The Kinetic Hourglass Data Structure for Computing the Bottleneck Distance of Dynamic Data

Elizabeth Munch* Elena Xinyi Wang† Carola Wenk‡

Abstract

The kinetic data structure (KDS) framework is a powerful tool for maintaining various geometric configurations of continuously moving objects. In this work, we introduce the kinetic *hourglass*, a novel KDS designed to compute the bottleneck distance for geometric matching problems. We detail the events and updates required for handling general graphs, accompanied by a complexity analysis. Furthermore, we demonstrate the utility of the kinetic hourglass by applying it to compute the bottleneck distance between two persistent homology transforms (PHTs) derived from shapes in $\mathbb{R}^2$, which are topological summaries obtained by computing persistent homology from every direction in $\mathbb{S}^1$.

1 Introduction

Motion is a fundamental property of the physical world. To address this challenge computationally, Basch et al. proposed the *kinetic data structure* (KDS) framework [6], designed to maintain various geometric configurations for continuously moving objects. The KDS framework has been applied to many geometric problems since it was introduced, including but not limited to finding the convex hull of a set of moving points in the plane [5], the closest pair of such a set [5], a point in the center region [1], kinetic medians and *kd*-trees [2], and range searching; see [19] for a survey. In this work, we extend the framework to the geometric matching problem. Specifically, we are interested in the min-cost matching of a weighted graph with continuously changing weights on the edges. The result is that the bottleneck cost can be updated, rather than recomputing the needed matching from scratch every time.

The static bottleneck distance between persistence diagrams has been studied extensively, both theoretically and practically [7, 14, 16, 22, 23]. Since the bottleneck distance between persistence diagrams can be formulated as a bipartite graph matching problem [15], the work in this abstract can be used to improve bottleneck distance computation for a vineyard [10, 24, 28], more generally in the case of the Persistent Homology Transform [11, 17, 26], or even more generally for Persistent Homology Bundles [4, 20, 21].

*Department of Mathematics, Department of Computational Mathematics, Science, and Engineering, Michigan State Universitymuncheli@msu.edu

†Institute of Geometry, Graz University of Technology, x.wang@tugraz.at

‡Department of Computer Science, Tulane University, cwenk@tulane.edu

2 Background

Broadly, we are interested in the following geometric matching problem. Given an undirected graph $G = (V, E)$, a *matching* M of G is a subset of the edges E such that no vertex in V is incident to more than one edge in M. A vertex v is said to be *matched* if there is an edge $e \in M$ that is incident to v. A matching is *maximal* if it is not properly contained in any other matching. A *maximum* matching M is a matching with the largest cardinality; i.e., for any other matching M', $|M| \geq |M'|$. A maximum matching is always maximal; the converse is not true.

For a bipartite graph $G = (X \sqcup Y, E)$ where $|X| = |Y| = n$ and $|E| = m$, a maximum matching is a *perfect* matching if every $v \in X \sqcup Y$ is matched, and $|M| = n$. This can be expressed as a bijection $\eta : X \to Y$. For a subset $W \subseteq X$, let $N(W)$ denote the *neighborhood* of W in G, the set of vertices in Y that are adjacent to at least one vertex of W. Hall's marriage theorem provides a necessary condition for a bipartite graph to have a perfect matching.

Theorem 1 (Hall's Marriage Theorem) *A bipartite graph $G = (X \sqcup Y, E)$ has a perfect matching if and only if for every subset W of X: $|W| \leq |N(W)|$.*

Building on Hall's Marriage Theorem, we extend our focus to weighted bipartite graphs. Given such a graph, a fundamental optimization problem is to identify matchings that minimize the maximum edge weight, known as the *bottleneck cost*.

Definition 2 *A weighted graph $\mathcal{G} = (G, c)$ is a graph G together with a weight function $c : E \to \mathbb{R}_+$. The* bottleneck cost *of a matching M for such a $\mathcal{G}$ is $\max\{c(e) \mid e \in M\}$. The* bottleneck edge *is the highest weighted edge in M, assuming this is unique. A perfect matching is* optimal *if its cost is minimal among all perfect matchings. An optimal matching is also called a* min-cost *matching.*

To find a maximum matching of a graph, we use augmenting paths.

Definition 3 *For a graph G and matching M, a path P is an* augmenting path *for M if:*

1. *the two end points of P are unmatched in M, and*
2. *the edges of P alternate between edges $e \in M$ and $e \notin M$.*

Theorem 4 (Berge's Theorem) *A matching M in a graph G is a maximum matching if and only if G contains no M-augmenting path.*

The existing algorithms that compute the bottleneck cost are derived from the Hopcroft-Karp maximum matching algorithm, which we briefly review [22]. Given a graph $G = (X \sqcup Y, E)$ where $|E| = n$, and an initial matching M, the algorithm iteratively searches for augmenting paths P. Each phase begins with a breadth-first search from all unmatched vertices in X, creating a layered subgraph of G by alternating between $e \in M$ and $e \notin M$. It stops when an unmatched vertex in Y is reached. From this layered graph, we can find a maximal set of vertex-disjoint augmenting paths of the shortest length. For each P, we augment M by replacing $M \cap P$ with $P \backslash M$. We denote this process as $\mathrm{Aug}(M, P) = M \setminus (M \cap P) \cup (P \setminus M)$. Note that $|\mathrm{Aug}(M, P)|$ is a matching with $|\mathrm{Aug}(M, P)| = |M|+1$, so we can repeat the above process until no more augmenting paths can be found. By Theorem 4, the resulting M is maximum. The algorithm terminates in $O(\sqrt{n})$ rounds, resulting in a total running time of $O(n^{2.5})$. This algorithm was later improved to $O(n^{1.5} \log n)$ for geometric graphs by Efrat et al. by constructing the layered graph via a near-neighbor search data structure [16].

2.1 Bottleneck Distance

Let X and Y be two sets of n points. We consider the bipartite graph obtained by adding edges between points whose weight $\mathrm{c} : E \to \mathbb{R}_{\geq 0}$ is at most λ:

$$G_\lambda = (X \sqcup Y, \{xy \mid \mathrm{c}(xy) \leq \lambda\}).$$

Definition 5 *The* optimal bottleneck cost *is the minimum λ such that G_λ has a perfect matching, denoted as $d_B(G)$.*

We are interested in computing the bottleneck distance in the context of *persistent homology*. Persistent homology is a multi-scale summary of the connectivity of objects in a nested sequence of subspaces; see [13] for an introduction. For the purposes of this section, we can define a *persistence diagram* to be a finite collection of points $\{(b_i, d_i)\}_i$ with $d_i \geq b_i$ for all i. Further details and connections to the input data will be given in Section 4.

Given two persistence diagrams X and Y, a partial matching is a bijection $\eta : X' \to Y'$ on a subset of the points $X' \subseteq X$ and $Y' \subseteq Y$; we denote this by $\eta : X \rightleftharpoons Y$. The cost of a partial matching is the maximum over the L_∞-norms of all pairs of matched points and the distance between the unmatched points to the diagonal:

$$c(\eta) = \max\big(\{\|x - \eta(x)\|_\infty \mid x \in X'\} \cup \{\tfrac{1}{2}|z_2 - z_1| \mid (z_1, z_2) \in (X \setminus X') \cup (Y \setminus Y')\}\big)$$

and the bottleneck distance is defined as $d_B(X, Y) = \inf_{\eta : X \rightleftharpoons Y} c(\eta)$.

We now reduce finding the bottleneck distance between persistence diagrams to a problem of finding the bottleneck cost of a bipartite graph. Let X and Y be two persistence diagrams given as finite lists of off-diagonal points. For any off-diagonal point $z = (z_1, z_2)$, the orthogonal projection to the diagonal is $z' = ((z_1+z_2)/2, (z_1+z_2)/2)$. Let $\overline{X}$ (resp. $\overline{Y}$) be the set of orthogonal projections of the points in X (resp. Y). Set $U = X \sqcup \overline{Y}$ and $V = Y \sqcup \overline{X}$. We define the complete bipartite graph $G = (U \sqcup V, U \times V, \mathrm{c})$, where for $u \in U$ and $v \in V$, the weight function c is given by

$$\mathrm{c}(uv) = \begin{cases} \|u - v\|_\infty & \text{if } u \in X \text{ or } v \in Y \\ 0 & \text{if } u \in \overline{X} \text{ and } v \in \overline{Y}. \end{cases}$$

An example of the bipartite graph construction is shown in Figure 1. This graph can be used to compute the bottleneck distance of the input diagrams because of the following lemma.

Lemma 6 (Reduction Lemma [15]) *For the above construction of G, $d_B(G) = d_B(X, Y)$.*

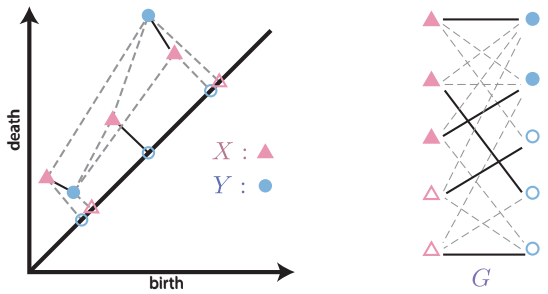

Figure 1: Construction of the bipartite graph G based on the persistence diagrams X and Y.

Naively, given a graph G of size n, we can compute the bottleneck distance by sorting the edge weights and performing a binary search for the smallest λ such that G_λ has a perfect matching in $O(n^2 \log n)$ time, which dominates the improved Hopcroft-Karp algorithm. Using the technique for efficient k-th distance selection for

a bi-chromatic point set under the L_∞ distance introduced by Chew and Kedem, this can be reduced to $O(n^{1.5}\log n)$ [8]. Therefore, the overall complexity to compute the bottleneck distance of a static pair of persistence diagrams is $O(n^{1.5}\log n)$.

2.2 Kinetic Data Structure

A kinetic data structure (KDS) [5, 18, 19] maintains a system of objects v that move along a known continuous *flight plan* as a function of time, denoted as $v = f(v)$. *Certificates* are conditions under which the data structure is accurate, and *events* track when the certificate fails and what corresponding *updates* need to be made. The certificates of the KDS form a proof of correctness of the current configuration function at all times. Updates are stored in a priority queue, keyed by event time. To advance to time t, we process all the updates keyed at times before t and pop them from the queue after updating. We continue until t is smaller than the first event time in the queue. Finally, we distinguish between *external events*, i.e., those affecting the configuration function (e.g., maximum or minimum values) we are maintaining, and *internal events*, i.e., those that do not affect the outcome.

The kinetization of a heap results in a *kinetic heap*, which maintains the priority of a set of objects as they change continuously. Maximum and minimum are both examples of priorities. A kinetic heap follows the properties of a static heap such that objects are stored as a balanced tree. The value of a node is stored as a function of time $f_X(t)$. The data structure augments a certificate for every pair of parent-child nodes A and B, which is only valid when $f_A(t) > f_B(t)$ (resp. $f_A(t) < f_B(t)$) for a max (resp. min) heap. Thus, the failure times of the certificate are the times t such that $f_A(t) = f_B(t)$, and updates are done by swapping A and B in the heap while making all relevant parent-child updates, see Figure 2. The kinetic max heap supports operations similar to a static heap, such as create-heap, find-max, insert, and delete. Insertion and deletion are done in $O(\log n)$ time.

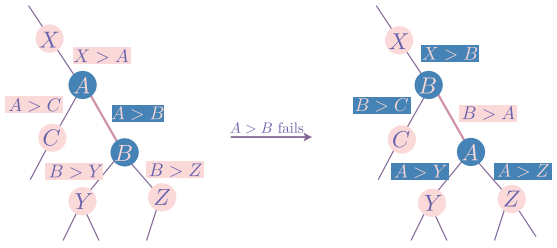

Figure 2: Kinetic heap event updates

More efficient solutions to this problem have been developed, such as the *kinetic hanger*, introduced by da Fonseca et al. in [12]. It modifies the kinetic heap by incorporating randomization to balance the tree, meaning all complexity results are expected rather than deterministic. As shown in [12], the kinetic hanger has $O(1)$ locality. The number of expected events in the affine motion case is $O(n\log n)$ and $O(\lambda_s(n)\log n)$ for n s-intersecting curves. The expected runtime is obtained by multiplying by $O(\log n)$ for each event. We use $\alpha(\cdot)$ to denote the inverse Ackerman function and λ_s is the length bound for the Davenport-Schinzel Sequence; see [3] for details.

3 Kinetic Hourglass

In this section, we introduce a new kinetic data structure that keeps track of the optimal bottleneck cost of a weighted graph $\mathcal{G} = (G, \mathrm{c})$ where c changes continuously with respect to time t. The *kinetic hourglass* data structure is composed of two kinetic heaps; in Section 3.3 we will give the details for replacing these with kinetic hangers. One heap maintains minimum priority, and the other maintains maximum. Assume we are given a connected bipartite graph $G = (V, E)$ with the vertex set $V = X \sqcup Y$, where $|X| = |Y| = n$; and edge set E, where $|E| = m$. If G is a complete bipartite graph, then $m = n^2$. The weight of the edges at time t is given by $\mathrm{c}^t : E \to \mathbb{R}_{\geq 0}$. Denote the weighted bipartite graph by $\mathcal{G}^t = (G, \mathrm{c}^t)$. The weights of those m edges are the objects we keep track of in our *kinetic hourglass*. We assume that these weights, called flight plans in the kinetic data structure setting, are given for all times $t \in [0, T]$.

Let $G^t_\delta \subseteq G$ be the portion of the complete bipartite graph with all edges with weight at most δ at time t; i.e., $V(G^t_\delta) = V$, and $E(G^t_\delta) = \{e \in E \mid \mathrm{c}^t(e) \leq \delta\}$. If the bottleneck distance $\hat{\delta}^t = d_B(\mathcal{G}^t)$ is known, we are focused on the bipartite graph $G^t_{\hat{\delta}}$, which we will denote by $\hat{G}^t$ for brevity. By definition, we know that there is a perfect matching in $\hat{G}^t$, which we denote as M^t, although we note that this is not unique. Further, there is an edge $\hat{e}^t \in M^t$ with $\mathrm{c}(\hat{e}^t) = \hat{\delta}^t$, which we call the *bottleneck edge*. This edge is unique as long as all edges have unique weights, except for when events happen. We separate the remaining edges into the sets

$$L^t = E(\hat{G}^t) \setminus E(M^t), \text{ and}$$
$$U^t = E \setminus E(\hat{G}^t) = \{e \in E \mid \mathrm{c}^t(e) > \hat{\delta}^t\}$$

so that $E = L^t \sqcup M^t \sqcup U^t$.

The kinetic hourglass consists of the following kinetic max heap and kinetic min heap. The *lower heap* $\mathcal{H}_L$ is the max heap containing $L^t \sqcup M^t = E(\hat{G}^t)$. The *upper heap* $\mathcal{H}_U$ is the min heap containing $U^t \cup \{\hat{e}\}$. See Figure 3 for an example of this construction. Note that $\mathrm{c}^t(\hat{e}) = \max\{\mathrm{c}^t(e) \mid e \in L^t \sqcup M^t\}$ and $\mathrm{c}^t(\hat{e}) <$

Scenario	Kinetic Heap	Kinetic Hanger
Lines	$O(n \log^2 n)$	$O(n \log^2 n)$
Line segments	$O(n\sqrt{m} \log^{3/2} m)$	$O(n\alpha(m) \log^2 m)$
s-intersecting curves	$O(n^2 \log^2 n)$	$O(\lambda_s(n) \log^2 n)$
s-intersecting curve segments	$O(mn \log^2 m)$	$O(n/m\lambda_{s+2}(m) \log^2 m)$

Table 1: Deterministic complexity of the kinetic heap and expected complexity of the kinetic hanger. Here, n is the total number of elements, m denotes the maximum number of elements stored at a given time, s is a constant.

$\min\{c^t(e) \mid e \in U^t\}$, meaning that $\hat{e}$ is the root for both heaps.

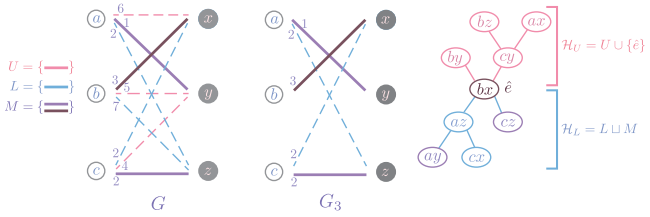

Figure 3: Illustration of construction of the kinetic hourglass.

3.1 Certificates

The certificates for the kinetic hourglass (i.e., properties held by the data structure for all $t \in [0, T]$) are

1. All max-heap certificates for $\mathcal{H}_L$ and min-heap certificates for $\mathcal{H}_U$.
2. Both heaps have the same root, denoted r^t.
3. The edge r^t is the edge with bottleneck cost; i.e., $r^t = \hat{e}^t$ where $c^t(\hat{e}^t) = \hat{\delta}^t$.

Assuming the certificates are maintained, r^t and $\hat{e}^t$ are the same edge. However, in the course of proofs, bottleneck edge of the matching is denoted by $\hat{e}^t$, while we use r^t for the edge stored in the root of the two heaps (or $r^t(\mathcal{H}_U)$ and $r^t(\mathcal{H}_L)$ when a distinction is needed).

3.2 Events

For a particular event time t, we denote the moment of time just before an event by $t^- = t - \varepsilon$ and the moment of time just after by $t^+ = t + \varepsilon$. For two edges, we write $a \preccurlyeq_t b$ to mean that $c^t(a) \leq c^t(b)$. The two heaps have their own certificates, events (both internal and external), and updates as in the standard setting. Our main task of this section is to determine which events in the heaps lead to an external event in the hourglass. We define the external events for the hourglass as those events in $\mathcal{H}_U$ and $\mathcal{H}_L$ which lead to changes of the root, r^t. Thus, internal events are those which do not affect the roots; i.e., $r^{t^-}(\mathcal{H}_U) = r^{t^+}(\mathcal{H}_U)$ and $r^{t^-}(\mathcal{H}_L) = r^{t^+}(\mathcal{H}_L)$.

We first show that an internal event of $\mathcal{H}_U$ or $\mathcal{H}_L$ is an internal event of the hourglass.

Lemma 7 *If the event at time t is an internal event of $\mathcal{H}_U$ or $\mathcal{H}_L$ and the kinetic hourglass satisfies all certificates at time t^-, then the edge giving the bottleneck distance for times t^- and t^+ is the same; that is, $\hat{e}^{t^-} = \hat{e}^{t^+}$.*

Proof. Following the previous notation, we have bottleneck distances before and after given by $\hat{\delta}^{t^-} = c^{t^-}(\hat{e}^{t^-})$ and $\hat{\delta}^{t^+} = c^{t^+}(\hat{e}^{t^+})$. Because we start with a correct hourglass, we know that $\hat{e}^{t^-} = r^{t^-}(\mathcal{H}_U) = r^{t^-}(\mathcal{H}_L) = r^{t^-}$. By definition, an internal event in either heap is a swap of two elements with a parent-child relationship but for which neither is the root so the roots remain unchanged; that is, $r^{t^-} = r^{t^+}$ so we denote it by r for brevity. This additionally means that no edge moves from one heap to the other, so the set of elements in each heap does not change and thus $G_{c^{t^-}(r)} = G_{c^{t^+}(r)}$. Again for brevity, we write this subgraph as Γ.

We need to show that this edge r is the one giving the bottleneck distance at t^+, i.e. $\hat{\delta}^{t^+} = c^{t^+}(r)$ or equivalently that $r = \hat{e}^{t^+}$. All edges of the perfect matching from t^-, M^{t^-}, are contained in Γ; thus M^{t^-} is still a perfect matching at time t^+. We show that any minimal cost perfect matching for t^+ must contain r, and thus M^{t^+} is a *minimal* cost perfect matching. Since r is $\hat{e}^{t^-}$, by removing r, $\Gamma \backslash \{r\}$ will cease to have a perfect matching, else contracting the minimality of the perfect matching at t^-. But as the order is unchanged, this further means that for time t^+, lowering the threshold for the subgraph $\Gamma = G_{c^{t^+}(r)}$ or equivalently removing the edge r will not have a perfect matching, finishing the proof. □

The remaining cases to consider are external events of $\mathcal{H}_U$ and $\mathcal{H}_L$, when a certificate in one of the heaps involving the root fails. This can be summarized in the three cases below. In each case, denote the root at time t^- as $r = r^{t^-}$.

1. (L-Event) Swap priority of r and an $e \in L^t$ in $\mathcal{H}_L$; i.e. $e \preccurlyeq_{t^-} r$ and $r \preccurlyeq_{t^+} e$.
2. (M-Event) Swap priority of r and an $e \in M^t$ in $\mathcal{H}_L$; i.e. $e \preccurlyeq_{t^-} r$ and $r \preccurlyeq_{t^+} e$.

3. (U-Event) Swap priority of r and an $e \in U^t$ in $\mathcal{H}_U$; i.e. $r \preccurlyeq_{t^-} e$ and $e \preccurlyeq_{t^+} r$.

In the remainder of this section, we consider each of those events and provide the necessary updates. The simplest update comes from the first event in the list, since we will show that no additional checks are needed.

Lemma 8 (L-Event) *Assume we swap priority of $r = r^{t^-}$ and an $e \in L^{t^-}$ at t; i.e. $e \preccurlyeq_{t^-} r$ and $r \preccurlyeq_{t^+} e$. Then e moves from $\mathcal{H}_L$ to $\mathcal{H}_U$, and r remains the root and is the edge with the bottleneck cost for t^+, i.e., $r^{t^+} = \hat{e}^{t^+}$.*

Proof. Denote $M = M^{t^-}$. In this case, e is an edge in the lower heap but $e \notin M$. Because $r \preccurlyeq_{t^+} e$, in order to maintain the heap certificates, either e needs to be inserted into the upper heap with r remaining as the root; or if e remains in the lower heap, it needs to become the new root. Note that although they swap orders, the graph thresholded at the cost of r at t^- and the graph thresholded at the cost of e at t^+ are the same; i.e. $G_{c^{t^-}(r)} = G_{c^{t^+}(e)}$. In addition, $G_{c^{t^+}(r)} = G_{c^{t^+}(e)} \setminus \{e\}$. However, since $e \notin M$, M is still a perfect matching in $G_{c^+(r)}$. If there exists a perfect matching in $G_{c^{t^+}(r)} \setminus \{r\}$, this would constitute a perfect matching at time t^- which has lower cost than M, contradicting the assumption that M is a minimal cost matching for that time. Thus M is a minimal cost matching for time t^+. □

Lemma 9 (M-Event) *Assume we swap priority of $r = r^{t^-}$ and an $e \in M^{t^-}$ at t; i.e. $e \preccurlyeq_{t^+} r$ and $r \preccurlyeq_{t^-} e$. Let $G' = \hat{G}^{t^-} \setminus \{e\}$ be the graph with $e = (u, v)$ removed. Exactly one of the following scenarios happens. (See Figure 4 and 5 for examples of the two scenarios.)*

1. *There exists an augmenting path P in G' from u to v. Then e moves into the upper heap ($e \in U^{t^+}$), the root remains the same ($\hat{e}^{t^+} = r^{t^+} = r$), and the matching is updated with the augmenting path, specifically $M^{t^+} = \mathrm{Aug}(M^{t^-}, P)$.*
2. *There is no such augmenting path. Then $M^{t^+} = M^{t^-}$ and $\hat{e}^{t^+} = r^{t^+} = e$; i.e. the only update is that r and e switch places in the lower heap.*

Lemma 10 (U-Event) *Assume we swap priority of $r = r^{t^-}$ and an $e \in U^{t^-}$ at t; i.e. $r \preccurlyeq_{t^-} e$ and $e \preccurlyeq_{t^+} r$. Let $G' = \hat{G}^{t^-} \cup \{e\} \setminus \{r\}$ be the graph with $r = (u, v)$ removed and e included. Exactly one of the following events happens.*

1. *There exists an augmenting path P from u to v. Then r moves into the upper heap ($r \in U^{t^+}$), e becomes the root ($\hat{e}^{t^+} = r^{t^+} = e$), and the matching is updated with the augmenting path, specifically $M^{t^+} = \mathrm{Aug}(M^{t^-}, P)$.*

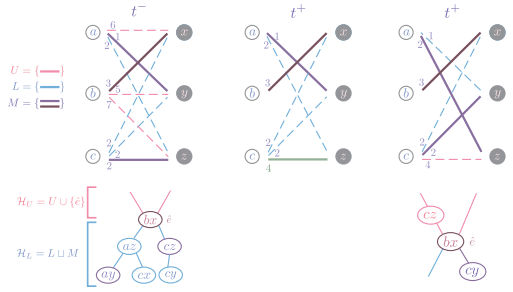

Figure 4: Illustration of Scenario 1 of and M-Event; see Lemma 9.

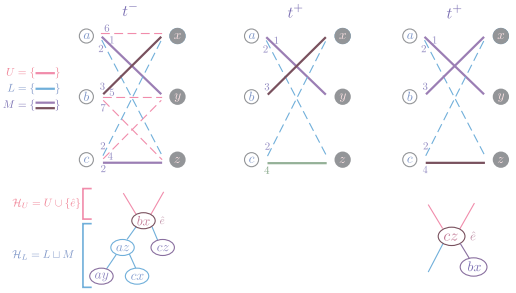

Figure 5: Illustration of Scenario 2 of M-Event; see Lemma 9.

2. *There is no such augmenting path. Then $M^{t^+} = M^{t^-}$ and e moves into the lower heap ($e \in L^{t^+}$), and the root remains the same ($\hat{e}^{t^+} = r^{t^+} = r$).*

We have included the proofs of the above lemmas in Appendix A and B. For further details, see [25]

3.3 Complexity and Performance Evaluation

The kinetic hourglass builds on the kinetic heap and hanger frameworks [12], with complexities derived by multiplying the number of events by the $O(\log n)$ event processing time (Table 1). In contrast, the hourglass requires $O(m)$ work per external event due to augmenting path searches, replacing the $\log n$ factor with m.

For a bipartite graph with m edges, we treat $m = n$ and analyze each edge's behavior only during its presence in a heap. This leads to complexities summarized in Table 2: for linear weights, $O(m^2\sqrt{m}\log^{3/2} m)$ using heaps and $O(m^2\alpha(m)\log m)$ with hangers; for s-intersecting curves, $O(m^3 \log m)$ and $O(m\lambda_{s+2}(m)\log m)$, respectively.

Despite being responsive, local, and compact, the hourglass is not efficient due to its linear per-event cost. We conjecture that amortized analysis may reduce this, though it remains an open problem.

Scenario	Kinetic Heap Hourglass	Kinetic Hanger Hourglass
Line segments	$O(m^{5/2}\log^{1/2} m)$	$O(m^2\alpha(m)\log m)$
s-intersecting curve segments	$O(m^3 \log m)$	$O(m\lambda_{s+2}(m)\log m)$

Table 2: Deterministic complexity of the kinetic heap hourglass and expected complexity of the kinetic hanger hourglass.

4 Kinetic Hourglass for Persistent Homology Transform

We apply the kinetic hourglass to compute the integrated bottleneck distance between two 0-dimensional persistent homology transforms (PHTs) [11, 17, 26], a directional transform shape signature derived from sublevel set filtrations. For simplicity, we focus on connected, embedded graphs in $\mathbb{R}^2$ whose PHTs are composed of 0-dimensional persistence diagrams.

Each unit vector $\omega \in \mathbb{S}^1$ induces a height function h_ω on the vertices of a simplicial complex K, resulting in a lower-star filtration and a persistence diagram $\mathrm{Dgm}(h_\omega^K)$. The PHT is the function $\mathrm{PHT}(K) : \omega \mapsto \mathrm{Dgm}(h_\omega^K)$. The integrated bottleneck distance between two PHTs is

$$d_B(\mathrm{PHT}(K_1), \mathrm{PHT}(K_2)) = \int_0^{2\pi} d_B\big(\mathrm{Dgm}(h_\omega^{K_1}), \mathrm{Dgm}(h_\omega^{K_2})\big)\, d\omega$$

which is stable under small vertex perturbations.

To enable tractable runtime analysis, we restrict our attention to *star-shaped* complexes, where each persistence vineyard exhibits *trivial geometric monodromy* [4]. In this case, each off-diagonal persistence point traces a continuous curve (vine) over $\mathbb{S}^1$.

We construct a complete bipartite graph G whose vertices correspond to vines and their diagonal projections from the PHTs of K_1 and K_2. The edge weights are determined by ℓ^∞ distances between points on the vines at a given direction ω. This reduces the integrated bottleneck distance computation to maintaining a minimum-cost matching in G across all directions ω. Figure 6 gives an example of the pipeline.

Given at most $2n$ vines from each complex, G has $O(n^2)$ edges, leading to $O(n^6 \log n)$ runtime for the kinetic hourglass using heaps. See Appendix C for full definitions of the filtration, stability, and geometric interpretation of the PHT.

5 Conclusions

In this paper, we introduced the *kinetic hourglass*, a novel kinetic data structure designed for maintaining the bottleneck distance for graphs with continuously changing edge weights. The structure incorporates two kinetic priority queues, which can be either the deterministic kinetic heap or the randomized kinetic hanger. Both versions are straightforward to implement.

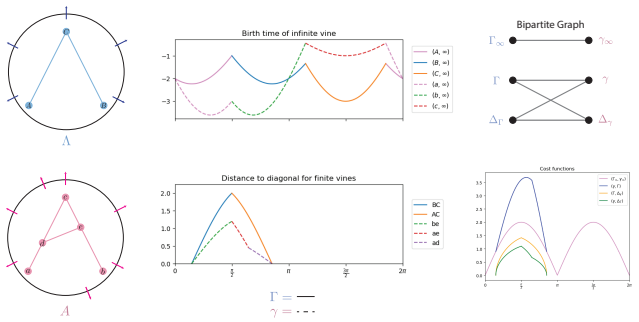

Figure 6: Example of edge weight functions in the bipartite graph induced by two PHTs.

In the future, we hope to improve the runtime for this data structure. In particular, the augmenting path search requires $O(n)$ time, falling short of the efficiency goals in the kinetic data structure (KDS) framework. Moreover, when comparing PHTs with n vertices, the kinetic hourglass holds n^2 elements, which can be computationally expensive. This method can immediately be extended to study the extended persistent homology transform (XPHT) to compare objects that have different underlying topologies [27], however there is much to be understood for the structure of the vines in the PHT and how this particular structure can deal with monodromy. Further, the nature of this data structure means that it is not immediately extendable to the Wasserstein distance case, however, we would like to build a modified version that will work for that case. Finally, since the kinetic hourglass data structure also has the potential to compare more general vineyards that extend beyond the PHT, it will be interesting in the future to find further applications.

Acknowledgment EXW acknowledges support from the Austrian Science Fund (FWF) grant number P 33765-N.

References

[1] P. K. Agarwal, M. De Berg, J. Gao, L. J. Guibas, and S. Har-Peled. Staying in the middle: Exact and approximate medians in R^1 and R^2 for moving points. In *CCCG*, pages 43–46, 2005.

[2] P. K. Agarwal, J. Gao, and L. J. Guibas. Kinetic medians and kd-trees. In R. Möhring and R. Raman, editors, *Algorithms — ESA 2002*, pages 5–17, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.

[3] P. K. Agarwal and M. Sharir. Davenport–schinzel sequences and their geometric applications. In J.-R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*, pages 1–47. North-Holland, Amsterdam, 2000.

[4] S. Arya, B. Giunti, A. Hickok, L. Kanari, S. McGuire, and K. Turner. Decomposing the persistent homology transform of star-shaped objects, 2024.

[5] J. Basch, L. J. Guibas, and J. Hershberger. Data structures for mobile data. *Journal of Algorithms*, 31(1):1–28, 1999.

[6] J. Basch, L. J. Guibas, and G. D. Ramkumar. Sweeping lines and line segments with a heap. In *Proceedings of the Thirteenth Annual Symposium on Computational Geometry*, SCG '97, page 469–471, New York, NY, USA, 1997. Association for Computing Machinery.

[7] S. Cabello, S.-W. Cheng, O. Cheong, and C. Knauer. Geometric Matching and Bottleneck Problems. In W. Mulzer and J. M. Phillips, editors, *40th International Symposium on Computational Geometry (SoCG 2024)*, volume 293 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 31:1–31:15, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[8] L. P. Chew and K. Kedem. Improvements on geometric pattern matching problems. In O. Nurmi and E. Ukkonen, editors, *Algorithm Theory — SWAT '92*, pages 318–325, Berlin, Heidelberg, 1992. Springer Berlin Heidelberg.

[9] D. Cohen-Steiner, H. Edelsbrunner, and J. Harer. Stability of persistence diagrams. *Discrete & Computational Geometry*, 37(1):103–120, Jan 2007.

[10] D. Cohen-Steiner, H. Edelsbrunner, and D. Morozov. Vines and vineyards by updating persistence in linear time. In *Proceedings of the Twenty-Second Annual Symposium on Computational Geometry*, SCG '06, page 119–126, New York, NY, USA, 2006. Association for Computing Machinery.

[11] J. Curry, S. Mukherjee, and K. Turner. How many directions determine a shape and other sufficiency results for two topological transforms. *Transactions of the American Mathematical Society, Series B*, 2018.

[12] G. D. da Fonseca, C. M. de Figueiredo, and P. C. Carvalho. Kinetic hanger. *Information Processing Letters*, 89(3):151–157, 2004.

[13] T. K. Dey and Y. Wang. *Computational Topology for Data Analysis*. Cambridge University Press, 2017.

[14] T. K. Dey and C. Xin. Computing Bottleneck Distance for 2-D Interval Decomposable Modules. In B. Speckmann and C. D. Tóth, editors, *34th International Symposium on Computational Geometry (SoCG 2018)*, volume 99 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 32:1–32:15, Dagstuhl, Germany, 2018. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[15] H. Edelsbrunner and J. Harer. *Computational Topology - an Introduction*. American Mathematical Society, 2010.

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

[17] R. Ghrist, R. Levanger, and H. Mai. Persistent homology and Euler integral transforms. *Journal of Applied and Computational Topology*, 2(1):55–60, Oct 2018.

[18] L. J. Guibas. Kinetic data structures: a state of the art report. In *Proceedings of the Third Workshop on the Algorithmic Foundations of Robotics on Robotics: The Algorithmic Perspective: The Algorithmic Perspective*, WAFR '98, page 191–209, USA, 1998. A. K. Peters, Ltd.

[19] L. J. Guibas and M. Roeloffzen. Modeling motion. In C. D. Toth, J. O'Rourke, and J. E. Goodman, editors, *Handbook of Discrete and Computational Geometry (3rd ed.)*, chapter 57, pages 1117 – 1134. Chapman and Hall/CRC, 2017.

[20] A. Hickok. Computing persistence diagram bundles, 2022.

[21] A. Hickok. Persistence diagram bundles: A multidimensional generalization of vineyards, Oct. 2022.

[22] J. E. Hopcroft and R. M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM J. Comput.*, 2(4):225–231, Dec 1973.

[23] M. Kerber, D. Morozov, and A. Nigmetov. Geometry helps to compare persistence diagrams. *ACM J. Exp. Algorithmics*, 22, Sep 2017.

[24] W. Kim and F. Mémoli. Spatiotemporal persistent homology for dynamic metric spaces. *Discrete & Computational Geometry*, 66(3):831–875, Oct 2021.

[25] E. Munch, E. X. Wang, and C. Wenk. The kinetic hourglass data structure for computing the bottleneck distance of dynamic data, 2025.

[26] K. Turner, S. Mukherjee, and D. M. Boyer. Persistent homology transform for modeling shapes and surfaces. *Information and Inference: A Journal of the IMA*, 3(4):310–344, 12 2014.

[27] K. Turner, V. Robins, and J. Morgan. The extended persistent homology transform of manifolds with boundary. *Journal of Applied and Computational Topology*, May 2024.

[28] L. Xian, H. Adams, C. M. Topaz, and L. Ziegelmeier. Capturing dynamics of time-varying data via topology. *Foundations of Data Science*, 4(1):1–36, 2022.

Appendix

A Proof of Lemma 9

Proof. Let $M' = M^{t^-} \setminus \{e\}$. Then M' is a matching of size $n-1$ in G' (hence it is not perfect), and the unmatched vertices are u and v.

If there exists an augmenting path P from u to v, augment M' by replacing $M' \cap P$ with $P \setminus M'$ to make a new matching M''. This increases $|M'|$ by 1 and thus M'' is a perfect matching. Note that if $r \in P$, then M'' is also a perfect matching in t^- with strictly cheaper cost than M^{t^-}; but this contradicts the assumption of r giving the bottleneck distance. Thus we can assume that $r \in M''$, and all edges in the lower heap have cost at most that of r. This means that r is still the bottleneck edge of the matching, i.e. $\mathrm{c}^{t^+}(r) = \max\{\mathrm{c}^{t^+}(e) \mid e \in M''\}$.

Assume instead that there is no augmenting path in G' from u to v. Then M' is a maximum matching for G' by Theorem 4, however it is not perfect. Note that because $G' = \hat{G}^{t^-} \setminus e$ and e and r have swapped places, we have that $G' = G_{\mathrm{c}^{t^+}(r)}$. Therefore, there is no perfect matching for $G_{\mathrm{c}^{t^+}(r)}$. However, M^{t^-} is a perfect matching at time t^- and $\hat{G}^{t^-} = G_{\mathrm{c}^{t^+}(e)}$, it still is a perfect matching at t^+ for $G_{\mathrm{c}^{t^+}(e)}$. Moreover, $\mathrm{c}^{t^+}(e) = \max\{\mathrm{c}^{t^+}(e) \mid e \in M^{t^+}\}$. Therefore, $e = \hat{e}^{t^+} = r^{t^+}$ becomes the root, and r becomes a child of the root e in $\mathcal{H}_L$. □

B Proof of Lemma 10

Proof. Let $M' = M^{t^-} \setminus \{r\}$, and again M' is a matching of size $n-1$ in G' (hence it is not perfect), and the unmatched vertices are u and v.

Similar to the M-Event, if there exists an augmenting path P from u to v, augment M' by replacing $M' \cap P$ with $P \setminus M'$ to make a new matching M''. This increases $|M'|$ by 1; thus, M'' is a perfect matching. Further, because $G' = \hat{G}^{t^-} \cup \{e\} \setminus \{r\}$ and e and r have swapped places, we have that $G' = G_{\mathrm{c}^{t^+}(e)}$. Moreover, $\mathrm{c}^{t^+}(e) = \max\{\mathrm{c}^{t^+}(e) \mid e \in M''\}$. Therefore $e = \hat{e}^{t^+} = r^{t^+}$, and r gets moved down to $\mathcal{H}_U$ and becomes a child of e.

Assume instead that there is no augmenting path in G' from u to v. Then M' is a maximum matching for G' by Theorem 4, however it is not perfect. Therefore, there is no perfect matching for $G_{\mathrm{c}^{t^+}(e)}$. However, M^{t^-} is a perfect matching at time t^- and $\hat{G}^{t^-} = G_{\mathrm{c}^{t^+}(r)} \setminus \{e\}$, it still is a perfect matching at t^+ for $G_{\mathrm{c}^{t^+}(r)}$. This is thus an internal event; we move e to $\mathcal{H}_L$ as a child of r, and r remains the bottleneck edge, i.e. $\mathrm{c}^{t^+}(r) = \max\{\mathrm{c}^{t^+}(e) \mid e \in M''\}$. □

C Persistent Homology Transform

C.1 Lower-Star Filtration and PHT Definition

Let K be a finite simplicial complex embedded in $\mathbb{R}^2$. For any direction $\omega \in \mathbb{S}^1$, the height function $h_\omega : |K| \to \mathbb{R}$ is defined by $h_\omega(x) = \langle x, \omega \rangle$. The lower-star filtration induced on the abstract complex K is given by:

$$h_\omega(\sigma) = \max\{h_\omega(v) \mid v \in \sigma\},$$

and the sublevelset at height a is the full subcomplex induced by vertices $\{v \in K \mid h_\omega(v) \le a\}$. Filtering K by h_ω produces a persistence diagram $\mathrm{Dgm}_k(h_\omega^K)$. We focus on $k = 0$, where the input can be assumed to be a connected graph, so each diagram contains a single point at infinity.

The persistent homology transform (PHT) is defined as:

$$\mathrm{PHT}(K) : \mathbb{S}^1 \to \mathcal{D}, \quad \omega \mapsto \mathrm{Dgm}(h_\omega^K),$$

where $\mathcal{D}$ is the space of persistence diagrams. The PHT is injective [11, 26], making it a faithful shape representation.

C.2 Stability of the Bottleneck Distance

Assume two geometric realizations $f_1, f_2 : K \to \mathbb{R}^2$ of the same abstract complex K. For a fixed ω, define $h_{i,\omega} = \langle f_i(\cdot), \omega \rangle$. Then:

$$\|h_{1,\omega} - h_{2,\omega}\|_\infty \le \max_{v \in V} \|f_1(v) - f_2(v)\|_\infty.$$

By the stability theorem for persistence diagrams [9], we have:

$$d_B(\mathrm{Dgm}(h_\omega^{K_1}), \mathrm{Dgm}(h_\omega^{K_2})) \le \max_{v \in V} \|f_1(v) - f_2(v)\|_\infty,$$

and integrating over ω yields the global bound on the PHT distance:

$$d_B(\mathrm{PHT}(K_1), \mathrm{PHT}(K_2)) \le 2\pi \max_{v \in V} \|f_1(v) - f_2(v)\|_\infty.$$

C.3 Point Tracking and Geometry of Diagrams

In a lower-star filtration, a k-dimensional homology class born at τ and dying at σ corresponds to a birth-death pair (b, d) in the diagram, where $b = h_\omega(v_b)$ and $d = h_\omega(v_d)$ for unique vertices v_b and v_d. Thus, each off-diagonal point can be traced to a pair of vertices.

As ω varies, each such point traces a curve:

$$x(\omega) = (\langle v_b, \omega \rangle, \langle v_d, \omega \rangle),$$

which parametrizes an ellipse. The projection to the diagonal is also a degenerate ellipse:

$$x'(\omega) = (\langle (v_b + v_d)/2, \omega \rangle, \langle (v_b + v_d)/2, \omega \rangle).$$

C.4 Trivial Geometric Monodromy and Star-Shaped Domains

Following [4], a simplicial complex K is *star-shaped* if there exists $c \in |K|$ such that for all $x \in |K|$, the segment $[c, x] \subseteq |K|$. For such K in general position, $\mathrm{PHT}(K)$ has trivial geometric monodromy: there exist continuous functions $\{\gamma_i : \mathbb{S}^1 \to \overline{\mathbb{R}}^2\}$ (vines) such that the off-diagonal points of $\mathrm{Dgm}(h_\omega^K)$ are exactly $\{\gamma_i(\omega) \mid \gamma_i(\omega) \notin \Delta\}$.

Theorem 11 ([4]) *Let K be a star-shaped complex with vertices in general position. Then $\mathrm{PHT}(K)$ has trivial geometric monodromy.*

Each vine either never intersects the diagonal or intersects it exactly once entering and once exiting. For connected K, one vine is of the form $(\tilde{\gamma}(\omega), \infty)$. A vertex v is extremal if it gives birth to a class for some ω, and the number of vines N is at most the number of such extremal vertices:

$$N \leq \left|\mathrm{ext}^K(V)\right|.$$

C.5 Matching Construction and Cost Function

Given PHTs for K_1 and K_2 with n_1 and n_2 vines respectively, we build a complete bipartite graph G with $2n_1 + 2n_2$ vertices—each vine and its diagonal projection. The edge weight between two vines γ_u and γ_v at direction ω is:

$$c(u,v) = \begin{cases} \|\gamma_u(\omega) - \gamma_v(\omega)\|_\infty & \text{if one or both are off-diagonal} \\ \|\gamma_u(\omega) - \Delta\|_\infty & \text{if } \gamma_v(\omega) \in \Delta \\ \|\gamma_v(\omega) - \Delta\|_\infty & \text{if } \gamma_u(\omega) \in \Delta \\ 0 & \text{if both are at } \Delta. \end{cases}$$

At each ω, this graph realizes the bipartite matching problem described in Sec. 2.1. With $n = \max(n_1, n_2)$, we have $O(n^2)$ edges, so the kinetic hourglass maintains $O(n^2)$ elements and runs in $O(n^6 \log n)$ time using heaps.