Wasserstein distance for persistence diagrams

Jacobus Leander Conradi
Vincent Rheinthal

16. September 2019

Lab Report

supervisor: Prof. Dr. Rolf Klein

secondary supervisor: Dr. Elmar Langetepe

Mathematisch-Naturwissenschaftliche Fakultät der
Rheinischen Friedrich-Wilhelms-Universität Bonn

Inhaltsverzeichnis

Kapitel 1

The problem

1.1 Introduction

In this report we want to take a look at the comparison of different types of image data. For this we consider the concept of persistent homology. Persistent homology can best be described as the "recognition of structure while squinting". In essence, we want to return a complex-looking data set to basic geometric forms. In particular, this approach is very intuitive and close to the subconscious recognition of structures that are characteristic of a human.
We first want to calculate a so called persistence diagram for a given set of points X and then find a suitable distance measure for such diagrams.

1.2 Persistent Homology

In order to calculate homologies, we first need a manifold, or in the case of a computer a necessarily discrete representation, a cell complex. Now let $X \subset \mathbb{R}^d$ be an arbitrary set of points. We begin by defining Čech-complexes.

Definition 1 (Čech-Komplex)**.** *Let $X \subset \mathbb{R}^d$ be a finite set, and $\varepsilon > 0$ arbitrary but fixed. We construct the Čech-complex $\check{\mathrm{C}}_\varepsilon(X)$ as follows. The 0-skeleton is simply X itself. And any subset $\sigma \subset X$ is in the $(|\sigma| - 1)$-skeleton iff the intersection $\bigcap_{x\in\sigma} B_\varepsilon(x)$ of balls with radius ε around every point of σ is non-empty.*

Note, that for any finite X there is $\mathcal{E} > 0$, such that for any $\varepsilon \geq \mathcal{E}$ we have $\check{\mathrm{C}}_\varepsilon(X) = \check{\mathrm{C}}_\mathcal{E}(X)$. Call this stabilizing cell-complex $C(X) := \check{\mathrm{C}}_\mathcal{E}(X)$. Define a filtration f on $C(X)$ via $f : C(X) \to \mathbb{R}, \sigma \mapsto \min\{\varepsilon | \sigma \in \check{\mathrm{C}}_\varepsilon(X)\}$. Since X is finite, and hence $C(X)$, then $\mathrm{im}(f)$ must also be finite. Let $\mathrm{im}(f) = \{\varepsilon_i | 0 \leq i \leq N\}$. We now get a sequence of cell-complexes

$$\check{\mathrm{C}}_{\varepsilon_1}(X) \hookrightarrow \check{\mathrm{C}}_{\varepsilon_2}(X) \hookrightarrow \ldots \hookrightarrow \check{\mathrm{C}}_{\varepsilon_{N-1}}(X) \hookrightarrow \check{\mathrm{C}}_{\varepsilon_N}(X).$$

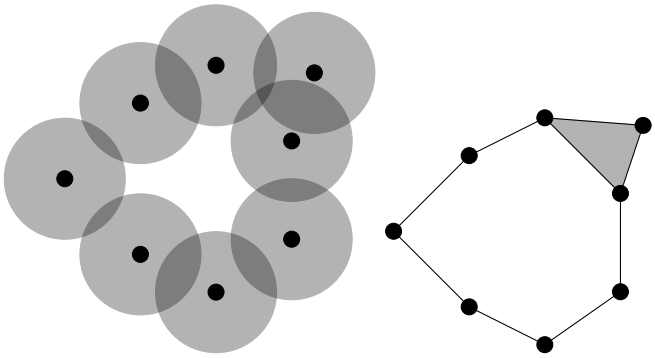

Abbildung 1.1: *Ein Čech-Komplex für ein gegebenes ε mit dem erzeugenden Datensatz.*

With this sequence we will now define persistent Homology.

Definition 2 (persistente Homologie)**.** *Let*

$$X_1 \overset{\iota_1}{\hookrightarrow} X_2 \overset{\iota_2}{\hookrightarrow} \ldots \overset{\iota_{n-2}}{\hookrightarrow} X_{n-1} \overset{\iota_{n-1}}{\hookrightarrow} X_n$$

be a sequence of cell-complex embeddings. For every $k \geq 0$ this implies a sequence of homology groups

$$H_k(X_1) \overset{\iota_1^*}{\to} H_k(X_2) \overset{\iota_2^*}{\to} \ldots \overset{\iota_{n-2}^*}{\to} H_k(X_{n-1}) \overset{\iota_{n-1}^*}{\to} H_k(X_n).$$

For every $0 \leq i < j \leq n$ we begin with $\mathcal{L}_i^j(X)_k$ the set of generators $\sigma \in coker(\iota_{i-1}^*)$, such that $\iota_{j-2}^* \circ \ldots \circ \iota_i^*(\sigma) \neq 0$ but $\iota_{j-1}^* \circ \ldots \circ \iota_i^*(\sigma) = 0$. Now to get the persistent homology from this, we need to apply the so called "elder rule". It may happen, that $\sigma_1 \in \mathcal{L}_i^l(X)_k$ and $\sigma_2 \in \mathcal{L}_j^l(X)_k$ exist (w.l.o.g. $i \leq j$), such that σ_1 equals σ_2 at some point t. In this case, we kill the "younger" one of the two, i.e. moving σ_2 to $\mathcal{L}_j^t(X)_k$, or removing completely, if $j = t$. We will call this reduced family $\{\mathcal{L}_i^j(X)_k | 0 \leq i < j \leq n, k \geq 0\}$ the persistent homology of $\{X_i\}$.

In essence persistent homology stores the lives and deaths of various generators, while increasing the maximum filtration value of elements in the cell complex.
Since in this report we limit ourselves to 2-dimensional point sets X, it is sufficient to limit the cell complex $C(X)$ to its 2-complex, since all higher homologies vanish.

One can geometrically define the calculation of these homologies for 2-dimensional data as follows. Let balls grow evenly around each point of X. Unite these and calculate the homology of the resulting manifold. This creates "holes" at different times, i.e. generators of the first homology, which disappear again at large ε.
We want to calculate this persistent homology for given sets of points efficiently, and will take a look at different approaches to calculate these.

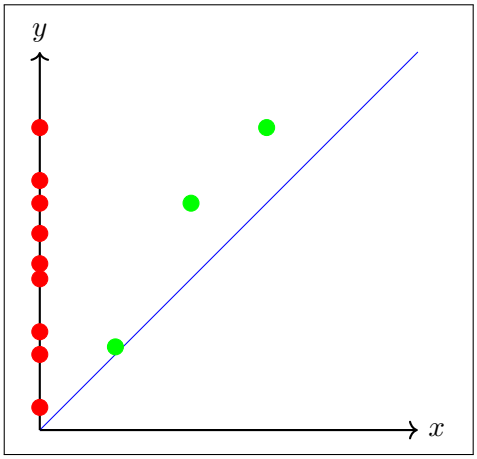

Abbildung 1.2: *Ein Persistenzdiagramm. Rote Punkte gehören zur nullten, und grüne Punkt zur ersten Homologie*

1.2.1 Persistence Diagrams

In order to work with persistent homology more intuitively we want to visualize these. A particular type of visualizatoin we want to work with are called persistence diagrams. Here we want to mark the point (t_1, t_2) in $\mathbb{R}^2$ for every creator who is born at a certain time t_1 and dies at a certain time t_2. In more general terms, for every $\sigma \in \mathcal{L}_i^j(X)_k$ mark $(\varepsilon_i, \varepsilon_j)$ in $\mathbb{R}^2$.

Note that the points marked in this way are always above the diagonal $\Delta = \{(x, x) \in \mathbb{R}^2 | x \in \mathbb{R}\}$, where points directly on the diagonal represent generators that die quickly after birth, and points far from the diagonal represent generators that live very long. Fig. 1.2 is an example for a persistence diagram which shows "holes" at three different times, i.e. generators of the first homology. Furthermore, one can see that the generators of the zeroth homology - i.e. one per connected component at every point in time - are all born at time 0.

1.3 Wasserstein Distance

Now let two such persistence diagrams be given. To define a distance measure, we consider the Wasserstein distance often used in probability theory. This is the "minimum effort to move one probability distribution to another". In our concrete application we define this as follows.

Definition 3 (Wassersteindistanz)**.** *Let two finite pointsets X and Y be*

given, with $|X| = |Y|$*. Then the Wasserstein distance is given by*

$$W_p(X,Y) = \min_{\varphi: X \to Y} \left(\sum_{x \in X} ||x - \varphi(x)||^p \right)^{\frac{1}{p}},$$

where the minimum is taken over all bijections $\varphi : X \to Y$*, with* $||\cdot||$ *being the* ℓ_2*-norm in* $\mathbb{R}^2$*.*

A special case $W_\infty(\cdot,\cdot)$ we will call the Bottleneck distance.
Notice the restraint $|X| = |Y|$. We want to adapt this such that differently sized sets can be compared with each other as well. For this we want to compare different approaches qualitatively.

1.4 Feature Proposals

As a final step, we will examine methods to find a representative set of points X for a given image to use as input for calculating persistent homology. Again, we want to compare different methods and compare them in the context of persistent homology.

Kapitel 2

Persistent Homology

In this chapter we want to focus on the implementation for the calculation of the Čech complex, as well as the calculation of persistent homology.

2.1 Čech-Complex

To calculate the Čech complex we first look at the Voronoi diagram for a given point set X in 2-dimensional space. Here $\mathbb{R}^2$ is divided into regions, so that in each region there is exactly one point x of X, and for every other point y of this region x is the nearest point of X to y.
From the Voronoi diagram we can extract the Čech complex. We add an edge between x and y from X to the cell complex when the Voronoi regions of x and y meet. The distance of x and y also gives us the filtration value $f(\{x, y\}) = \frac{||x-y||}{2}$ of the edge. If you add all these edges, you get the planar dual-graph, the DeLauney triangulation of the points from X.
It gets more complicated with faces, i.e. 2 cells. Here we have to make an important case distinction. If more than 2 Voronoi regions touch in a point, this corresponds to a circuit in the DeLauney trinagulation and a 2 cell in the Čech complex. But what exactly is the filtration value? This is where the following case distinction comes into play. Looking at the convex hull of the points $x_1, \ldots, x_k \in X$, whose Voronoi regions touch each other, it can happen that the point v where the Voronoi regions touch each other is inside or outside the polytope. If the point v is inside the polytope, the filtration value must be chosen as the distance of v to all x_i, which is the same for all x_i. Because as soon as this value is exceeded, the hole in the geometric presentation vanishes, hence the face must be inserted into the complex.
In the other case, there is no point in time where the circuit given by the points $x_1, \ldots, x_k$ in the geometric representation is "around a hole" - i.e. never is a generator -, so the face must be inserted as soon as the last edge closes the circle in $x_1, \ldots, x_k$. In other words, the filtration value is $max_{\{x_i,x_y\} \in C(X)} ||x_i - x_j||$. We want to call these 2 cells *degenerated*.

```
public class Voronoi {
   //stores voronoi vertices, where more than 2 regions touch
   private PointD[] vertices = null;
   private VEdge[] edges = null;
   ...
   private void compute(int width, int height) {
      VoronoiResults results =
          org.kynosarges.tektosyne.geometry.Voronoi.findAll(
      sites, new RectD(0, 0, width, height));
      vertices = results.voronoiVertices;
      //transform output of library to our own data types
      ...
   }
   ...
}
public class ActionGenerator {
   @NotNull
   public List<Action> generate() {
      //generates the list of elements added to the cell
          complex sorted by their filtration values
      ...
      voronoi.forEachVertex(this::computeVertex);
      voronoi.forEachEdge(this::computeEdge);
      actions.sort(Action::compareTo);
      return actions;
   }
   private void computeVertex(@NotNull PointD vertex, int
       index) {
      //create list of actions given the Voronoi Diagram
      ...
      VEdge[] edges = voronoi.getEdges(edgeIndices);
      PointD[] sites = getSites(edges);
      if (Util.isInside(vertex, sites)) {
         actions.add(new FaceAction(...));
         return;
      }
      ...
      actions.add(new EdgeFaceAction(...));
   }
}
```

Abbildung 2.1: *Codesnippet of the generation of* $C(X)$

For the implementation in Java we decided to use a library which calculates the Voronoi diagram. We calculate the planar dualgraph based on the library, as shown in Fig. 2.1 Furthermore, we see that in the case of degenerated 2-faces we use a `EdgeFaceAction` to insert the longest edge of the circle and the 2-cell in the same time step.

2.2 Calculation of persistent Homology

Since the 1 skeleton is the same as a graph, we save the complex as a graph, and remember which relations are generated by 2 cells. And because the second homology is always zero, due to our choice of 2-dimensional data, this suffices.

To respect the "elder rule", we need to make a choice for 0 cells which vertices are older than other vertices. Since the choice has no influence on the zero homology persistence diagram, we number the vertices randomly. We use this numbering when calculating the first homology as an orientation of the edges.

The main task in our implementation is the calculation of the first homology. For the zeroth we only have to remember whenever we add an edge whether the two vertices are in the same or different connected components. To maintain the elder rule for generators, we remember for each node which is the oldest one in each connected component. If the added edge is between two different connected components, we update for one of the connected components which the new oldest vertex is, and remember that a generator of the zeroth homology has died.

Since generators of the first homology live in the kernel of the boundary operator $H_1(C(X)) \to H_0(C(X))$, and these are just described by all circuits, we must find all possible circles in the graph, and then find out which circuits die via linear combinations of these. To find circuits, we check whenever we add an edge $e = (v, w)$ whether v and w are in the same context component. If this is the case, we find a v, w path P in $G \setminus \{e\}$. Then we remember $P \cup \{e\}$ as a new generator. At the time this generator is added, e is completely new and is not yet contained in any edge of a 2 cell or other circle, so this circle is definitely a living generator for a little time at least.

If a 2 cell is added, a relation is noted, namely the unique circle that describes the border of the 2 cell. Every time a 2 cell is added, we have to check if a circle dies. For this we check the kernel of the matrix:

$$\begin{Bmatrix} c_{11} & c_{21} & \dots & c_{n1} & r_{11} & \dots & r_{k1} \\ c_{12} & c_{22} & \dots & c_{n2} & r_{12} & \dots & r_{k2} \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ c_{1m} & c_{2m} & \dots & c_{nm} & r_{1m} & \dots & r_{km} \end{Bmatrix}$$

Where $c_{ij} = \sum_{e_j \in P_i} 1 - \sum_{e_j * \in P_i} 1$ and $r_{ij} = \sum_{e_j \in R_i} 1 - \sum_{e_j^* \in R_i} 1$, where

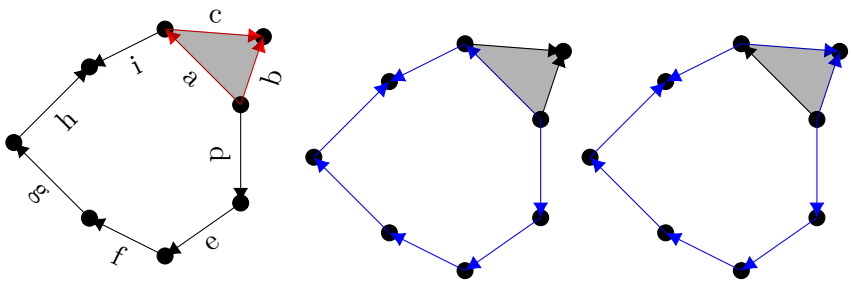

Abbildung 2.2: Beispiel der Kreisvektoren. In Rot der Kreis der Relation, in Blau zwei Kreise, die durch die rote Relation miteinander identifiziert werden

P_i is the generator circuits and R_i is the relation circuits, and e_j is an edge and e_j^* is the inverted edge.

An example in Fig. 2.2. We are given two circles in blue, and a relation in red. If we number the edges as in the first picture and select all circle orientations counter clockwise we get the following matrix:

$$A = \begin{Bmatrix} 1 & 0 & -1 \\ 0 & 1 & 1 \\ 0 & -1 & -1 \\ -1 & -1 & 0 \\ -1 & -1 & 0 \\ -1 & -1 & 0 \\ -1 & -1 & 0 \\ -1 & -1 & 0 \\ 1 & 1 & 0 \end{Bmatrix}$$

And thus gets a non-empty kernel, $A(1, 1, -1)^T = 0$ So we know that the two generators are identified with each other and can remove the younger one out of the generator list.

We now know when to add edges and faces and how to find all possible generators and how to find "dead" generators. Putting this together we are now able to calculate the first persistent homology.

2.2.1 Optimizations

Da uns dieses Verfahren zu langsam war (5 Sekunden für 100 Knoten, 20 Sekunden für 200 Knoten) haben wir das ganze optimiert, indem wir den Graphen zu geschickten Zeiten kontrahieren, um die Berechnungskomplexität niedrig zu halten. So erhalten wir fast lineare Laufzeit, mit 10 Sekunden für 1000 Knoten.

Kapitel 3

Distance Measures

First of all we want to extend the Wasserstein distance to differently sized sets. A first idea was to minimize $\varphi : X \to Y$ for $|X| < |Y|$ via injections $\varphi : X \to Y$, and to introduce an error term for every point in Y that is not hit. This seemed promising.
Define $\gamma : \mathbb{R}^2 \to \mathbb{R}$, with $(x, y) \mapsto y - x$. The motivation behind this definition is that this is exactly the vertical distance from a point (x, y) to the diagonal. Then we define

$$W'_p(X, Y) := \min_{\varphi: X \to Y} \left(\sum_{x \in X} |||x - \varphi(x)||^p + \sum_{y \in Y \setminus \in(\varphi)} \gamma(y)^p \right)^{\frac{1}{p}}.$$

This approach seemed promising at first, but the error term was too big. The problem is that $||x - \varphi(x)||$ is the ℓ_2 norm from the distance of x and $\varphi(x)$, whereas $\gamma(y)$ is the ℓ_1 norm from the distance of y and Δ. If you define $\gamma(x, y) = \frac{y-x}{\sqrt{2}}$ instead, you get a fair distribution. However we noticed that this distance is too restrictive. We also want to allow points from X not to be mapped. So now we define the final version of our modified Wasserstein distance, where we apply the error term to any point from X that is not mapped, as well as any point from Y that isnt hit.

$$V_p(X, Y) := \min_{Z \subset X} \min_{\varphi: Z \to Y} \left(\sum_{z \in Z} ||z - \varphi(z)||^p + \sum_{y \in (Y \setminus \mathrm{im}(\varphi)) \cup X \setminus Z} \gamma(y)^p \right)^{\frac{1}{p}}.$$

We call the bottleneck distance $B(X, Y) := V_\infty(X, Y)$. Since we now have three different Wasserstein distances, we want to make clear, the $V_p(\_, \_)$ will be the distance in question from now on.

3.1 Efficient Calculation

We quickly noticed that the whole problem is closely related to the Optimal Mapping problem. We want to make use of the knowledge we have of the

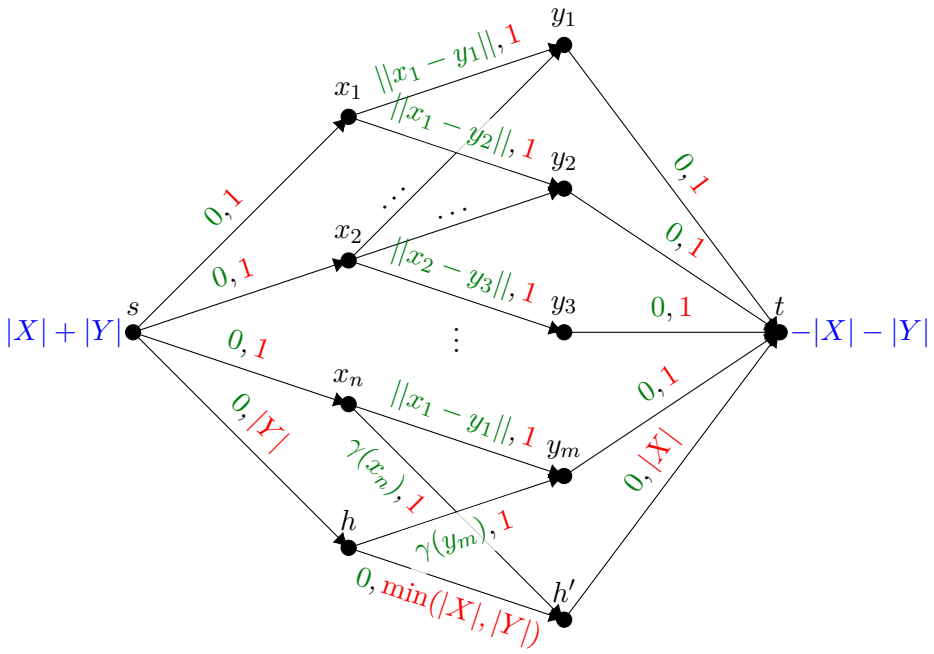

Abbildung 3.1: An example of a constructed graph. Flow requirements are blue, capacities red and costs green.

"Optimal Mapping" problem and present the problem of minimizing over exponentially many mappings as a flow problem. So for $V_p(X, Y)$ we define the following graph. $V := X \cup Y \cup s, t, h, h'$ and $E := (X \cup h) \times (Y \cup h') \cup s \times (X \cup h) \cup (Y \cup h') \times t$.

Weights are selected as follows. For edges in $s \times (X \cup h) \cup (Y \cup h') \times t \cup \{(h, h')\}$ they are always 0. For edges of the form $(x, y) \in X \times Y$ the costs are exactly $||x - y||$, and for edges of the form $(x, h') \in X \times \{h'\}$ resp. $(h, y) \in \{h\} \times Y$ we select $\gamma(x)$ or $\gamma(y)$. As capacities we choose $|Y|, |Y|$ and $\min(|X|, |Y|)$ for $(s, h), (h', t)$ and (h, h') respectively and 1 for all other edges. And finally the flow requirements $b(s) = -b(t) = |X| + |Y|$, and $b(v) = 0$ for all others. In order to solve the Wasserstein distance via a min-cost flow instance, we must first prove that for every feasible integer flow there is a $Z \subset X$ and map $\varphi : Z \to Y$, where the cost of the flow is equal to the error term of φ and vice versa. From this it follows that the cost of a min-cost-flow is equal to the Wasserstein distance. Note that this cost only applies to $p = 1$. For $1 < p < \infty$ exponentiate the cost of all edges by p and return the pth root of the cost at the end.

Lemma 1. *The cost of a min-cost-flow for the above graph $G = (V, E)$ is equal to the Wasserstein distance $V_p(X, Y)$.*

Beweis. Let $f : E \to \mathbb{Z}_{\geq 0}$ be a feasible flow. For Z choose all vertices x from X, such that $f((x, y)) = 1$ for a $y \in Y$ and set $\varphi(x) = y$ for this y. In less mathy terms, φ is given by the selected flow edges of f, between X and Y. Since the cost of the edges of the form (x, h') or (h, y) selected by the flow is exactly $\gamma(x)$ and $\gamma(y)$, $V_p(X, Y)$ is a lower bound for the min-cost-flow value.
For the other direction we consider for given Z, φ the flow given by $f(x, y) = 1$ iff $x \in Z$ and$\varphi(x) = y$. For all $x \in X \setminus Z$ we set $f((x, h')) = 1$ and for $y \in Y \setminus \mathrm{im}(\varphi)$ we set $f((h, y)) = 1$. Thus one receives a feasible flow, with the same costs. □

And as we know, the min-cost-flow problem can be calculated in polynomial time with an algorithm like Edmonds-Karp.
To calculate the Bottleneck distance, we consider the subgraph given by all edges with weights smaller than a given value. Since the Bottleneck distance is given by the cheapest edge, so that with all cheaper edges a feasible flow is still possible, one can determine the Bottleneck distance with logarithmically many calls of a min-cut algorithm with a procedure like binary search. This reduction follows from the fact, that the error term that is to be minimized in the Wasserstein distance is the same as the p-norm. The Bottleneck distance hence is the same as the ∞-norm, and is described by the maximum. Hence the Bottleneck distance reduces to finding the value $min_{\varphi: Z \subset X \to Y} max_{x \in Z} ||x - \varphi(x)||$.
This leaves us with the following approximations for runtime. Let X and Y again be arbitrary. Then the graph consists of $n = 2 + |X| + |Y|$ vertices and $m = 3 + 2|X| + 2|Y| + |X||Y|$ edges. Assuming a runtime of $O(nm^2)$ for Edmonds-Karp, we have a runtime of $O(|X|^2|Y|^3 + |X|^3|Y|^2)$ for the Wasserstein distance. For the Bottleneck distance we get a runtime of $O((\log(|X||Y|))(|X|^2|Y|^3 + |X|^3|Y|^2))$.

Kapitel 4

Feature Proposals

4.1 Feature Detection

4.2 Feature Selection

Conclusion