# Computing the Shift-Invariant Bottleneck Distance for Persistence Diagrams

Don Sheehy* Oliver Kisielius† Nicholas Cavanna‡

**Abstract**

We define an algorithm that can compute the minimum of the bottleneck distance between two persistence diagrams over all diagonal shifts, in $O(n^{3.5})$ time. When applied to log-scale persistence diagrams, this is a scale-invariant version of bottleneck distance.

## 1 Introduction

A persistence diagram is a set of points in the plane that describes the changes in topology of the sublevel sets of a function. Each point's coordinates represent the birth and death of a topological feature. Often, persistence diagrams are generated from other geometric data sets and can serve as data summaries. They have risen to prominence in topological data analysis for their ability to capture multi-scale structure in a way that is invariant to distance-preserving transformations.

The stability theory of persistence diagrams implies that for small changes in the inputs, the persistence diagrams will have correspondingly small changes with respect to the bottleneck distance. This distance is defined in terms of a minimal matching between two diagrams that allows points to be matched with the diagonal. This distance is used as the foundation of all approximation results in persistent homology.

Persistence diagrams from metric inputs are sensitive to scaling of the input data. One way to combat this is to use log-scale persistence diagrams, as in [5]. In such a diagram, the prominence of a feature—its distance to the diagonal—is determined by the ratio of the death and birth times of the original diagram. This eliminates the artificial inflation of prominence that would result from a change in units.

Even log-scale persistence diagrams cannot recognize that two diagrams are generated by the same metric input measured in different units. Although the prominence of the features will remain the same, the two diagrams will differ by a shift along the diagonal. To resolve this, we introduce a new pseudometric, the *shifted bottleneck distance* on persistence diagrams that minimizes over all possible shifts, thus adding scale-invariance to the resulting metric space of diagrams. In the language of Euclidean geometry, this makes persistent homology useful not only for congruence but also similarity.

We give the formal definition of the shifted bottleneck distance and the proof of its metric properties. It is stable in the sense proven for bottleneck distance in [2]. Then, we show how to compute the distance in polynomial time.

### 1.1 Persistent Homology—A Quick Example

The results in this paper do not depend on a deep understanding of persistent homology, and we refer the reader to the accessible survey by Edelsbrunner and Morozov [3] for more background. We give a simple example here to show a common way that geometric points are turned into persistence diagrams that capture multiscale structure.

For the point set $P$ shown in Figure 1, we will compute the persistent homology of the sublevel sets of the function $r_P : \mathbb{R}^2 \to \mathbb{R}$, which is the distance to the set $P$:

$$r_P(x) = \min_{p \in P} \|x - p\|. \tag{1}$$

The sublevel sets of $r_P$ are topologically equivalent to subcomplexes of the Delaunay triangulation of $P$. As one considers larger scales (i.e. sublevels of $r_P$ for larger thresholds), one obtains larger and larger subcomplexes of $\mathrm{Del}\,(P)$. The persistence algorithm will convert this growing sequence of complexes into a persistence diagram, $\mathrm{Dgm}(r_P)$, as shown in Figure 2. Each point in $\mathrm{Dgm}(r_P)$ is a pair $(b, d)$ representing the birth and death of a topological feature. In general, for a filtration based on distance from a finite point set, one can use log-scale diagrams for features of any dimension except 0. The eye-catching features of $P$—two cycles—appear at different scales, and in the original persistence diagram, the inside of the big cycle dwarfs the other features. In the log-scale diagram, both cycles are prominent. Both diagrams are shown in Figure 2.

## 2 Defining Shifted Bottleneck Distance

The only distance we consider between points in the plane is the infinity metric, $\mathrm{d}_\infty$.

$$\mathrm{d}_\infty\left((x, y), (x', y')\right) = \max\{|x - x'|, |y - y'|\} \tag{2}$$

*Computer Science Department, University of Connecticut, don.r.sheehy@gmail.com

†Computer Science Department, University of Connecticut, oliver.kisielius@uconn.edu

‡Computer Science Department, University of Connecticut, nicholas.j.cavanna@uconn.edu

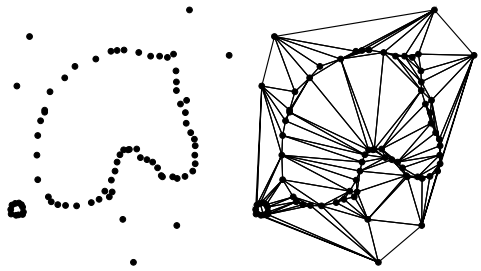

Figure 1: A point set $P$ with its Delaunay triangulation $\mathrm{Del}\,(P)$

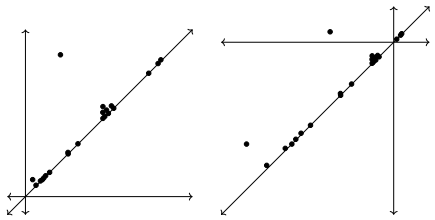

Figure 2: Persistence diagram from the filtration of $P$. Each point is a (birth, death) pair. On the left is the original persistence diagram. On the right is the log-scale diagram. The two cycles look similarly prominent on the log-scale.

### 2.1 Shifted Points and Shifted Bottleneck Distance

Fix $p = (x, y)$, a point in the plane, and fix $s \in \mathbb{R}$. Define the *image of $p$ under shift $s$* as

$$p_s = (x + s, y + s). \tag{3a}$$

Define the image of an entire multiset $A$ of points in $\mathbb{R}^2$ under shift $s$ as the multiset

$$A_s = \{p_s \mid p \in A\}. \tag{3b}$$

If $p$ is an off-diagonal point, then $\delta(p)$ denotes the orthogonal projection of $p$ onto the diagonal.

$$\delta((x, y)) = (\frac{x+y}{2}, \frac{x+y}{2}) \tag{4}$$

Define $\Delta$, *the diagonal*, to be the multiset containing each point $(x, x)$ in $\mathbb{R}^2$ with infinite multiplicity.

Let $A$ be a finite multiset in the plane, with $x < y$ for all $(x, y)$ in $A$. Denote by $\hat{A}$ the infinite *persistence diagram* $A \cup \Delta$. We assume all persistence diagrams have finitely many off-diagonal points.

For two multisets of points $A$ and $B$ in $\mathbb{R}^d$, the *bottleneck distance* $\mathrm{d_B}\,(A, B)$ is defined as follows:

$$\mathrm{d_B}\,(A, B) = \min_{M} \max_{\langle a,b\rangle \in M} \mathrm{d}_\infty\,(a, b) \tag{5}$$

where $M$ ranges over all perfect matchings between $A$ and $B$.

For multisets $A$ and $B$ of points in the plane, define the *shifted bottleneck distance* of $A$ and $B$:

$$\mathrm{d_{SB}}\,(A, B) = \min_{s\in\mathbb{R}} \mathrm{d_B}\,(A_s, B) \tag{6}$$

Given finite multisets $A$ and $B$, our algorithm computes $\mathrm{d_{SB}}\,(\hat{A}, \hat{B})$.

**Lemma 1** *If $A$ and $B$ are finite multisets of points, then $\mathrm{d_{SB}}\,(A, B)$ is well-defined.*

*If $A$ and $B$ are finite multisets of points with $x < y$ for all $(x, y)$ in $A \cup B$, then $\mathrm{d_{SB}}\,(\hat{A}, \hat{B})$ is well-defined.*

**Proof.** Let $r = \inf_{s\in\mathbb{R}} \mathrm{d_B}\,(A_s, B)$. We need to show $\mathrm{d_B}\,(A_s, B) = r$ for some real shift $s$.

It's clear that $\mathrm{d_B}\,(A_s, B)$ is a continuous function of $s$. This means it is sufficient to demonstrate a closed, bounded set $S$ such that $r = \inf_{s\in S} \mathrm{d_B}\,(A_s, B)$. The set

$$S = \bigcup_{\langle a,b\rangle \in A\times B} \{s \in \mathbb{R} \mid \mathrm{d}_\infty\,(a_s, b) \le r\} \tag{7}$$

will suffice. $S$ is closed and bounded because it is a finite union of closed intervals.

We show that $\mathrm{d_{SB}}\,(\hat{A}, \hat{B})$ is well-defined by a similar argument. Let $\hat{r} = \inf_{s\in\mathbb{R}} \mathrm{d_B}\,(\hat{A}_s, \hat{B})$.

We can assume that $A$ and $B$ are nonempty and that $\hat{r} < \max_{p\in A\cup B} \mathrm{d}_\infty\,(p, \delta(p))$, because otherwise there is nothing to show. With that assumption, we use the same argument as before, with the same set $S$. □

If (6) used inf instead of min, then $\mathrm{d_{SB}}$ might be well-defined for more inputs. However, it is useful to know that there always exists some $s$ such that

$$\mathrm{d_{SB}}\,(\hat{A}, \hat{B}) = \mathrm{d_B}\,(\hat{A}_s, \hat{B}).$$

**Lemma 2** *Let $X$, $Y$, and $Z$ be persistence diagrams or finite sets of points.*

$$\mathrm{d_{SB}}\,(X, Z) \le \mathrm{d_{SB}}\,(X, Y) + \mathrm{d_{SB}}\,(Y, Z) \tag{8}$$

*In other words, shifted bottleneck distance satisfies the triangle inequality.*

**Proof.** Let $s_0$ and $s_1$ be shifts such that

$$\mathrm{d_{SB}}\,(X, Y) = \mathrm{d_B}\,(X_{s_0}, Y) \tag{9}$$
$$\mathrm{d_{SB}}\,(Y, Z) = \mathrm{d_B}\,(Y_{s_1}, Z) \tag{10}$$

Since $\mathrm{d_B}$ is a metric, we have

$$\mathrm{d_B}\,(X_{s_0+s_1}, Z) \le \mathrm{d_B}\,(X_{s_0+s_1}, Y_{s_1}) + \mathrm{d_B}\,(Y_{s_1}, Z) \tag{11}$$
$$= \mathrm{d_B}\,(X_{s_0}, Y) + \mathrm{d_B}\,(Y_{s_1}, Z) \tag{12}$$
$$= \mathrm{d_{SB}}\,(X, Y) + \mathrm{d_{SB}}\,(Y, Z) \tag{13}$$

(We reach (12) by applying (3a), and we get (13) by applying (9) and (10).) Now we apply (6) to get (14).

$$\mathrm{d_{SB}}\,(X, Z) \le \mathrm{d_B}\,(X_{s_0+s_1}, Z) \tag{14}$$

Combining (13) and (14) yields (8). □

It's typical to compare two persistence diagrams using bottleneck distance, thanks to the following stability result proven in [2].

Let $\mathrm{Dgm}(f)$ denote the persistence diagram of sublevel sets of $f$. The main theorem of [2] states that, assuming some conditions on the topological space $X$ and the continuous functions $f, g : X \to \mathbb{R}$, we have

$$\mathrm{d_B}\left(\mathrm{Dgm}(f), \mathrm{Dgm}(g)\right) \leq \|f - g\|_\infty \tag{15}$$

Since it's obvious that in general $\mathrm{d_{SB}}\left(\hat{A}, \hat{B}\right) \leq \mathrm{d_B}\left(\hat{A}, \hat{B}\right)$, the stability result by [2] must hold for $\mathrm{d_{SB}}$ as well:

**Theorem 3** *Let $X$ be a topological space $X$, and let $f, g : X \to \mathbb{R}$ be functions. If $X$, $f$, and $g$ satisfy the conditions for the main stability result of [2], then we have the same result for shifted bottleneck distance.*

$$\mathrm{d_{SB}}\left(\mathrm{Dgm}(f), \mathrm{Dgm}(g)\right) \leq \|f - g\|_\infty \tag{16}$$

### 2.2 Related Work

We make use of some ideas from prior work concerning a different pseudo-metric, which we'll call *general shifted bottleneck distance*, or $\mathrm{d_{GSB}}$.

$$\mathrm{d_{GSB}}\left(X, Y\right) = \min_{t \in \mathbb{R}^2} \mathrm{d_B}\left(\{x + t \mid x \in X\}, Y\right) \tag{17}$$

Here $X$ and $Y$ are finite sets of points in the plane.

The earliest relevant algorithm for $\mathrm{d_{GSB}}$ is by Alt et al. in [1]. They compute $\mathrm{d_{GSB}}$ in time $O(n^6 \log n)$. To do this, they first make an $O(n^6)$ time decision algorithm that, given $X$, $Y$, and $r$, tests whether $\mathrm{d_{GSB}}\left(X, Y\right) \leq r$. Then they generate and sort all $O(n^6)$ possible answers and find the correct answer with a binary search.

One idea of theirs that we adopt is their subroutine in which a bipartite matching is repeatedly maximized (and pruned) while the set of available edges changes incrementally. This involves $O(n^4)$ invocations of the Hopcroft-Karp augmenting paths algorithm at a cost of $O(n^2)$ time per augmenting path. That's lower than the cost of computing a matching from scratch $O(n^4)$ times.

Efrat et al. improved this result by using geometry to optimize the augmenting-path routine [4]. They use near-neighbor structure to represent edges implicitly during the graph searches of the Hopcroft-Karp, which results in running time of $\log n$ per node of the graph, and thus $O(n \log n)$ per augmenting path. Like Alt et al., they find $O(n^4)$ total augmenting paths, so their algorithm runs in $O(n^5 \log n)$ time.

Efrat et al. also use the optimized Hopcroft-Karp algorithm to compute bottleneck distance between finite point sets [4], and Kerber et al. use the same technique to compute bottleneck distance between persistence diagrams in $O(n^{1.5} \log n)$ time [6].

To minimize $\mathrm{d_B}$ over all two-dimensional shifts, Alt et al. pay quadratic time just to reduce the problem to a one-dimensional problem in polar coordinates. Their key idea is to guess ($O(n^2)$ times) which edge is the bottleneck. Knowing that $\langle x, y\rangle \in X \times Y$ is the bottleneck, you can test whether $\mathrm{d_{GSB}}\left(X, Y\right) \leq r$ by testing only shifts $t$ such that $\mathrm{d}_\infty\left(x + t - y\right) = r$. (This works for Euclidean distance as well.) Then only a one-dimensional value, the angle from $y$ to $x+t$, is unknown. And so they compute $O(n^2)$ *critical angles* at which another edge has value exactly $r$, and they check for a matching at each critical angle.

Our algorithm is faster. Since we compute $\mathrm{d_{SB}}$, we have a one-dimensional parameter from the beginning, the shift, so we need not spend $O(n^2)$ immediately. Furthermore, in our setting we can process the *critical shifts* only once, reducing the radius and reordering the critical shifts on the fly. Because we needn't perform a binary search, our full algorithm resembles the decision-only version of the other algorithms.

## 3 Background

Throughout the remaining discussion we refer to $A$ and $B$, the finite input multisets to our algorithm. These are to be distinguished from the infinite sets $\hat{A}$ and $\hat{B}$.

### 3.1 Diagonal-Perfect Matchings

Edges with at least one end on the diagonal are called *diagonal edges*. Edges in $A \times B$ are *non-diagonal edges*.

A finite matching $M$ between $\hat{A}$ and $\hat{B}$ is *diagonal-perfect* if the degree in $M$ of each point in $A \sqcup B$ equals the multiplicity of that point. For such a matching $M$, the *value* of $M$ is the minimum, over all shifts, of the greatest edge length in $M$.

$$\texttt{value}\left(M\right) = \min_{s \in \mathbb{R}} \max_{\langle a,b\rangle \in M} \mathrm{d}_\infty\left(a_s, b\right)$$

An *$r$-matching* is a diagonal-perfect matching $M$ between $\hat{A}$ and $\hat{B}$ with $\texttt{value}\left(M\right) = r$. Clearly such a matching is a certificate that $\mathrm{d_{SB}}\left(\hat{A}, \hat{B}\right) \leq r$. A *less-than-$r$-matching* is an $r'$-matching for some $r' < r$.

If $M$ is a diagonal-perfect matching between $\hat{A}$ and $\hat{B}$, then the union of $M$ with any perfect matching from $\Delta$ to $\Delta$ is a perfect matching. In particular, if you extend $M$ by adding edges of the form $\langle x, x_s\rangle \in \Delta$, where $s$ is the optimal shift for $M$, then the value of the resulting perfect matching is $\texttt{value}\left(M\right)$.

As proven in [6], if an $r$-matching exists, then one exists that contains no "skew" diagonal edges. A *non-skew diagonal edge* is an edge $\langle p, \delta(p)\rangle$ or $\langle \delta(p), p\rangle$. This includes $\langle p, p\rangle$ where $p = \delta(p)$.

### 3.2 Working with the Diagonal

The addition of points on the diagonal in the definition of persistence diagrams is useful for stability results, but requires special consideration in our algorithm. Bottleneck distance between diagrams can be reduced to bottleneck distance between finite sets using (18).

$$\mathrm{d_B}\left(\hat{A}, \hat{B}\right) = \mathrm{d_B}\left(A \cup \delta(B), B \cup \delta(A)\right) \tag{18}$$

However, this does not work for shifted bottleneck distance. It is not true that $\mathrm{d_{SB}}\left(\hat{A}, \hat{B}\right) = \mathrm{d_{SB}}\left(A \cup \delta(B), B \cup \delta(A)\right)$, because you can't match $B$ to the shifted image $\delta(B)_s$ or match $A_s$ to $\delta(A)$. Instead, we must handle the diagonal as a special case. As noted in [6], it is faster to give the diagonal special treatment, because all but $O(n)$ edges involving the diagonal can be ignored.

**Diagonal-to-Diagonal Edges** Augmenting paths discovered by the Hopcroft-Karp algorithm can include diagonal-to-diagonal ($\Delta$-to-$\Delta$) edges. Because $\hat{A}$ and $\hat{B}$ include every point of the diagonal with infinite multiplicity, the length of the longest $\Delta$-to-$\Delta$ edge in any matching can be made arbitrarily small via augmenting paths in $\Delta \times \Delta$. This lets us consider the length of $\Delta$-to-$\Delta$ edges to be zero. The diagonal parts of $\hat{A}$ and $\hat{B}$ form a complete bipartite graph on zero-length edges. As a result, we represent $\hat{A} \cap \Delta$ and $\hat{B} \cap \Delta$ as two nodes $\Delta_A$ and $\Delta_B$, both with infinite multiplicity. We identify any edge $\langle a, \delta(a)\rangle$ with $\langle a, \Delta_B\rangle$, and similarly we identify $\langle \delta(b), b\rangle$ with $\langle \Delta_A, b\rangle$.

As noted in [6], the near-neighbor search structure (used in the optimized Hopcroft-Karp algorithm) can be adapted to handle $\Delta_A$ and $\Delta_B$.

## 4 A Kinetic Data Structure Approach

The main algorithm will look at an increasing sequence of shifts. At different shifts, edges will appear or disappear. These are the *events* we want to track. Moreover, as we discover better matchings, our upper bound on the radius $r$ decreases. Changing the radius reorders future events, i.e. the partition $P_r$. In this section, we will define the events and introduce an *event queue* that provides access to the events in the correct order.

### 4.1 Searching for a Better Matching

Let $r$ be an upper bound on $\mathrm{d_{SB}}\left(\hat{A}, \hat{B}\right)$. The *left shift* $\lambda$ of a non-diagonal edge $\langle (a_x, a_y), (b_x, b_y)\rangle$ at radius $r$ is

$$\lambda(\langle (a_x, a_y), (b_x, b_y)\rangle, r) = \max\{b_x - r - a_x, b_y - r - a_y\}.$$

Similarly, the *right shift* $\rho$ is

$$\rho\left(\langle (a_x, a_y), (b_x, b_y)\rangle, r\right) = \min\{b_x + r - a_x, b_y + r - a_y\}.$$

For any edge $e = \langle a, b\rangle$ and shift $s$ such that $\mathrm{d}_\infty(a_s, b) < r$, we have $\lambda(e, r) < s < \rho\left(e, r\right)$. Let $P_r$ denote the set of all left and right shifts, i.e.

$$P_r = \{\lambda(e, r) \mid e \in A \times B\} \cup \{\rho\left(e, r\right) \mid e \in A \times B\}$$

If $s_0, \ldots, s_k$ are the shifts of $P_r$, where $s_0 < \cdots < s_k$, then $\textsc{intvl}\, P_r$ is the set of open intervals $\{(s_i, s_{i+1}) \mid i \in \{0, \ldots, k-1\}\}$.

The *set of available non-diagonal edges* for a shift $s$ and radius $r$ is:

$$\mathcal{E}(r, s) = \{\langle a, b\rangle \in A \times B \mid \mathrm{d}_\infty\left(a_s, b\right) < r\}$$

At radius $r$, the *set of available non-skew diagonal edges*, which includes $\langle \Delta_A, \Delta_B\rangle$ when $r > 0$, is:

$$\mathcal{D}(r) = \{e \in A \times \{\Delta_B\} \cup \{\Delta_A\} \times B \cup \{\langle \Delta_A, \Delta_B\rangle\} \mid \mathrm{d}_\infty(e) < r\}$$

For each interval $(s_i, s_{i+1})$ in $\textsc{intvl}\, P_r$, we have a graph

$$G(r, s_i) = \mathcal{E}(r, \frac{s_i + s_{i+1}}{2}) \cup \mathcal{D}(r).$$

The graph contains the diagonal edges and the available non-diagonal edges at a shift inside the interval $(s_i, s_{i+1})$. The choice of the midpoint is arbitrary, and indeed, $G(r, s_i) = \mathcal{E}(r, s) \cup \mathcal{D}(r)$ for any $s_i < s < s_{i+1}$. This implies the following lemma. (Proofs of these facts can be found in the appendix.)

**Lemma 4** *If $M$ is a less-than-$r$-matching, then $M \subseteq G(r, \lambda(e, r))$ for some $e$ in $A \times B$.*

**Lemma 5** *For any edge $e$ and radii $r' < r$, we have $G(r', \lambda(e, r')) \subseteq G(r, \lambda(e, r))$.*

### 4.2 The Event Queue

In this section, we will describe the event queue data structure. The events provided by this structure are *L-events* and *R-events*. An event `e` of either type stores an edge `e.edge`. An L-event also stores a shift `e.shift` such that $G(r, \lambda(\texttt{e.edge}, r)) = \mathcal{E}(r, \texttt{e.shift}) \cup \mathcal{D}(r)$.

The event queue `Q` holds three stacks of edges. The stack `Q.D` contains all non-skew diagonal edges, including $\langle \Delta_A, \Delta_B\rangle$, in decreasing order by length. (Longer edges are popped first.) The stacks `Q.L` and `Q.R` contain the edges of $A \times B$ sorted increasing by $\lambda$ and $\rho$ respectively (at radius 0). The order within those stacks does not depend on the radius, because for any $x, r \in \mathbb{R}$, $\lambda(e, r + x) - \lambda(e, r) = x$ and $\rho\left(e, r - x\right) - \rho\left(e, r\right) = x$.

The method `Q.nextevent`$(r)$ goes like this: If the top of `Q.D` has length $r$ or greater, return an R-event for `Q.D.pop()`. If $\rho\left(\texttt{Q.R}, r\right) \le \lambda(\texttt{Q.L}, r)$, return an R-event for `Q.R.pop()`. Otherwise, return an L-event for `Q.L.pop()`. Also remove from `Q.L` any edges with the same left shift as the edge popped.

**Input:** Event queue Q, radius $r$
1 **if** $|\texttt{Q.D}| > 0$ *and* $\mathrm{d}_\infty(\texttt{Q.D.top}()) \geq r$ **then**
2 | **return** RightEvent(edge $\leftarrow$ Q.D.pop())
3 **if** $\rho(\texttt{Q.R.top}(), r) \leq \lambda(\texttt{Q.L.top}(), r)$ **then**
4 | **return** RightEvent(edge $\leftarrow$ Q.R.pop())
5 Let e $\leftarrow$ Q.L.pop()
6 **while** $|\texttt{Q.L}| > 0$ *and* $\lambda(\texttt{e}, 0) = \lambda(\texttt{Q.L.top}(), 0)$ **do**
7 | Q.L.pop()
8 Let t $\leftarrow \rho(\texttt{Q.R.top}(), r)$
9 **if** $|\texttt{Q.L}| > 0$ **then**
10 | Set t $\leftarrow \min\{\texttt{t}, \lambda(\texttt{Q.L.top}(), r)\}$
11 **return**
LeftEvent(edge $\leftarrow$ e, shift $\leftarrow (\lambda(\texttt{e}, \texttt{r}) + \texttt{t}) \div 2$)

**Algorithm 1:** Q.nextevent($r$)

**Theorem 6** *Let* Q *be the event queue, and let* $r_1 \geq r_2 \geq \cdots \geq r_n$ *be a nonincreasing sequence of radii. Say that* Q.nextevent($r_i$) *is called for* $i$ *from 1 to* $n$ *in order, and suppose the call to* Q.nextevent($r_n$) *returns an L-event whose shift is* $s$*. Then after the sequence of calls, we have the following.*

$$\mathcal{D}(r_n) = \texttt{Q.D} \tag{19}$$
$$\mathcal{E}(r_n, s) = \texttt{Q.R} \setminus \texttt{Q.L} \tag{20}$$

*Here,* Q.L*,* Q.R*, and* Q.D *are treated as sets.*

The proof is in the appendix.

**Corollary 7** *A modified version of Theorem* 6 *holds, where in the sequence of operations,* e $\leftarrow$ Q.nextevent($r$) *may be followed immediately by* Q.L.push(e.edge)*, provided* e *is an L-event.*

**Proof.** Reinserting the edge undoes the previous operation and has no other side effects. □

### 4.3 Reducing the Radius

To reduce the radius after we find a perfect matching, we use the method Q.newradius(), which returns

$$\max\{\mathrm{d}_\infty(\texttt{Q.D.top}()), \frac{1}{2}(\lambda(\texttt{Q.L.top}(), 0) - \rho(\texttt{Q.R.top}(), 0))\}.$$

**Lemma 8** *The invocation* Q.nextevent($r$) *returns an R-event if and only if* $r \leq$ Q.newradius()*.*

**Proof.** Consider Algorithm 1. Q.nextevent($r$) returns an R-event from line 2 if and only if $r \leq \mathrm{d}_\infty(\texttt{Q.D.top}())$. Otherwise, Q.nextevent() returns an R-event if and only if $\rho(\texttt{Q.R.top}(), r) \leq \lambda(\texttt{Q.L.top}(), r)$. In fact $\rho(\texttt{Q.R.top}(), r) \leq \lambda(\texttt{Q.L.top}(), r)$ exactly when $r \leq \frac{1}{2}(\lambda(\texttt{Q.L.top}(), 0) - \rho(\texttt{Q.R.top}(), 0))$, with equality only when $r = \frac{1}{2}(\lambda(\texttt{Q.L.top}(), 0) - \rho(\texttt{Q.R.top}(), 0))$. Thus $r$.nextevent() an L-event is returned, by line 11, if and only if $r >$ Q.newradius(). □

**Lemma 9** *Suppose the event queue* Q *is in a state such that* Q.nextevent($r$) *would return an L-event with shift* $s$*, and there is a less-than-*$r$*-matching* $M$ *in* $G(r, s)$*. Then*

$$r > \texttt{Q.newradius}() \geq \texttt{value}(M) \geq \mathrm{d_{SB}}(\hat{A}, \hat{B}).$$

**Proof.** Lemma 8 gives us $r >$ Q.newradius(), since Q.nextevent($r$) would return an L-event. For any $r > r' >$ Q.newradius(), call $s'$ the shift of the L-event resulting from Q.nextevent($r'$). Because Q.nextevent($r$) and Q.nextevent($r'$) have equivalent effects on the state of Q, Theorem 6 says $G(r', s') = G(r, s)$. Thus $r' >$ value($M$) for any $r' >$ Q.newradius(), and so Q.newradius() $\geq$ value($M$). The last inequality, value($M$) $\geq \mathrm{d_{SB}}(\hat{A}, \hat{B})$, holds for any diagonal-perfect $M$. □

## 5 The Algorithm

Here we state the main algorithm and prove its correctness and running time in the real RAM model.

### 5.1 Algorithm for Shifted Bottleneck Distance

Algorithm 2 computes the shifted bottleneck distance as follows. Let Q be the event queue. Set the radius $r$ to be an upper bound on $\mathrm{d_{SB}}(\hat{A}, \hat{B})$, say the length of the longest non-skew diagonal edge. Maintain a bipartite matching M, initially empty, between $\hat{A}$ and $\hat{B}$. While Q is not empty, get the next event e from Q. If e is an R-event, remove e.edge from M. Otherwise, e is an L-event: Augment M using the geometrically-optimized version of Hopcroft-Karp (as in [4] and [6]), and if M is now diagonal-perfect, then reinsert e.edge into Q.L and reduce $r$ to Q.newradius(). Finally, return $r$, which now equals $\mathrm{d_{SB}}(\hat{A}, \hat{B})$.

**Input:** Multisets $A$ and $B$ representing diagrams $\hat{A}$ and $\hat{B}$
1 Let $r \leftarrow \max\{\mathrm{d}_\infty(x, \delta(x)) \mid x \in A \cup B\}$
2 Let Q be the event queue
3 Let M be an empty matching
4 **while** Q.L, Q.R *and* Q.D *are nonempty* **do**
5 | Let e $\leftarrow$ Q.nextevent($r$)
6 | **if** e *is an R-event* **then**
7 | | Remove e.edge from M
8 | **else**
9 | | Use augmenting paths to maximize M at shift e.shift and radius $r$
10 | | **if** M *is diagonal-perfect* **then**
11 | | | Q.L.push(e.edge)
12 | | | $r \leftarrow$ Q.newradius()
13 **return** $r$

**Algorithm 2:** $\mathrm{d_{SB}}(\hat{A}, \hat{B})$

### 5.2 Correctness in the Real RAM Model

**Lemma 10** *After line 9 executes,* M *is a maximum matching in* $G(r, \lambda(\texttt{e.edge}, r))$.

**Proof.** If $\texttt{M} \subseteq G(r, \lambda(\texttt{e.edge}, r))$ before line 9, then M is a maximum matching in $G(r, \lambda(\texttt{e.edge}, r))$ after line 9 because the augmenting path algorithm .

It will suffice to show that $\texttt{M} \subseteq G(r, \lambda(\texttt{e.edge}, r))$ whenever the execution reaches line 9. We proceed by induction. For the base case, in the first execution of line 9, the matching M is initially empty.

In the inductive case, we have $\texttt{M} \subseteq G(r, \lambda(\texttt{e.edge}, r))$ after line 9. By Corollary 7, this is equivalent to $\texttt{M} \subseteq \texttt{Q.D} \cup \texttt{Q.R} \setminus \texttt{Q.L}$. This still holds just before line 9 next executes, because Q.L has not increased and every edge popped from Q.D or Q.R has been removed from M. □

**Theorem 11** *Given persistence diagrams A and B, Algorithm 2 outputs* $\mathrm{d_{SB}}(\hat{A}, \hat{B})$ *in time* $O(n^{3.5})$ *where* $n = |A| + |B|$.

**Proof.** Each iteration makes progress toward termination. For iterations where we reinsert an edge into Q.L, we set $r$ to Q.newradius(), guaranteeing an R-event will be processed next (Lemma 8). In all other cases, we shrink Q.L, Q.R, or Q.D. Thus the outer loop executes at most $2|A||B| + |A| + |B|$ times. As in [6], line 9 takes time $O((|A| + |B|)^{1.5})$ per augmenting path. We find at most $|A| + |B|$ paths the first time we augment the matching, and subsequently we find at most one path per event, as in [4]. Thus the total running time is $O((|A| + 1)(|B| + 1)(|A| + |B|)^{1.5})$, i.e. $O(n^{3.5})$.

Initially, $r = \texttt{value}(\mathcal{D}(\infty)) \geq \mathrm{d_{SB}}(\hat{A}, \hat{B})$. (Note $\mathcal{D}(\infty)$ is diagonal-perfect.) Because M is diagonal-perfect at line 12, Lemma 9 tells us that $r \geq \mathrm{d_{SB}}(\hat{A}, \hat{B})$ always and that $r$ always decreases at line 12.

We reinsert an edge $e$ in Q.L unless $G(r, \lambda(e, r))$ contains no diagonal-perfect matching. When the radius decreases from $r$ to $r'$, we get $G(r', \lambda(e, r')) \subseteq G(r, \lambda(e, r))$ for each edge $e$ by Lemma 5. So by induction, there is never a diagonal-perfect matching in $G(r, \lambda(e, r))$ for any edge $e$ in $A \times B \setminus \texttt{Q.L}$ at the start of the loop. The base case is vacuous.

When we exit the loop, we have $G(r, \lambda(e, r)) = \mathcal{D}(r)$ for every edge $e$ in Q.L. (This is vacuous if Q.L is empty; if Q.D is empty, then $r = 0$; otherwise, Q.R is empty, and Theorem 6 applies.) So we know there is no diagonal-perfect matching in $G(r, \lambda(e, r))$ for any edge $e$ in $A \times B$. Thus $r \leq \mathrm{d_{SB}}(\hat{A}, \hat{B})$ by Lemma 4. Now $r \leq \mathrm{d_{SB}}(\hat{A}, \hat{B}) \leq r$, and so $r = \mathrm{d_{SB}}(\hat{A}, \hat{B})$. □

### 5.3 A Constant-Factor Improvement

At line 6 of Algorithm 1, whenever several edges in Q.L have the same left shift, we discard all but one of them. With negligible extra effort, we set e to be the edge maximizing $\rho(e, 0)$. Then before line 9 of Algorithm 2, we test, for the event e, whether $\lambda(\texttt{e.edge}, r) \geq \rho(\texttt{e.edge}, r)$, and if so we skip the rest of the iteration. (In particular, we skip the expensive line 9.)

If $\lambda(\texttt{e.edge}, r) \geq \rho(\texttt{e.edge}, r)$, then e.edge is not in $G(r, \lambda(\texttt{e.edge}, r))$, and neither are any other edges with the same left shift as e.edge. This means $G(r, \lambda(\texttt{e.edge}, r)) \subseteq G(r, \lambda(e', r))$, where $e'$ is the previous edge popped from Q.L. Since we have no hope of finding a diagonal-perfect matching, it is sound to skip the rest of the iteration.

## 6 Implementation

We have implemented Algorithm 2 with Python 3. Our near-neighbor structure uses a kd-tree. Our implementation is slow for even small point sets. (This is consistent with the $O(n^{3.5})$ running time.) For inputs of sizes 32 (i.e. $|A| = |B| = 32$), 64, and 128, the computation takes about two seconds, 15 seconds, and two minutes. To compute shifted bottleneck distance for medium or large point sets, we will need a faster algorithm.

Our implementation needed some tweaking to account for floating point errors. The research-grade code is available on request.

## References


[1] H. Alt, K. Mehlhorn, H. Wagener, and E. Welzl. Congruence, similarity, and symmetries of geometric objects. In *Proceedings of the Third Annual Symposium on Computational Geometry*, SCG '87, pages 308–315, New York, NY, USA, 1987. ACM.

[2] D. Cohen-Steiner, H. Edelsbrunner, and J. Harer. Stability of persistence diagrams. *Comput. Geom*, 37:103–120, 2007.

[3] H. Edelsbrunner and D. Morozov. Persistent homology: Theory and practice. In *European Congress of Mathematics Kraków, 2 - 7 July, 2012*, pages 31 – 50. European Mathematical Society Publishing House, 2012.

[4] A. Efrat, A. Itai, and M. Katz. Geometry helps in bottleneck matching and related problems. *Algorithmica*, 31(1):1–28, 2001.

[5] B. Hudson, G. L. Miller, S. Y. Oudot, and D. R. Sheehy. Topological inference via meshing. In *Proceedings of the 26th ACM Symposium on Computational Geometry*, pages 277–286, 2010.

[6] M. Kerber, D. Morozov, and A. Nigmetov. Geometry helps to compare persistence diagrams. *CoRR*, abs/1606.03357, 2016.

## Appendix

**Lemma 12** *If two shifts $s_0$ and $s_1$ lie in the same (open) interval of* $\textsc{intvl}(P_r)$*, then* $\mathcal{E}(r, s_0) = \mathcal{E}(r, s_1)$.

**Proof.** Suppose (WLOG) that $s_0 < s_1$ and that we have some edge $e = \langle a, b\rangle \in \mathcal{E}(r, s_0) \setminus \mathcal{E}(r, s_1)$. Then, since $\mathrm{d}_\infty(a_s, b)$ is a continuous function of $s$, the intermediate value theorem tells us there is a shift $s_0 < s' < s_1$ such that $\mathrm{d}_\infty(a_{s'}, b) = r$.

This $s'$ must be either $\lambda(e, r)$ or $\rho(e, r)$, and so $s_0$ and $s_1$ do not lie in the same interval of $\textsc{intvl}(P_r)$. □

There are a few prerequisites to Lemma 4.

**Lemma 13** *Let $s_{i-1} < s < s_i < s' < s_{i+1} \in \mathbb{R}$, where $s_{i-1}, s_i, s_{i+1}$ are consecutive elements of $P_r$. Then $\mathcal{E}(r, s_i) \subseteq \mathcal{E}(r, s) \cap \mathcal{E}(r, s')$.*

**Proof.** Suppose for contradiction there is an edge $e = \langle a, b\rangle$ in $\mathcal{E}(r, s_i) \setminus \mathcal{E}(r, s)$. Lemma 12 tells us $e \notin \mathcal{E}(r, t)$ whenever $s_{i-1} < t < s_i$. Thus for the continuous function $f(t) = \mathrm{d}_\infty(a_t, b)$, we have $f(t) \geq r$ for $s_{i-1} < t < s_i$ but $f(s_i) < r$, which is impossible. Therefore $\mathcal{E}(r, s_i) \subseteq \mathcal{E}(r, s)$.

A similar argument shows $\mathcal{E}(r, s_i) \subseteq \mathcal{E}(r, s')$. □

**Lemma 14** *Let $s_i < s_{i+1} < s_{i+2}$ be consecutive elements of $P_r$ such that $s_{i+1}$ is not the left shift of any edge. If $s_i < s < s_{i+1} < s' < s_{i+2}$, then $\mathcal{E}(r, s') \subseteq \mathcal{E}(r, s)$.*

*Consequently, if there is no diagonal-perfect matching in $G(r, s_i)$, then there is no such matching in $G(r, s_{i+1})$.*

**Proof.** The statement $\mathcal{E}(r, s') \subseteq \mathcal{E}(r, s)$ holds because if we have an edge $e \in \mathcal{E}(r, s') - \mathcal{E}(r, s)$, then $\lambda(e, r) = s_{i+1}$, which violates the premise.

The second statement is immediate. □

**Proof.** [Lemma 4] Let $M$ be a less-than-$r$-matching. We know $M \subseteq \mathcal{E}(r, s) \cup \mathcal{D}(r)$ for some shift $s$. Consider two cases:

1. $\min P_r < s < \max P_r$. Lemma 13 lets us assume WLOG that $s_i < s < s_{i+1}$ for consecutive elements $s_i, s_{i+1}$ of $P_r$. Then Lemma 12 tells us $\mathcal{E}(r, s) \cup \mathcal{D}(r) = G(r, s_i)$.

   If $s_i$ is not a left shift, we can apply Lemma 14 to show that $M \subseteq G(r, s_{i-1})$. We iterate this until we reach a left shift. (If we never reach a left shift, then $M \subseteq \mathcal{D}(r)$.)
2. $s \leq \min P_r$ or $s > \max P_r$. This means $\mathcal{E}(r, s) = \emptyset$, so $M \subseteq \mathcal{D}(r) \subseteq G(r, t)$ for any $t$ in $P_r$.

□

**Proof.** [Lemma 5] Pick some small offset $t$ such that $G(r', \lambda(e, r')) = \mathcal{E}(r', \lambda(e, r') + t) \cup \mathcal{D}(r')$ and $G(r, \lambda(e, r)) = \mathcal{E}(r, \lambda(e, r) + t) \cup \mathcal{D}(r)$. It is clear that $\mathcal{D}(r') \subseteq \mathcal{D}(r)$.

Let $e'$ be an edge in $\mathcal{E}(r', \lambda(e, r') + t)$. This means

$$\lambda(e', r') < \lambda(e, r') + t < \rho(e', r').$$

Subtracting $(r - r')$ from all three sides yields

$$\lambda(e', r) < \lambda(e, r) + t < \rho(e', r) - 2(r - r').$$

This means $e' \in \mathcal{E}(r, \lambda(e, r) + t) \subseteq G(r, \lambda(e, r))$, since $r - r' > 0$. Thus $G(r', \lambda(e, r')) \subseteq G(r, \lambda(e, r))$. □

**Proof.** [Theorem 6] Let $\{\mathtt{e}_i\}_{1 \leq i \leq n}$ be the results of the $n$ successive calls $\mathtt{Q.nextevent}(r_i)$.

Since $\mathtt{e}_n$ is an L-event, we know the condition at line 1 is false during the call $\mathtt{Q.nextevent}(r_n)$. Therefore, there are no edges in $\mathtt{Q.D}$ of length $r_n$ or less, and so $\mathtt{Q.D} \subseteq \mathcal{D}(r_n)$. Because $r_i \geq r_n$ for all $i < n$, we have only popped edges longer than $r_n$ from $\mathtt{Q.D}$. Thus $\mathcal{D}(r_n) \subseteq \mathtt{Q.D}$, and (19) follows.

Because the order of $\mathtt{Q.L}$ is independent of the radius, and because we pop all edges with the same left shift as $\mathtt{e}_n\mathtt{.edge}$, we have

$$\mathtt{Q.L} = \{e \in A \times B \mid \lambda(e, r_n) > \lambda(\mathtt{e}_n\mathtt{.edge}, r_n)\}.$$

If $\mathtt{Q.L}$ is nonempty, then $\mathtt{e}_n\mathtt{.shift} < \lambda(\mathtt{Q.L.top}(), r_n)$. Thus

$$\mathtt{Q.L} = \{e \in A \times B \mid \lambda(e, r_n) > \mathtt{e}_n\mathtt{.shift}\}. \tag{21}$$

Let

$$S = \{e \in A \times B \mid \lambda(\mathtt{e}_n\mathtt{.edge}, r_n) < \rho(e, r_n)\}.$$

We will prove $\mathtt{Q.R} = S$, which implies

$$\mathtt{Q.R} = \{e \in A \times B \mid \mathtt{e}_n\mathtt{.shift} < \rho(e, r_n)\}. \tag{22}$$

Because the call $\mathtt{Q.nextevent}(r_n)$ returns an L-event, we know $\mathtt{Q.R} \subseteq S$ because of line 3. Fix an edge $e$ in $A \times B \setminus \mathtt{Q.R}$. Let $\mathtt{e}_i$ be the R-event such that $e = \mathtt{e}_i\mathtt{.edge}$. At the time when $\mathtt{Q.nextevent}(r_i)$ returns $\mathtt{e}_i$, we have $\rho(e, r_i) \leq \lambda(\mathtt{Q.L.top}(), r_i) \leq \lambda(\mathtt{e}_n\mathtt{.edge}, r_i)$. Because $r_i \leq r_n$, we get $\rho(e, r_n) \leq \lambda(\mathtt{e}_n\mathtt{.edge}, r_n)$, which means $e$ is not in $S$. This means $S \subseteq \mathtt{Q.R}$, and so $S = \mathtt{Q.R}$.

From (21) and (22), we get

$$\begin{aligned}\mathtt{Q.R} \setminus \mathtt{Q.L} &= \{e \in A \times B \mid \lambda(e, r_n) < \mathtt{e}_n\mathtt{.shift} < \rho(e, r_n)\} \\ &= \mathcal{E}(r_n, \mathtt{e}_n\mathtt{.shift}).\end{aligned}$$

□