$. This will imply that the three invariants hold at iteration $i+1$ as well. For the first and third invariant, this is trivial. For the second, simply observe that deleting a cluster never invalidates the three properties of Definition 10; thus, $(C_{i+1}, \ldots, C_k)$ will be a (β,γ)-convex clustering of $X_{i+1} = C_{i+1} \cup \ldots \cup C_k$.

Thus, we prove that $\widehat{C_i} = C_i$. To this end, consider the subgraph G^* and its node set V^* computed at line 3. Let $\mathcal{C}_i = (C_i, \dots, C_k)$, and let $\mathcal{C}_i^* = (C_i \cap X^*, \dots, C_k \cap X^*)$. Since ε_i is the smallest radius of all clusters in $\mathcal{C}_i$, then by Lemma 21 with $\varepsilon^* = \varepsilon_i$, $\mathcal{C}_i^*$ is a (β,γ)-convex clustering for X_i with radius ε_i (Definition 1). Furthermore, by construction $\boldsymbol{s}^*$ contains one seed for each nonempty cluster in $\mathcal{C}^*$. Therefore, by Lemma 8, RecoverSingleCluster$(G^*, \varepsilon, \boldsymbol{s}^*, i)$ returns C_i, so $\widehat{C_i} = C_i$.

The bound on the number of queries is straightforward. ■

Appendix D. Supplementary material for Section 6

D.1. GetEpsilons and proof of Theorem 12

Theorem 12 *Suppose $\mathcal{C}$ is (β,γ)-convex (Definition 10). Then, the cluster radii $\varepsilon_1, \dots, \varepsilon_k$ can be learned using $\mathcal{O}(k \log n)$* SEED *queries in time $\mathcal{O}(m\,\alpha(m,n) + kn \log n)$, where $\alpha(m,n)$ is the functional inverse of the Ackermann function.*[9]

We start with a simple routine for testing the connectedness of a cluster using SEED queries.

Algorithm 7: IsConnected(G, i)

```
1 $u :=$ SEED$(V(G), i)$
2 $U :=$ the connected component of $u$ in $G$
3 return (SEED$(V(G) \setminus U, i) =$ NIL)
```

Claim 1 *If $V(G) \cap C_i \neq \emptyset$, then* IsConnected$(G, i)$ *uses two* SEED *queries and returns* TRUE *if and only if $d_G(x,y) < \infty$ for all $x, y \in C_i$.*

Let T be a minimum spanning tree of the weighted graph $\mathcal{G}$. For any $\varepsilon > 0$, let $T(\varepsilon)$ be the forest obtained by keeping only the edges (x,y) of T such that $d(x,y) \le \varepsilon$. Recall the following basic fact:

Claim 2 *The connected components of $T(\varepsilon)$ are the connected components of $G_X(\varepsilon)$.*

As a consequence, we have:

Claim 3 IsConnected$(T(\varepsilon), i) =$ IsConnected$(G_X(\varepsilon), i)$, *for any $i \in [k]$ and any $\varepsilon > 0$.*

We introduce the algorithm for learning the radius of a single cluster.

Lemma 22 *If T is a MST of $\mathcal{G} = (X, \mathcal{E}, d)$, then* GetEpsilon$(T, i)$ *returns ε_i in time $\mathcal{O}(n \log n)$ using $\mathcal{O}(\log n)$* SEED *queries.*

Proof It is straightforward to see that the algorithm stops within $\mathcal{O}(\log n)$ iterations, since $\boldsymbol{w}$ has at most $m = \mathcal{O}(n^2)$ entries and $(hi - lo)$ decreases by a constant factor at each iteration. For the running time, since T has $\mathcal{O}(n)$ edges, every call to IsConnected$(T(w_{mid}), i)$ takes time $\mathcal{O}(n)$. This gives the time bound of $\mathcal{O}(n \log n)$.

9. For all practical purposes, α can be considered constant. For instance, $\alpha(m,m) \le 4$ for all $m \le \frac{1}{8}2^{2^{2^{65536}}}$.

Algorithm 8: GetEpsilon(T, i)

```
1 $\boldsymbol{w} := (w_0, w_1, \ldots, w_\ell)$, the distinct edge weights of $T$ in increasing order, with $w_0 = 0$
2 $lo := 0, \quad hi := \ell$
3 while $w_{lo} < w_{hi}$ do
4     $mid := \lfloor \frac{lo+hi}{2} \rfloor$
5     if IsConnected$(T(w_{mid}), i)$ then $hi := mid$ else $lo := mid + 1$
6 return $w_{hi}$
```

Now we show that the algorithm returns ε_i. By Lemma 20, this is equivalent to prove that the algorithm returns $w^* = \min\{w \in \boldsymbol{w} : C_i \text{ is connected in } G_X(w)\}$. Consider the beginning of a generic iteration, when the test $w_{lo} < w_{hi}$ is performed. We claim that $w^* \le w_{hi}$. To this end, observe that C_i is connected in $G_X(w_{hi})$. This is true since it holds at the beginning of the first iteration, when $w_{hi} = w_\ell$, and because at each iteration hi is set to mid only if IsConnected$(T(w_{mid}), i) =$ TRUE. We now claim that $w^* \ge w_{lo}$. This holds since $w^* \ge w_0 = 0$ at the first iteration, and because at each iteration lo is set to $mid + 1$ only if IsConnected$(T(w_{mid}), i) =$ FALSE. Therefore, when the algorithm stops, we have $w_{lo} = w_{hi} = w^*$, as claimed. ■

We conclude with the algorithm to learn all the radii. We denote by $\mathrm{MST}(m)$ the time needed for computing the MST of a connected graph (note that we can always assume $\mathcal{G}$ is connected, otherwise we can just compute its connected components in time $\mathcal{O}(m)$ and use each one of them in turn). It is known that $\mathrm{MST}(m) = O(m\,\alpha(m,m))$, where $\alpha(m,m)$ is the classic functional inverse of Ackermann's function (Chazelle, 2000). Lemma 23 below follows immediately from these observations.

Algorithm 9: GetEpsilons$(\mathcal{G} = (X, \mathcal{E}, d), k)$

```
1 $T := \mathrm{MST}(\mathcal{G})$
2 for $i = 1, \ldots, k$ do
3     $\widehat{\varepsilon_i} :=$ GetEpsilon$(T, i)$
4 return $\widehat{\varepsilon_1}, \ldots, \widehat{\varepsilon_k}$
```

Lemma 23 GetEpsilons$(\mathcal{G}, k)$ *returns the radii* $\varepsilon_1, \ldots, \varepsilon_k$ *in time* $\mathcal{O}(m\,\alpha(m,n) + kn\log n)$ *using* $\mathcal{O}(k\log n)$ SEED *queries.*

D.2. Proof of Theorem 13

Theorem 13 *Suppose* $\mathcal{C}$ *is* (β,γ)*-convex (Definition 10), and let* $R = \log(\frac{4}{\beta\gamma})\,\mathrm{dens}(X)$*. If only one between* β *and* γ *is unknown, then we can recover* $\mathcal{C}$ *with a multiplicative overhead of* $\mathcal{O}(R)$ *in both query cost and running time, plus* $\mathcal{O}(k^2 R)$ SCQ *queries and* $\mathcal{O}(kR)$ SEED *queries. This applies to each one of our algorithms (i.e., with radii that are identical or not, known or unknown).*

Proof Suppose first β is unknown and γ is known. Recall that $\beta \le 1$. We make a succession of guesses $\widehat{\beta} = 2^{-j}$ for $j = 0, 1, \ldots$. For each guess, we run our algorithm with $\beta = \widehat{\beta}$ and look at the output clustering $\widehat{\mathcal{C}}$. Clearly, if $\mathcal{C}$ is (β,γ)-convex, then $\mathcal{C}$ is $(\widehat{\beta},\gamma)$-convex for any $\widehat{\beta} \le \beta$ as well.

Thus, as soon as $\widehat{\beta} \leq \beta$, our algorithm will return $\widehat{\mathcal{C}} = \mathcal{C}$. So, after each run, we need only to check whether $\widehat{\mathcal{C}} = \mathcal{C}$, and stop in the affirmative case.

To check whether $\widehat{\mathcal{C}} = \mathcal{C}$, we do as follows. First, we check if $|\widehat{\mathcal{C}}| \neq |\mathcal{C}|$. If this is the case, then the only possibility for $\widehat{\mathcal{C}} \neq \mathcal{C}$ is that some cluster C_i intersects both $\widehat{C}$ and $X \setminus \widehat{C}$, for some cluster $\widehat{C} \in \widehat{\mathcal{C}}$. Therefore, we take each cluster $\widehat{C} \in \widehat{\mathcal{C}}$ in turn. We then take any node $x \in \widehat{C}$, and we learn the label of i with $\mathcal{O}(k)$ SCQ queries. Then, we invoke SEED$(X \setminus \widehat{C}, i)$. If we get a node in return, we know that C_i has points in $\widehat{C}$ and $X \setminus \widehat{C}$, and therefore $\widehat{\mathcal{C}} \neq \mathcal{C}$. Otherwise, we continue to the next cluster. If the outputs of the SEED are all NIL, then we deduce that $\widehat{\mathcal{C}} = \mathcal{C}$.

The process will stop with $\widehat{\beta} \geq \frac{1}{2}\beta$, which happens after $R = \mathcal{O}(\log \mathcal{M}^*(\frac{\beta\gamma}{2}))$ rounds. Since $\mathcal{M}^*(\frac{\beta\gamma}{2}) \leq \left(\frac{4}{\beta\gamma}\right)^{\text{dens}(X)}$, see Section 3, then $\log \mathcal{M}^*\left(\frac{\beta\gamma}{2}\right) = \mathcal{O}(\text{dens}(X) \log\left(\frac{4}{\beta\gamma}\right))$. At each round, the algorithm uses k^2 SCQ queries plus $2k$ SEED queries. Thus, in total we use $\mathcal{O}(k^2 R)$ SCQ queries and $\mathcal{O}(kR)$ SEED queries. The case with γ is unknown and β is known is symmetric. ■

Appendix E. Supplementary material for Section 7

E.1. Running time with identical radii

We prove:

Theorem 24 RecoverClustering$(X, \varepsilon, \boldsymbol{s})$ *runs in time* $\mathcal{O}(k^2(n+m))$.

Proof First, recall from Section 7 that computing $G = G_X(\varepsilon)$ takes time $\mathcal{O}(n+m)$. Then, each call to RecoverSingleCluster$(G, \varepsilon, \boldsymbol{s}, i)$ takes time $\mathcal{O}(k(n+m))$ by Lemma 29. Since there are k clusters, the claim follows. ■

In the rest of this appendix we prove Lemma 29, through a sequence of intermediate steps.

Lemma 25 MBS(Z, ε, u_i) *runs in time* $\mathcal{O}(n+m)$.

Proof First, we construct $G(\beta\varepsilon)$ by thresholding G, which takes time $\mathcal{O}(n+m)$. Then, we keep only the edges of $G(\beta\varepsilon)$ which have both endpoints in Z, which takes again time $\mathcal{O}(n+m)$. Once we have $G_Z(\beta\varepsilon)$, listing its connected components takes once again time $\mathcal{O}(n+m)$. ■

Lemma 26 *Given a simple C_i-prefixed path π,* FindCutEdge(π) *runs in time* $\mathcal{O}(\log n)$.

Proof Straightforward, see Observation 1 and the code of FindCutEdge. ■

Lemma 27 ClusterSeparator(G, u_i, u_j) *runs in time* $\mathcal{O}(n+m)$.

Proof Computing $d_G(u_i, x)$ and $d_G(u_j, x)$ for all $x \in G$ takes time $\mathcal{O}(n+m)$ using a BFS from u_i and u_j. Thereafter, we can compute Z in time $\mathcal{O}(n)$. Running MBS(Z, ε, u_i) takes time $\mathcal{O}(n+m)$, see above. Finally, the loop at line 4 takes time $\mathcal{O}(n)$. ■

Lemma 28 *In* RecoverSingleCluster$(G, \varepsilon, \boldsymbol{s}, i)$*, each call to* FindNewSeed$(G, R_i, \varepsilon, \boldsymbol{s}, \boldsymbol{u}, i)$ *takes time $\mathcal{O}(k(n+m))$. By adapting both algorithms, this can be reduced to $\mathcal{O}(n+m)$ while adding at most an additive $\mathcal{O}(n+m)$ to the running time of each iteration of* RecoverSingleCluster.

Proof Let us start with the $\mathcal{O}(k(n+m))$ bound given by a "naive" implementation of FindNewSeed. At line 1, we compute $\boldsymbol{s} \cap R_i$ in time $\mathcal{O}(k|R_i|) = \mathcal{O}(kn)$ and perform the check in constant time. At line 2, we make $|\boldsymbol{u}| \le k$ iterations. At each iteration we compute Z in time $\mathcal{O}(n+m)$ with a BFS from u, then we run MBS(Z, ε, u) in time $\mathcal{O}(n+m)$ by Lemma 25, and possibly we search for $x \in Z \setminus Z_i$ which takes time $\mathcal{O}(n)$. Hence the entire loop of line 2 takes time $\mathcal{O}(k(n+m))$. Finally, at line 6 we compute the set $\{(x,y) \in \Gamma(R_i) \,:\, \forall u \in \boldsymbol{u} \,:\, d_G(u,x) \ge \frac{2}{\gamma} + 1\}$. To this end, we compute the set $\{x \in R_i \,:\, \forall u \in \boldsymbol{u} \,:\, d_G(u,x) \ge \frac{2}{\gamma} + 1\}$, which takes time $\mathcal{O}(k(n+m))$ using a BFS from u. For each such x in this set, we list all its edges $(x,y) \in E(G)$. If we find any such edge with $y \notin R_i$, we return y, else we return NIL. Thus, this part takes $\mathcal{O}(k(n+m))$. Therefore, a single call to FindNewSeed takes time $\mathcal{O}(k(n+m))$. Since FindNewSeed is called at most k times, this gives a total running time of $\mathcal{O}(k^2(n+m))$.

Let us now see how to reduce to $\mathcal{O}(n+m)$ the running time of FindNewSeed, by adding at most $\mathcal{O}(n+m)$ to each iteration of RecoverSingleCluster. First, consider line 1 of FindNewSeed. We keep $\boldsymbol{s}$ updated so as to ensure that $\boldsymbol{s} \cap R_i = \boldsymbol{s} \setminus \{s_i\}$. In this way, we can run line 1 in constant time. Towards this end, we modify RecoverSingleCluster as follows. First, we store s_i separately form $\boldsymbol{s}$ in a dedicated variable. Second, after updating G_i and R_i at line 10 of RecoverSingleCluster, we replace $\boldsymbol{s}$ with $\boldsymbol{s} \cap R_i$. This is done by taking an empty dictionary $\boldsymbol{s}'$, taking every node $x \in R_i$, and adding x to $\boldsymbol{s}'$ if $x \in \boldsymbol{s}$. The whole operation takes time $\mathcal{O}(n)$ by using dictionaries with $O(1)$ time per lookup and update. Summarizing, we spend an additional $\mathcal{O}(n)$ time at each iteration of RecoverSingleCluster, and line 1 of FindNewSeed will run in constant time.

Now consider the loop at line 2 of FindNewSeed. We modify RecoverSingleCluster so as to keep track of the set of nodes:

$$\overline{Z(\boldsymbol{u})} = \{x \in R_i \setminus C_i \,:\, \exists u \in \boldsymbol{u} \,:\, d_G(u,x) < \frac{2}{\gamma} + 1\}$$

Note that line 2 of FindNewSeed detects precisely if $\overline{Z(\boldsymbol{u})} \neq \emptyset$, in which case it returns any $x \in \overline{Z(\boldsymbol{u})}$. Thus, if we have $\overline{Z(\boldsymbol{u})}$, we can replace the entire block at line 2 with an equivalent block that runs in time $\mathcal{O}(1)$. To keep track of $\overline{Z(\boldsymbol{u})}$, we initialize it to an empty set, using a dictionary with $O(1)$ lookup and access time. Then, after updating G_i and R_i at line 10, we perform the following operations. First, we make RecoverSingleCluster compute $Z = Z(u_i)$. Second, we run MBS(Z, ε, u_i) to obtain $Z_i(u_i)$. Third, we compute $\overline{Z_i(u_i)} := Z(u_i) \setminus Z_i(u_i)$. Fourth, we add $\overline{Z_i(u_i)}$ to $\overline{Z(\boldsymbol{u})}$. Finally, we keep in $\overline{Z(\boldsymbol{u})}$ only those $x \in \overline{Z(\boldsymbol{u})}$ such that $x \in R_i$. This can be done by creating a new dictionary, adding to it each $x \in R_i$ such that $x \in \overline{Z(\boldsymbol{u})}$, and overwriting $\overline{Z(\boldsymbol{u})}$ with that dictionary, which requires time $\mathcal{O}(n)$ in total. Therefore, we add $\mathcal{O}(n+m)$ to each iteration of RecoverSingleCluster, and line 2 of FindNewSeed will take time $\mathcal{O}(n)$.

Finally, we have line 6 of FindNewSeed. Here, we want to perform the check in time $\mathcal{O}(n+m)$. To this end, at the beginning of the first iteration of RecoverSingleCluster, we mark all nodes of R_i as *active*. Then, at each iteration, after RecoverSingleCluster has computed (u_i, u_j), for all $x \in R_i$ we compute $d_G(u_i, x)$ and if $d_G(u_i, x) < \frac{2}{\gamma} + 1$ then we change the mark of x to *inactive*. This takes time $\mathcal{O}(n+m)$, using a BFS from u_i. Then, at line 6 of FindNewSeed, we only need to sweep over all $x \in R_i$ and, if x is active, list its edges (x,y) until finding $y \notin R_i$ (a check which takes again time $O(1)$ by storing R_i as a dictionary). This gives a total time bound of $\mathcal{O}(n+m)$ for the block at line 6, and once again we add only a $\mathcal{O}(n+m)$ to each iteration of RecoverSingleCluster.

The proof is complete. ■

Lemma 29 RecoverSingleCluster$(G, \varepsilon, \boldsymbol{s}, i)$ *runs in time* $\mathcal{O}(k(n+m))$.

Proof Computing G_i and R_i at any point along the algorithm takes time $\mathcal{O}(n+m)$. Now consider each iteration of the main loop. By Lemma 28, we can make FindNewSeed$(G, R_i, \varepsilon, \boldsymbol{s}, \boldsymbol{u}, i)$ run in time $\mathcal{O}(n+m)$ while increasing the overall running time of the iteration of RecoverSingleCluster by $\mathcal{O}(n+m)$. ShortestPath$(G[R_i], s_i, s_h)$ runs in time $\mathcal{O}(n+m)$, as it is simply a BFS on $G[R_i]$. FindCutEdge$(\pi(s_i, s_h))$ runs in time $\mathcal{O}(\log n)$, see Observation 26. Adding u_i to $\boldsymbol{u}$ takes constant time. ClusterSeparator(G, u_i, u_j) runs in time $\mathcal{O}(n+m)$, see Lemma 27. Therefore each iteration of RecoverSingleCluster takes time $\mathcal{O}(n+m)$. At most k iterations are made, concluding the proof. ■

E.2. Running time with different radii

Theorem 30 RecoverClustering2$(X, \boldsymbol{\varepsilon}, \boldsymbol{s})$ *runs in time* $\mathcal{O}(k^2(n+m))$.

Proof Let us consider each one of the k iterations of the algorithm. To compute G^*, we perform a BFS by ignoring any edge $(x, y) \in \mathcal{G}$ with $d(x, y) > \varepsilon^*$. The resulting runtime is $\mathcal{O}(n+m)$. The SEED part takes time $\mathcal{O}(k) = \mathcal{O}(n)$, as does the construction of $\boldsymbol{s}^*$. The call to RecoverSingleCluster takes time $\mathcal{O}(k(n+m))$ by Lemma 29. Writing $\widehat{C_i}$ in the output takes time $\mathcal{O}(n)$. Summing over all iterations gives the bound. ■

Appendix F. Supplementary material for Section 8

F.1. Proof of Theorem 14

Theorem 14 (Dependence on $\mathrm{dens}(X)$.**)** *Choose any* $\beta, \gamma \in (0, 1)$. *There is a distribution of* (β, γ)*-convex* 2*-clusterings* $\mathcal{C}$ *(Definition 1 or Definition 10), where* $n = |X| = 2^{\mathrm{dens}(X)}$ *is arbitrarily large, such that any algorithm (even randomized) needs* $\Omega(2^{\mathrm{dens}(X)})$ SCQ *and/or* SEED *queries to recover* $\mathcal{C}$ *with constant probability. This holds even if* $\beta, \gamma, \varepsilon$ *are known.*

Proof Let $\mathcal{G} = (X, \mathcal{E}, d)$ where $(X, \mathcal{E})$ is the complete graph on n nodes and $d = 1$, and let $\mathcal{C} = (C_1, C_2)$ be a uniform random partition of X. We claim that any such $\mathcal{C}$ is (β, γ)-convex according to both Definition 1 and Definition 10. Take indeed $\varepsilon = 1$ and let $G = G_X(\varepsilon)$. The connectivity of $G[C_1]$ and $G[C_2]$ holds trivially (they are complete graphs). The local metric margin holds as well, since any two distinct points $x, y \in X$ satisfy $d(x, y) = d > \beta d$, as $\beta < 1$. To see that geodesic convexity holds, too, note that for any $x, y \in C_1$ we have $d_G(x, y) \leq 1$ and any (simple) path between x and y that contains a point in $X \setminus C_1$ has length at least $2 > (1+\gamma) d_G(x, y)$. Finally, note that $G_X(\varepsilon')$ is an independent set for any $\varepsilon' < \varepsilon$, proving that $\mathcal{C}$ is (β, γ)-convex according to Definition 10 as well.

Now, since $\mathcal{C}$ is chosen uniformly at random among all partitions of X, one can see that $\Omega(n)$ SCQ or SEED queries are necessary to recover $\mathcal{C}$ with constant probability. To see this, note that as long as x has not been returned by some SEED query or has not bee queried via SCQ, then x belongs to one of C_1 and C_2 with equal probability. Finally, $\mathrm{dens}(X) = \log_2 \mu(X) \leq \log_2 n$ since $\mu(X) \leq |X|$. Thus $n = 2^{\mathrm{dens}(X)}$, which proves the thesis. ■

F.2. Proof of Theorem 15

Theorem 15 (Necessity of seeds.) *Choose any $\beta, \gamma \in (0, 1]$. There is a distribution of (β, γ)-convex 2-clusterings $\mathcal{C}$ (Definition 1 or Definition 10), where $X \subseteq \mathbb{R}^2$ with $n = |X|$ arbitrarily large and d the Euclidean distance, such that any algorithm (even randomized) needs $\Omega(n)$ SCQ queries to recover $\mathcal{C}$ with constant probability if no seed nodes are given. This holds even if $\gamma, \alpha, \varepsilon$ are known.*

Proof Let $\mathcal{G} = (X, \mathcal{E}, d)$ where $(X, \mathcal{E})$ is the complete graph and $d(x, y)$ is the Euclidean distance in $\mathbb{R}^2$. We let $X = \text{UP} \cup \text{LOW}$, see Figure 4, where:

$$\text{UP} = \bigcup_{j=1}^{n/3} \{(2j, 1)\}, \quad \text{LOW} = \bigcup_{j=1}^{2n/3} \{(j, 0)\}$$

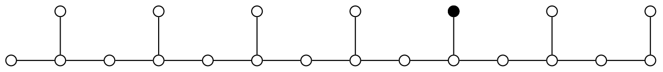

Figure 4: the graph $G_\varepsilon(X)$ for $\varepsilon = 1$. All points are in C_1, save for the filled point in C_2, chosen uniformly at random in UP.

Now choose a point z uniformly at random from UP, and set $C_1 = X \setminus \{z\}$ and $C_2 = \{z\}$. One can check that all properties of Definition 1 and Definition 10 are satisfied for $\varepsilon = 1$. In particular, in $G_X(\varepsilon)$, no simple path between two points of C_1 contains z. Hence, $\mathcal{C}$ is (β, γ)-convex. Clearly, $\Omega(n)$ queries are needed to find z (which is equivalent to recovering $\mathcal{C}$) with constant probability, even if γ, α and the ε are known. ■

F.3. Proof of Theorem 16

Theorem 16 (Cost of learning the radii.) *For any $k \geq 2$, and any sufficiently large n, there exists a distribution of $(1/2, 1)$-convex k-clusterings (Definition 1 or Definition 10) on n points such that any algorithm (even randomized) needs $\Omega(k \log \frac{n}{k})$ SEED and/or SCQ queries to learn the radii of all clusters with constant probability.*

Proof We start with the simpler case $k = 2$. Let $\mathcal{G} = (X, \mathcal{E}, d)$ be a path with increasing edge weights, that is:

$$X = [n] \tag{25}$$
$$\mathcal{E} = \{(j, j+1) \,:\, j = 1, \ldots, n-1\} \tag{26}$$
$$d(j, j+1) = 1 + \frac{\beta j}{n}, \quad (j, j+1) \in \mathcal{E} \tag{27}$$

Choose j^* uniformly at random in $\{2, \ldots, n-1\}$. We let $C_1 = \{1, \ldots, j^*\}$, and $C_2 = \{j^*, \ldots, n\}$.

First, we prove that C_1 and C_2 are (β, γ)-convex with radii respectively $\varepsilon_1 = d(j^* - 1, j^*)$ and $\varepsilon_2 = d(n-1, n)$. Recall Definition 1. For the connectivity, clearly $\rho(G_{C_1}(\varepsilon_1)) = \rho(G_{C_2}(\varepsilon_2)) = 1$. For the local metric margin, note that, by the choice of β and d, we have $d \geq 1$ but $\varepsilon_1, \varepsilon_2 \leq \frac{3}{2}$. Thus, for any two distinct $x, y \in X$, we have $d(x,y) > \beta \max(\varepsilon_1, \varepsilon_2)$. Therefore the local metric margin is satisfied. For geodesic convexity, note that there is only one edge between C_1 and C_2 in $\mathcal{G}$, thus no simple path can exist between two points of one cluster that intersects the other cluster. Thus, the properties of Definition 1 are satisfied by $\varepsilon_1, \varepsilon_2$.

To show that Definition 10 is satisfied as well, we have to prove that $\varepsilon_1, \varepsilon_2$ are the smallest such values; by Lemma 20 this implies that ε_1 and ε_2 are the radii of respectively C_1 and C_2. To this end, simply note that $\varepsilon_1 = \min\{\zeta : \rho(G_{C_1}(\zeta)) = 1\}$, since $d(j^* - 1, j^*) = \varepsilon_1$ and $d(j, j+1) \le \varepsilon_1$ for all $j = 1, \ldots, j^* - 1$. Similarly, $\varepsilon_2 = \min\{\zeta : \rho(G_{C_2}(\zeta)) = 1\}$, since $d(n-1, n) = \varepsilon_2$ and $d(j, j+1) \le \varepsilon_1$ for all $j = j^*, \ldots, n-1$.

Finally, we prove that any algorithm needs $\Omega(\log n)$ queries to learn ε_1. Clearly, if the algorithm learns ε_1 then it can also output the index j^*, which is a function of ε_1. Therefore, we show that finding j^* requires $\Omega(\log n)$ queries.

First, we show that SEED is as powerful as SCQ. Consider any set of points $U \subseteq X$. Recall that, when $U \cap C_i \neq \emptyset$, $\textsc{seed}(U, i)$ is allowed to return *any* node in $U \cap C_i$. Therefore, we let $\textsc{seed}(U, i)$ return $\min(U \cap C_i)$ if $i = 1$, and $\max(U \cap C_i)$ if $i = 2$. Now, observe that $\min(U \cap C_1) = \min U$ and $\max(U \cap C_1) = \max U$. Therefore, the output of $\textsc{seed}(U, i)$ can be emulated using SCQ. If $i = 1$, then we run $\textsc{scq}(\min(U), 1)$; if the response is $+1$, then we return $\min(U)$, else we return NIL. For $i = 2$, we do the same, but using $\max(U)$.

Therefore, SEED and SCQ are equivalent in this case. Since each call to SCQ reveals at most one bit of information, and j^* is chosen uniformly at random in a set of cardinality $\Omega(n)$, we need $\Omega(\log n)$ in order to learn j^* with constant probability.

In order to extend the construction to any $k \ge 2$, simply take $K = \frac{k}{2}$ disjoint weighted paths on $\frac{n}{K}$ nodes each (without loss of generality we can assume k is even). Each such path is weighted as in the construction above, with the weights of the h-th path all smaller than the weights of the $(h+1)$-th path. For each path, we draw j^* uniformly at random like above, and form two clusters. The same proof used above shows that, to learn the radii of all clusters with constant probability, any algorithm uses at least $\Omega(k \log \frac{n}{k})$ queries. ■