Sketching Persistence Diagrams

Donald R. Sheehy
North Carolina State University, Raleigh, NC, USA

Siddharth Sheth
North Carolina State University, Raleigh, NC, USA

Abstract

Given a persistence diagram with n points, we give an algorithm that produces a sequence of n persistence diagrams converging in bottleneck distance to the input diagram, the ith of which has i distinct (weighted) points and is a 2-approximation to the closest persistence diagram with that many distinct points. For each approximation, we precompute the optimal matching between the ith and the $(i+1)$st. Perhaps surprisingly, the entire sequence of diagrams as well as the sequence of matchings can be represented in $O(n)$ space. The main approach is to use a variation of the greedy permutation of the persistence diagram to give good Hausdorff approximations and assign weights to these subsets. We give a new algorithm to efficiently compute this permutation, despite the high implicit dimension of points in a persistence diagram due to the effect of the diagonal. The sketches are also structured to permit fast (linear time) approximations to the Hausdorff distance between diagrams – a lower bound on the bottleneck distance. For approximating the bottleneck distance, sketches can also be used to compute a linear-size neighborhood graph directly, obviating the need for geometric data structures used in state-of-the-art methods for bottleneck computation.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases Bottleneck Distance, Persistent Homology, Approximate Persistence Diagrams

Digital Object Identifier 10.4230/LIPIcs.SoCG.2021.57

Related Version *Full Version*: https://arxiv.org/abs/2012.01967

Funding This research was supported by the NSF under grant CCF-2017980.

1 Introduction

Persistent homology (PH) is a topological invariant with a built-in metric. Thus, qualitative shape information (topology) becomes quantitative (distances). This is why PH is so useful as a meta-analysis tool; it can map an entire data set to a single point in a metric space, i.e., a *persistence diagram* (PD). The complexity of computing the distance between PDs is determined by the complexity of the PDs themselves, which are multisets of pairs of numbers. The exact distance is computed by finding the minimum bottleneck matching between the sets. Naturally, smaller diagrams lead to faster computation.

In this paper, we will explore methods for sketching PDs, producing much smaller diagrams while maintaining some guaranteed proximity to the original PD. Given a PD D with n distinct (nondiagonal) points, we will produce a sequence of PDs $D_0, \ldots, D_n$ where each D_i has i distinct points. Let ε_i be the bottleneck distance $\mathrm{d}_B(D, D_i)$ for all i. The sequence has the property that $\varepsilon_0 \geq \varepsilon_1 \geq \cdots \geq \varepsilon_n = 0$. In other words, the sequence approaches D in the bottleneck distance. Moreover, the triangle inequality then gives a guarantee that for any PD X and all i

$$|\mathrm{d}_B(X, D) - \mathrm{d}_B(X, D_i)| \leq \varepsilon_i.$$

In addition to computing the sequence of diagrams, we will also compute the optimal matching $M_i : D_i \to D_{i+1}$. Thus, given a matching $M : X \to D_i$, we will be able to compute a matching $M_i \circ M : X \to D_{i+1}$ whose bottleneck cost is only increased by at most ε_{i+1}.

We will show how this can be used to efficiently reuse much of the work in computing a bottleneck distance to D_i when computing the distance to the finer approximation D_{i+1}. Surprisingly, the total space required to represent the sequence of diagrams as well as the sequence of matchings will only be $O(n)$. So, for a constant factor extra space, the PD can be stored in a way that allows for fast approximate distance computations.

Our approach is based on a greedy permutation (also known as a farthest-point traversal) of the points in the persistence diagram combined with a weighting scheme. For a PD with n points, we will produce n different approximations, where the ith approximation has i distinct weighted points. During construction, we precompute both the bottleneck distance between the successive approximations, but also the matching that realizes this distance. Thus, we have bounds on the error associated to any given approximation.

The main motivation for our work is to improve data analysis on spaces of persistence diagrams. In many proximity search problems, it is often less important to compute distances than it is to guarantee that distances are larger than some bound in order to prune a search. In simple metric spaces where distances are inexpensive to compute or are part of the input, this is usually accomplished by direct comparison. For expensive to compute metrics such as the bottleneck distance, we would benefit from fast approximations.

2 Background

The extended plane is the set $\mathbb{R}^2_\infty := (\mathbb{R} \cup \{\infty\})^2$. Let $p = (p_b, p_d)$ be a point in $\mathbb{R}^2_\infty$. The subscripts b and d on the coordinates refer to "birth" and "death" respectively. The distance between points $p, q \in \mathbb{R}^2$ is defined using the L_∞-norm

$$\|p - q\| = \max\{|p_b - q_b|, |p_d - q_d|\}.$$

The distance between between points in $\mathbb{R}^2_\infty$ is defined similarly with the usual care required for arithmetic on ∞.

A persistence diagram (PD) is a multiset of points in $\mathbb{R}^2_\infty$ that contains the diagonal $\{(x, x) \mid x \in \mathbb{R}^\infty\}$ with infinite multiplicity. For any multiset X, let $\underline{X}$ denote the underlying set, and for any $x \in \underline{X}$, let $\text{mult}(x)$ be the multiplicity of x in X.

Let A and B be PDs. A bijection $M : A \to B$ is called a *matching*. The *bottleneck cost* of M is $\max_{a \in A} \|a - M(a)\|$. The *bottleneck distance* between PDs A and B is defined to be the minimum bottleneck cost over all matchings $M : A \to B$.

Every matching $M : A \to B$ induces a *transportation plan*. This is a function $T : \underline{A} \times \underline{B} \to \mathbb{Z}$ that counts the number of edges between a point $a \in \underline{A}$ and $b \in \underline{B}$. More generally a function $T : \underline{A} \times \underline{B} \to \mathbb{Z}$ is a transportation plan between multisets A and B if for all $a \in \underline{A}$ and all $b \in \underline{B}$, we have

$$\sum_{b' \in \underline{B}} T(a, b') = \text{mult}(a) \text{ and } \sum_{a' \in \underline{A}} T(a', b) = \text{mult}(b).$$

The bottleneck cost of a matching is easily computed from the transportation plan. For this reason, it is not uncommon to see the bottleneck distance presented in terms of either. Both matchings and transportation plans will be useful in this paper. Matchings have the advantage that their composition is canonically defined. Transportation plans have the advantage that they are simpler to represent and therefore reduce space usage. Every transportation plan represents an equivalence class of matchings.

The bottleneck distance is a special case of the Wasserstein distance. For a given matching M, the p-Wasserstein cost is

$$\text{cost}_p(M) = \left(\sum_{x \in X} \mathrm{d}(x, M(p))^p \right)^{1/p}.$$

The bottleneck distance is the case of $p = \infty$.

2.1 Greedy Permutations

Given two subsets A and B in a metric space, the *Hausdorff distance* between A and B is defined to be

$$\mathrm{d}_H(A, B) := \max\{\sup_{a \in A} \inf_{b \in B} \mathrm{d}(a, b), \sup_{b \in B} \inf_{a \in A} \mathrm{d}(a, b)\}.$$

Let (X, d) be any metric space and let $P \subseteq X$ be finite. Let P be ordered $(p_0, \ldots, p_{n-1})$. The ith *prefix* of the ordering is denoted $P_i = \{p_0, \ldots, p_{i-1}\}$. The ordering is a *greedy permutation* if for all $i \in \{1, \ldots, n-1\}$,

$$\min_{q \in P_i} \mathrm{d}(p_i, q) = \mathrm{d}_H(P_i, P).$$

In other words, each point p_i is the farthest point from the prefix P_i. For this reason, greedy permutations are also known as farthest point samples. The point p_0 may be chosen arbitrarily.

For each p_i, the distance $\varepsilon_i = \mathrm{d}(p_i, P_i)$ is called the *insertion radius*. By convention $\varepsilon_0 = \infty$. By construction, for a greedy permutation, we have

$$\varepsilon_0 \geq \varepsilon_1 \geq \cdots \geq \varepsilon_{n-1}.$$

The *spread* Δ of a point set is the ratio of the largest to smallest pairwise distances. If the points are arranged in a greedy permutation, then the spread is at most $\frac{2\varepsilon_1}{\varepsilon_n}$. We define the spread for a persistence diagram similarly, except that we ignore multiplicity and treat the entire diagonal as a single point.

3 Related Work

The state-of-the-art for computing the bottleneck distance between persistence diagrams is the work of Kerber et al. [17]. They borrow the geometric insight from the work of Efrat et al. [12], in which the Hopcroft-Karp matching algorithm [16] is combined with geometric data structures to avoid constructing the entire bipartite graph. With this approach Efrat et al. reduce the execution time of the matching algorithm from $O(n^{2.5})$ to $O(n^{1.5} \log n)$. Kerber et al. use this idea to give a similar running time for computing the bottleneck distance of persistence diagrams.

Kerber et al. also adapt Bertsekas' auction algorithm [2] to find an approximate Wasserstein distance between diagrams, including the adaptation by Bertsekas and Castanon [3] that works with multiplicity.

Our approach is based on greedy permutations of the points as originally presented for clustering (see [14, 11]). An efficient algorithm to compute such permutations comes from the work of Clarkson [8] and his data structures for nearest neighbor search. Har-Peled

and Mendel [15] showed that Clarkson's algorithm runs in $2^{O(d)}n\log(\Delta)$ time and $O(2^{O(d)}n)$ space, where d is the doubling dimension and Δ is the spread. Our work extends Clarkson's algorithm and uses it to compute greedy permutations of PDs.

There are many novel applications of the bottleneck and Wasserstein distance, or its approximation. Fasy et al. [13] consider the problem of approximating nearest neighbors of PDs. Mumey [19] provides an approach to approximate nearest bottleneck distance queries over a finite point set by indexing a set of points to create a database and a *trie*-based data structure. Soler et al. [22] track topological changes in time by finding matchings in a lifted Wasserstein metric. Vidal et al. [23] provide an iterative algorithm to calculate progressively accurate approximations of the Wasserstein barycenters on a space of PDs. All of these approaches could benefit from faster distance computation.

There are several previous approaches that transforming PDs into another representation to make certain tasks computationally simpler. Bubenik [4] introduces one such form called persistence landscape which enables statistical inference via standard statistical tests. Adams et al. [1] show that PDs can be converted to a vector representation called a persistence image. Divol and Lacombe [10] give a framework to study PDs by converting them to partial optimal transport problems and expressing them as Radon measures on the upper half plane. Lacombe et al. [18] describe a scalable framework to compute standard properties of PDs by reformulating them as optimal transport problems.

Additionally, there has been significant work in representing PDs as vectors leading to a combination of machine learning theory with topological data analysis. Carrière et al. [6] define a stable topological point signature for shapes by extracting vectors from PDs. Many machine learning techniques require the underlying space to be a Hilbert space. It is shown by Reininghaus [20] that it is possible to use topological information encapsulated in PDs in all kernel-based machine learning methods. The kernel thus defined, however, performs poorly when encumbered with a large number of training vectors. Zeppelzauer [24] attempts to overcome these limitations by analysing 3D surfaces using persistence image descriptors of Adams et al. [1]. Carrière et al. [5] implement a provably stable sliced Wasserstein kernel and an algorithm to approximate it.

4 From Hausdorff to Bottleneck Approximations

The main idea of this paper is to use greedy permutations to give approximations to a persistence diagram. The greedy permutation is often used to give approximations that are close in Hausdorff distance. If A and B are PDs, then it is possible that $\mathrm{d}_H(\underline{A},\underline{B}) = 0$ and $\mathrm{d}_B(A,B)$ is large. Therefore, one should not, in general, expect that a good Hausdorff approximation will give a good bottleneck approximation. The same holds for other Wasserstein metrics.

There is one important case in which we *can* relate the Hausdorff distance and the Bottleneck distance – when one diagram is a subset of the other and the multiplicities of its points have been carefully adjusted. This section gives the construction and proves the equivalence of the Hausdorff and Bottleneck distances for that case.

Natural Reweighting

A *reweighting* of a PD A is a new persistence diagram A' formed by assigning new multiplicities to the points. Thus, if A is a reweighting of A, then $\underline{A} = \underline{A'}$.

Given $A \subseteq B$, the *natural reweighting* of A with respect to B is one that assigns a multiplicity to each point a in $\underline{A}$ according to the number of points in B having a as their nearest neighbor. That is, for $a \in \underline{A}$, we have $\mathrm{mult}(a)$ is the number of points of B that are closer to a than to any other point of $\underline{A}$. Ties can be broken arbitrarily, possibly resulting in non-uniqueness.

▶ **Lemma 1.** *Let A and B be PDs with $A \subseteq B$. If A' is a natural reweighting of A, then*

$$\mathrm{d}_B(A', B) = \mathrm{d}_H(\underline{A}, \underline{B}).$$

Proof. All points in $A \cap B$ will be matched to themselves. The points of $B \setminus A$ will be matched to their nearest neighbor. This exactly matches each point of A' with the correct multiplicity. This is a matching whose bottleneck is $\max_{b \in B} \mathrm{d}(b, A) = \mathrm{d}_H(A, B)$. This matching must be optimal, because every edge from $b \in B$ is the globally shortest possible to a point in A. ◀

Optimal Transport

The natural reweighting can be understood in terms of optimal transport. Specifically, for $A \subseteq B$, the natural reweighting A' corresponds to the transportation plan that moves every point b of B to its nearest point $\mathrm{NN}_{\underline{A}}(b)$ in $\underline{A}$. Formally, the transportation plan is

$$T(a, b) := \begin{cases} \mathrm{mult}(b) & \text{if } a = \mathrm{NN}_{\underline{A}}(b) \\ 0 & \text{otherwise.} \end{cases}$$

The transportation plan T is optimal for any Wasserstein metric. Any other transportation plan would suffer increased cost for each point of B that is not moved to its nearest neighbor.

5 Greedy Permutations of PDs

Given a PD, the diagonal and the multiplicity of points makes it impossible to give a greedy permutation directly. With a slight adjustment, we can define a greedy permutation of the nondiagonal points of a PD. Let D be a PD and let the nondiagonal points of $\underline{D}$ be ordered $p_0, \ldots, p_{n-1}$. The ith approximate diagram D_i is the natural reweighting of the ith prefix of the ordering with the diagonal added to make it a PD. So, D_0 is the empty diagram consisting of just the diagonal and D_n is D. The ordering is greedy if for all $i \in \{0, \ldots, n-1\}$,

$$\min_{q \in D_i} \|p_i - q\| = \mathrm{d}_H(\underline{D_i}, \underline{D}).$$

The sequence $(D_0, \ldots, D_n)$ is called a *greedy PD sketch* of D.

By always starting with the diagonal, the choice of p_0 is not arbitrary as is the case of greedy permutations in other metrics. The permutation will always start with p_0 as a point of maximum persistence. Also, it is not relevant to consider multiplicities when finding greedy permutations of PDs. Once all the distinct points have been added, the Hausdorff distance will be zero.

Because D_i is the natural reweighting with respect to D, the result is a sequence of bottleneck approximations to D. Up to a factor of two, these approximations are optimal for their size in the following sense.

▶ **Theorem 2.** *Let D be a persistence diagram and let $(D_0, \ldots, D_n)$ be a greedy PD sketch of D. For all i, let Opt_i be the PD with i distinct points that minimizes $\mathrm{d}_B(D, \mathrm{Opt}_i)$. Then, for all i, the approximation D_i satisfies*

$$\mathrm{d}_B(D, D_i) \leq 2\mathrm{d}_B(D, \mathrm{Opt}_i).$$

Proof. The proof follows the same pattern as the standard proof that greedy permutations yield 2-approximate k-centers in metric spaces [14, 11]. Let $r = \mathrm{d}_B(D, \mathrm{Opt}_i)$. Any point of D paired with the diagonal in the optimal matching with Opt_i can also be matched with

the diagonal in the matching with D_i. So, it will suffice to consider points of D matched to nondiagonal points. By construction, the distance between any pair of nondiagonal points in D_i is at least $\min_{q \in D_i} \|p_i - q\| = \mathrm{d}_H(\underline{D_i}, \underline{D})$. So, if any two points $a, b \in D_i$ are matched to the same nondiagonal point q in Opt_i, then, by the triangle inequality,

$$\mathrm{d}_H(\underline{D_i}, \underline{D}) \leq \|a - b\| \leq \|a - q\| + \|b - q\| \leq 2r.$$

So, we may assume that the bottleneck matching that realizes $\mathrm{d}_B(D, \mathrm{Opt}_i)$ matches each point of D_i with a unique point of Opt_i. Using the triangle inequality, every point of D is within $2r$ of a point of D_i. Therefore, we have shown that $\mathrm{d}_H(\underline{D_i}, \underline{D}) \leq 2r$, and so, by Lemma 1, it follows that $\mathrm{d}_B(D_i, D) \leq 2r$. ◀

The preceding theorem shows the sense in which the PD sketch is approximately optimal for its size. When used as a proxy for a full diagram, there is a clear upper bound on the error; using D_i instead of D introduces at most ε_i error as shown in the following lemma.

▶ **Lemma 3.** *Let X and D be persistence diagrams and let i be a nonnegative integer. Let D_i be the ith PD in a greedy PD sketch of D and let $\varepsilon_i = \mathrm{d}_H(\underline{D_i}, \underline{D})$. Then,*

$$|\mathrm{d}_B(X, D) - \mathrm{d}_B(X, D_i)| \leq \varepsilon_i.$$

Proof. Because D_i is the natural reweighting of a subset of D, Lemma 1 implies that $\mathrm{d}_B(D, D_i) = \mathrm{d}_H(\underline{D}, \underline{D_i}) = \varepsilon_i$. The desired inequality, then follows from the triangle inequality for the bottleneck distance. ◀

5.1 Transportation Plans

In addition to the approximate PDs, we also want to compute a matching or a transportation plan that take us from D_i to D_{i+1}. The difference between these PDs is

- the addition of point p_i,
- some mass moves from points in D_i to p_i, and
- some mass moves from the diagonal to p_i.

By the definition of the natural reweighting, the movement of mass is just a count of how many points of D have p_i as their nearest neighbor. By the triangle inequality in the plane, any such transportation plan must be optimal as the source and target of every unit of mass is known and the straight lines have minimal cost.

The natural reweighting of D_{i+1} is with respect to D rather than D_i. As a result, the bottleneck distance between D_i and D_{i+1} may be larger than the bottleneck distance between D_i and D. Specifically, we have

$$\varepsilon_i = \mathrm{d}_H(\underline{D_i}, \underline{D_{i+1}}) \leq \mathrm{d}_B(D_i, D_{i+1}) \leq \varepsilon_i + \varepsilon_{i+1}.$$

The sketch for a PD with n points contains $n + 1$ PDs and n transportation plans. With a little care, this can be represented in $O(n)$ space as shown in the following theorem.

▶ **Theorem 4.** *Given a persistence diagram D with n points, the greedy PD sketch and the optimal transportation plans can be represented in $O(n)$ space.*

Proof. Without the multiplicities, the sequence of diagrams is represented by the greedy permutation of the points. The multiplicities are encoded in the transportation plans from D_i to D_{i+1}. In such a transportation plan, all the mass that moves is shifted to the newly

added point p_i. Say that a point $q \in D_i$ is a neighbor of p_i if it is one of the points that shifts some of its mass to p_i. That is, there is some point $x \in D$ such that $NN_{D_i}(x) = q$ and $NN_{D_{i+1}}(x) = p_i$. By the greedy ordering,

$$\|x - p_i\| < \|x - q\| \leq \varepsilon_i.$$

Moreover, the greedy permutation guarantees that the neighbors of p_i are all ε_i-separated. So, the squares of side-length ε_i centered at the neighbors of p_i are all disjoint and contained in the square of side length $5\varepsilon_i$ centered at p_i. It follows that there are at most 25 neighbors including the diagonal. Thus, the ith transportation plan can be represented by a list of at most 25 neighbors along with an integer for each counting the amount of mass moved. In total this is $O(n)$ numbers to store. ◀

6 Modifying Clarkson Algorithm for Greedy Permutations of Persistence Diagrams

It is possible to compute the greedy permutation of a persistence diagram using the standard quadratic time algorithm [14, 11]. The naive approach is simply to treat the entire diagonal as the first point in the permutation. To get a faster algorithm, a simple and effective approach due to Clarkson can compute a greedy permutation in $O(n \log \Delta)$ time in low-dimensional metrics. However, the inclusion of the diagonal in a persistence diagram is an obstacle to direct application of Clarkson's algorithm. The reason is that treating the diagonal as a point breaks the triangle inequality, e.g., consider two points that are both close to the diagonal but far from each other (see Fig. 1). If one enforced the triangle inequality, the impact would be that the diagram could appear to have high doubling dimension; all n points could be one unit from the diagonal and thus equidistant (exactly two units) from each other. In this section, we will show how to modify the algorithm so that we can achieve the same $O(n \log \Delta)$ running time for persistence diagrams. The main insight is to augment the PD with its projection to the diagonal and modify the way distances are computed. We will also give an example where the direct application of Clarkson's algorithm devolves to quadratic time.

6.1 Overview of Clarkson's Algorithm

In his work on nearest neighbor search in metric spaces [7, 8, 9], Clarkson developed several data structures based on a kind of discrete Voronoi diagram. Points are inserted into the structure one at a time and the uninserted points are assigned to their nearest neighbors among the inserted points. The set of points whose nearest neighbor is a point p are called the *(discrete) Voronoi cell* of p and are denoted $\mathrm{Vor}(p)$. A neighborhood graph is constructed on the inserted points with edges between points p and q representing the possibility that inserting a point in $\mathrm{Vor}(p)$ would alter $\mathrm{Vor}(q)$ or vice versa. As points are inserted, updates to the structure remain local with respect to this neighborhood graph. Some extra edges are maintained to simplify construction, pruning only those between points whose distance is more than some constant times their radii.

The Voronoi cells provide exactly the information required to compute both the natural reweighting as well as the optimal transportation plans. The improvements in running time come from the sparsity of the neighborhood graph. That sparsity will also translate into the sparsity of the PD sketch.

The simplest version of Clarkson's data structure is called *sb* and was implemented in C. It was originally designed to use a random permutation of the input points, but after experimental analysis, it was shown that performance improved when using a greedy

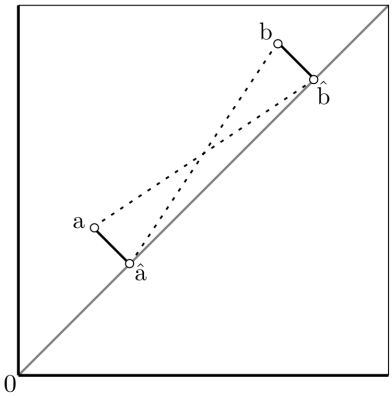

Figure 1 The triangle inequality fails when treating the diagonal as a point. If $\mathrm{d}(a,b) \gg \mathrm{d}(a,\hat{a})$ and $\mathrm{d}(a,b) \gg \mathrm{d}(b,\hat{b})$ then the triangle inequality fails as $\mathrm{d}(a,b) > \mathrm{d}(a,\hat{a}) + \mathrm{d}(\hat{a},b)$.

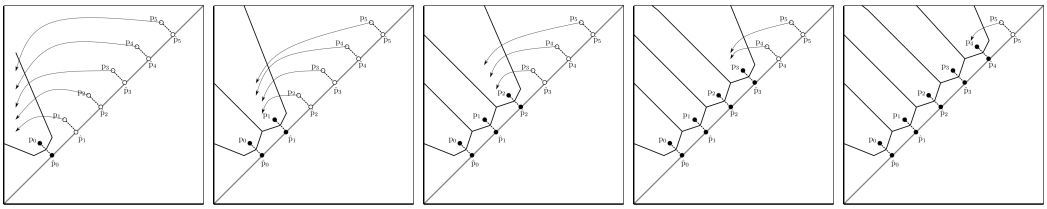

Figure 2 In this example, Clarkson's algorithm devolves to quadratic time. As points are added p_0 to p_5, all other points have to be checked to see if they move.

permutation [8]. The data structure also finds the greedy permutation efficiently as part of the construction: knowing the Voronoi cells of the points added so far makes it easy to add the next farthest point by storing the Voronoi cells in a priority queue. Later, Har-Peled and Medel showed that this algorithm runs in $O(n \log \Delta)$ time in doubling metrics. A Python implementation of this algorithm is also available [21].

Consider a PD in which all the points are on a line one unit from the diagonal and are spaced out two units apart. With a slight perturbation, any given ordering can be the greedy permutation. If the ordering is sorted along the line, then each new point would require checking all the other points to see if they move. Thus the total running time is quadratic, even though the spread is linear. This is illustrated in Fig. 2. The reason this seems to violate the $O(n \log \Delta)$ running time is that the big-O hides a term exponential in the doubling dimension. If the distances are replaced with shortest path distances to enforce the triangle inequality, the doubling dimension of this example becomes $\log n$ because the the path from a point to the diagonal plus the diagonal to another point results in all points equidistant. So, in order to compute the greedy permutation in subquadratic time using Clarkson's algorithm, one must treat the diagonal differently.

6.2 Using Projections

Let D be a persistence diagram. Let X be the nondiagonal points of D and let $\hat{X}$ be the projections of the points of X to the diagonal. For each $a \in X$, we will write $\hat{a}$ for its projection. Ultimately, we will compute the greedy permutation of $X \cup \hat{X}$ and retain only the ordering on X.

Let S be a subset of $X \cup \hat{X}$. The points $\hat{X}$ naturally partition the diagonal into segments where for any $\hat{x} \in S \cap \hat{X}$,

$$\mathrm{seg}_S(\hat{x}) := \{\hat{y} \mid \text{for all } \hat{z} \in S \cap \hat{X}, \|\hat{x} - \hat{y}\| \leq \|\hat{z} - \hat{y}\|\}.$$

When partitioning the points into discrete Voronoi cells, we will consider the Voronoi cells of the segments for the diagonal points. Formally,

$$\mathrm{d}_S(x, y) := \begin{cases} \min_{\hat{z} \in \mathrm{seg}_S(x)} \|\hat{z} - y\| & \text{if } x \in \hat{X} \\ \|x - y\| & \text{otherwise.} \end{cases}$$

This way of computing distances does not give a metric, but it will give the correct notion of discrete Voronoi cells, defined as

$$\mathrm{Vor}_S(x) := \{y \in X \cup \hat{X} \mid \text{ for all } z \in S, \mathrm{d}_S(x, y) \leq \mathrm{d}_S(z, y)\}.$$

Note that the addition of more points of $\hat{X}$ into S will not affect the Voronoi cells of points in $S \cap X$. This is the desired invariant because it means that the union of the Voronoi cells of the diagonal points only depends on the nondiagonal points. The points of $S \cap \hat{X}$ serve only to partition what would otherwise be one large cell associated to the diagonal.

The *projection distance* is

$$r(x, y) := \begin{cases} \|y - \hat{y}\| & \text{if } x \in \hat{X} \text{ and } y \in X \\ \|x - y\| & \text{otherwise.} \end{cases}$$

The *radius* of a Voronoi cell of a point $x \in S$ is

$$\mathrm{rad}(x) := \max_{y \in \mathrm{Vor}_S(x)} r(x, y).$$

The modified Clarkson algorithm then computes the greedy PD sketch by maintaining these Voronoi cells. The next point added at each step is the farthest point (in projection distance) in the Voronoi cell of maximum radius. The neighborhood graph connects two Voronoi cells as long as their distance is less than four times the larger of their radii.

The key to the analysis of Clarkson's algorithm is that the neighborhood graph maintains constant degree. One must keep enough edges so that the following two conditions hold.

1. The Voronoi cell of a newly inserted point is contained in the union of the cells of the neighbors of its parent, i.e., the nearest neighbor just prior to insertion.
2. The neighbors of a newly inserted point are contained in the union of the neighbors of neighbors of its parent just prior to insertion.

Our changes to how distances are computed produce only a small change to the analysis of Clarkson's algorithm, requiring us to store edges in the neighborhood graph up to four times the radius (rather three times the radius as in the original). The full version gives the details on why this small change is sufficient (by repeated use of the triangle inequality).

▶ **Theorem 5.** *The modified Clarkson algorithm restricted to the nondiagonal points gives a greedy PD sketch.*

Proof. It suffices to prove that each time a nondiagonal point p_i is added, it is the farthest point from D_i. The algorithm explicitly chooses the farthest point at each step, so we need only observe that the inclusion of the projections do not change the Voronoi cells of the nondiagonal points. In other words, the projections do not change the order in which the nondiagonal points are added. This follows from the definition of the projection distance. ◀

▶ **Theorem 6.** *The greedy PD sketch can be computed in $O(n \log \Delta)$ time.*

Proof. Using the projections, the analysis of the running time is the same as the standard analysis as given in Har-Peled and Mendel [15] and also in Clarkson [8]. The key idea is to count the number of times any point is considered for moving into a new Voronoi cell. By a volume packing argument, this can happen only a constant number of times before the maxmimum radius goes down by at least a factor of 2. The one caveat is that the projection could artificially increase the spread. This is resolved by stopping the algorithm as soon as all of the nondiagonal points are inserted. This happens at scale $\frac{1}{\Delta}$ times the diameter and therefore, the total running time is $O(n \log \Delta)$ as desired. ◀

A natural improvement in many instances would be to stop early and return a prefix of the PD sketch. In particular, if one has other sources of error – such as the grid size in persistence images [1] – one need not compute the full sketch. For constant precision, the running time will be linear.

▶ **Corollary 7.** *Let D be a PD, and let $R_{\max}$ be the maximum persistence of any point in D. For a fixed precision ε, a partial greedy PD sketch $(D_0, \ldots, D_k)$ can be computed such that*

$$\mathrm{d}_B(D_k, D) \leq \varepsilon$$

in $O(n \log \frac{R_{\max}}{\varepsilon})$.

7 Updating a Matching

Starting from a greedy PD sketch of D and a second PD X, an approximation D_i in the sketch can be used to estimate $\mathrm{d}_B(X, D)$. In doing so, an optimal bottleneck matching between D_i and X is computed. Choosing a larger value of i will increase the precision. In this section, we show how to bootstrap the computation that has already been done for D_i to get good matching between X and D_{i+1}. The principle idea is to compose a matching $X \to D_i$ with a matching $D_i \to D_{i+1}$ that is consistent with the transportation plan T_i. The result will not necessarily be the optimal matching $X \to D_{i+1}$, but it will satisfy the cost guarantee of Lemma 3. Thus, in many cases, most of the work of finding the bottleneck matching will already be done.

Naive Update

The transportation plan $T_i : \underline{D_i} \times \underline{D_{i+1}} \to \mathbb{Z}$ encodes an equivalence class of matchings. Any matching $M_i : D_i \to D_{i+1}$ that has $T_i(q, q')$ edges from q to q' is in this class. One can easily choose such a matching arbitrarily and compose it with the previously computed optimal matching $M : X \to D_i$. That is, $M_i \circ M : X \to D_{i+1}$ is a matching. Because M_i has bottleneck cost at most ε_i, the new matching will increase in cost by at most ε_i (by the triangle inequality).

A Short Auction

It is possible to choose a locally optimal matching update consistent with T_i. This is perhaps most easily understood in terms of the p-Wasserstein cost of the matching. Minimizing this cost is equivalent to minimizing the pth power of the cost. So, replacing one edge xq with an edge xp_i in a matching M' results in the following change.

$$\text{cost}_p(M')^p - \text{cost}_p(M)^p = \text{d}(x, p_i)^p - \text{d}(x, q)^p.$$

Thus, for each such edge xq, we compute the corresponding pth power change in cost. Then, we update the edges in order of this change. This resembles an abbreviated version of the auction algorithm [2].

For bottleneck matchings, one cannot assign costs using the pth power of the distances, because $p = \infty$ in that case. Instead, one can find the optimal matching from the neighbors of q in the matching M to the points q and p_i (with multiplicities). This only requires sorting the $O(n)$ edges by their length. The bottleneck matching can be found in the minimum prefix of this ordering that has sufficiently many edges incident to the points of X and the points q and p_i (by Hall's Theorem). We call the resulting matching, *the locally optimal matching consistent with* T_i.

In either case, the time is bounded by the cost of sorting the points matched to q for each q that shifts some mass to p_i in the transportation plan T_i. We have already shown that there are only at most 25 such points q, but with multiplicity, there could be as many as $|X| = \Theta(n)$ edges to consider. This makes the overall, worst-case running time $O(n \log n)$ for an update to the matching.

Amortizing the Cost of Updates

Although the worst case cost of updating the matching when going from D_i to D_{i+1} is $O(n \log n)$, not every such update can be that expensive. Below, we show that a sequence of these updates from D_i to D_j such that $\varepsilon_i \leq 2\varepsilon_j$ will also only take $O(n \log n)$ time. This means adding points in the sketch to halve the error (double the precision) is asymptotically the same (in the worst-case) as updating the matching for one new point.

▶ **Theorem 8.** *Let $(D_0, \ldots, D_n)$ be a greedy PD sketch of D. For any i, k such that $\varepsilon_i \leq 2\varepsilon_k$, the total cost of computing the locally optimal matchings M_j consistent with T_j for all $i \leq j \leq k$ can be computed in $O(n \log n)$ time.*

Proof. The total cost is $\sum_{q \in Q} \text{mult}(q) \log(\text{mult}(q))$ where Q is the set of points whose mass changes at some point in the sequence of updates. Recall that by the definition of the natural reweighting used in the greedy PD sketch, the multiplicity of a point $q \in D$ is equal to the number of points in a discrete Voronoi cell at some point in the construction. If $T_j(q, p_j) > 0$, then q and p_j have neighboring Voronoi cells in some step of the construction. This means that every point in $\text{Vor}(q)$ is touched when p_j is inserted to see if it must be move. So, as we observed in the analysis of the construction (Theorem 6), any given point can be touched at most $O(1)$ times before the insertion radius decreases by a factor of 2. As the edges incident to any point q in the matching are in correspondence with the points of $\text{Vor}(q)$, it follows that each such edge participates in at most $O(1)$ updates. In other words, $\sum_{q \in Q} \text{Vor}(q) = \sum_{q \in Q} \text{mult}(q) = O(n)$. So, the total cost is

$$\sum_{q \in Q} \text{mult}(q) \log(\text{mult}(q)) \leq \sum_{q \in Q} \text{mult}(q) \log(n) = O(n \log n). \qquad \blacktriangleleft$$

8 Filtered Neighborhood Graphs

Matchings are computed on neighborhood graphs. In this section, we show how the greedy sketch computation can also simplify the construction of these graphs. We then show how this same construction allows for fast Hausdorff approximation between PD sketches, leading to fast lower bounds on the bottleneck distance.

The *α-neighborhood graph* on a set P is the graph

$$\text{Nbrhd}(P, \alpha) := (P, \{(p_i, p_j) \mid \mathrm{d}(p_i, p_j) \leq \alpha\}).$$

Let λ and γ be constants. If the points of P are all pairwise λ apart, then the degree of any vertex in $\text{Nbrhd}(P, \gamma\lambda)$ cannot exceed $(2\gamma + 1)^2$. This follows because the squares of side length λ centered at the neighbors will be disjoint and contained in the square of side length $(2\gamma + 1)\lambda$.

Let $P = (p_0, \ldots, p_{n-1})$ be a set ordered according to a greedy permutation. The *γ-filtered neighborhood graph* on P is the graph with vertices P and edges (p_j, p_i) whenever $i < j$ and $\mathrm{d}(p_i, p_j) < \gamma\lambda_j$. Because P_j is a λ_j-net for all j, there will be at most $(2\gamma + 1)^2$ neighbors of p_j that precede it in the greedy permutation. Thus, the total number of edges is $(2\gamma + 1)^2 n$. Moreover, the graph contains $\text{Nbrhd}(P_i, \gamma\lambda_i)$ for all i.

When computing the greedy permutation, one can compute the filtered neighborhood graph at the same time in the same asymptotic running time. This is the underlying idea in the Clarkson algorithm (the graph is an undirected version of the *sb* data structure).

For a pair of compact sets P, Q, the bipartite α-neighborhood graph is

$$\text{BiNbrhd}(P, Q, \alpha) := (P \sqcup Q \mid \{(p, q) \in P \times Q \mid \mathrm{d}(p, q) \leq \alpha)$$

The Hausdorff distance between P and Q is the minimum α such that $\text{BiNbrhd}(P, Q, \alpha)$ contains no isolated vertices. The bottleneck distance between P and Q is the minimum α such that $\text{BiNbrhd}(P, Q, \alpha)$ contains a perfect matching. If P and Q are persistence diagrams, one adds the diagonal as an extra vertex to each set and adds edges from points to the diagonal if their projection to the diagonal is within α.

The main way that "geometry helps" for matching problems in the plane is that one can use a geometric data structure to implicitly store this graph. However, when working with approximations, one can show that the bipartite neighborhood graph has linear size and can be computed in linear time if one has already precomputed the filtered neighborhood graphs of the two sets. Below, we explain the construction.

▶ **Lemma 9.** *Let $R, B \subset \mathbb{R}^2$. If $\text{BiNbrhd}(R_\lambda, B_\lambda, \gamma\lambda)$ has an isolated vertex, then $\mathrm{d}_H(R, B) \geq \lambda(\gamma - 1)$.*

Proof. Without loss of generality, let $x \in R_\lambda$ be an isolated vertex. Then, there are no points of B_λ within distance $\lambda\gamma$ of x and so, $\mathrm{d}_H(R, B_\lambda) \geq \lambda\gamma$. Because B_λ is a λ-net, $\mathrm{d}_H(B, B_\lambda) \leq \lambda$. Therefore, by the triangle inequality, $\mathrm{d}_H(R, B) \geq \lambda\gamma - \lambda = \lambda(\gamma - 1)$. ◀

▶ **Theorem 10.** *Let R, B be PDs and let λ and γ be constants such that $\lambda(\gamma - 1) \geq \mathrm{d}_H(R, B)$. Given the greedy permutations of R and B as well as the corresponding $(2\gamma + 1)$-filtered neighborhood graphs, the $\text{BiNbrhd}(R_\lambda, B_\lambda, \gamma\lambda)$ can be computed in linear time.*

Proof. The algorithm will be incremental, adding the points in order of their insertion radii. At each step i, we maintain $G_i = \text{BiNbrhd}(R_{\lambda_i}, B_{\lambda_i}, \gamma\lambda_i)$, where λ_i is the insertion radius of the newly inserted point. When inserting p_i, we add its neighbors G_i. Without loss of generality, assume $p_i \in R$. Let y be the nearest neighbor of p_i in $\text{Nbrhd}(R_i, 2\gamma(\lambda_i))$. So

$\mathrm{d}(y, p_i) = \lambda_i$. There are only a constant number of neighbors. Then, let a be any neighbor of y in G_i (which are contained in the neighbors of y in G_{i-1}). The neighbor a must exist because of Lemma 9 and our choice of λ. By the triangle inequality,

$$\mathrm{d}(a, b) \leq \mathrm{d}(a, y) + \mathrm{d}(y, x) + \mathrm{d}(x, b) \leq \lambda_i\gamma + \lambda_i + \lambda_i\gamma = \lambda_i(2\gamma + 1).$$

It follows that a and b are neighbors in $\mathrm{Nbrhd}(B_{\lambda_i}, \lambda_i(2\gamma + 1))$. So, the neighbors of p_i can all be found among the neighbors of y. Because the degrees are constant and edges that are too long for the current value of λ_i can be deleted as they are encountered the neighbors of p_i can be found in amortized constant time.

One pass over the edges suffices to remove any that are longer than $\lambda\gamma$. Thus, the total running time is linear. ◀

▶ **Theorem 11.** *Given the greedy permutations of R and B as well as the corresponding $(2\gamma + 1)$-filtered neighborhood graphs, an approximation of $\mathrm{d}_H(R, B)$ to within a factor of $1 \pm \frac{1}{\gamma}$ can be computed in linear time.*

Proof. If the algorithm from Theorem 10 is run until it either inserts all the points or discovers an isolated vertex, it will produce a graph with a linear number of edges. By iterating over the edges, one can find, for each vertex, the distance to the nearest adjacent vertex in the graph. The maximum of these distances indicates the distance $r = \lambda\gamma$ at which a vertex becomes isolated and is the desired approximation. By Lemma 9, we know that $\mathrm{d}_H(R, B) \geq \lambda(\gamma - 1) = r(1 - \frac{1}{\gamma})$. Similarly, because $\mathrm{d}_H(R, B_\lambda) \leq \lambda$, the triangle inequality implies that $\mathrm{d}_H(R, B) \leq \mathrm{d}_H(R, B\lambda) + \lambda = \lambda(\gamma + 1) = r(1 + \frac{1}{\gamma})$. ◀

9 Conclusions and Future Work

We have presented an efficient method to preprocess a persistence diagram so that one can extract guaranteed approximations with any number of distinct points. It remains to explore the relationship between greedy PD sketches and other PD simplicifcations such as persistence landscapes [4] or persistence images [1]. In particular, it is possible to construct a persistence landscape or a persistence image from a PD sketch, possibly much faster than with the entire diagram. Another application where many PD distance computations are used is in the computation of Wasserstein barycenters. In Vidal et al. [23], a kind of sketch is used to dramatically speed up the Wasserstein barycenter computation. In that case, they use the subset of points of highest persistence. Although the approximation guarantees we prove are only applicable to the bottleneck distance, it seems reasonable that they should also be applicable to other Wasserstein metrics. Indeed, we give the matching update for these metrics in Section 7.

In future work, we will incorporate these sketches into a data structure that supports standard proximity search queries including (approximate) nearest neighbor search, metric range search, and metric range counting. This is the main target of our work as it is a case where there is immediate benefit to finding fast approximate distances with bounds on the error to prune a search.

References

1 Henry Adams, Tegan Emerson, Michael Kirby, Rachel Neville, Chris Peterson, Patrick Shipman, Sofya Chepushtanova, Eric Hanson, Francis Motta, and Lori Ziegelmeier. Persistence images: A stable vector representation of persistent homology. *J. Mach. Learn. Res.*, 18(1):218–252, 2017.

2 Dimitri P. Bertsekas. The auction algorithm: A distributed relaxation method for the assignment problem. *Annals of Operations Research*, 14(1):105–123, 1988.

3 Dimitri P. Bertsekas and David A. Castanon. The auction algorithm for the transportation problem. *Annals of Operations Research*, 20(1):67–96, December 1989. doi:10.1007/BF02216923.

4 Peter Bubenik. Statistical topological data analysis using persistence landscapes. *The Journal of Machine Learning Research*, 16(1):77–102, 2015.

5 Mathieu Carrière, Marco Cuturi, and Steve Oudot. Sliced Wasserstein kernel for persistence diagrams. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 664–673, International Convention Centre, Sydney, Australia, 06–11 August 2017. PMLR. URL: http://proceedings.mlr.press/v70/carriere17a.html.

6 Mathieu Carrière, Steve Y. Oudot, and Maks Ovsjanikov. Stable topological signatures for points on 3d shapes. *Computer Graphics Forum*, 34(5):1–12, August 2015. doi:10.1111/cgf.12692.

7 Kenneth L. Clarkson. Nearest neighbor queries in metric spaces. *Discrete & Computational Geometry*, 22(1):63–93, 1999.

8 Kenneth L. Clarkson. Nearest neighbor searching in metric spaces: Experimental results for "sb(s)". Preliminary version presented at ALENEX99, 2003.

9 Kenneth L. Clarkson. Nearest-neighbor searching and metric space dimensions. In Gregory Shakhnarovich, Trevor Darrell, and Piotr Indyk, editors, *Nearest-Neighbor Methods for Learning and Vision: Theory and Practice*, pages 15–59. MIT Press, 2006.

10 Vincent Divol and Theo Lacombe. Understanding the topology and the geometry of the space of persistence diagrams via optimal partial transport. *Journal of Applied and Computational Topology*, 2020.

11 M.E. Dyer and A.M. Frieze. A simple heuristic for the p-centre problem. *Operations Research Letters*, 3(6):285–288, 1985.

12 A. Efrat, A. Itai, and M. J. Katz. Geometry helps in bottleneck matching and related problems. *Algorithmica*, 31(1):1–28, September 2001. doi:10.1007/s00453-001-0016-8.

13 Brittany Terese Fasy, Xiaozhou He, Zhihui Liu, Samuel Micka, David L. Millman, and Binhai Zhu. Approximate nearest neighbors in the space of persistence diagrams. *CoRR*, abs/1812.11257, 2018. arXiv:1812.11257.

14 Teofilo F. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theor. Comput. Sci.*, 38:293–306, 1985.

15 Sariel Har-Peled and Manor Mendel. Fast construction of nets in low-dimensional metrics and their applications. *SIAM Journal on Computing*, 35(5):1148–1184, January 2006. doi:10.1137/S0097539704446281.

16 John E. Hopcroft and Richard M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on Computing*, 2(4):225–231, December 1973. doi:10.1137/0202019.

17 Michael Kerber, Dmitriy Morozov, and Arnur Nigmetov. Geometry helps to compare persistence diagrams. *ACM Journal of Experimental Algorithmics*, 22(1):1.4:1–1.4:20, 2017.

18 Theo Lacombe, Marco Cuturi, and Steve OUDOT. Large scale computation of means and clusters for persistence diagrams using optimal transport. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31, pages 9770–9780. Curran Associates, Inc., 2018. URL: https://proceedings.neurips.cc/paper/2018/file/b58f7d184743106a8a66028b7a28937c-Paper.pdf.

19 Brendan Mumey. Indexing point sets for approximate bottleneck distance queries. *CoRR*, abs/1810.09482, 2018. arXiv:1810.09482.

20 Jan Reininghaus, Stefan Huber, Ulrich Bauer, and Roland Kwitt. A stable multi-scale kernel for topological machine learning. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, page 4741–4748. IEEE, June 2015. doi:10.1109/CVPR.2015.7299106.

21 Donald R. Sheehy. greedypermutations, 2020. URL: https://github.com/donsheehy/greedypermutation.

22 Maxime Soler, Melanie Plainchault, Bruno Conche, and Julien Tierny. Lifted wasserstein matcher for fast and robust topology tracking. In *2018 IEEE 8th Symposium on Large Data Analysis and Visualization (LDAV)*, page 23–33. IEEE, October 2018. doi:10.1109/LDAV.2018.8739196.

23 Jules Vidal, Joseph Budin, and Julien Tierny. Progressive wasserstein barycenters of persistence diagrams. *IEEE Transactions on Visualization and Computer Graphics*, 26:151–161, 2020.

24 Matthias Zeppelzauer, Bartosz Zieliński, Mateusz Juda, and Markus Seidl. *Topological Descriptors for 3D Surface Analysis*, volume 9667 of *Lecture Notes in Computer Science*, page 77–87. Springer International Publishing, 2016. doi:10.1007/978-3-319-39441-1_8.