A Scalable Approach for Mapper via Efficient Spatial Search

Anonymous authors
Paper under double-blind review

Abstract

Topological Data Analysis (TDA) is a branch of applied mathematics that studies the shape of high dimensional datasets using ideas from algebraic topology. The Mapper algorithm is a widely used tool in Topological Data Analysis, used for uncovering hidden structures in complex data. However, existing implementations often rely on naive and inefficient methods for constructing the open covers that Mapper is based on, leading to performance issues, especially with large, high-dimensional datasets. In this study, we introduce a novel, more scalable method for constructing open covers for Mapper, leveraging techniques from computational geometry. Our approach significantly enhances efficiency, improving Mapper's performance for large high-dimensional data. We will present theoretical insights into our method and demonstrate its effectiveness through experimental evaluations on well-known datasets, showcasing substantial improvements in running time compared to existing approaches. We implemented our method in a new Python library called *library-omitted-for-anonymity*, which is freely available at `link-omitted-for-anonymity`, providing a powerful tool for TDA practitioners and researchers.

1 Introduction

In recent years, Topological Data Analysis (TDA) has gained significant traction in the field of data science due to its ability to extract valuable insights from complex datasets. TDA uses topological methods that are resilient to noise and dimensionality, making it a robust mathematical framework for data analysis. A well-knonw technique in TDA is the *Mapper algorithm*. Mapper provides a visual representation of data in the form of a graph, called *Mapper graph*, enabling easy exploration and interpretation. Unlike conventional algorithms, such as *clustering algorithms* or *Principal Component Analysis (PCA)*, Mapper excels at visualizing data by preserving their connected components, making it very effective for shape analysis and pattern discovery. The effectiveness of Mapper was initially demonstrated in the analysis of medical data, as showcased in the pioneering work by Singh et al. Singh et al. (2007). Since then, Mapper has proven to be a versatile and powerful tool for data exploration, capable of uncovering hidden patterns even in high-dimensional datasets.

Data exploration is an interactive process that requires constant fine-tuning and adjustments to obtain relevant information. Therefore, the running time performance of software for Mapper is essential for its widespread adoption. The original description of Mapper (Singh et al., 2007) includes what has now become a standard approach, involving the construction of an *open cover* made of overlapping hyperrectangles, also known as *standard cubical cover*. Currently, researchers and developers have access to several established open-source libraries for Mapper. However, these libraries work well with low dimensional lenses, but their approach is often inefficient in higher dimensions. More specifically:

1. The standard cubical cover is often assembled from open covers obtained in lower dimensional projections. This implies that the number of steps grows exponentially with the dimension, and this is computationally unfeasible;

2. Points in dimension k can fall in the intersection of up to 2^k hypercubes, resulting in complex Mapper graphs that are hard to explore, and often have more nodes than points in the dataset.

These crucial points has been consistently overlooked and neglected, reinforcing the misconception that Mapper is inefficient with high-dimensional data. Motivated by these limitations, recent advancements in the field have led to the development of a wide family of *Mapper-type* algorithms, each proposing a distinct adaptation of the original concept. For instance, *Ball Mapper* (Dłotko, 2019) and *Mapper on Ball Mapper* (Dłotko et al., 2023) construct the open cover by creating an ϵ-net (Gonzalez, 1985), adopting open balls centered in the points of the dataset instead of evenly-spaced hyperrectangles. Additionally, specialized variations like *NeuMapper* (Geniesse et al., 2022), designed specifically for neuroscience data, adopt a more complex approach. This method, partially inspired by Ball Mapper, employs an intrinsic metric derived from *reciprocal kNN*. These adaptations all shift towards changing the way open covers are built, improving the performance, but also introducing some randomness and implicit choices due to the ϵ-net construction (Gonzalez, 1985). While this is often acceptable, there are cases where using a cubical cover with uniform overlap is beneficial, especially given its foundational role in many Mapper-related results. For instance, one key advantage of the standard cubical cover is being able to estimate optimal parameters (Carrière et al., 2018), minimizing the need for time-consuming manual fine-tuning.

In this work, we introduce a novel and more efficient approach to computing Mapper-type algorithms, leveraging concepts from computational geometry, aimed at solving the problems of currently available implementations (points 1 and 2). Our method uses a greedy adaptation of ϵ-net (Gonzalez, 1985; Dłotko, 2019), called *proximity-net*, to construct a subcover of the standard cubical cover, preserving the evenly overlapping open sets of the standard cubical cover, as defined in the original Mapper implementation. Moreover, we show how we can improve the overall efficiency of Mapper by adopting specialized data structures for spatial search like *metric trees* (Clarkson, 2006; Brin, 1995; Yianilos, 1993; Uhlmann, 1991). We also provide a theoretical analysis on the complexity of building cubical covers, obtaining an upper bound that explicitly incorporates the *doubling dimension* of the dataset (Krauthgamer and Lee, 2004). We present theoretical insights into our method, supported by experimental evaluations on well-known datasets, highlighting significant improvements in running time compared to the standard approach. Additionally, we introduce our open-source library, *library-omitted-for-anonymity* (Anonymous, 2024), available at `link-omitted-for-anonymity`. To the best of our knowledge, it is the only library implementing this approach. Finally, we compare our method with existing topological data analysis libraries, including *Kepler Mapper* (Van Veen, 2019) and *giotto-tda* (Tauzin et al., 2021). Performance tests demonstrate the advantages of our method in terms of scalability and efficiency, underscoring its potential for large-scale applications.

2 Preliminaries

We start with preliminary definitions and notations necessary for understanding the rest of this work. While some of these concepts are fundamental in topology and widely available in various resources, we include them here for the sake of completeness.

Remark 1. *In this work, we analyze datasets* X *that are finite and discrete samples of unknown topological spaces* $\tilde{X}$*. The topology of a finite, discrete space is trivial, as it only depends on its cardinality. Hence, we are interested in the topology of* $\tilde{X}$*, not* X*. We use the sample* X *to infer the shape of* $\tilde{X}$*.*

Definition 1. *Let* X *be a topological space. A pseudo-metric on* X *is a map* $d\colon X \times X \to \mathbb{R}$ *such that*

- $d(x, y) \geq 0$ *for every* $x, y \in X$*, and* $d(x, x) = 0$ *for every* $x \in X$*;*
- d *is symmetric, i.e.* $d(x_1, x_2) = d(x_2, x_1)$ *for every* $x_1, x_2 \in X$*;*
- d *satisfies the triangle inequality, i.e.:* $d(x_1, x_3) \leq d(x_1, x_2) + d(x_2, x_3)$ *for every* $x_1, x_2, x_3 \in X$*.*

We say that d *is a metric when* $d(x, y) = 0$ *implies* $x = y$*. An open ball of center* $p \in X$ *and radius* $\epsilon > 0$ *is defined as* $B_d(p, \epsilon) = \{x \in X | d(p, x) < \epsilon\}$*.*

Definition 2. *Let* $f\colon X \to Y$ *be a map. If* d *is a pseudo-metric on* Y*, the pullback of* d *under* f *is the pseudo-metric* f^*d*, defined by setting for each* $u, v \in X$

$$(f^*d)(u, v) = d(f(u), f(v)).$$

Definition 3. *The* L_∞*-distance on* $\mathbb{R}^k$ *is the metric denoted with* d_∞ *defined by setting*

$$d_\infty(x, y) = \max_{i=1,\ldots,n} (|x_i - y_i|)$$

for every $x, y \in \mathbb{R}^k$*.*

The notion of ϵ-net from Gonzalez (1985) is well-known in computational geometry (see also Clarkson (2006)), and we report it here since we will use it in the rest of this work.:

Definition 4. *Let* (X, d) *be a pseudo-metric space. An* ϵ*-net on* X *is a subset* $N \subseteq X$ *such that:*

- $d(x, y) \geq \epsilon$ *for every* $x, y \in X$ *whenever* $x \neq y$
- *For every* $x \in X$ *there exists* $y \in N$ *such that* $d(x, y) < \epsilon$

An ϵ-net corresponds directly to an open cover consisting of open balls. Given that the primary objective of this work is to construct open balls, it is important to understand the methodology for building an ϵ-net. This can be achieved by employing a greedy algorithm, as detailed in Gonzalez (1985) (see Algorithm 1).

Definition 5. *Let* (X, d) *be a pseudo-metric space. The doubling measure of* X *is the least* $\lambda > 0$ *such that every ball in* X *can be covered with* λ *balls of half diameter. The doubling dimension of* X *is* $dim(X) = \log_2 \lambda$*.*

The notions of ϵ-net and doubling dimension are bound together by Proposition 1 that gives an estimation of the cardinality of any ϵ-net, and can be found in many resources, for example Krauthgamer and Lee (2004); Clarkson (2006).

Proposition 1. *Let* (X, d) *be a pseudo-metric space, and let* N *be an* ϵ*-net for* X*. Then for every ball* $B(p, R)$ *in* X *we have* $|N \cap B(p, R)| = \mathcal{O}((R/\epsilon)^{dim(X)})$*.*

As we will see later, after some adaptation, we will use the same ideas of Proposition 1 to prove Theorem 2, where the open cover is one of the core steps of Mapper, and cannot be derived from an ϵ-net.

Algorithm 1 ϵ-net construction

```
Require: Let $X$ be a topological space, and let $d$ be a metric on $X$.
Ensure: An $\epsilon$-net of $X$
  $N \leftarrow \emptyset$
  while $d(X \setminus N, N) > 0$ do
      Take $p \in X \setminus N$ maximizing $d(p, N)$
      $N \leftarrow N \cup \{p\}$
  end while
  return $N$
```

2.1 Mapper algorithm

In this subsection, we provide a concise overview of Mapper, based on its original formulation (Singh et al., 2007). Mapper operates on a dataset X and its output is determined by the following steps:

1. Let f be a *lens*, defined as any continuous map $f\colon X \to Y$, where Y is a parameter space. Common choices for the lens f include *statistics* of any order, *projections*, *entropy*, *density*, *eccentricity*, and more.
2. Next, we proceed by constructing an *open cover* for $f(X)$. In other words, we create a collection $\{U_\alpha\}_\alpha$ of open sets such that their union *covers* the entire image $f(X)$, i.e., $f(X) = \bigcup_\alpha U_\alpha$. It is important to note that the sets in this open cover may intersect with one another, and they inherit their topology from the space Y.
3. For each element U_α in the selected cover, we define V_α as the preimage of U_α under the function f. It is clear that the collection $\{V_\alpha\}_\alpha$ forms an open cover of X. Next, we proceed by applying a user-specified *clustering algorithm*, in order to partition each open set V_α into a disjoint union of clusters, denoted as $V_\alpha = \amalg_\beta C_{\alpha,\beta}$. The resulting family $\{C_{\alpha,\beta}\}_{\alpha,\beta}$ is referred to as a *refined open cover* for X.
4. We construct the *Mapper graph* as the undirected graph $G = (V, E)$ defined by the following rule: the set V contains a vertex $v_{\alpha,\beta}$ for every local cluster $C_{\alpha,\beta}$, while the set E contains the edge $e = (v_{\alpha_1,\beta_1}, v_{\alpha_2,\beta_2})$ only if their corresponding local clusters intersect, i.e., when $C_{\alpha_1,\beta_1} \cap C_{\alpha_2,\beta_2} \neq \emptyset$.

The theoretical foundation of Mapper is rooted in the *Nerve Theorem* (Borsuk, 1948; Weil, 1952), which requires that every intersection of finitely many open sets are either empty or *simply connected*. For Mapper, this means that any intersection of clusters must be empty or simply connected. However, Nerve Theorem does not always apply for open covers obtained from Mapper, since clustering may produce clusters that don't correspond to connected components, and may not preserve simple-connectedness. Nevertheless, when the Nerve Theorem applies, then X and its Mapper graph have the same number of connected components.

2.2 Standard Cubical Cover

In the original definition of Mapper (Singh et al., 2007), the authors use an open cover defined by two parameters: the *length* w of the intervals and the *overlap* $p \in (0, 1)$, which is the fraction of

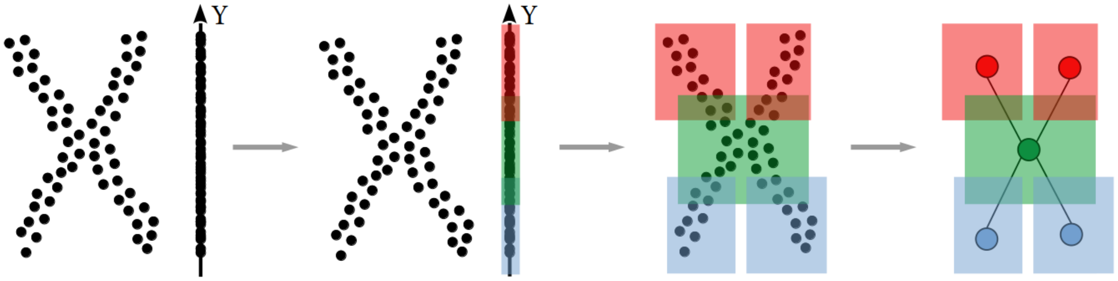


Figure 1: An example of a dataset X as an X-shaped 2D point cloud, with Mapper steps shown from left to right. The lens is the projection on the Y-axis and its image is covered by three open sets. Each open set is then pulled back and clustered. Clusters from the same open set share the same color.

w that corresponds to the length δ of the intersection of any two adjacent intervals in the cover. Some sources call such cover *cubical cover*, some others call it implicitely *standard cover*. In the rest of this work we will refer to such cover as *standard cubical cover*, and to any of its subcovers as *cubical cover*. In the remainder of this subsection we'll denote with Y the space $f(X) \subseteq \mathbb{R}^k$ on which the cover is constructed.

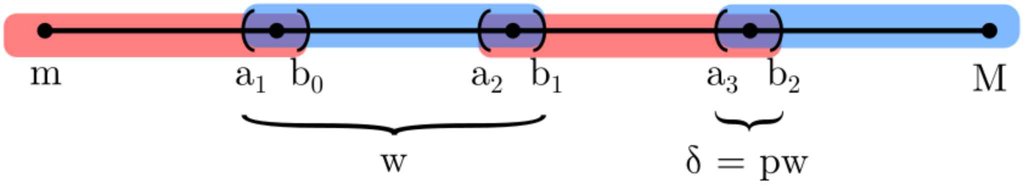


Figure 2: A visual representation of a standard cubical cover in dimension one with $n = 4$. The value $w = \frac{M-m}{n(1-p)}$ correspond to the length of every interval, while $\delta = pw$ corresponds to the length of the overlap of any two adjacent intervals.

Definition 6. *Let $0 < n \in \mathbb{N}$ and $p \in (0,1)$. Let $Y \subseteq \mathbb{R}^k$ compact and let Y_i be the projection of Y on the i-axis. The standard cubical cover of Y with n intervals and p overlap is the collection of open sets*

$$\mathcal{CC}_Y^{n,p} = \left\{ R \cap Y \neq \emptyset \middle| R \in \prod_{i=1}^{k} \mathcal{CC}_{n,p}^{Y_i} \right\}$$

where $\mathcal{CC}_{n,p}^{Y_i}$ is defined as

$$\mathcal{CC}_{n,p}^{Y_i} = \{(a_j, b_j) \cap Y_i | j = 0, \dots, n-1\}$$

with

$$\begin{aligned} a_j &= m + j(w - \delta) - \delta/2 \\ b_j &= m + (j+1)(w - \delta) + \delta/2, \end{aligned}$$

where $m = min(Y_i)$ and $M = max(Y_i)$, $w = \frac{M-m}{n(1-p)}$ and $\delta = pw$.

Remark 2. *We report here some facts that easily follow from the definition in the case of a one-dimensional standard cubical cover for some $Y \subseteq [m, M]$.*

- *$b_i - a_i = w$ for every $i = 0, \ldots, n-1$;*
- *$a_0 = m - \delta/2$ and $b_{n-1} = M + \delta/2$;*
- *$b_i - a_{i+1} = \delta$ for every $i = 0, \ldots, n-1$.*
- *$[m, b_0 - \delta/2), \ldots, [a_i + \delta/2, b_i - \delta/2), \ldots [a_{n-1} + \delta/2, M]$ is a partition of $[m, M]$.*

In this work we use a notion of standard cubical cover that simplifies some computations, but it's worth to point out that this definition may slightly diverge in literature. For example, some authors and software libraries, require that the standard cubical cover of $[n, M]$ satisfies $a_0 = m$ and $b_{n-1} = M$. However, it's also important to remark that our version of standard cubical cover is compatible with the results from Carrière et al. (2018).

From Definition 6, we can readily devise an algorithm that initially computes the standard cubical cover for each projection independently. Subsequently, we assemble these individual open covers into an open cover for the topological space Y. We will refer to this approach as *naive construction* of standard cubical cover (see Algorithm 2).

Algorithm 2 Naive construction

```
Require: $Y$ finite point cloud, $0 < n \in \mathbb{N}$, $p \in (0, 1)$.
Ensure: $\mathcal{CC}_Y^{n,p}$.
  for $i = 1, \ldots, k$ do
      $\mathcal{CC}_{n,p}^{Y_i} \leftarrow \{I_{i,0}, \ldots, I_{i,n-1}\}$ be the standard cubical cover on the projection of $Y$ on the $i$-axis;
  end for
  $\mathcal{CC}_Y^{n,p} \leftarrow \{R = \prod_{i=1}^k I_{i,j_i} | R \neq \emptyset, 0 \leq j_i \leq n-1\}$.
  return $\mathcal{CC}_Y^{n,p}$
```

Algorithm 2 presents a straightforward yet inefficient method for obtaining the standard cubical cover of a dataset Y. Indeed, this approach becomes computationally expensive for high-dimensional datasets. This is also true in the case of Mapper, when we construct the standard cubical cover on high dimensional lenses. Even when many products are empty, their number can grow rapidly and introduce additional computational overhead to the entire Mapper process. To illustrate this issue, which is well-known in literature, consider the following example: if $Y \subset \mathbb{R}^k$ lies along the diagonal, an appropriate cover for Y could be achieved using a small number of rectangles, proportional to the number of intervals n. However, Algorithm 2 would construct an open cover for each projection initially and then iterate through all possible rectangles, resulting in a total of n^k steps. As we will demonstrate later in Section 3, the primary contribution of this work is the resolution of this issue through the adoption of a more efficient algorithm. Instead of relying on projections, this algorithm iterates over a significantly smaller number of open sets, comparable to $n^{dim(f(X))} \ll n^k$.

2.3 Ball Cover

When Y is contained in $\mathbb{R}^k$, we can use rectangles with centers in $\mathbb{R}^k$ to form an open cover, as in the standard cubical cover. However, this approach is not applicable to arbitrary metric spaces, where the concept of a rectangle may be undefined. In such cases, open balls centered at Y can be used as an alternative.

Definition 7. *Given a dataset Y and a metric d on Y, a ball cover of radius $r > 0$ on Y is any open cover where every set is an open ball of radius r.*

Remark 3. *It's important to note that the notion of standard cubical cover is not a special case of the notion of ball cover. This distinction is often misunderstood because balls in the L_∞-distance are cubical. However, in the case of a ball cover, the balls are centered at the points of Y, whereas in a standard cubical cover, the centers are evenly spaced and are centered at points in $\mathbb{R}^k$ which may not be contained in Y.*

The concept of constructing a ball cover in the context of Mapper is not novel. It was introduced as an alternative to the original Mapper, known as *Ball Mapper*. This method, referred to as ϵ-*net*, employs a greedy algorithm to cover the dataset with balls until full coverage is achieved (Dłotko, 2019; Dłotko et al., 2023). It is important to note that the term ϵ-*net* is also used in computational geometry to describe a different, albeit related, concept, as defined in Definition 4 from Gonzalez (1985). Despite this potential for confusion, it is crucial to emphasize that the collection of centers obtained from the ϵ-net in Dłotko (2019); Dłotko et al. (2023) is indeed a proper ϵ-net according to Gonzalez (1985). Thus, depending on the context, distinguishing between the two notions is relatively straightforward.

Remark 4. *The number of nodes in the Ball Mapper graph corresponds to the cardinality of the open cover constructed on Y. Therefore, estimating this cardinality can provide insights into the Ball Mapper graph and on the complexity of Ball Mapper. While any open cover of Y has a cardinality bounded by $|Y|$, this bound is not very informative when Y is large. However, more useful bounds can be identified. Using ϵ-net, it's possible to cover Y with a number of open balls proportional to $(1/\epsilon)^{dim(Y)}$, where $dim(Y)$ is the doubling dimension of Y (see Proposition 1).*

2.4 Vantage Point Trees

Given a *query point* q and a *query radius* ϵ, a *range query* is a function that returns the set of points within distance ϵ from q, i.e. the points in the ball $B(q, \epsilon)$. There are many ways to perform range queries efficiently, using different algorithms and data structures. A well-known example is the *kd-tree* (Friedman et al., 1977), which partitions the space in a hierarchical tree-like structure that allows to reduce the number of distance computations employing the triangle inequality. The use of kd-trees in Mapper-type algorithms has been explored in Dłotko (2019), where the author notes that their effectiveness for Ball Mapper may be limited, particularly in high-dimensional spaces where Ball Mapper typically operates. However, we believe that the approach presented in *Mapper on Ball Mapper* from Dłotko et al. (2023) could benefit from incorporating a specialized data structure for range queries. In this case, the open cover is constructed on the space $f(X)$, which is often lower-dimensional than the original space X. Using a data structure optimized for range queries could thus offer a significant performance boost. In our study, we aim to address diverse scenarios by using any lens function $f\colon X \to Y$ with no restriction on the space Y. Importantly, Y need not be strictly Euclidean or coordinate-based; it can encompass any domain where a meaningful notion of distance is defined. For all these reasons we decided to chose *vp-trees* instead of kd-trees (Yianilos, 1993; Brin, 1995). A *vantage-point tree*, or *vp-tree*, is a binary tree data structure where each internal node organizes the points of the space according to their distance from a chosen point, called *vantage point*. Each internal node stores a tuple (p, r) as a reference to the ball $B(p, r)$ where p is the chosen vantage point, and p's descendants satisfy the *vp-tree property*: for every left descendant y we have $d(p, y) \leq r$, and for every right descendant z we have $d(p, z) \geq r$.

The procedure used to build a vp-tree can be sketched in this way: given a dataset Y we first chose a *vantage point* p from Y, then split Y into two equally-sized subsets: those points that are closer to p, and those that are farther. Repeating the procedure on the two halves we obtain two trees L, obtained from the first half, and R obtained from the second one. The result is then obtained as the binary tree rooted at p, with L as left child and R as right child. We will refer to this procedure as the *build_vptree* function (see Algorithm 3 and Figure 3).

Algorithm 3 *build_vptree*(Y, d)

```
Require: Let $Y = [y_0, \ldots, y_{n-1}]$ be a dataset, and let $d$ be a metric on $Y$.
Ensure: build_vptree$(Y, d)$ returns a vp-tree on $(Y, d)$.
  if $Y = \emptyset$ then:
      return $\emptyset$                                            ▷ the empty tree
  else
      $p \leftarrow$ choose in $Y$.                                  ▷ chose randomly
      Move $p$ at the head of $Y$, such that $y_0 = p$.
      Let $\rho = \text{median}_{y \in Y} d(p, Y)$.
      Reorder $Y$ such that $d(p, y_i) \leq \rho$ for $i < n/2$ and $d(p, y_i) \geq \rho$ for $i \geq n/2$.
      $L \leftarrow build\_vptree([y_1, \ldots, y_{n/2-1}], d)$
      $R \leftarrow build\_vptree([y_{n/2}, \ldots, y_{n-1}], d)$
      return $Tree\{root = (p, \rho), left = L, right = R\}$
  end if
```

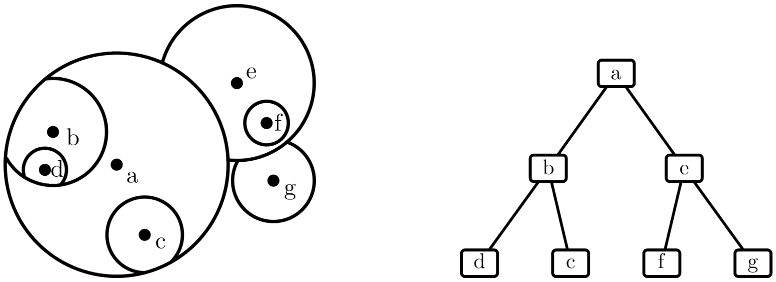


Figure 3: Consider a dataset $Y = \{a, b, c, d, e, f, g\} \subseteq \mathbb{R}^2$. A vp-tree is built by recursively selecting a vantage point and radius. The left image shows the regions centered around vantage points, while the right shows the vp-tree structure. The invariant property of vp-trees is evident: left descendants lie within the ball defined by the vantage point, and right descendants fall outside.

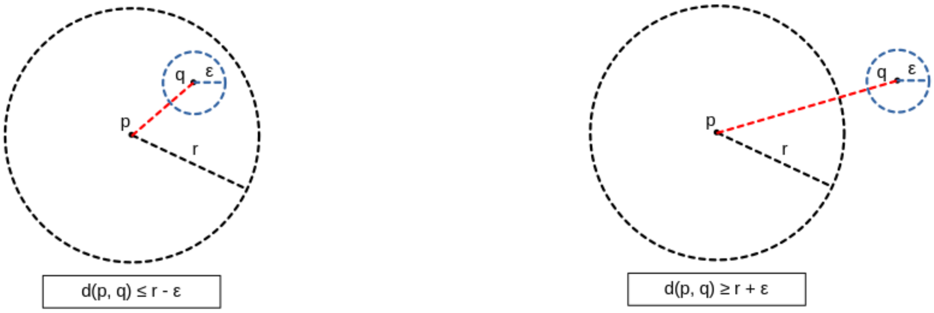
Remark 5. *Building a balanced vp-tree on a dataset Y takes $\mathcal{O}(|Y| \log |Y|)$ time in total.*

After a vp-tree is built, we can perform range queries by descending from the root (see Algorithm 4). Say we want to perform a range query for a point q and radius ϵ. Let (p, r) be the tuple stored at any internal node while visiting the vp-tree. Using the triangle inequality it's possible to skip some of p's children when some conditions are met. In particular, we can do this in two situation: when $B(q, \epsilon) \subseteq B(p, r)$ we need to visit only the left child (see Figure 4a), and when $B(q, \epsilon) \cap B(p, r) = \emptyset$ we need to visit only the right child (see Figure 4b).

Range queries can be significantly more efficient with vp-trees than with linear scans. A linear scan requires going through all the points in Y, which takes $|Y|$ steps in total. On the other hand, with vp-trees, a range query usually takes less steps, since we can often skip one child from the search due to the triangle inequality satisfied by the metric (see Figure 4a and Figure 4b). Giving a general estimation of the average time complexity of range queries via vp-trees is particularly challenging due to its dependency on the dataset (Brin, 1995), we can only state that it is bounded between $\mathcal{O}(\log |Y|)$ and $\mathcal{O}(|Y|)$. However, when the query radius is sufficiently small, we expect to fall often in a cases where we can skip a branch from the range query. In such cases the time complexity becomes closer to $\mathcal{O}(\log |Y|)$.

Algorithm 4 *range_query*(T, q, ϵ)

```
Require: Let $Y$ be a dataset, and $d$ a metric on $Y$. Let $T = build\_vptree(Y, d)$. Let $q$ be a query point, and let $\epsilon > 0$ be a query radius.
Ensure: The open ball $B_d(q, \epsilon)$.
  if $T$ is empty or terminal then
      return $\{y \in T.leaves \mid d(q, y) < \epsilon\}$
  else
      $(p, r) \leftarrow T.root$
      $S \leftarrow \emptyset$
      if $r < d(p, q) + \epsilon$ then
          $S \leftarrow S \cup range\_query(T.right, q, \epsilon)$
      end if
      if $r > d(p, q) - \epsilon$ then
          $S \leftarrow S \cup range\_query(T.left, q, \epsilon)$
      end if
      return $S$
  end if
```



(a) When $B(q, \epsilon) \subseteq B(p, r)$ (equivalent to $d(p, q) \leq r - \epsilon$), we skip the right child, since right descendants are outside the range query.

(b) When $B(q, \epsilon) \cap B(p, r) = \emptyset$ (equivalent to $d(p, q) \geq r + \epsilon$), we skip the left child, since left descendants are outside the range query.

Figure 4: The two conditions when we need to visit only one child during range queries.

It is worth emphasizing that the construction of the ball cover can be improved by leveraging vp-trees alone. Specifically, one can first construct a vp-tree T on the dataset. Then, in the ϵ-net algorithm, the open balls are generated using range queries on T (see Algorithm 5).

3 Cubical Cover in Higher Dimensions

In this section, we outline the main contributions of this work. To begin, it is essential to introduce some notation.

Definition 8. *Let $Y \subseteq \mathbb{R}^k$ compact. Let $m_i = \min_{y \in Y} y_i$, and $M_i = \max_{y \in Y} y_i$, with $m_i < M_i$ for $i = 1 \dots k$. We define $\sigma_Y : \mathbb{R}^k \to \mathbb{R}^k$ by setting for every $y = (y_i)_{i=1,\dots,k} \in \mathbb{R}^k$*

$$\sigma(y) = \left(\frac{y_i - m_i}{M_i - m_i} \right)_{i=1\dots,k}.$$

Algorithm 5 Ball Cover via vp-trees

```
Require: Let $(Y, d)$ be a metric space. Let $r \geq 0$.
Ensure: A ball cover $\mathcal{B}$ of radius $r$.
  $T \leftarrow build\_vptree(Y, d)$                    ▷ Algorithm 3
  $N \leftarrow \emptyset$
  $\mathcal{C} \leftarrow \emptyset$
  while $N \neq Y$ do
      Take $p \in Y \setminus N$
      $B \leftarrow range\_query(T, p, r)$              ▷ Algorithm 4
      $N \leftarrow N \cup B$
      $\mathcal{C} \leftarrow \mathcal{C} \cup \{B\}$
  end while
  return $\mathcal{C}$
```

Remark 6. *In the settings of Definition 8, the map* $\sigma_Y \colon \mathbb{R}^k \to \mathbb{R}^k$ *is a bijection that maps* Y *to the hypercube* $[0,1]^k$*. The map* $\sigma^{-1} \colon \mathbb{R}^k \to \mathbb{R}^k$ *is given by setting for each* $y = (y_i)_{i=1,\ldots,k} \in \mathbb{R}^k$

$$\sigma_Y^{-1}(y) = (m_i + y_i(M_i - m_i))_{i=1,\ldots,k}\,.$$

Definition 9. *Let* $\rho_n \colon \mathbb{R}^k \to \left(\frac{1}{2n} + \frac{1}{n}\mathbb{Z}\right)^k$ *be the map defined by setting for every* $y = (y_i)_{i=1,\ldots,k} \in \mathbb{R}^k$

$$\rho_n(y) = \left(\frac{\lfloor n y_i \rfloor + 1/2}{n}\right)_{i=1,\ldots,k}.$$

Following Definition 6, we define a helper function that maps each point in Y to its closest hypercube in the standard cubical cover. Specifically, this means the function assigns the hypercube whose center is the nearest neighbor to the point among all other hypercube centers.

Definition 10. *Let* $0 < n \in \mathbb{N}$ *and let* $p \in (0,1)$*. Consider the interval* $[m, M] \subseteq \mathbb{R}$ *and let* $w = \frac{M-m}{n(1-p)}$ *and* $\delta = pw$*. Let* $a_i = m + i(w - \delta) - \delta/2$ *and* $b_i = m + (i+1)(w-\delta) + \delta/2$*. We define the cubical proximity function* $CP_{[m,M]}(n,p) \colon [m, M] \to \mathcal{P}([m, M])$ *by setting for every* $y \in [m, M]$

$$CP_{[m,M]}(n,p)(y) = [m, M] \cap (a_i, b_i) \iff a_i + \delta/2 \leq y < b_i - \delta/2.$$

For any $Y \subseteq \mathbb{R}^k$ *compact we can define for every* $y = (y_i)_{i=1,\ldots,k} \in Y$

$$CP_Y(n,p)(y) = Y \cap \prod_{i=1}^{k} CP_{Y_i}(n,p)(y_i).$$

where Y_i *is the projection of* Y *on the* i*-axis.*

Remark 7. *For every* $y \in [m, M]$ *there exists only one* j *such that* $y \in [a_j + \delta/2, b_j - \delta/2)$*, which can be easily computed as*

$$j = n \left\lfloor \frac{y - m}{M - m} \right\rfloor.$$

We can extend this to higher dimension. In case of $Y \subseteq \mathbb{R}^k$*, for every* $y = (y_i)_{i=1,\ldots,k} \in Y$*, for each* i *there exists only one* $j_i = n \left\lfloor \frac{y_i - m_i}{M_i - m_i} \right\rfloor$ *such that* $y_i \in [a_{j_i} + \delta_i/2, b_{j_i} - \delta_i/2)$*, and we have*

$$CP_Y(n,p)(y) = Y \cap \prod_{i=1}^{k} (a_{j_i}, b_{j_i}).$$

This setting gives a well-defined notion for $CP_Y(n,p)$, since the intervals $[a_i + \delta/2, b_i - \delta/2)$ are a partition of $[m, M]$ (see Remark 2). We can finally state one of the main contributions of our methodology, in the form of the following result.

Theorem 1. *Let $Y \subseteq \mathbb{R}^k$ compact. Let $0 < n \in \mathbb{N}$ and let $p \in (0,1)$. Then for every $y \in Y$ we have*

$$CP_Y(n,p)(y) = B_{\sigma_Y^* d_\infty}\left((\sigma_Y^{-1} \circ \rho_n \circ \sigma_Y)(y), \frac{1}{2n-2np}\right).$$

Proof. We have $CP_Y(n,p)(y) = \prod_{i=1}^{k}(a_{j_i}, b_{j_i})$ where $j_i = n\left\lfloor \frac{y_i - m_i}{M_i - m_i}\right\rfloor$. Then

$$\begin{aligned}
\sigma_Y(CP_Y(n,p)(y)) &= \prod_{i=1}^{k}\left(\frac{a_{j_i} - m_i}{M_i - m_i}, \frac{b_{j_i} - m_i}{M_i - m_i}\right) \\
&= \prod_{i=1}^{k}\left(\frac{j_i + 1/2}{n} - \frac{1}{2n-2np}, \frac{j_i + 1/2}{n} + \frac{1}{2n-2np}\right) \\
&= B_{d_\infty}\left(\frac{j_i + 1/2}{n}, \frac{1}{2n-2np}\right) \\
&= B_{d_\infty}\left(\rho_n(\sigma_Y(y)), \frac{1}{2n-2np}\right)
\end{aligned}$$

Therefore

$$\begin{aligned}
CP_Y(n,p)(y) &= \sigma_Y^{-1} B_{d_\infty}\left(\rho(\sigma_Y(y)), \frac{1}{2n-2np}\right) \\
&= B_{\sigma_Y^* d_\infty}\left((\sigma_Y^{-1} \circ \rho \circ \sigma_Y)(y), \frac{1}{2n-2np}\right)
\end{aligned}$$

□

Theorem 1 suggests how we can construct the hypercubes of the standard cubical cover as open balls under a scaled L_∞-distance. This insight leads to an immediate improvement in constructing the standard cubical cover: first, a vp-tree T is built using the scaled L_∞-distance. Then, after identifying all the hypercubes and their centers (noting that some centers may not correspond to points in the dataset), the points within each hypercube can be efficiently retrieved using range queries on T centered at these points (see Algorithm 6).

Algorithm 6 Standard Cubical Cover via vp-trees

```
Require: Let $Y \subseteq \mathbb{R}^k$, let $0 < n \in \mathbb{N}$ and $p \in (0, 0.5]$.
Ensure: The standard cubical cover $\mathcal{CC}_Y^{n,p}$.
  $T \leftarrow build\_vptree(Y, \sigma_Y^* d_\infty)$                          ▷ Algorithm 3
  $L \leftarrow (\sigma_Y^{-1} \circ \rho_n \circ \sigma_Y)(Y)$
  $\mathcal{C} \leftarrow \left\{ range\_query\left(T, l, \frac{1}{2n-2np}\right) \,\middle|\, l \in L \right\}$     ▷ Theorem 1, Algorithm 4
  return $\mathcal{C}$
```

Remark 8. *It is important to note that vp-trees were our first choice due to their flexibility, as they can be used in any metric or pseudo-metric space. However, when Y is a Euclidean domain contained within $\mathbb{R}^k$, it may be beneficial to explore other data structures that offer efficient spatial search. One such structure is R-trees, which are particularly well-suited for constructing the hypercubes of cubical covers (Guttman, 1984).*

While this improvement is significant, it is still insufficient. Although it reduces the number of steps compared to Algorithm 2, a single point in the dataset may still lie in the intersection of up to 2^k open hypercubes. This detail is often overlooked but has critical implications: the standard cubical cover could, in principle, contain more open sets than there are points in the dataset. This issue becomes particularly pronounced in higher dimensions, where such open covers tend to produce Mapper graphs that are too complex to provide meaningful insights.

3.1 Estimating Cardinality

As a consequence of Theorem 1, we developed a more efficient method for constructing the elements of $\mathcal{CC}_Y^{n,p}$ using vp-trees. In this subsection, we provide an estimate of the cardinality of $\mathcal{CC}_Y^{n,p}$, which also allows us to assess the overall complexity of its construction. Theorem 2 establishes an upper bound on the cardinality of a minimal subcover of $\mathcal{CC}_Y^{n,p}$, while Corollary 1 extends this result to derive an upper bound on the cardinality of $\mathcal{CC}_Y^{n,p}$.

Theorem 2. *Let $Y \subseteq \mathbb{R}^k$. Then, there exist a subcover $\mathcal{C} \subseteq \mathcal{CC}_Y^{n,p}$ with cardinality*

$$|\mathcal{C}| \leq \left(2n \cdot \frac{2-p}{p}\right)^{\dim(Y)}.$$

Proof. Initially we establish some notation that will make the proof easier. Let $\delta = \sigma_Y^* d_\infty$ and let $\psi_n = \sigma_Y^{-1} \circ \rho_n \circ \sigma_Y$. Under the metric δ, Y is contained in a k-dimensional hypercube of side 1, and ψ_n acts as an approximation function that maps Y to a regular grid of side $\epsilon_n = \frac{1}{2n}$. Then, as a consequence of Theorem 1, the collection $\mathcal{CC}_Y^{n,p}$ consists of the balls $B_\delta^Y(\psi_n(y), r_n)$ for each $y \in Y$, where the radius is $r_n = \frac{1}{2n-2np}$.

First, it's easy to observe that $\delta(y, \psi_n(y)) \leq \epsilon_n$ for every $y \in Y$ and every n. Therefore every ball $B_\delta^Y(\psi_n(y), r_n)$ is contained within the ball $B_\delta^Y(y, r_n + \epsilon_n)$, which has the same center y but a larger radius to account for the approximation error introduced by ψ_n. Therefore, for any chosen m, this gives us our first inclusion:

$$B_\delta^Y(\psi_m(y), r_m) \subseteq B_\delta^Y(y, r_m + \epsilon_m).$$

By recursively applying the notion of doubling dimension, the ball $B_\delta^Y(y, r_m + \epsilon_m)$ can be iteratively covered by λ^s balls of radius $\frac{r_m+\epsilon_m}{2^s}$, where s is the depth of the iteration and λ is the doubling measure of Y under the metric δ. Therefore, we have:

$$B_\delta^Y(y, r_m + \epsilon_m) \subseteq \bigcup_{j=1}^{\lambda^s} B_\delta^Y\left(y_j, \frac{r_m + \epsilon_m}{2^s}\right),$$

where $\{y_j\}_j$ are the centers of the covering balls and λ is the doubling measure of Y. If we now consider any $n \geq m$, using the same argument as in the first inclusion, we can write

$$B_\delta^Y\left(y_j, \frac{r_m + \epsilon_m}{2^s}\right) \subseteq B_\delta^Y\left(\psi_n(y_j), \frac{r_m + \epsilon_m}{2^s} + \epsilon_n\right),$$

which holds for any choice of s. If we then chose s such that $\frac{r_m+\epsilon_m}{2^s} + \epsilon_n \leq r_n$ we can further claim that

$$B_\delta^Y\left(\psi_n(y_j), \frac{r_m + \epsilon_m}{2^s} + \epsilon_n\right) \subseteq B_\delta^Y\left(\psi_n(y_j), r_n\right).$$

The inequality $\frac{r_m+\epsilon_m}{2^s} + \epsilon_n \leq r_n$ can be easily solved in s, and gives $s \geq \log_2\left(\frac{n}{m} \cdot \frac{2-p}{p}\right)$, which holds when we set $s = \left\lceil \log_2\left(\frac{n}{m} \cdot \frac{2-p}{p}\right)\right\rceil$. After this, we can finally set $L = \lambda^s$ and give the following estimate:

$$L = \lambda^s = 2^{dim(Y)\cdot s} \leq 2^{dim(Y)\cdot\left[1+\log_2\left(\frac{n}{m}\cdot\frac{2-p}{p}\right)\right]} = \left(2 \cdot \frac{n}{m} \cdot \frac{2-p}{p}\right)^{dim(Y)}.$$

Summing up and chaining the inclusions together, we obtain the following

$$B_\delta^Y(\psi_m(y), r_m) \subseteq \bigcup_{j=1}^{L} B_\delta^Y\left(y_j, \frac{r_m+\epsilon_m}{2^s}\right) \subseteq \bigcup_{j=1}^{L} B_\delta^Y\left(\psi_n(y_j), r_n\right).$$

Finally, setting $m = 1$ and $I_j = B_\delta^Y\left(\psi_n(y_j), r_n\right)$, we have $L \leq \left(2n \cdot \frac{2-p}{p}\right)^{\mathrm{dim}(Y)}$ and

$$Y \subseteq B_\delta^Y(\psi_1(y), r_1) \subseteq \bigcup_{j=1}^{L} B_\delta^Y\left(\psi_n(y_j), r_n\right) = \bigcup_{j=1}^{L} I_j,$$

which concludes the proof. ☐

Corollary 1. *Let* $Y \subseteq \mathbb{R}^k$*, then*

$$|\mathcal{CC}_Y^{n,p}| \leq 3^k \cdot \left(2n \cdot \frac{2-p}{p}\right)^{dim(Y)}.$$

Proof. Theorem 2 states that is always possible to find a subcover $\mathcal{S} \subseteq \mathcal{CC}_Y^{n,p}$ where $|\mathcal{S}| \leq \left(2n \cdot \frac{2-p}{p}\right)^{dim(Y)}$. Since $\mathcal{S}$ covers Y, every other interval $I \in \mathcal{CC}_Y^{n,p}$ must intersect some interval in $\mathcal{S}$. Therefore we can write

$$\mathcal{CC}_Y^{n,p} = \bigcup_{I\in\mathcal{S}} \mathcal{A}_I,$$

where $\mathcal{A}_I = \{J \in \mathcal{CC}_Y^{n,p} \mid J \cap I \neq \emptyset\}$. It's easy to see that for dimensionality reasons $|\mathcal{A}_I| \leq 3^k$, therefore we can claim that

$$|\mathcal{CC}_Y^{n,p}| \leq |\mathcal{S}| \cdot 3^k \leq 3^k \cdot \left(2n \cdot \frac{2-p}{p}\right)^{dim(Y)},$$

and this concludes the proof. ☐

Theorem 2 asserts that a minimal subcover of $\mathcal{CC}_Y^{n,p}$ has cardinality bounded by a value that depends solely on n, p, and $dim(Y)$. This upper bound is *intrinsic* as it is independent from the dimension of the *feature space* $\mathbb{R}^k$. Conversely, the inequality in Corollary 1 is not intrinsic, as it also depends on k, yet it justifies why proximity-net runs in far fewer steps than n^k. Notably, this upper bound is a very rough estimation and could potentially be improved, as the factor 3^k is significantly higher than what is typically observed. While a smaller factor might be achievable, it remains unclear how such an improvement would be influenced by the specific dataset.

3.2 Construction via Proximity-Net

In this work, we introduce a generalization of ϵ-net that we call *proximity-net*. This modified algorithm is a greedy procedure that takes a single parameter, that we call *proximity function* (see Definition 11), and covers the dataset with a collection of sets.

Definition 11. *A proximity function on Y is a map $b\colon Y \to \mathcal{P}(Y)$ such that $p \in b(p)$ for any $p \in Y$.*

Remark 9. *The cubical proximity function $CP_Y(n, p)$ from Definition 10 is a proximity function according to Definition 11.*

The difference with respect to ϵ-net is that the sets obtained from proximity-net are built by applying the proximity function, and therefore are not required to be open balls (see Algorithm 7). This choice brings improved flexibility and allows to build diverse types of open covers by applying the same procedure to a properly chosen parameter. In this section we will see how we can obtain both the ball cover and a cubical cover using proximity-net. More importantly, deriving a cubical cover from proximity-net effectively addresses the flaw of Algorithm 2, as the number of open balls is expected to be significantly fewer than n^k (Corollary 1).

Algorithm 7 proximity-net

```
Require: Let $Y$ be a dataset, and let $b$ be a proximity function on $Y$.
Ensure: An open cover of $Y$
  $S \leftarrow Y$, as a set            ▷ $S$ is the set that tracks the points of $Y$ not covered yet
  $C \leftarrow \emptyset$
  while $S \neq \emptyset$ do
      Take a point $p \in S$            ▷ Randomly or according to some heuristic
      $B \leftarrow b(p)$
      Add $B$ to $C$
      for $q \in B$ do                  ▷ All the points in $B$ are now covered
          Remove $q$ from $S$
      end for
  end while
  return $C$
```

It's worth to point out that the original ϵ-net can be obtained by supplying proximity-net with the *ball proximity function* defined as in Definition 12, and further optimize it using vp-trees (see Algorithm 8).

Definition 12. *Let $Y \subseteq Y'$ and let d be a pseudo-metric on Y'. For each $\epsilon > 0$ we define the function $BP_Y(d, \epsilon)\colon Y' \to \mathcal{P}(Y)$ by setting*

$$BP_Y(d, \epsilon)\colon y \mapsto Y \cap B_d(y, \epsilon).$$

for every $y \in Y'$. Moreover, the restriction of $BP_Y(d, \epsilon)$ on Y is a proximity function that we call ball proximity function.

Remark 10. *Definition 12 allows to use the same notation BP_Y when we want to construct a ball with a center that is not contained in Y, but in an eventually larger space Y'. We will use this in Theorem 1.*

We can improve ϵ-net algorithm by first building a vp-tree T on the dataset to be covered. After that we can call proximity-net (Algorithm 7) by supplying a function that for each point p performs a range query on T. This approach (Algorithm 8) is eventually faster than the original ϵ-net approach.

Algorithm 8 ϵ-net via proximity-net and vp-trees

```
Require: Let $Y$ be a dataset and $d$ a pseudo-metric on $Y$. Let $\epsilon > 0$ be a chosen radius.
Ensure: A ball cover on $Y$.
  $T \leftarrow build\_vptree(Y, d)$                                  ▷ Algorithm 3
  $\pi \leftarrow p \mapsto range\_query(T, p, \epsilon)$             ▷ Definition 12, Algorithm 4
  $C \leftarrow proximity\text{-}net(X, \pi)$                         ▷ Algorithm 7
  return $C$
```

Remark 11. *The ability to work with pseudo-metrics, rather than just metrics, is an invaluable feature of vp-trees that we can leverage in our implementation. In the setting of Mapper on Ball Mapper, we have a lens* $f\colon X \to Y$ *and a metric* d *on* Y*. Mapper on Ball Mapper is obtained by taking the pullback of the open sets of Ball Mapper under the lens* f*. This is equivalent to apply Algorithm 8 to the input dataset* $Y = X$ *under the pullback pseudo-metric* f^*d*. This brings a practical benefit in terms of time and space, since the pullback cover is already obtained in this way, without constructing it explicitely from a cover on* $f(X)$*.*

Under an appropriate choice of proximity function, we can construct a cubical cover using proximity-net, while keeping the number of open sets limited, as in the case of ϵ-net (see Remark 4), eliminating the performance degradation encountered in Algorithm 2.

Remark 12. *As a result of Theorem 1, we can claim that*

$$y \mapsto B_{\sigma_Y^* d_\infty}\left((\sigma_Y^{-1} \circ \rho_n \circ \sigma_Y)(y), \frac{1}{2n - 2np}\right)$$

is a proximity function.

By leveraging Theorem 1 we can finally summarize our methodology for computing this cover efficiently (see Algorithm 9):

1. First we use Algorithm 3 to construct a vp-tree T on Y using the pseudo-metric $\sigma_Y^* d_\infty$. The range query method on T (Algorithm 4) is then equivalent to computing the proximity function $BP_Y\left(\sigma_Y^* d_\infty, \epsilon\right)$ for any choice of ϵ.

2. Once T has been constructed, we run proximity-net algorithm (Algorithm 7) by supplying the proximity function $BP_Y\left(\sigma_Y^* d_\infty, \frac{1}{2n-2np}\right)(\sigma_Y^{-1} \circ \rho_n \circ \sigma_Y)$ which by definition is equivalent to

   $$y \mapsto B_{\sigma_Y^* d_\infty}\left((\sigma_Y^{-1} \circ \rho_n \circ \sigma_Y)(y), \frac{1}{2 - 2p}\right),$$

   and therefore can be efficiently computed as

   $$y \mapsto range\_query\left(T, (\sigma_Y^{-1} \circ \rho_n \circ \sigma_Y)(y), \frac{1}{2n - 2np}\right)$$

   using the vp-tree T constructed in the previous step. As stated by Theorem 1, this is equivalent to computing $CP_Y(n, p)(y)$.

Algorithm 9 Cubical Cover via proximity-net and vp-trees

```
Require: Let $Y \subseteq \mathbb{R}^k$, let $0 < n \in \mathbb{N}$ and $p \in (0, 0.5]$.
Ensure: A cubical cover $\mathcal{C} \subseteq \mathcal{CC}_Y^{n,p}$.
    $T \leftarrow build\_vptree(Y, \sigma_Y^* d_\infty)$                                ▷ Algorithm 3
    $\pi = y \mapsto range\_query(T, (\sigma_Y^{-1} \rho_n \sigma_Y)(y), \frac{1}{2n-2np})$    ▷ Definition 10, Theorem 1, Algorithm 4
    $\mathcal{C} \leftarrow proximity\text{-}net(\pi)$                                    ▷ Algorithm 7
    return $\mathcal{C}$
```

Since proximity-net is a greedy algorithm that selects a distinct element of $\mathcal{CC}_Y^{n,p}$ at each step, estimating the cardinality of $\mathcal{CC}_Y^{n,p}$ also provides an upper bound on the total number of iterations of proximity-net. Consequently, the algorithm produces an open cover of Y, where each open set corresponds to one of the hyperrectangles from the original standard cubical cover $\mathcal{CC}_Y^{n,p}$. This refined open cover may contain fewer open sets than the original cubical cover, thanks to the application of proximity-net. Despite its smaller size, the cover remains sufficient to encompass the entire dataset. Moreover, the Mapper graph derived from this open cover retains its informativeness while being potentially easier to visualize and analyze, particularly in higher dimensions (see Figure 5).

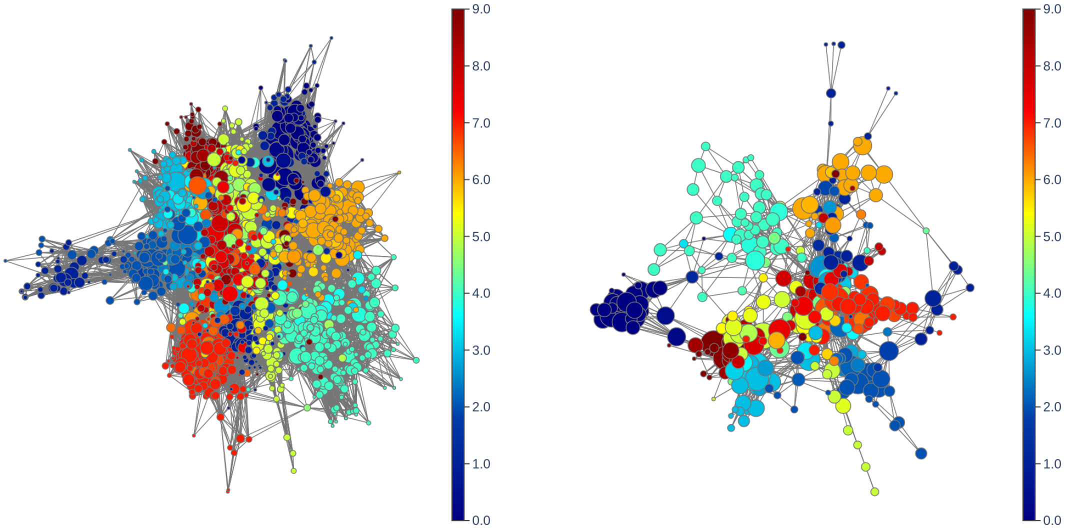


Figure 5: These plots show two Mapper graphs from the *Digits* dataset (1797 instances, 64 features) (Alpaydin and Kaynak, 1998). Both use PCA with four principal components as the lens, a cubical cover with five intervals and 50% overlap, and KMeans clustering with two clusters. The left plot uses the standard cubical cover, while the right uses the cubical cover generated by proximity-net. Node colors represent the average digit values. The right plot is more compact, easier to navigate, and better reveals relationships in data.

3.3 Experimental Results

To evaluate the benefits of the approach outlined in Algorithm 8 and supported by Theorem 2 and Corollary 1, we conducted a series of programmatic experiments. Initially, we developed a Python library called *library-omitted-for-anonymity* (Anonymous, 2024) based on the approach presented in this work. Subsequently, we compared it against other open-source libraries. The motivation behind creating *library-omitted-for-anonymity* was to explore alternative methods for constructing open covers for Mapper and eventually implement a more efficient approach. While major open-source implementations like *Python Mapper (v0.1.17)* (Müllner and Babu, 2013), *GUDHI (v3.9.0)* (Carrière, 2021), *Kepler Mapper (v2.0.1)* (Van Veen, 2019), and *giotto-tda (v0.6.0)* (Tauzin et al., 2021) can theoretically handle high-dimensional lenses, they all rely on Algorithm 2 which has known limitations, as previously discussed. The root cause of this issue lies in their source code: a common thread among these libraries is the use of the `itertools.product` function. This function, described in Python's official documentation available at `https://docs.python.org/3/library/itertools.html#itertools.product`, is used to perform a nested loop on each one-dimensional open cover, which corresponds to what Algorithm 2 does.

The results obtained from comparing *library-omitted-for-anonymity (v0.7.3)* (Anonymous, 2024) with *Kepler Mapper (v2.1.0)* (Van Veen, 2019) and *giotto-tda (v0.6.2)* (Tauzin et al., 2021) are reported as plots within this section and more extensively as tables in Appendix B fore reproducibility. These results align with the expected behavior and demonstrate a clear superiority of our approach in terms of scalability with respect to the lens dimension.

To evaluate the performance and scalability of our approach, we conducted a series of measurements to compute the Mapper graph's running time. During these benchmarks, we consistently kept constant number of intervals and overlap, while systematically varying the lens dimension. We compared the running time of three different libraries, namely *Giotto-TDA* (Tauzin et al., 2021), *Kepler Mapper* (Van Veen, 2019), and *library-omitted-for-anonymity* (Anonymous, 2024). This comparative analysis provides valuable insights into the behavior of these implementations when dealing with high-dimensional data. Our experiments were conducted on Fedora 37 using Python 3.10, leveraging *giotto-tda* 0.6.2, *Kepler Mapper* 2.1.0, and *library-omitted-for-anonymity* 0.7.3. All experiments have been run on a PC equipped with an i5-4590 CPU @ 3.30GHz with 4x4GB DDR3 1600Mhz, in dual channel configuration. To ensure the reliability of our benchmarks, we used well-known datasets publicly available at the UCI Machine Learning Repository (Dua and Graff). For each dataset we ran Mapper using overlap fraction p ranging in the set $\{0.125, 0.25, 0.5\}$ and using 10 intervals on each feature. This choice is arbitrary, but was enough to get informative Mapper graphs, especially with low-dimensional lenses, with every dataset we used. To minimize the effect of clustering on the benchmarks, we chose a trivial clustering algorithm that for each input dataset creates a single cluster. In all these benchmarks we excluded the running time used for plotting the Mapper graph.

The first experiment involves creating a 1-dimensional dataset embedded in dimension k, referred to as the *Line* dataset in plots and tables. This is a toy experiment where the dataset consists of 10000 points lying on the diagonal of the hypercube $[0,1]^k$, with a small random noise. As expected, compared with *kepler-mapper* and *giotto-tda*, the running time of *library-omitted-for-anonymity* on this dataset demonstrates the advantage of our approach especially in higher dimensions. Subsequently, we conducted additional experiments that more closely resemble the typical experiences one might have when using libraries for Mapper. To simplify the process, we used *Principal Component Analysis* (PCA) as the lens, with the number of components ranging from 1 to 5. As the number of PCA components k increases, we expect the difference with the doubling dimension of the image to increase too. Such limited range was enough to appreciate a remarkable performance advantage of *library-omitted-for-anonymity* across all our experiments. In the following plots, we present the running times with a linear scale on the main axes, complemented by a logarithmic scale on the inset plots. This dual-scale approach provides a clearer and more insightful representation of the data, allowing us to accurately capture both the overall trend and finer details of the running times. By using a logarithmic scale in the inset plots, we can effectively highlight the behavior of the running times over a wide range of values, making it easier to discern subtle variations that might be overlooked with a purely linear scale.

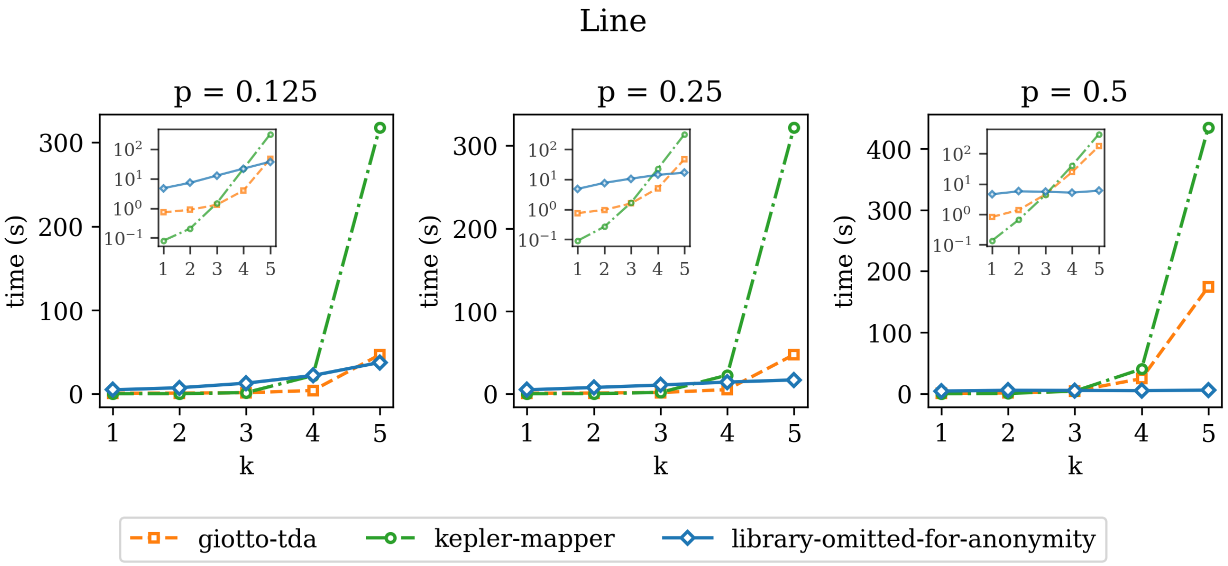


Figure 6: *Line* dataset: 10000 instances, variable number of features (k).

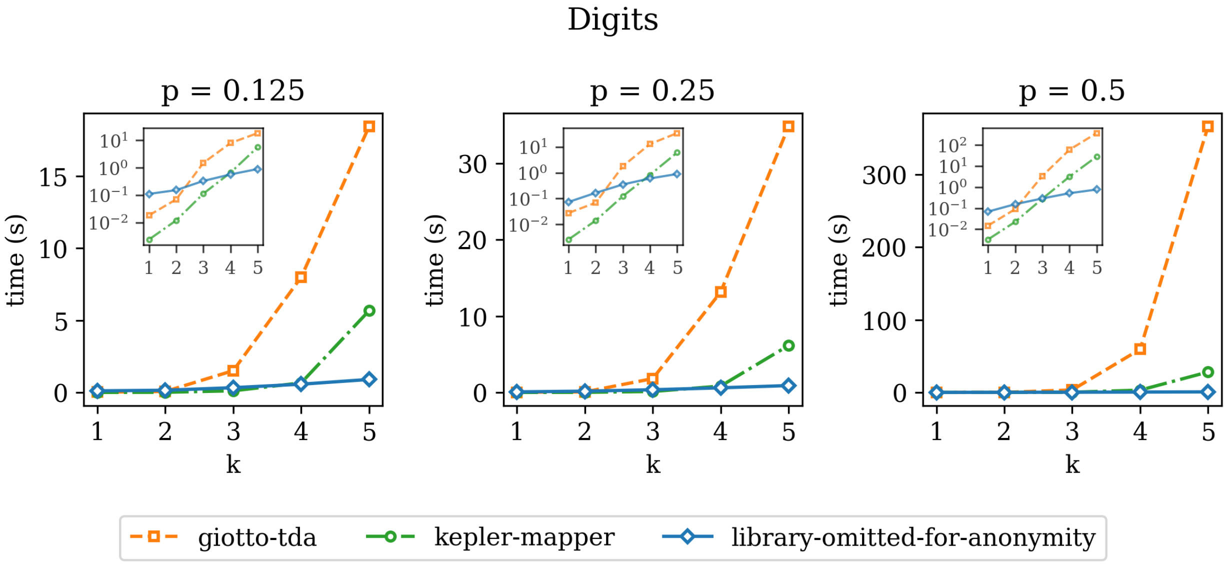


Figure 7: *Digits* dataset: 1797 instances, 64 features (Alpaydin and Kaynak, 1998). PCA with variable number of principal components (k).

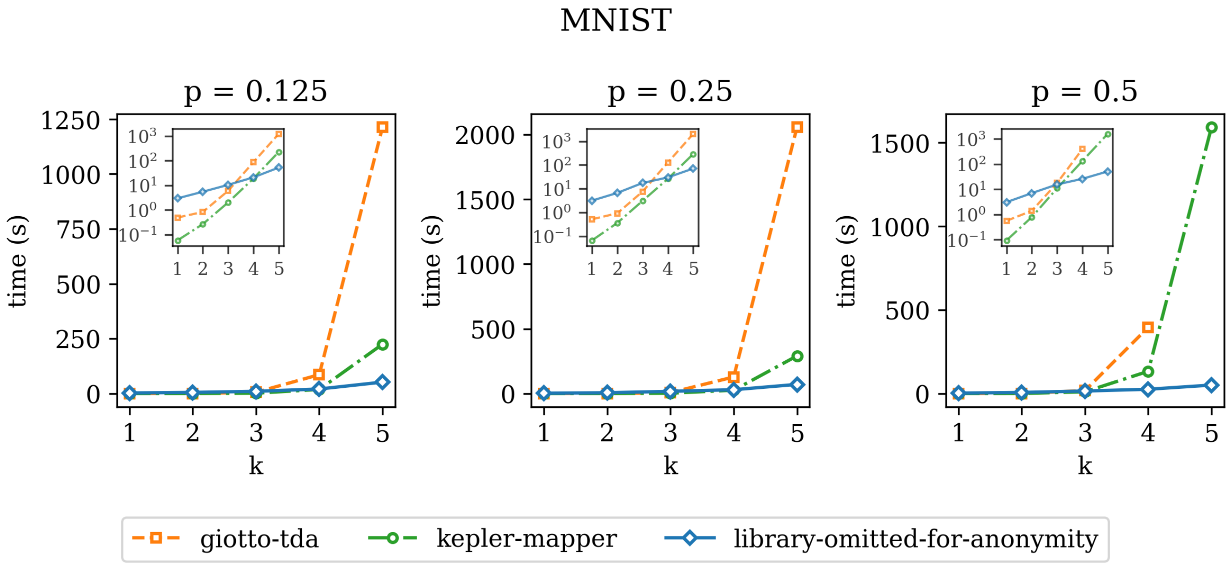


Figure 8: *MNIST* dataset: 70000 instances, 784 features (LeCun and Cortes, 2010). PCA with variable number of principal components (k).

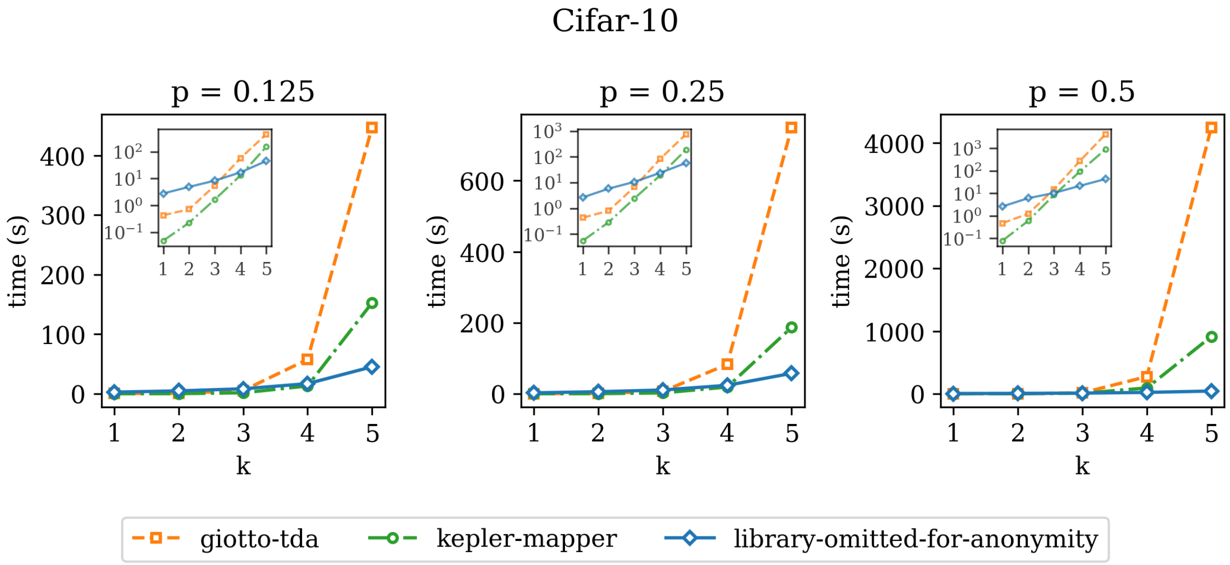


Figure 9: *Cifar-10* dataset: 60000 instances, 1024 features (Krizhevsky, 2009). PCA with variable number of principal components (k).

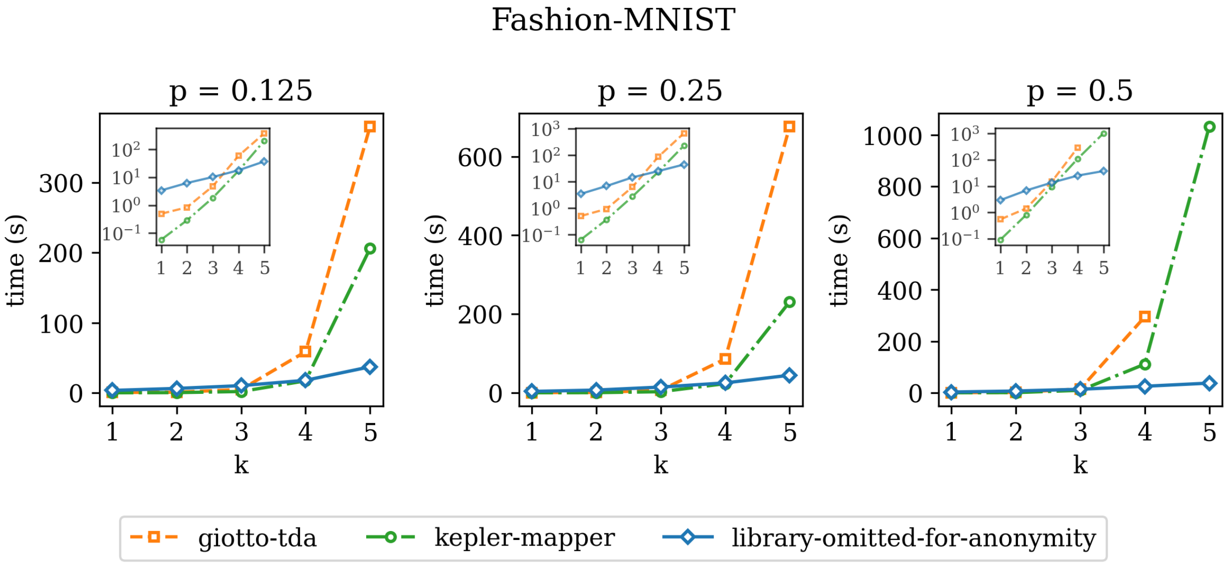


Figure 10: *Fashion-MNIST* dataset: 70000 instances, 784 features (Xiao et al., 2017). PCA with variable number of principal components (k).

References

E. Alpaydin and C. Kaynak. Optical Recognition of Handwritten Digits. UCI Machine Learning Repository, 1998. DOI: https://doi.org/10.24432/C50P49.

A. Anonymous. library-omitted-for-anonymity, July 2024. URL `link-omitted-for-anonymity`.

K. Borsuk. On the imbedding of systems of compacta in simplicial complexes. *Fundamenta Mathematicae*, 35(1):217–234, 1948. URL `http://eudml.org/doc/213158`.

S. Brin. Near neighbor search in large metric spaces. In *Proceedings of the 21th International Conference on Very Large Data Bases*, VLDB '95, page 574–584, San Francisco, CA, USA, 1995. Morgan Kaufmann Publishers Inc. ISBN 1558603794.

M. Carrière. Cover complex. In *GUDHI User and Reference Manual*. GUDHI Editorial Board, 3.4.1 edition, 2021. URL `https://gudhi.inria.fr/doc/3.4.1/group__cover__complex.html`.

M. Carrière, B. Michel, and S. Oudot. Statistical analysis and parameter selection for mapper. *Journal of Machine Learning Research*, 19(12):1–39, 2018. URL `http://jmlr.org/papers/v19/17-291.html`.

K. L. Clarkson. Nearest-Neighbor Searching and Metric Space Dimensions. In *Nearest-Neighbor Methods in Learning and Vision: Theory and Practice*. The MIT Press, 03 2006. ISBN 9780262256957. doi: 10.7551/mitpress/4908.003.0005. URL `https://doi.org/10.7551/mitpress/4908.003.0005`.

D. Dua and C. Graff. Uci machine learning repository. University of California, Irvine, School of Information; Computer Sciences. URL `http://archive.ics.uci.edu/`.

P. Dłotko. Ball mapper: a shape summary for topological data analysis, 2019.

P. Dłotko, D. Gurnari, and R. Sazdanovic. Mapper-type algorithms for complex data and relations, 2023.

J. H. Friedman, J. L. Bentley, and R. A. Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Trans. Math. Softw.*, 3(3):209–226, sep 1977. ISSN 0098-3500. doi: 10.1145/355744.355745. URL `https://doi.org/10.1145/355744.355745`.

C. Geniesse, S. Chowdhury, and M. Saggar. NeuMapper: A scalable computational framework for multiscale exploration of the brain's dynamical organization. *Network Neuroscience*, 6(2): 467–498, 06 2022. ISSN 2472-1751. doi: 10.1162/netn_a_00229.

T. F. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science*, 38:293–306, 1985. ISSN 0304-3975. doi: https://doi.org/10.1016/0304-3975(85)90224-5. URL `https://www.sciencedirect.com/science/article/pii/0304397585902245`.

A. Guttman. R-trees: a dynamic index structure for spatial searching. *SIGMOD Rec.*, 14(2): 47–57, June 1984. ISSN 0163-5808. doi: 10.1145/971697.602266. URL `https://doi.org/10.1145/971697.602266`.

A. A. Hagberg, D. A. Schult, and P. J. Swart. Exploring network structure, dynamics, and function using networkx. In G. Varoquaux, T. Vaught, and J. Millman, editors, *Proceedings of the 7th Python in Science Conference*, pages 11 – 15, Pasadena, CA USA, 2008.

C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, Sept. 2020. doi: 10.1038/s41586-020-2649-2. URL `https://doi.org/10.1038/s41586-020-2649-2`.

J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3): 90–95, 2007. doi: 10.1109/MCSE.2007.55.

P. T. Inc. Collaborative data science, 2015. URL `https://plot.ly`.

R. Krauthgamer and J. R. Lee. Navigating nets: simple algorithms for proximity search. In *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '04, page 798–807, USA, 2004. Society for Industrial and Applied Mathematics. ISBN 089871558X.

A. Krizhevsky. Learning multiple layers of features from tiny images. 2009. URL `https://api.semanticscholar.org/CorpusID:18268744`.

Y. LeCun and C. Cortes. MNIST handwritten digit database. 2010. URL `http://yann.lecun.com/exdb/mnist/`.

D. Müllner and A. Babu. Python mapper: An open-source toolchain for data exploration, analysis, and visualization. 2013. URL `http://danifold.net/mapper`.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, and M. Blondel. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

G. Singh, F. Memoli, and G. Carlsson. Topological Methods for the Analysis of High Dimensional Data Sets and 3D Object Recognition. In M. Botsch, R. Pajarola, B. Chen, and M. Zwicker, editors, *Eurographics Symposium on Point-Based Graphics*. The Eurographics Association, 2007. ISBN 978-3-905673-51-7. doi: 10.2312/SPBG/SPBG07/091-100.

G. Tauzin, U. Lupo, L. Tunstall, J. B. Pérez, M. Caorsi, A. M. Medina-Mardones, A. Dassatti, and K. Hess. giotto-tda: A topological data analysis toolkit for machine learning and data exploration. *Journal of Machine Learning Research*, 22(39):1–6, 2021. URL `http://jmlr.org/papers/v22/20-325.html`.

J. K. Uhlmann. Satisfying general proximity / similarity queries with metric trees. *Information Processing Letters*, 40(4):175–179, 1991. ISSN 0020-0190. doi: https://doi.org/10.1016/0020-0190(91)90074-R. URL `https://www.sciencedirect.com/science/article/pii/002001909190074R`.

e. a. Van Veen. Kepler mapper: A flexible python implementation of the mapper algorithm. *Journal of Open Source Software*, 4(42):1315, 2019. doi: 10.21105/joss.01315.

A. Weil. Sur les théorèmes de de rham. *Commentarii mathematici Helvetici*, 26:119–145, 1952. URL `http://eudml.org/doc/139040`.

H. Xiao, K. Rasul, and R. Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017.

P. N. Yianilos. Data structures and algorithms for nearest neighbor search in general metric spaces. In V. Ramachandran, editor, *Proceedings of the Fourth Annual ACM/SIGACT-SIAM Symposium on Discrete Algorithms, 25-27 January 1993, Austin, Texas, USA*, pages 311–321. ACM/SIAM, 1993. URL `http://dl.acm.org/citation.cfm?id=313559.313789`.

A Appendix A: Library Overview

Throughout the development process of *library-omitted-for-anonymity*, one of the objectives was to create an API that is easy to understand and use. For this reason we adopted an object-oriented approach taking inspiration from the well-known *scikit-learn* APIs (Pedregosa et al., 2011), since we expect some good level of familiarity with it from the intended user base of *library-omitted-for-anonymity*. Additionally, we made efforts to keep the API of *library-omitted-for-anonymity* similar to the APIs provided by *giotto-tda* and *Kepler Mapper*, allowing users to smoothly transition between these libraries and leverage their existing knowledge. By considering these factors, we aim to provide a user-friendly and seamless experience for users of *library-omitted-for-anonymity*, making it easier for them to explore and use the library's full potential.

We have implemented our own version of the vp-tree data structure and optimized it for our specific use-case: our implementation allows each leaf of the vp-tree to contain multiple items by stopping the construction when the splitting circle is *small*, either in terms of its cardinality or in terms of its radius (smaller than a given threshold). This optimization is beneficial both for range queries and for K-nearest neighbor (KNN) queries. When, during a search, the visited node becomes smaller than the query, the search operation collapses into a faster brute force linear scan.

The implementation of *library-omitted-for-anonymity* relies on several dependencies, including `networkx` (Hagberg et al., 2008), `numpy` (Harris et al., 2020), `matplotlib` (Hunter, 2007), and `plotly` (Inc., 2015). Overall, the software dependencies in *library-omitted-for-anonymity* are crucial for its functionality and enable users to generate Mapper graphs and visualize them effectively:

- `networkx` is used to generate and manipulate the Mapper graph, which is the primary result of the algorithm.
- `numpy` is necessary for numeric computations, particularly for the `CubicalCover` function.
- `matplotlib` and `plotly` are used to create plots for the Mapper graph, providing visualization options.

Additionally, there is a weaker dependency on `sklearn` (Pedregosa et al., 2011) which is used only for testing, ensuring that the implementation aligns with widely-used machine learning standards. The `sklearn` library is used to check that the custom-defined estimators in *library-omitted-for-anonymity* are compatible with `sklearn`. An extensive amount of effort was devoted to ensure a good level of automation during development, especially for testing, which is performed using GitHub actions. At the time of writing code coverage is around 96%.

For more in depth information, examples, tutorials, and documentation, the interested reader can visit `https://library-omitted-for-anonymity.readthedocs.io/en/main/`.

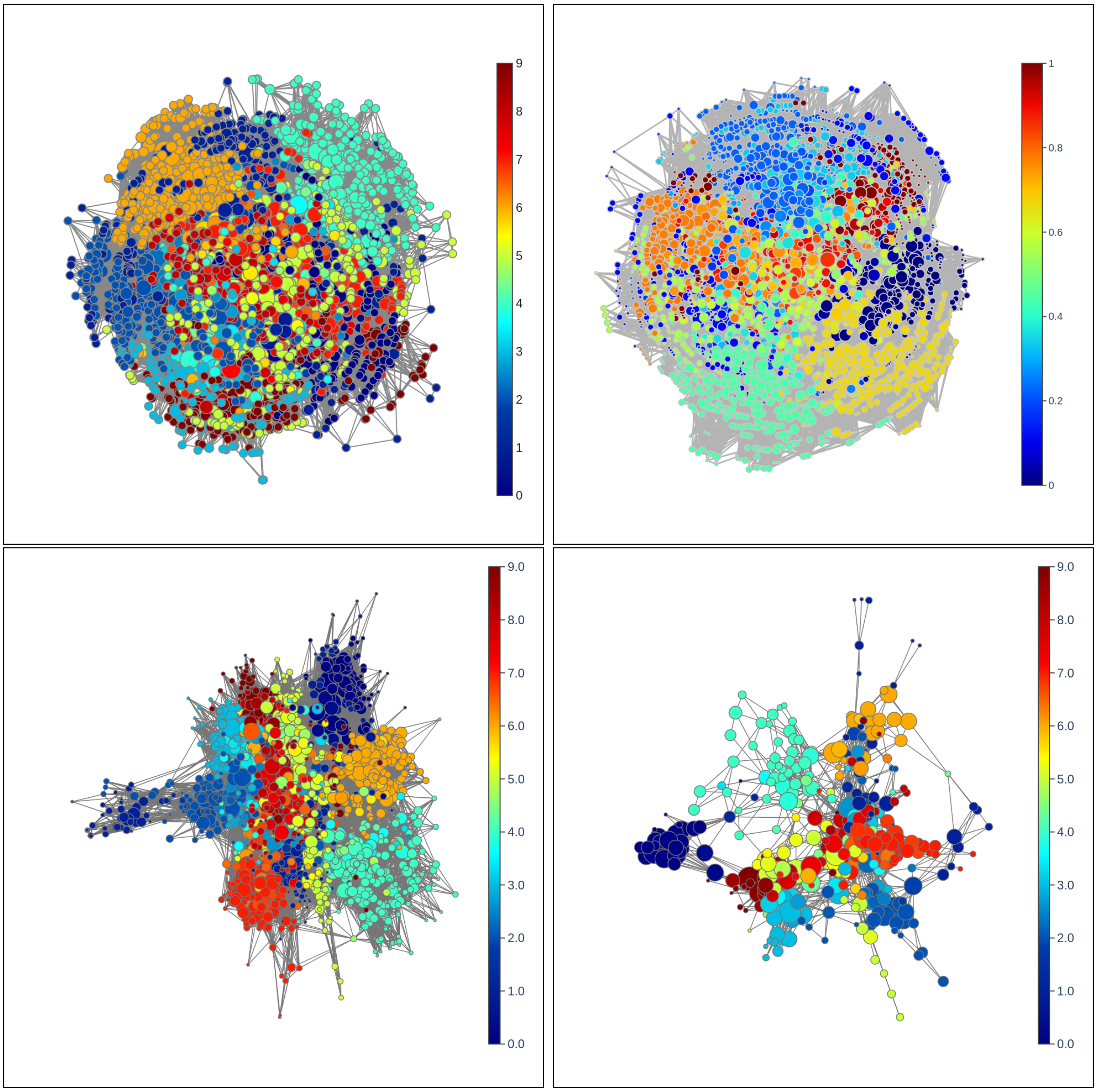


Figure 11: For comparison, the top row displays the Mapper graphs generated using *giotto-tda* and *kepler-mapper*, respectively. In these two cases the Mapper graphs contain more than 3000 nodes and are hard to read. The bottom row presents the results produced by *library-omitted-for-anonymity* for the standard cubical cover and the cubical cover based on proximity-net, respectively. All settings are consistent with those in Figure 5.

B Appendix B: Additional Tables

In the following tables all running time measurements are presented in seconds to provide a clear understanding of the performance. We used the following abbreviations: GT for *Giotto-TDA*, KM for *Kepler Mapper* and XX for *library-omitted-for-anonymity*. The symbol OOM means that an *Out Of Memory* event was encountered and the running time could not be registered.

Line									
	$p = 0.125$			$p = 0.25$			$p = 0.5$		
k	GT	KM	XX	GT	KM	XX	GT	KM	XX
1	0.73	0.08	4.83	0.75	0.09	4.61	0.81	0.14	4.69
2	0.88	0.21	7.2	0.97	0.27	8.48	1.51	0.66	5.36
3	1.31	1.46	11.73	1.61	1.76	10.31	4.63	4.27	5.51
4	4.12	22.14	22.44	5.21	22.78	13.3	24.44	40.0	5.44
5	45.43	319.23	33.91	47.39	318.04	16.3	173.17	433.16	5.61
6	OOM	4340.13	54.74	OOM	4461.53	19.06	OOM	5196.72	5.41
7	OOM	56692.15	87.32	OOM	56407.57	25.5	OOM	60917.2	6.13
8	OOM	OOM	121.6	OOM	OOM	27.64	OOM	OOM	5.96
9	OOM	OOM	188.21	OOM	OOM	36.54	OOM	OOM	5.9
10	OOM	OOM	224.08	OOM	OOM	39.2	OOM	OOM	6.29

Table 1: *Line* dataset: 10000 instances, variable number of features

Digits									
	$p = 0.125$			$p = 0.25$			$p = 0.5$		
k	GT	KM	XX	GT	KM	XX	GT	KM	XX
1	0.43	0.05	2.79	0.44	0.06	2.66	0.48	0.08	2.64
2	0.72	0.23	4.89	0.81	0.29	5.82	1.25	0.6	6.11
3	5.45	1.65	8.35	6.71	2.43	10.66	15.08	8.49	10.41
4	57.64	13.15	17.02	82.75	18.99	23.83	273.35	92.33	21.55
5	446.6	152.57	45.25	749.74	187.13	57.55	4247.41	913.06	43.5

Table 2: *Digits* dataset: 1797 instances, 64 features (Alpaydin and Kaynak, 1998)

MNIST									
	$p = 0.125$			$p = 0.25$			$p = 0.5$		
k	GT	KM	XX	GT	KM	XX	GT	KM	XX
1	0.49	0.06	2.97	0.51	0.07	3.15	0.56	0.09	3.06
2	0.83	0.27	5.49	0.92	0.36	6.65	1.39	0.78	7.01
3	5.98	2.02	10.43	7.6	3.08	17.5	17.71	10.91	15.51
4	86.26	18.89	20.95	128.32	26.35	29.94	395.66	133.18	25.97
5	1213.06	224.99	52.41	2055.24	290.85	71.1	OOM	1590.47	51.12

Table 3: *MNIST* dataset: 70000 instances, 784 features (LeCun and Cortes, 2010)

Cifar-10									
	$p = 0.125$			$p = 0.25$			$p = 0.5$		
k	GT	KM	XX	GT	KM	XX	GT	KM	XX
1	0.02	0.0	0.11	0.03	0.0	0.07	0.01	0.0	0.07
2	0.07	0.01	0.15	0.07	0.01	0.17	0.09	0.02	0.16
3	1.51	0.11	0.33	1.81	0.12	0.35	3.38	0.28	0.29
4	7.99	0.66	0.57	13.15	0.83	0.61	59.46	3.16	0.52
5	18.43	5.67	0.9	34.86	6.15	0.9	366.43	28.13	0.77

Table 4: *Cifar-10* dataset: 60000 instances, 1024 features (Krizhevsky, 2009)

Fashion-MNIST									
	$p = 0.125$			$p = 0.25$			$p = 0.5$		
k	GT	KM	XX	GT	KM	XX	GT	KM	XX
1	0.49	0.06	3.4	0.51	0.06	3.48	0.54	0.09	2.94
2	0.84	0.29	6.25	0.94	0.37	6.89	1.42	0.81	6.85
3	4.79	1.84	10.22	6.3	2.74	14.52	15.17	9.52	13.69
4	58.74	17.07	17.86	86.02	23.2	25.11	294.7	111.08	25.27
5	380.09	206.06	37.1	676.18	231.01	44.56	OOM	1032.43	37.52

Table 5: *Fashion-MNIST* dataset: 70000 instances, 784 features (Xiao et al., 2017)