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

Elizabeth Munch^{1, 2}, Elena Xinyi Wang¹, and Carola Wenk³

¹Department of Computational Mathematics, Science, and Engineering, Michigan State University

²Department of Mathematics, Michigan State University

³Department of Computer Science, Tulane University

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 implementation 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) [6], designed to maintain various geometric configurations for continuously moving objects. The KDS frameworks have 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 [21] 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.

This is related to the problem of comparing persistence diagrams, a central concept in topological data analysis (TDA). Intuitively, a persistence diagram is a visual summary of the topological features of a given object. We can learn how different or similar two objects are by comparing their persistence diagrams. One of the most common comparisons is to compute the bottleneck distance, a dual of the min-cost matching problem in graphs. Improving the algorithm to compute the bottleneck distance has been studied extensively, both theoretically and practically [15, 22, 18, 23, 8].

Vineyards are continuous families of persistence diagrams that represent persistence for time-series of continuous functions [12, 27, 24]. A specific case of vineyards is the persistent homology transform (PHT) [13, 19, 25], which is the family of persistence diagrams of a shape in \mathbb{R}^d computed from every direction in \mathbb{S}^{d-1} . The PHT possesses desirable properties such as

continuity, injectivity, and stability. However, an efficient method for comparing two PHTs has not been thoroughly investigated. In particular, existing approaches to compute the bottleneck distance between two PHTs of shapes in \mathbb{R}^2 rely on sampling various directions, which only provides approximate results. To date, there is no known solution for computing the exact bottleneck distance. This work presents the first method to address this challenge, providing a precise and efficient solution.

Our contribution: We construct a new kinetic data structure that maintains a min-cost matching and the bottleneck distance of a weighted bipartite graph with continuously changing weight functions. We evaluate the data structure based on the four standard metrics in the KDS framework. We provide a specific case study where the weighted graph is computed from two persistent homology transforms, resulting in the first exact distance computation between two PHTs.

The paper is organized as follows: we cover the bottleneck matching problem's background and the kinetic data structures' preliminaries in Section 2. In Section 3, we iterate the events and updates necessary to maintain the KDS and the overall complexity analysis. Finally, we focus in Section 4 on our case study of the persistent homology transform for a special case of objects, called star-shaped objects, as defined in [4].

2 Background

Broadly, we are interested in a 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 the matching with the largest cardinality; i.e., for any other matching M', $|M| \ge |M'|$. A maximum matching is always maximal; the reverse is not true.

For a 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 2.1 (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 2.2. 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 2.3. 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 2.4 (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 $P \setminus M$. We denote this process as $Aug(M,P) = M \setminus (M \cap P) \cup (P \setminus M)$. Note that |Aug(M,P)| = |M| + 1, so we can repeat the above process until no more augmenting paths can be found. By Theorem 2.4, 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 [18].

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 $c: E \to \mathbb{R}_{>0}$ is at most λ :

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

Definition 2.5. 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 [16] 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:

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

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

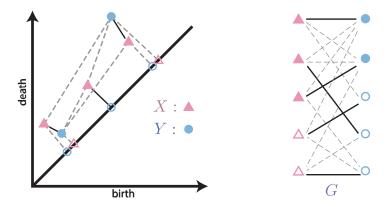


Figure 1: Construction of the bipartite graph G based on the persistence diagrams X and Y.

the complete bipartite graph $G = (U \sqcup V, U \times V, \mathbf{c})$, where for $u \in U$ and $v \in V$, the weight function c is given by

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

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 2.6 (Reduction Lemma [17]). For the above construction of G, $d_B(G) = d_B(X,Y)$.

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 $O(n^{1.5} \log n)$ [10]. 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) [20, 5, 21] 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. Finally, 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.

A KDS is evaluated by four measures. We say a quantity is *small* if it is a polylogarithmic function of n, or is $O(n^{\varepsilon})$ for arbitrarily small ε , and n is the number of objects. A KDS is considered *responsive* if the worst-case time required to fix the data structure and augment a failed certificate is small. *Locality* refers to the maximum number of certificates in which any one value is involved. A KDS is local if this number is small. A *compact* KDS is one

where the maximum number of certificates used to augment the data structure at any time is O(n polylog n) or $O(n^{1+\varepsilon})$. 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. We consider the ratio between the worst-case total number of events that can occur when the structure is advanced to $t = \infty$ and the worst-case number of external to the data structure. The second number is problem-dependent. A KDS is efficient if this ratio is small.

We next explore kinetic solutions to upper- and lower-envelope problems. The deterministic algorithm uses a kinetic heap, while the more efficient kinetic hanger adds randomness.

2.2.1 Kinetic Heap

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. If v is a child node of u, then u has higher priority than v. In the example of a max heap, that means u > v. In a kinetic max heap, the value of a node is stored as a function of time $f_X(t)$. The data structure augments a certificate [A > B] for every pair of parent-child nodes A and B. It is only valid when $f_A(t) > f_B(t)$. Thus, the failure times of the certificate are scheduled in the event queue at time t such that $f_A(t) = f_B(t)$. When a certificate [A > B] fails, we swap A and B in the heap and make 5 parent-child updates. This is the total number of certificates in which any pair of A and B are present. 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.

This KDS satisfies the responsiveness criteria because, in the worst case, each certificate failure only leads to O(1) updates. Locality, compactness, and efficiency depend on the behavior of $f_X(t)$. The analysis below assumes the case of affine motion where $f_X(t) = at + b$. In the worst case, each node is present in only three certificates, one with its parent and two with children, resulting in 3 events per node. For compactness, the total number of scheduled events is n-1 for n nodes in the kinetic heap. It is also efficient, where the maximum number of events processed is $O(n \log n)$ for a tree of height $O(\log n)$. The total time complexity in the linear case is $O(n \log^2 n)$ [6]. The results for locality and compactness hold for the more general setting of s-intersecting curves, where each pair of curves intersect at most s times, $s \in \mathbb{Z}$. The efficiency is unknown in this case, and the time complexity is $O(n^2 \log^2 n)$ [7].

When the functions are only defined within an interval of time, we call them segments. We perform an insertion when a new segment appears and a deletion when a segment disappears. In the scenario of affine motion segments, the complexity of the kinetic heap is $O(n\sqrt{m}\log^{3/2}m)$, n is the total number of elements and m is the maximum number of elements stored at a given time. In the case of s-intersecting curve segments, the complexity of the kinetic heap is $O(mn\log^2 m)$.

2.2.2 Kinetic Hanger

More efficient solutions to this problem have been developed, such as the kinetic *heater* and *tournament* [7, 5]. While both improve some aspects of the heap, they each have downsides. The kinetic heater increases the space complexity and is difficult to implement, and the tournament has a high locality bound. We instead use the *kinetic hanger*, introduced by da Fopnseca et

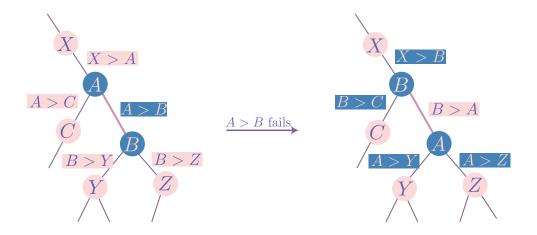


Figure 2: Kinetic heap event updates

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.

al. in [14]. It modifies the kinetic heap by incorporating randomization to balance the tree, meaning all complexity results are expected rather than deterministic.

Given an initial set of elements S, sorted by decreasing priorities, we construct a hanger hanger(S) by so-called hanging each element at the root. To hang an element e at node v, if no element is assigned to v, we assign e to v and return. Otherwise, we use a random seed r to choose a child c_r of v and recursively hang e at c_r . Insertion is similar to hanging - it only requires the additional step of comparing the priorities of e and e', where e' is the element already assigned to v. Deletion can be done by removing the desired element and going down the tree, replacing the current element with its child with the highest priority.

As shown in [14], the kinetic hanger has O(1) locality. The number of expected events in the affine motion case is $O(n^2 \log n)$ 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. When the functions are partially defined, the complexities are $O(n\alpha(m) \log^2 m)$ for line segments and $O(n/m\lambda_{s+2}(m) \log^2 m)$ for curve segments.

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 $\mathcal{G} = (G, c)$ where c changes continuously with respect to

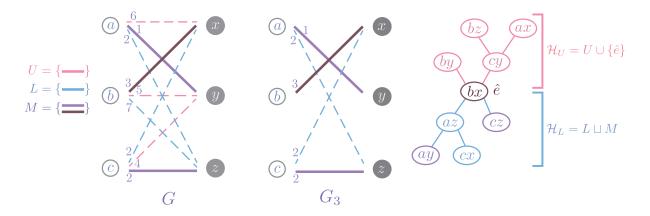


Figure 3: Illustration of construction of the kinetic hourglass.

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 $c^t : E \to \mathbb{R}_{\geq 0}$. Denote the weighted bipartite graph by $\mathcal{G}^t = (G, c^t)$. 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 $G_{\delta}^t \subseteq G$ be the portion of the complete bipartite graph with all edges with weight at most δ at time t; i.e., $V(G_{\delta}^t) = V$, and $E(G_{\delta}^t) = \{e \in E \mid c^t(e) \leq \delta\}$. If the bottleneck distance $\hat{\delta}^t = d_B(\mathcal{G}^t)$ is known, we are focused on the bipartite graph $G_{\hat{\delta}}^t$, 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 $c(\hat{e}^t) = \hat{\delta}^t$, which we call the *bottleneck edge*. This edge is unique as long as all edges have unique weights. We separate the remaining edges into the sets

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

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 $U^t \cup \{\hat{e}\}$. Note that $c^t(\hat{e}) = \max\{c^t(e) \mid e \in L^t \sqcup M^t\}$ and $c^t(\hat{e}) < \min\{c^t(e) \mid e \in U^t\}$, meaning that \hat{e} is the root for both heaps.

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 $a \leq_t b$ 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 3.1. 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, $\Gamma \setminus \{r\}$ 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. $e \leq_{t^-} r$ and $r \leq_{t^+} e$.
- 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 3.2 (*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 3.3 (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 $(\hat{c}^{t^+} = r^{t^+} = r)$, and the matching is updated with the augmenting path, specifically $M^{t^+} = \operatorname{Aug}(M^{t^-}, P)$.
- 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.

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. $c^{t^+}(r) = \max\{c^{t^+}(e) \mid e \in M''\}$.

Assume instead that there is no augmenting path in G' from u to v. Then M' is a maximum matching for G' by Theorem 2.4, however it is not perfect. Note that because $G' = \hat{G}^{t^-} \setminus e$ and e and r have swapped places, we have that $G' = G_{c^{t^+}(r)}$. Therefore, there is no perfect matching for $G_{c^{t^+}(r)}$. However, M^{t^-} is a perfect matching at time t^- and $\hat{G}^{t^-} = G_{c^{t^+}(e)}$, it still is a perfect matching at t^+ for $G_{c^{t^+}(e)}$. Moreover, $c^{t^+}(e) = \max\{c^{t^+}(e) \mid e \in M^{t^+}\}$. Therefore, $e = \hat{e}^{t^+} = r^{t^+}$ becomes the root, and r becomes a child of the root e in \mathcal{H}_L .

Lemma 3.4 (*U*-Event). Assume we swap priority of $r = r^{t^-}$ and an $e \in U^{t^-}$ at t; i.e. $r \leq_{t^-} e$ and $e \leq_{t^+} r$. 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 $M^{t^+} = \operatorname{Aug}(M^{t^-}, P)$.

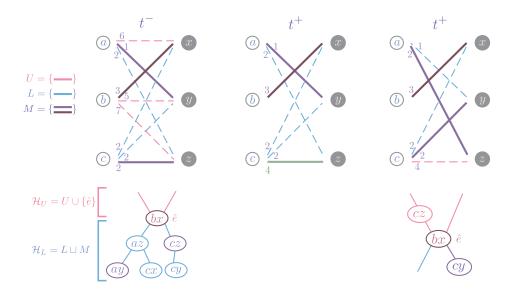


Figure 4: Illustration of Scenario 1 of and M-Event; see Lemma 3.3.

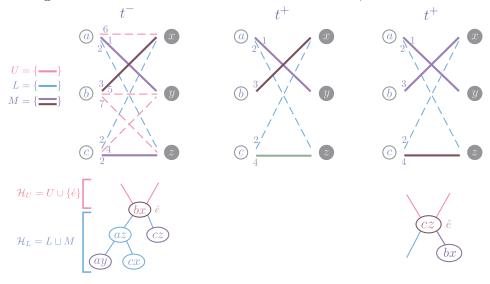


Figure 5: Illustration of Scenario 2 of M-Event; see Lemma 3.3.

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 $(\hat{e}^{t^+} = r^{t^+} = r)$.

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 $G' = G_{c^{t^+}(e)}$. Moreover, $c^{t^+}(e) = \max\{c^{t^+}(e) \mid e \in M''\}$. 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 2.4, however it is not perfect. Therefore, there is no perfect matching for $G_{c^{t+}(e)}$. However, M^{t^-} is a perfect matching at time t^- and $\hat{G}^{t^-} = G_{c^{t+}(r)} \setminus \{e\}$, it still is a perfect matching at t^+ for $G_{c^{t+}(r)}$. 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. $c^{t^+}(r) = \max\{c^{t^+}(e) \mid e \in M''\}$.

3.3 Complexity and Performance Evaluation

The kinetic hourglass's complexity analysis largely follows that of the kinetic heap and kinetic hanger [14]. Recall the time complexities of the structures summarized in Table 1 are obtained by multiplying the number of events by the time to process each event, $O(\log n)$. This is the key difference between those data structures and the kinetic hourglass.

In the kinetic hourglass structure, we maintain two heaps or hangers at the same time. For a bipartite graph G of size m, we maintain the m edges in the kinetic hourglass. It is also the worst-case maximum number of elements in the max-heap \mathcal{H}_L and the min-heap \mathcal{H}_U . We therefore use m=n in our analysis. Within a single heap, we can think of this procedure as focusing on the cost of an edge at time t as $c^t(e)$ only given for its time inside the heap. This means that when we evaluate the runtime for the heap (or hanger) structures, we can think of the function on each edge as being a curve segment, rather than defined for all time.

The time complexities for the kinetic hourglass are summarized in Table 2 and we describe them further here. The main change in time complexity results from the augmenting path search required at every external event. Note that in both the L- and M-events from Lemma 3.3 and 3.4, exactly only one iteration of augmenting path search is required, so this takes O(|E|) = O(m) per iteration. Therefore, we replace a $\log n$ factor with m. When c(e) is linear, the complexity of the kinetic hourglass is $O(m^2\sqrt{m}\log^{3/2}m)$ using heaps, and is $O(m^2\alpha(m)\log m)$ using hangers, since the arrangement of m lines has complexity $O(m^2)$. If the weight function is non-linear, we are in the s-intersecting curve segments scenario. The resulting runtimes are $O(m^3\log m)$ and $O(m\lambda_{s+2}(m)\log m)$, using heaps and hangers respectively. The constant s is determined by the behavior of the weight function.

While this data structure is responsive, local, and compact, following directly from the analysis of kinetic heap and hanger, it fails to be efficient due to the linear time complexity required by each external event of the heap. We conjecture that this can be improved via amortization analysis, which involves investigating the ratio between the number of internal and external events. However, this is a non-trivial problem and has yet to be addressed in existing literature [14].

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

In the following section, we apply our kinetic hourglass to compute the bottleneck distance between two persistent homology transforms [25, 13, 19], a specific case of persistence vineyard [12]. The structure of the PHT can be quite complex, even for simple embedded graph inputs [4]. Thus, we discuss the general PHT, but then following [4], focus here on a simple case of input structures where we have additional knowledge of the PHT vineyard. In this section, we assume a basic knowledge of persistent homology; see [16] for additional details.

4.1 The Persistent Homology Transform

Given a finite geometric simplicial complex K in \mathbb{R}^2 with $|K| \subset \mathbb{R}^2$ denoting the geometric realization in the plane, every unit vector $\omega \in \mathbb{S}^1$ corresponds to a height function in direction ω defined as

$$h_{\omega}: |K| \to \mathbb{R}$$
$$x \mapsto \langle x, v \rangle$$

where $\langle \cdot, \cdot \rangle$ denotes the inner product. This induces a filtration $\{h_{\omega}^{-1}(-\infty, a] \mid a \in \mathbb{R}\}$ of |K|, however we instead use the the combinatorial representation given on the abstract complex K. The lower-star filtration [17] is defined on the abstract simplicial complex by

$$h_{\omega}: K \to \mathbb{R}$$

 $\sigma \mapsto \max\{h_v(u) \mid u \in \sigma\}.$

We abuse notation and write both functions as h_{ω} since the sublevelset $h_{\omega}^{-1}(-\infty, a]$ has the same homotopy type in both cases, thus does not affect the persistent homology calculations. Note also that the combinatorial viewpoint means we can also define $h_{\omega}^{-1}(-\infty, a]$ as the full subcomplex of K given by the vertex set $V_a = \{v \in V \mid h_{\omega}(v) \leq a\}$.

We say K is generic if there are no two parallel and distinct lines each passing through pairs of vertices in K. Then for a generic K, all vertices have distinct function values for h_{ω} except for ω s that are perpendicular to any line connecting a pair of vertices, whether or not this is an edge. Fix ω away from this set. Then the function induces a sorted order of the vertices by function value, and we denote the full subcomplex defined by the first i vertices by $K(\omega)_i$. Finally, the nested sequence of complexes

$$\emptyset = K(v)_0 \subseteq K(v)_1 \subseteq \cdots \subseteq K(v)_n = K$$

is the lower star filtration. The resulting k-dimensional persistence diagram computed by filtering K by the sub-level sets of h_{ω} is denoted $\mathrm{Dgm}_k(h_{\omega}^K)$.

In the context of this work, we are focusing on 0-dimensional diagrams, denoted as $\mathrm{Dgm}(h_{\omega}^K)$ for simplicity. Dimension 0 homology is entirely determined by the 0- and 1-dimensional simplices, so we can assume our input K is an embedded graph. We assume that the input

embedded graph is connected, so each $\mathrm{Dgm}(h_{\omega}^K)$ has exactly one feature that dies at time $t=\infty$.

Definition 4.1. The persistent homology transform of $K \subset \mathbb{R}^2$ is the function

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

where $h_{\omega}^K: K \to \mathbb{R}$, $h_{\omega}^K(x) = \langle x, \omega \rangle$ is the height function on K in direction ω , and \mathcal{D} is the space of persistence diagrams.

The PHT has been shown to be injective: for K_1 , $K_2 \subset \mathbb{R}^d$, if $PHT(K_1) = PHT(K_2)$, then $K_1 = K_2$ [25, 13, 19]. This makes it a useful representation of shape in data analysis contexts. However, in those contexts, we wish to be able to compare two of these representations to encode their similarity. To this end, we are interested in comparing the persistent homology transforms of two input complexes, K_1 and K_2 , via an extension of the bottleneck distance using the framework of the kinetic hourglass.

Definition 4.2. The *integrated bottleneck distance* between $PHT(K_1)$ and $PHT(K_2)$ is defined as

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

For a fixed direction ω , the bottleneck distance between the two persistence diagrams can be formulated as a geometric matching problem as described in Section 2.1. Moreover, it is stable in the following sense. Assume that we have two geometric simplicial complexes K_1 and K_2 with the same underlying abstract complex. Then we can think of the geometric embeddings as functions $f_1, f_2 : K \to \mathbb{R}^2$ and so for a fixed $\omega \in \mathbb{S}^1$, we write $h_{i,\omega} : K \to \mathbb{R}$ for the respective height functions. We can use the fact that we have a lower star filtration along with the Cauchy-Schwartz inequality to see that $||h_{1,\omega} - h_{2,\omega}||_{\infty} \le \max_{v \in V} ||f_1(v) - f_2(v)||_{\infty}$. Then by the stability theorem [11], the distance between the diagrams is bounded by

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

for all $\omega \in \mathbb{S}^1$. Integrating this inequality immediately gives the following result for the integrated bottleneck distance.

Proposition 4.3. For two finite, connected geometric simplicial complexes with the same underlying abstract complex,

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

A choice of total order on the simplices of K induces a pairing on the simplices of K, where for a pair (τ, σ) , $\dim(\sigma) = \dim(\tau) + 1$, and a $\dim(\tau)$ -dimensional homology class was born with the inclusion of τ and dies with the inclusion of σ . Because the lower star filtration, the function values of these simplices are given by the unique maximum vertex $v_{\sigma} = \{v \in \sigma \mid f_{\omega}(v) \geq f_{\omega}(u) \ \forall u \in \sigma)\}$ and so the function value when this simplex was added is given by $f_{\omega}(v_{\sigma})$, and v_{σ} does not change for nearby σ that do not pass through one of the non-generic directions. For this reason, we can associate each finite point (b, d) in the persistence diagram

 $\operatorname{Dgm}_k(h_\omega^K)$ to a vertex pair (v_b, v_d) with $f_\omega(v_b) = b$ and $f_\omega(v_d) = d$. This pair is well-defined in the sense that given a point in the diagram, there is exactly one such pair. Further, because of the pairing on the full set of simplicies, every birth vertex appears in exactly one such pair, although here a death vertex could be associated to multiple points in the diagram. For any infinite points in the diagram of the form (b, ∞) , we likewise have a unique vertex v_b again with $f_\omega(v_b) = b$.

Write a given filtration direction as $\omega = (\cos(\theta), \sin(\theta)) \in \mathbb{S}^1$ and fix a point $x \in \mathrm{Dgm}(h_\omega^K)$ with birth vertex $v_a \in K$ located at coordinates $(a_1, a_2) \in \mathbb{R}^2$ and death vertex $v_b \in K$ at coordinates $(b_1, b_2) \in \mathbb{R}^2$. This means the corresponding point in $x \in \mathrm{Dgm}(h_\omega^K)$ has coordinates which we denote as

$$x(\omega) = (a_1 \cos \theta + a_2 \sin v, b_1 \cos v + b_2 \sin \theta).$$

Notice that for some interval of directions $I_x \subset \mathbb{S}^1$, the vertices v_a and v_b will remain paired, and so for that region, we will have the point $x(\omega')$ in the diagram for all directions $\omega' \in I_x$. Viewing this as a function of angle θ , we observe that $x(\omega)$ is a parametrization of an ellipse, meaning that the point $x(\omega)$ in the diagram will trace out a portion of an ellipse in the persistence diagram plane. Further, the projection of this point to the diagonal has the form

$$x'(\omega) = \left(\frac{a_1 + b_1}{2}\cos\theta + \frac{a_2 + b_2}{2}\sin\theta, \frac{a_1 + b_1}{2}\cos\theta + \frac{a_2 + b_2}{2}\sin\theta\right),\,$$

which is also a parameterization of an ellipse, albeit a degenerate one.

4.2 Kinetic Hourglass for PHT

In this section, we show how the kinetic hourglass data structure investigated in Sec. 3 can be applied to compute the exact distance between the 0-dimensional PHTs of two geometric simplicial complexes in \mathbb{R}^2 , and provide an exact runtime analysis of the kinetic hourglass by explicitly determining the number of curve crossings in the flight plans for the edges in the bipartite graph. Here, we work in the restricted case of *star-shaped* simplicial complexes to avoid the potential for complicated monodromy in the PHT.

Following [4], an embedded simplicial complex K is called star-shaped if there is a point $c \in |K|$ such that for every $x \in |K|$, the line segment between c and x is contained in |K|. In this context, we call c a (non-unique) center of M. Let $\overline{\mathbb{R}} = \mathbb{R} \cup \{\infty\}$. We say that a function $F: \mathbb{S}^1 \to \mathcal{D}$ has trivial geometric monodromy if there exist maps $\{\gamma_i : \mathbb{S}^1 \to \overline{\mathbb{R}}^2\}$ (called vines) such that the off-diagonal points of $PHT(\omega)$ are exactly the set $\Gamma(\omega) = \{\gamma_i(\omega) \mid \gamma_i(\omega) \notin \Delta\}$.

Theorem 4.4 ([4]). Let K be a star-shaped simplicial complex whose vertices are in general position. Then the 0-dimensional persistent homology transform PHT(K) has trivial geometric monodromy.

Thus, assume we have the vines $\{\gamma_i\}_{i=1}^N$ for the PHT of a star-shaped K. We assume that these vines are written so that they either never touch the diagonal $\Delta = \{(x,x) \in \mathbb{R}\}$, or they each correspond to exactly one entrance and exit from Δ and so are off-diagonal for some interval I_{γ_i} . Further, assume that K is connected so that exactly one of these vines, say γ_1 , is the infinite vine of the form $(\tilde{\gamma}(\omega), \infty)$ for $\tilde{\gamma} : \mathbb{S}^1 \to \mathbb{R}$. For a geometric simplicial complex K, a vertex $x \in K$ is called *extremal* if it gives birth to a 0-dimensional class for some direction ω . The set of extremal vertices is denoted as $|\operatorname{ext}^K(V)|$. We show below that the number of vines N is at most $|\operatorname{ext}^K(V)|$.

To bound the number of vines N, we call a vertex v in a geometric simplicial complex K an extremal vertex if it is not in the convex hull of its neighbors following [9]. Write $\operatorname{ext}(V)$ for the set of vertices of K which are extremal. It is an immediate corollary of the merge tree version Lemma 2.5 of [9] that a vertex in K gives birth to a 0-dimensional class for some direction ω if and only if it is an extremal vertex. We define the external edges of an extremal vertex v to be the one or two edges that are incident to v and are part of the boundary of the convex hull formed by v and its neighbors. The remaining edges incident to v are called internal edges. For an extremal vertex, we can find the interval of directions I_v for which it gives birth for all $\omega \in I_v$. We note that this is a connected interval in the circle, so we have the starting and ending direction ω_1 and ω_2 for which the point exits and enters the diagonal with $I_v = [\omega_1, \omega_2] \subset \mathbb{S}^1$. ω_1 and ω_2 are the normal vectors of the edges that form the largest angle incident to v. We can write a piecewise-continuous path associated to the vertex v as $\mu_v : I_v \to \mathbb{R}^2$; $\omega \mapsto x(\omega)$ for the point $x(\omega)$ in the diagram with birth vertex v, and note that each piece of this path is a portion of an ellipse.

Lemma 4.5. For an extremal vertex v, if $\mu_v(\omega) \in \Delta$ then $\omega = \omega_1$ or $\omega = \omega_2$.

Proof. Let ω_1 , ω_2 be the normal vectors of the external edges (v, u_1) , (v, u_2) respectively, and $\mu_v(\omega) = (h_\omega(v), h_\omega(v'))$ where v is the birth-causing vertex and v' is the death causing vertex. Since v is birth-causing, all other neighbors of v have higher height values along direction ω : $\{h_\omega(u) > h_\omega(v) | u \in N_K(v), u \neq v'\}$. Because $\mu_v(\omega) \in \Delta$, then $h_\omega(v) = h_\omega(v')$. Namely $\langle v, \omega \rangle = \langle v', \omega \rangle$. This implies that ω is a normal vector to the edge defined by v and v'. Furthermore, no neighbor of v lies in the half plane defined by (v, v') and $-\omega$. Therefore, (v, v') must be part of the convex hull formed by v and $N_K(v)$. We thus conclude that v' is either u_1 or u_2 , and $\omega = \omega_1$ or $\omega = \omega_2$.

Because every point in the diagram $PHT(\omega)$ at any fixed direction is uniquely associated to an extremal vertex, we can see that the off diagonal points of the PHT can also be written as $M(\omega) = \{\gamma_i(\omega) \mid \gamma_i(\omega) \notin \Delta\}$, and so $M(\omega) = \Gamma(\omega)$. Thus there is an injective map from the set of finite vines γ_i to the vertex giving birth to the corresponding points at the time it comes out of the diagonal. This means that the number of vines is at most the number of extremal vertices, i.e. $N \leq |\operatorname{ext}(V)|$.

Given two star-shaped complexes K_1 and K_2 , say the two PHTs are given by the vines $\{\gamma_{1,i}\}_{i=1}^{n_1}$ and $\{\gamma_{2,j}\}_{j=1}^{n_2}$ respectively and write the projection of the respective vine to the diagonal as $\gamma_{k,i}^{\Delta}$. We then construct the complete bipartite graph G with vertex set $U \cup V$ where U and V each have $n_1 + n_2$ vertices, each associated to one of the vines from either vineyard. However, in U we think of these as being the vines $\{\gamma_{1,i}\}_i$ followed by the projections of each vine $\{\gamma_{2,j}^{\Delta}\}_j$ to the diagonal; the reverse is assumed for V. The basic idea is that the cost of an edge between two vertices is based on the distance between the location of the paths (or projection of the paths for the diagonal representatives) so that at any fixed ω , this graph is the bipartite graph described in Section 2.1. Specifically, writing γ_u for the vine associated to vertex, u, the edge weights are given by

$$c(u,v) = \begin{cases} \|\gamma_u(\omega) - \gamma_v(\omega)\|_{\infty} & \text{if } \gamma_u(\omega) \notin \Delta \text{ or } \gamma_v(\omega) \notin \Delta \\ \|\gamma_u(\omega) - \Delta\|_{\infty} & \text{if } \gamma_v(\omega) \in \Delta \\ \|\gamma_v(\omega) - \Delta\|_{\infty} & \text{if } \gamma_u(\omega) \in \Delta \\ 0 & \text{else.} \end{cases}$$

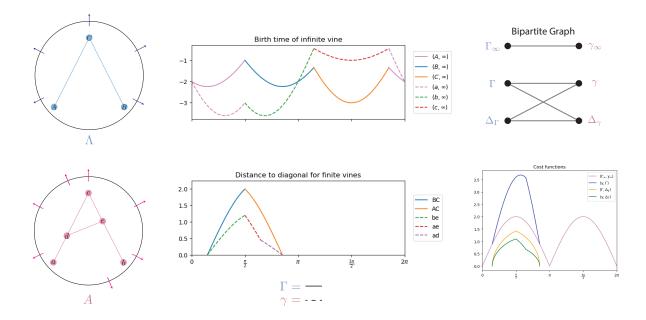


Figure 6: An example of the weight function c. The two figures in the middle visualize the behavior of the vines. On the top right is the bipartite graph representation, and on the bottom are the cost functions.

See Figure 6 for an example of the weight functions.

Given the above construction, let $n = \max\{|\operatorname{ext}^{K_1}(V)|, |\operatorname{ext}^{K_2}(V)|\}$, we have at most 2n nodes in each vertex set U and V and thus at most $4n^2$ edges in G. Therefore, the kinetic heap hourglass maintains $4n^2$ elements, resulting in a runtime of $O((4n^2)^3 \log(4n^2)) = O(n^6 \log n)$.

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.

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 [26], 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.

References

- [1] Pankaj K Agarwal, Mark De Berg, Jie Gao, Leonidas J Guibas, and Sariel 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] Pankaj K. Agarwal, Jie Gao, and Leonidas J. Guibas. Kinetic medians and kd-trees. In Rolf Möhring and Rajeev Raman, editors, *Algorithms ESA 2002*, pages 5–17, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.
- [3] Pankaj K. Agarwal and Micha 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] Shreya Arya, Barbara Giunti, Abigail Hickok, Lida Kanari, Sarah McGuire, and Katharine Turner. Decomposing the persistent homology transform of star-shaped objects, 2024.
- [5] Julien Basch, Leonidas J Guibas, and John Hershberger. Data structures for mobile data. Journal of Algorithms, 31(1):1–28, 1999.
- [6] Julien Basch, Leonidas J. Guibas, and G. D. Ramkumar. Sweeping lines and line segments with a heap. In *Proceedings of the Thirteenth Annual Symposium on Computational Ge*ometry, SCG '97, page 469–471, New York, NY, USA, 1997. Association for Computing Machinery.
- [7] Julien Basch, Leonidas J. Guibas, and G. D. Ramkumar. Reporting red—blue intersections between two sets of connected line segments. *Algorithmica*, 35(1):1–20, Jan 2003.
- [8] Sergio Cabello, Siu-Wing Cheng, Otfried Cheong, and Christian Knauer. Geometric Matching and Bottleneck Problems. In Wolfgang Mulzer and Jeff 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.
- [9] Erin Wolf Chambers, Elizabeth Munch, Sarah Percival, and Elena Xinyi Wang. A distance for geometric graphs via the labeled merge tree interleaving distance. arXiv preprint arXiv:2407.09442, 2024.
- [10] L. Paul Chew and Klara Kedem. Improvements on geometric pattern matching problems. In Otto Nurmi and Esko Ukkonen, editors, Algorithm Theory — SWAT '92, pages 318–325, Berlin, Heidelberg, 1992. Springer Berlin Heidelberg.
- [11] David Cohen-Steiner, Herbert Edelsbrunner, and John Harer. Stability of persistence diagrams. Discrete & Computational Geometry, 37(1):103–120, Jan 2007.
- [12] David Cohen-Steiner, Herbert Edelsbrunner, and Dmitriy Morozov. Vines and vineyards by updating persistence in linear time. In *Proceedings of the Twenty-Second Annual Sym*posium on Computational Geometry, SCG '06, page 119–126, New York, NY, USA, 2006. Association for Computing Machinery.

- [13] Justin Curry, Sayan Mukherjee, and Katharine Turner. How many directions determine a shape and other sufficiency results for two topological transforms. *Transactions of the American Mathematical Society, Series B*, 2018.
- [14] Guilherme D. da Fonseca, Celina M.H. de Figueiredo, and Paulo C.P. Carvalho. Kinetic hanger. *Information Processing Letters*, 89(3):151–157, 2004.
- [15] Tamal K. Dey and Cheng Xin. Computing Bottleneck Distance for 2-D Interval Decomposable Modules. In Bettina Speckmann and Csaba 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.
- [16] Tamal Krishna Dey and Yusu Wang. Computational Topology for Data Analysis. Cambridge University Press, 1 edition, 2017.
- [17] Herbert Edelsbrunner and John Harer. Computational Topology an Introduction. American Mathematical Society, 2010.
- [18] A. Efrat, A. Itai, and M. J. Katz. Geometry helps in bottleneck matching and related problems. *Algorithmica*, 31(1):1–28, Sep 2001.
- [19] Robert Ghrist, Rachel Levanger, and Huy Mai. Persistent homology and Euler integral transforms. *Journal of Applied and Computational Topology*, 2(1):55–60, Oct 2018.
- [20] Leonidas 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.
- [21] Leonidas J Guibas and Marcel Roeloffzen. Modeling motion. In Csba D. Toth, Joseph O'Rourke, and Jacob E. Goodman, editors, *Handbook of Discrete and Computational Geometry (3rd ed.)*, chapter 57, pages 1117 1134. Chapman and Hall/CRC, 2017.
- [22] John E. Hopcroft and Richard M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. SIAM J. Comput., 2(4):225–231, Dec 1973.
- [23] Michael Kerber, Dmitriy Morozov, and Arnur Nigmetov. Geometry helps to compare persistence diagrams. ACM J. Exp. Algorithmics, 22, sep 2017.
- [24] Woojin Kim and Facundo Mémoli. Spatiotemporal persistent homology for dynamic metric spaces. Discrete & Computational Geometry, 66(3):831–875, Oct 2021.
- [25] Katharine Turner, Sayan Mukherjee, and Doug M. Boyer. Persistent homology transform for modeling shapes and surfaces. *Information and Inference: A Journal of the IMA*, 3(4):310–344, 12 2014.
- [26] Katharine Turner, Vanessa Robins, and James Morgan. The extended persistent homology transform of manifolds with boundary. Journal of Applied and Computational Topology, May 2024.
- [27] Lu Xian, Henry Adams, Chad M. Topaz, and Lori Ziegelmeier. Capturing dynamics of time-varying data via topology, 2022.