Efficient Graph Reconstruction and Representation Using Augmented Persistence Diagrams

Brittany Terese Fasy* Samuel Micka† David L. Millman‡ Anna Schenfisch§ Lucy Williams¶

Abstract

Persistent homology is a tool that can be employed to summarize the *shape* of data by quantifying homological features. When the data is an object in $\mathbb{R}^d$, the (augmented) persistent homology transform ((A)PHT) is a family of persistence diagrams, parameterized by directions in the ambient space. A recent advance in understanding the PHT used the framework of reconstruction in order to find finite a set of directions to faithfully represent the shape, a result that is of both theoretical and practical interest. In this paper, we improve upon this result and present an improved algorithm for graph—and, more generally one-skeleton—reconstruction. The improvement comes in reconstructing the edges, where we use a radial binary (multi-)search. The binary search employed takes advantage of the fact that the edges can be ordered radially with respect to a reference plane, a feature unique to graphs.

1 Introduction

At the heart of inverse problems in the field of topological data analysis is the following question: *how many persistence diagrams are needed to faithfully represent a shape?* Since the introduction of persistence diagrams, it has been known that many "shapes" can share the same persistence diagram. With enough persistence diagrams, we arrive at a set of parameterized diagrams (that is, diagrams labeled by direction) that uniquely, or *faithfully* represents the underlying shape. Moreover, the set of parameterized diagrams is *faithful* if and only if it can be used to *reconstruct* the underlying shape.

The foundation for asking the question of how many diagrams are needed for a faithful representation was first introduced in [16], where Turner et al. defined the Persistent Homology Transform (PHT) and the Euler characteristic curve transform (ECCT). These transforms map a simplicial complex embedded in $\mathbb{R}^d$ (a shape) to sets of persistence diagrams (respectively, Euler characteristic curves) parmeterized by $\mathbb{S}^{d-1}$, the set of all directions in $\mathbb{R}^d$. They showed that, up to mild general position assumptions, no two simplicial complexes can correspond to the same PHT or ECCT, i.e., they showed that the uncountably infinite sets making up the PHT and ECCT faithfully represent the shape.

However, the PHT and ECCT are uncountably infinite sets of diagrams (and Euler characteristic curves). Thus, to bridge the gap between the theory and what can be used in practice, a discretization of the PHT was needed; several papers stepped up to the challenge and proved that there exists a finite discretizations of topological transforms that are faithful for simplicial and cubical complexes [1,2,4,5,7,8,12]. Beyond proving the existence of these finite faithful sets, Belton et al. [2] explicitly give an algorithm for using an oracle to reconstruct graphs embedded in $\mathbb{R}^d$ with n vertices. Their reconstruction uses $n^2-n+d+1$ augmented persistence diagrams in $O(dn^{d+1} + n^4 + (d+n^2)\Pi)$ time [2, Theorem 17], where $\Theta(\Pi)$ is the time complexity it takes for an oracle to produce answer a persistence diagram querry. Since the direction-labeled diagram set can be used for reconstructing the underlying graph, it is a faithful discretization of the augmented PHT (APHT).

In the current work, we give a faithful discretization of the APHT using $\Theta(d+m\log n)$ diagrams and $\Theta(dn^{d+1}+d\Pi + n^2 + m\log n(\log n + d + \Pi))$ time. This result is an improvement over [2], both in the size of the set and in the speed of reconstruction. The crux is in the improvmeent to edge reconstruction. While the method of [2] uses a linear scan of all possible edges for each vertex, resulting in a quadratic number of diagrams needed, here we show that we can detect edges with $\Theta(m\log n)$ diagrams using a radial binary multi-search.

2 Background and Tools

In this section, we provide definitions of the tools used in the remainder of the paper. We make use of standard notation such as using e_i for the ith standard basis vector in $\mathbb{R}^d$, where $1 \le i \le d$. We use the notation (V, E) for a graph and its vertex and edge sets, and use $n = |V|$ and $m = |E|$. We assume the reader is familar with

*School of Computing and Department of Mathematical Sciences, Montana State University, brittany.fasy@montana.edu

†Mathematics & Computer Science Department, Western Colorado University, smicka@western.edu

‡School of Computing, Montana State University, david.millman@montana.edu

§Department of Mathematical Sciences, Montana State University, annaschenfisch@montana.edu

¶School of Computing, Montana State University, luciawilliams@montana.edu

standard concepts in topology (such as simplicial complexes, simplicial homology, Betti numbers), and note that further information can be found in [6] for a general introduction and in [7, Section 2.1] for a detailed definition of the augmented persistence diagram.

First, we define our general postition assumption.

Assumption 1 *Let $V \subset \mathbb{R}^d$ be a finite set with $d \geq 2$. We say V is in* general position *if the following properties are satisfied:*

(i) Every set of $d+1$ points is affinely independent.

(ii) No three points are colinear after orthogonal projection into the space $\pi(\mathbb{R}^d)$, where $\pi : \mathbb{R}^d \to \mathbb{R}^2$ is the orthogonal projection onto the plane spanned by the first two basis elements, e_1 and e_2.

(iii) Every point has a unique height with respect to the direction e_2.

We call a graph GP-immersed[1] *iff its vertex set is in general position in $\mathbb{R}^d$.*

We note that (iii) is not strictly necessary, however, it is convenient for simplicity of exposition. How to handle this degeneracy is discussed in Appendix B.

Given a graph GP-immersed in $\mathbb{R}^d$, we can filter the graph based on the height in any direction s in the sphere of directions $\mathbb{S}^{d-1}$. To do so, we assign each simplex a height. A vertex $v \in V$ is assigned the height $s \cdot v$, and an edge $[v_0, v_1] \in E$ is assigned the height $\max\{s \cdot v_0, s \cdot v_1\}$. This function, mapping vertices and edges to heights, is known as a *filter* function, which we use to compute persistent homology.

2.1 Persistence and the Oracle Framework

Given a filtered simplicial complex (that is, a simplicial complex with each simplex assigned a "height"), the corresponding *augmented persistence diagram (APD)* is a record of of all homological events throughout the filtration. A *birth event* is the introduction or appearance of a new homological feature, and a *death event* is the merging of two features. A death is paired with the most recent of the birth-labeled features that it merges together, creating a birth-death pairing, leaving the remaining feature labeled by the elder birth height. In an APD, every simplex corresponds to exactly one event (resulting in some pairings where the birth and death heights are equal). As a result, by construction, APDs contain at least one event at the height of each vertex.

For a simplicial complex (e.g., a graph) GP-immersed in $\mathbb{R}^d$ and a direction in S^{d-1}, we use the *lower-star filtration*: the nested sequence of graphs that arise by looking at all simplices at or below a given height and allowing that height to grow from $-\infty$ to ∞. Throughout this paper, we denote the i-dimensional APD by $\widehat{\mathcal{D}}_i(s)$ and write $\widehat{\mathcal{D}}(s) = \sqcup_i \widehat{\mathcal{D}}_i(s)$, omitting the graph itself from the notation (as it is always clear from context).[2]

The *(augmented) persistence homology transform ((A)PHT)* is the set of (augmented) diagrams of lower-star filtrations in all possible directions, parameterized by the direction. That is, the set $X = \{(\widehat{\mathcal{D}}(s), s)\}_{s \in \mathbb{S}^{d-1}}$. A *faithful discretization* is a finite subset of X from which all other elements of X can be deduced (and, by [16], corresponds to a unique simplicial complex). The introduction of the (augmented) persistent homology transform has sparked related research in applications of shape comparison [3,9–11,14,15,17,18]. As such, finding a minimal faithful discretization is important for the applicability of the (A)PHT. In what follows, we will only consider APDs, and we may shorten notation and refer to an APD by the word *diagram*.

In this work, we assume an oracle framework. That is, we assume that we have no knowledge of the shape itself, but we have access to an oracle from which we can query directional diagrams.

Definition 2 (Oracle) *For a graph (V, E) GP-immersed in $\mathbb{R}^d$ and a direction $s \in \mathbb{S}^{d-1}$, the operation* `Oracle`(s) *returns the diagram $\widehat{\mathcal{D}}(s)$. We define $\Theta(\Pi)$ to be the time complexity of this oracle query and note that the space complexity of $\widehat{\mathcal{D}}(s)$ is $\Theta(n+m)$. We assume that the data structure returned by the oracle allows queries for specific birth or death values in $\Theta(\log n)$ time (for example, the we could have two arrays of persistences points, one sorted by birth values and one sorted by death values).*

2.2 Constructions and Data Structures

In this subsection, we introduce the edge arc object and other definitions useful for computing properties of immersed graphs. Throughout this paper, we project points in $\mathbb{R}^d$ to the (e_1, e_2)-plane. As a result, we use "above (below)" without stating with respect to which direction as shorthand for "above (below) with respect to the direction e_2." This direction is intentionally chosen (and used in our GP assumption), as it corresponds to our intuition of above (below) in the figures. When we measure an angle of a vector x, denoted $\measuredangle x$, we mean the angle that $\pi(x)$ makes with the positive e_1 axis.

Given a direction s and a vertex in a graph immersed in $\mathbb{R}^d$, we classify each edge (v, v') as either an "incoming" edge, when v' is below v with respect to s, or an "outgoing" edge, when v' is above v with respect to s. Note that all incoming edges have the same height as the vertex with respect to the e_2 direction.

[1] Here, we use *immersed* rather than *embedded* in order to allow intersections of edges. Note, however, that this can only happen when $d = 2$.

[2] When calculating diagrams, we count $\widehat{\mathcal{D}}(s)$ as one diagram, not multiple.

Definition 3 (Indegree) *Let (V, E) be a graph GP-immersed in $\mathbb{R}^d$. Let $v \in V$ and $s \in \mathbb{S}^{d-1}$. The* indegree of v in direction s, *denoted* $\texttt{Indeg}(v, s)$, *is the number of edges incident to v with height $s \cdot v$.*

The following lemma relates the number of edges at a given height to points in the APD.

Lemma 4 (Edge Count) *Let (V, E) be a graph and let $c \in \mathbb{R}$. Let $f\colon V \sqcup E \to \mathbb{R}$ be a filter function. Then, the edges in E with a function value of c are in one-to-one correspondence with the following multiset of points in $\widehat{\mathcal{D}}(f)$, the diagram corresponding to f:*

$$\begin{aligned}\big\{(b,d) \in \widehat{\mathcal{D}}_1(f) \text{ s.t. } b = c\big\} \\ \cup \big\{(b,d) \in \widehat{\mathcal{D}}_0(f) \text{ s.t. } d = c\big\}.\end{aligned} \tag{1}$$

In other words, each edge corresponds to either a birth of a one-dimensional homological feature or a death of a zero-dimensional feature in $\widehat{\mathcal{D}}(f)$. For more details and a generalized proof, see [7, Appendix A].

If f is a lower-star filtration in direction $s \in \mathbb{S}^{d-1}$, we note that whenever a vertex v has a unique height with respect to a direction s, the cardinality of the multiset above is exactly $\texttt{Indeg}(v, s)$.

Lemma 5 (Indegree Computation) *Let (V, E) be a graph GP-immersed in $\mathbb{R}^d$. Let $v \in V$ and let $s \in \mathbb{S}^{d-1}$ such that $s \cdot v \neq s \cdot v'$ for any $v' \neq v \in V$. Then, $\texttt{Indeg}(v, s)$ can be computed via the oracle using one diagram and $\Theta(\log n + \Pi)$ time.*

Proof. Let $\widehat{\mathcal{D}} = \texttt{Oracle}(s)$. By the assumption on s, the height of v with respect to the direction s is unique. Hence, we know that any edge at height $c = s \cdot v$ must be incident to v. Thus, by the definition of indegree, an edge has the height c if and only if it contributes to the indegree of v in direction s. Using Lemma 4, we count these edges by counting one-dimensional births and zero-dimensional deaths at height c. Since $\widehat{\mathcal{D}}_0$ and $\widehat{\mathcal{D}}_1$ are sorted by both birth and death values (see Definition 2) and since $\widehat{\mathcal{D}}$ has $\Theta(n+m)$ points, searching for these events takes $\Theta(\log n + \log m)$. Adding $\Theta(\Pi)$ for the oracle query and recalling that $m = O(n)$, the total runtime is $\Theta(\log n + \Pi)$. □

We conclude this section by introducing a data structure, the *edge arc object*; see Table 1 for a summary of the attributes of an edge arc and Figure 1 for an example. An edge arc represents the region in the (e_1, e_2)-plane centered at v that is swept out between the two angles α_1 and α_2 (the word 'arc' is referring to the arc of angles between α_1 and α_2, where the angle is measured with respect to the postive e_1 axis). We only consider edge arcs in the upper half-space, with respect to the e_2 direction, so the maximal edge arc is the upper half-plane and the start and stop angles always satisfy $0 \leq \alpha_1 \leq \alpha_2 \leq \pi$. An edge arc stores an array

Table 1: Attributes of the edge arc object.

EA	**Edge Arc**
v	Vertex around which the edge arc is centered
(α_1, α_2)	Start and stop angles of the arc, with respect to the e_1 direction
verts	Array of vertices in arc radially ordered clockwise in (e_1, e_2)-plane
count	Number of edges incident to v within the arc

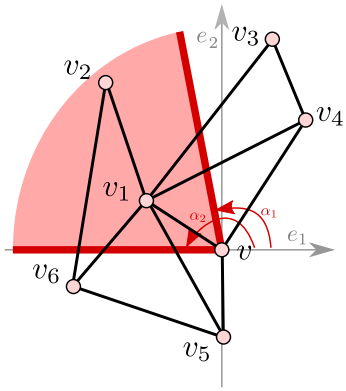


Figure 1: An edge arc EA centered at vertex $EA.v = v$. Other attributes of the edge arc include its start and stop angles, $EA.\alpha_1 = 1.75$ radians and $EA.\alpha_2 = \pi \approx 3.14$ radians, the array of vertices $EA.verts = \{v_1, v_2\}$, and the count of edges $EA.count = 1$. Here, we also see that $\texttt{Indeg}(v, e_1) = 1$ and $\texttt{Indeg}(v, e_2) = 2$.

of vertices sorted radially clockwise about $\pi(v)$ in the (e_1, e_2)-plane in decreasing angle with the e_1-direction. By construction, the first vertex in the array must be closest to α_2 and the last closest to α_1. The edge arc also stores the count of edges of E that have vertices from *verts* as endpoints. In implementation, the angles α_1 and α_2 do not need to be stored directly, but we include them in psuedocode and discussions for clarity.

Given some arc EA centered at vertex $v \in V$, we need to be able to compute $EA.count$, the number of edges contained EA that are adjacent to v. The following lemma provides such a computation. We omit a proof because it is a straightforward adaptation of [7, Theorem 16] and [2, Lemma 13].

Lemma 6 (Arc Count) *Let (V, E) be a graph GP-immersed in $\mathbb{R}^d$. Let EA be an edge arc object, and let $v = EA.v$. Let $s \in \mathbb{S}^{d-1}$ be the direction perpendicular to α_2 so that the arc is entirely below $s \cdot v$. Let E_* denote the edges with height $s \cdot v$ that are not contained in EA. If no vertex in V is at the same height as v in direction s, then*

$$EA.count = \texttt{Indeg}(v, s) - |E_*|.$$

As an illustration, again, consider Figure 1. Consider the direction $s = e^{i(\alpha_2 - \pi/2)}$, which is perpendicular to $e^{i\alpha}$ by construction. In addition, the edge arc is below $s \cdot v$ (specifically, all vertices in $EA.verts$ are below $s \cdot v$). Then, $\texttt{Indeg}(v, s) = 2$ and $E_* = \{[v, v_5]\}$. By Lemma 6, $EA.count = \texttt{Indeg}(v, s) - |E_*| = 2 - 1 = 1$. When we say that a list of vertices or edges is sorted clockwise around v, we mean that the list is sorted clockwise (cw) around $\pi(v)$ once projected into the (e_1, e_2)-plane with the largest angle first.

3 Fast Reconstruction

In this section, we provide an algorithm to reconstruct a graph (and, more generally, a one-skeleton of a simplicial complex) using the oracle. We start with an algorithm to find the edges, provided the vertex locations are known. We end with describing the complete graph reconstruction method.

3.1 Fast Edge Reconstruction

In this subsection, we assume we have a graph (V, E), where the vertex set V is known, but E is unknown. Using the oracle and the known vertex locations, we provide a reconstruction algorithm to find all edges in E (Algorithm 3). This algorithm is a sweepline algorithm in direction e_2 that, for each vertex processed in the sweep, performs a radial binary multi-search (Algorithm 2). This search is enabled by an algorithm that splits an edge arc object into two edge arcs, each containing half of the vertices (Algorithm 1). We provide the algorithms and relevant theorem statements here, but defer the proofs to Appendix A.

Algorithm 1 $\texttt{SplitArc}(EA, bigedges, \theta)$

Input: EA, an edge arc; $bigedges$, an array of all edges $(EA.v, v') \in E$ such that $\measuredangle\pi(v' - EA.v) < EA.\alpha_1$; θ, the minimum angle defined by any three vertices in $\pi(V)$

Output: EA_ℓ and EA_r, edge arcs satisfying the properties in Theorem 7

1: $n_v \leftarrow |EA.verts|$
2: $mid \leftarrow \lceil \frac{n_v}{2} \rceil$
3: $\alpha \leftarrow \measuredangle\pi(EA.verts[mid] - EA.v) - \theta/2$
4: $s \leftarrow e^{i(\alpha - \frac{\pi}{2})}$
5: $m_\ell \leftarrow \texttt{Indeg}(EA.v, s) - |\{b \in bigedges \mid \measuredangle b < \pi + \alpha\}|$
6: $m_r \leftarrow EA.count - m_\ell$
7: $EA_\ell \leftarrow$ edge arc where $EA_\ell.v = EA.v$, $EA_\ell.\alpha_1 = EA.\alpha_1$, $EA_\ell.\alpha_2 = \alpha$, $EA_\ell.verts = EA.verts[: mid]$, and $EA_\ell.count = m_\ell$
8: $EA_r \leftarrow$ edge arc where $EA_r.v = EA.v$, $EA_r.\alpha_1 = \alpha$, $EA_r.\alpha_2 = EA.\alpha_2$, $EA_r.verts = EA.verts[mid + 1 :]$, and $EA_r.count = m_r$
9: **return** (EA_ℓ, EA_r)

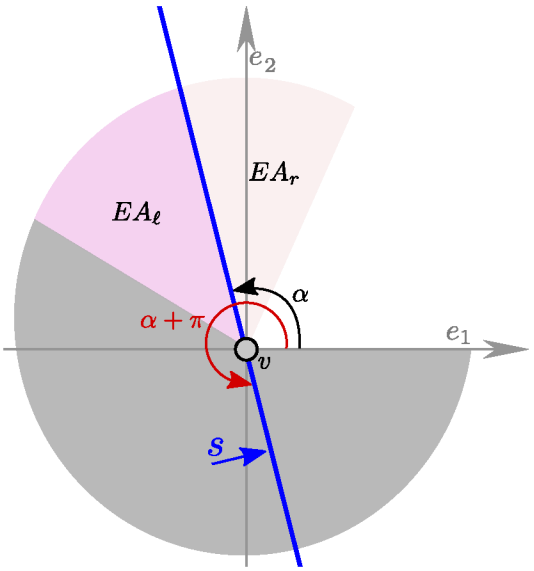


Figure 2: The splitting of edge arc EA into EA_ℓ and EA_r, as in Algorithm 1. The large gray region is the region containing all edges of $bigedges$. That is, all edges whose angle with the positive e_1-axis is at least $EA.\alpha_1$. On Line 5 of the algorithm, we compute the number of edges in EA_ℓ by first computing the indegree of $EA.v$ in direction s from the diagram in direction s, then we subtract the number of edges in $bigedges$ that are below the height of $EA.v$ in direction s (i.e., below the blue line). By the pigeonhole principal, we find $EA_r.count = EA.count - EA_\ell.count$.

In Algorithm 1, we find a direction s in the (e_1, e_2)-plane such that half of the vertices in $EA.verts$ are above v and half are below v with respect to the direction s. This allows us to create a new edge arcs corresponding to each half; see Figure 2. The properties of Algorithm 1 are described in the following theorem.

Theorem 7 (Arc Splitting) *Algorithm 1 uses one diagram and $\Theta(\log n + d + \Pi)$ time to split EA into two new edge arcs EA_ℓ and EA_r with the properties:*

(i) The sets $EA_r.verts$ and $EA_\ell.verts$ partition the vertex set $EA.verts$ such that the vertices in $EA_\ell.verts$ come before those in $EA_r.verts$, with respect to the clockwise ordering around $EA.v$.

(ii) $|EA_\ell.verts| = \lceil \frac{1}{2}|EA.verts| \rceil$.

(iii) $|EA_r.verts| = \lfloor \frac{1}{2}|EA.verts| \rfloor$.

In Algorithm 2, we use Algorithm 1 to find all outgoing edges from a given vertex. In particular, the algorithm maintains a stack of edge arc objects. When processing an edge arc (the while loop in Lines 4–16), we are determining which of the vertices in $verts$ form edges with v. If an edge arc has $count = 0$, it contains

Algorithm 2 `UpEdges`$(v, V_v, in_v, \theta, \widehat{\mathcal{D}})$

Input: $v \in V$; V_v, array of all vertices in V above v, ordered clockwise; in_v, array of all incoming edges of v, sorted radially clockwise; θ, the minimum angle formed by any three vertices in $\pi(V)$; and $\widehat{\mathcal{D}}$, the APD in direction e_2
Output: array of all outgoing edges of v

1: *indeg* ← indegree of v in direction $-e_2$.
2: *eastack* ← a stack of edge arc objects, initialized with a single edge arc A, where $A.v = v$, $A.\alpha_1 = 0$, $A.\alpha_2 = \pi$, $A.count = indeg$, and $A.verts = V_v$
3: $E_v \leftarrow \emptyset$
4: **while** *eastack* is not empty **do**
5: $EA \leftarrow eastack.pop()$
6: **if** $EA.count = 0$ **then**
7: **Continue** to top of while loop
8: **end if**
9: **if** $|EA.count| = |EA.verts|$ **then**
10: Append $v \times EA.verts$ to E_v, in order
11: **Continue** to top of while loop
12: **end if**
13: $(EA_\ell, EA_r) \leftarrow$ `SplitArc`$(EA, in_v \cup E_v, \theta)$
14: Push EA_r onto *eastack*
15: Push EA_ℓ onto *eastack*
16: **end while**
17: **return** E_v

no edges, and it can be ignored (Lines 6–8). If it has *count* exactly equal to the number of vertices in *verts*, each vertex in *verts* must form an edge with v (Lines 9–12). Otherwise, as demonstrated in Figure 3, the edge arc is split in half using Algorithm 1 and each half is put on the stack to be processed.

Theorem 8 (Finding Edges Above a Vertex) *Algorithm 2 finds the sorted array of edges above v using $\Theta(\deg(v)\log n)$ augmented persistence diagrams in $\Theta((\deg(v)\log n)(\log n + d + \Pi))$ time.*

Finally, our main algorithm (Algorithm 3) is a sweepline algorithm, where we consider the vertices in increasing order of their e_2-coordinates and find the outgoing edges of the vertex being considered.

Theorem 9 (Edge Reconstruction) *Let (V, E) be a graph GP-immersed in $\mathbb{R}^d$. Given V, Algorithm 3 reconstructs E using $\Theta(m \log n)$ augmented persistence diagrams in $\Theta(n^2 + m\log n(\log n + d + \Pi))$ time.*

3.2 Putting it Together: Full Reconstruction

The results of Section 3.1 are related to just part of the full process of reconstruction, since reconstruction begins with no knowledge of the underlying simplicial complex. Identifying the location of all vertices is the first step, and is one that has been previously examined in detail. In particular, Belton et al. provide an algorithm to reconstruct V in $\Theta(dn^{d+1} + d\Pi)$ time and $d+1$ oracle queries; see [2, Algorithm 1 & Theorem 9]. Together with Theorem 9, we obtain the following runtime and diagram count for a full reconstruction process.

Algorithm 3 `FindEdges`(V)

Input: V, array of all vertices in the unknown graph
Output: E, array of all edges in the unknown graph

1: $\widehat{\mathcal{D}} \leftarrow$ `Oracle`$(-e_2)$
2: $E \leftarrow \{\}$
3: *vertsabove* ← for each $v \in V$, an array clockwise ordering all vertices in V that are above v
4: $\theta \leftarrow$ min angle defined by any three vertices of $\pi(V)$
5: **for** v in V, in increasing height in direction e_2 **do**
6: *inedges* ← clockwise sorted array of edges in E incident to v
7: $E+=$ `UpEdges`$(v, vertsabove[v], inedges, \theta, \widehat{\mathcal{D}})$
8: **end for**
9: **return** E

Theorem 10 (Graph Reconstruction) *Using an oracle, we can reconstruct an unknown graph immersed in $\mathbb{R}^d$ using $\Theta(d + m\log n)$ diagrams in $\Theta(dn^{d+1} + d\Pi + n^2 + m\log n(\log n + d + \Pi))$ time.*

We omit a proof of Theorem 10, as it simply combines the results of [2, Theorem 9] and Theorem 9 of the current paper. Observing that the methods presented here are immediately applicable in the reconstruction of one-skeletons of general simplicial complexes, we have the following corollary:

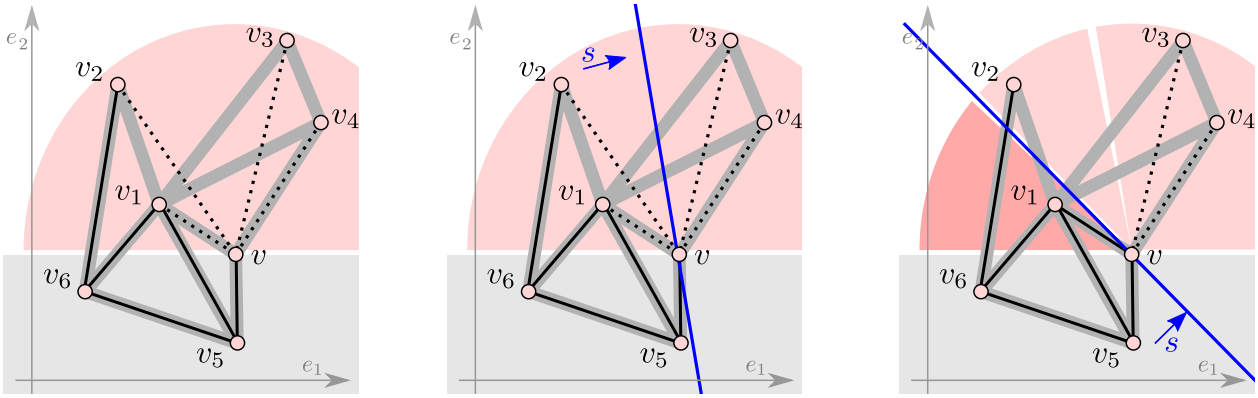
Corollary 11 (One-Skeleton Reconstruction) *Let K be an unknown simplicial complex GP-immersed in $\mathbb{R}^d$. Algorithm 1 of [2] and Algorithm 3 of the current paper reconstruct the one-skeleton of K using $\Theta(d + m\log n)$ augmented persistence diagrams in $\Theta(dn^{d+1} + d\Pi + n^2 + m\log n(\log n + d + \Pi))$ time.*

Finally, we note that embedding a graph (or simplicial complex) in $\mathbb{R}^2$ is a special case, as $m = O(n)$ and d is constant. In addition, by [2, Theorem 6], vertex reconstruction of a graph embedded in $\mathbb{R}^2$ can be done with three diagrams and $\Theta(n\log n + \Pi)$ time. Hence, we obtain a result for plane graph reconstruction:

Corollary 12 (Reconstruction in $\mathbb{R}^2$) *We can use an oracle to reconstruct the one-skeleton of an unknown simplicial complex embedded in $\mathbb{R}^2$ using $O(n\log n)$ diagrams and $O(n^2 + n\Pi\log n)$ time.*

4 Discussion

One way of proving that a discretization of the APHT is faithful is through the method of *reconstructing* the



(a) At vertex v. (b) Split edge arc. (c) Split again.

Figure 3: We demonstrate one step of Algorithm 2. (a) By assumption, we initially know $[v_5, v] \in E$. From Line 1 of Algorithm 2 we also know that two of the four vertices above v are adjacent to v. Thus, we create an edge arc object EA with $EA.count = 2$, and $EA.verts = (v_1, v_2, v_3, v_4)$. (b) In Algorithm 1, we choose a direction s such that half of the vertices in EA are below v. We use this split to create two edge arcs, EA_r and EA_ℓ, corresponding to the pink shaded regions on the right and left of the blue line defined by s. We push EA_r onto a stack to be processed later and focus on the arc EA_ℓ. Since two edges contribute to v's indegree in direction s and one is the known edge $[v_5, v]$, we have $EA.count = 2-1 = 1$. (c) Next, we find a new direction s that splits $EA_\ell.verts$ into two sets of size one. We push the set above s onto our stack. The edge arc containing only v_1 also has $EA.count = 2 - 1 = 1$, so $[v_1, v] \in E$. After all steps of Algorithm 2 are applied to find the edges above a particular vertex, Algorithm 2 is then applied to the next highest vertex, eventually processing every vertex in V in a sweep (Algorithm 3).

underlying simplicial complex. That is, by showing that the underlying simplicial complex can be recovered with the data of the discretization alone. In this paper, we take that approach and provide an algorithm for reconstructing a graph immersed in $\mathbb{R}^d$. We use fewer persistence diagrams than presented in alternate approaches. For example, the algorithm that we present for edge reconstruction (when the vertex locations are known) uses $\Theta(m \log n)$ diagrams. In contrast, [2, Theorem 16] uses $n^2 - n$ diagrams. Note that, for a very dense edge set, that is, when $m = \Theta(n^2)$, the method in [2, Theorem 16] uses fewer diagrams. However, if $m = O(n)$, as is common in many complexes, the represntation computed in this paper has fewer diagrams. Moreover, we emphasize that the number of diagrams is not exponential in the ambient dimension.

One might hope to use binary search strategies to reconstruct a simplicial complexe, but the methods presented here are unique to one-skeletons. Radially ordering higher dimensional simplices is not well-defined, and this issue prevents the methods presented here from being immediately transferrable. On the other hand, with the representation in this paper being output-sensitive (as opposed to testing if every pair of vertices is a simplex), we have hope for the discretization of the (A)PHT of a simplicial complex immersed in $\mathbb{R}^d$ being proportional to the size of the complex itself.

We also observe that not all diagrams used in our reconstruction algorithms were strictly necessary (i.e., the set of diagrams used were not a minimal faithful set). One straightforward way to reduce the number of diagrams used without altering the method much would be to split the region above a vertex in the sweep into arcs that contain exactly the same number of edges as vertices, or no edges. This property can then be validated by a simple difference of indegrees. In ongoing work, we hope to make these claims precise. We also hope to extend our methods to use topological descriptors that are *not* dimension-returning (such as augmented Euler Characteristic curves).

References

[1] R. L. Belton, B. T. Fasy, R. Mertz, S. Micka, D. L. Millman, D. Salinas, A. Schenfisch, J. Schupbach, and L. Williams. Learning simplicial complexes from persistence diagrams. In *Canadian Conference on Computational Geometry*, August 2018. Also available at arXiv:1805.10716.

[2] R. L. Belton, B. T. Fasy, R. Mertz, S. Micka, D. L. Millman, D. Salinas, A. Schenfisch, J. Schupbach, and L. Williams. Reconstructing embedded graphs from persistence diagrams. *Computational Geometry: Theory and Applications*, 2020.

[3] P. Bendich, J. S. Marron, E. Miller, A. Pieloch, and S. Skwerer. Persistent homology analysis of brain artery trees. *The Annals of Applied Statistics*, 10(1):198, 2016.

[4] L. M. Betthauser. *Topological Reconstruction of Grayscale Images*. PhD thesis, University of Florida, 2018.

[5] J. Curry, S. Mukherjee, and K. Turner. How many directions determine a shape and other sufficiency results for two topological transforms. arXiv:1805.09782, 2018.

[6] H. Edelsbrunner and J. Harer. *Computational Topology: An Introduction*. American Mathematical Society, 2010.

[7] B. T. Fasy, S. Micka, D. L. Millman, A. Schenfisch, and L. Williams. A faithful discretization of the augmented persistent homology transform. 2022. arXiv:1912.12759.

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

[9] C. Giusti, E. Pastalkova, C. Curto, and V. Itskov. Clique topology reveals intrinsic geometric structure in neural correlations. *Proceedings of the National Academy of Sciences*, 112(44):13455–13460, 2015.

[10] P. Lawson, A. B. Sholl, J. Q. Brown, B. T. Fasy, and C. Wenk. Persistent homology for the quantitative evaluation of architectural features in prostate cancer histology. *Scientific Reports*, 9, 2019.

[11] Y. Lee, S. D. Barthel, P. Dłotko, S. M. Moosavi, K. Hess, and B. Smit. Quantifying similarity of pore-geometry in nanoporous materials. *Nature Communications*, 8:15396, 2017.

[12] S. A. Micka. *Searching and Reconstruction: Algorithms with Topological Descriptors*. PhD thesis, Montana State University, 2020.

[13] D. L. Millman and V. Verma. A slow algorithm for computing the Gabriel graph with double precision. *Proceedings of the 23rd Annual Canadian Conference on Computational Geometry*, 2011.

[14] A. H. Rizvi, P. G. Camara, E. K. Kandror, T. J. Roberts, I. Schieren, T. Maniatis, and R. Rabadan. Single-cell topological RNA-seq analysis reveals insights into cellular differentiation and development. *Nature Biotechnology*, 35(6):551, 2017.

[15] G. Singh, F. Mémoli, and G. E. Carlsson. Topological methods for the analysis of high dimensional data sets and 3d object recognition. *SPBG*, 91:100, 2007.

[16] 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, 2014.

[17] S. Tymochko, E. Munch, J. Dunion, K. Corbosiero, and R. Torn. Using persistent homology to quantify a diurnal cycle in hurricanes. *Pattern Recognition Letters*, 2020.

[18] Y. Wang, H. Ombao, and M. K. Chung. Statistical persistent homology of brain signals. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1125–1129. IEEE, 2019.

A Algorithmic Proofs

In this appendix, we provide the proofs omitted from Section 3. These proofs provide justification for the runtimes, diagram compelxity, and correctness of the algorithms presented in this paper.

A.1 Proof of Theorem 7

Theorem 7 (Arc Splitting) *Algorithm 1 uses one diagram and $\Theta(\log n + d + \Pi)$ time to split EA into two new edge arcs EA_ℓ and EA_r with the properties:*

(i) The sets $EA_r.verts$ and $EA_\ell.verts$ partition the vertex set $EA.verts$ such that the vertices in $EA_\ell.verts$ come before those in $EA_r.verts$, with respect to the clockwise ordering around $EA.v$.

(ii) $|EA_\ell.verts| = \lceil \frac{1}{2}|EA.verts| \rceil$.

(iii) $|EA_r.verts| = \lfloor \frac{1}{2}|EA.verts| \rfloor$.

Proof. For the runtime, we walk through the algorithm and analyze the time and diagram complexity of each line. In Lines 1–3, we find the angle α that splits $EA.verts$ into two equal sets, then in Line 4 compute a direction s orthogonal to α. See Figure 2. Lines 1–4 use no diagrams and can be done in constant time when restricting our attention to the (e_1, e_2)-plane. However, we need s to be a direction in $\mathbb{R}^d$ (as opposed to only in the (e_1, e_2)-plane), so the computation takes $\Theta(d)$ time.[3] Specifically, s is the vector

$$s = e^{\frac{1}{2}i(2\alpha - \pi - \theta)} \tag{2}$$
$$= \left(\cos\left(\alpha - \frac{1}{2}\pi - \frac{1}{2}\theta\right), \sin\left(\alpha - \frac{1}{2}\pi - \frac{1}{2}\theta\right), 0, 0, \ldots, 0\right).$$

To compute m_ℓ in Line 5, we compute $\texttt{Indeg}(v, s)$ then subtract the cardinality of the set $S := \{b \in \textit{bigedges} \mid \measuredangle b < \pi + \alpha\}$, where $\measuredangle b$ is taken to mean the angle b makes with the e_1-axis, when viewed as a vector with $EA.v$ as the origin. By Lemma 5, we compute $\texttt{Indeg}(v, s)$ via the oracle using one diagram and $\Theta(\log n + \Pi)$ time. Since *bigedges* is sorted and since s lies in the (e_1, e_2)-plane, we can find the set S in $\Theta(\log(|\textit{bigedges}|))$ time. The subtraction in Line 5 takes constant time, as does Line 6.

In Lines 7 and 8, we create two edge arc objects. The time complexity of creating them is proportional to the size of the obejcts themselves. All attributes of edge arc objects, except the array of vertices (*verts*), are constant size. By construction, $EA_\ell.verts$ and $EA_r.verts$ split $EA.verts$ into two sets, which can be done naïvely in $\Theta(d|EA.verts|)$ time by walking through $EA.verts$ and storing each one explicitly. However, we improve this to $\Theta(\log |EA.verts|)$ time if we have a globally accessible array of vertices (sorted cw around v) and just computes the pointers to the beginning and end of the subarrays corresponding to the *verts* attributes of the new edge arc objects. In total, Algorithm 1 and takes $\Theta(d + \log n +$

[3] With some clever data structures, this $\Theta(d)$ can be reduced to constant time. For example, we could require vectors in $\mathbb{R}^2$ are automatically padded with 0's to become vectors in $\mathbb{R}^d$ when needed. However, this is out of the scope of the real RAM model of computation.

$\Pi + \log(|bigedges|) + 1 + \log(|EA.verts|)) = \Theta(\log n + d + \Pi)$ time and uses uses one diagram.

Now that we have walked through the algorithm and established the runtime and diagram complexity, we prove correctness. To do so, we first show that EA_ℓ and EA_r are edge arc objects. In particular, this means showing that they have the correct values for *count* and *verts*. We prove this for EA_ℓ; the proof for EA_r follows a similar argument.

EA_ℓ.count: We must show that $EA_\ell.count$ is the number of edges in EA_ℓ incident to $EA_\ell.v$. By Lemma 4, the value returned from $\texttt{Indeg}(EA.v, s)$ counts all edges incident to $EA.v$ and below $s \cdot EA.v$ in direction s. By Lemma 6, this is exactly the total number of edges in EA_ℓ plus edges $(EA.v, v') \in E_v$ for which $s \cdot v' < s \cdot EA.v$. Thus, by subtracting $|\{b \in bigedges \mid \measuredangle b < \pi + \alpha\}|$ from $\texttt{Indeg}(EA.v, s)$ on Line 5, we are left with m_ℓ, the number of edges incident to $EA.v$ contained in EA_ℓ. Setting $EA_\ell.count = m_\ell$ on Line 7, we see that $EA_\ell.count$ is correct.

EA_ℓ.verts: We must show that $EA_\ell.verts$ contains an array of all verices contained in EA_ℓ radially ordered clockwise. This follows from the fact that $EA.verts$ is all vertices contained in EA ordered clockwise, so when we restrict $EA.verts$ to $EA.verts[: mid]$ on Line 7, we are eliminating vertices not contained in EA_ℓ, so $EA_\ell.verts$ is correct.

Next, we prove Statement (i). Recall that $EA.verts$ orders the vertices in decreasing angle with e_1. In Line 3, $\measuredangle\pi(EA.verts[mid] - EA.v)$ is the angle made by $EA.v$ with the middle vertex. We tilt this angle by $\theta/2$ on Line 3 to obtain angle α. By construction of α,

$$\measuredangle\pi(EA.verts[mid] - EA.v) > \alpha.$$

By definition of θ, the angle α satisfies:

$$\alpha > \measuredangle\pi(EA.verts[mid + 1] - EA.v).$$

Since the array $EA.verts$ is sorted, all vectors in the set $\pi(EA.verts[: mid] - EA.v)$ have an angle of at least α with e_1 and all vectors in $\pi(EA.verts[: mid] - EA.v)$ have an angle of at most α.

By Lines 1–2 and Line 5, we know that $EA.verts$ contains the first $m = \lceil \frac{1}{2}|EA.verts| \rceil$ vertices in $EA.verts$. Hence, Statement (ii) holds. Statement (iii) follows from Statements (i) and (ii). □

A.2 Proof of Theorem 8

Theorem 8 (Finding Edges Above a Vertex) *Algorithm 2 finds the sorted array of edges above v using $\Theta(\deg(v) \log n)$ augmented persistence diagrams in $\Theta((\deg(v) \log n)(\log n + d + \Pi))$ time.*

Proof. First, we analyze the time complexity of the algorithm and the number of diagrams it requires. By Lemma 5, Line 1 can be computed in $\theta(\log n)$ time (since we are given the diagram and do not need an additional oracle query). Storing $A.verts$ by storing a pointer to V_v, we initialize *eastack* and E_v in Lines 2 and 3 in constant time.

To analyze the complexity of the loop in Lines 4–16, we first note that this is a radial binary multi-search. When processing an edge arc, we decide whether all edges have been found or if we need to split the edge arc. If there is only one edge in the arc (i.e., $EA.count = 1$), then this loop is a binary search for an edge, using the angle with e_1 in the (e_1, e_2)-plane as the search key. When $EA.count > 1$, we search for all edges, finding them in decreasing angle order (since arcs with larger angles are added after arcs of smaller angles). The if statement in Lines 9–12 is where the edges are added to E_v. Note that this shortcuts additional edge arc splitting by stopping the process once we find that the number of edges in the arc is equal to the number of potential vertices that can form the edges. As a result, each edge above v contributes to $O(\log n)$ edge arcs being added to *eastack* and, in the case that every other vertex is incident to an edge with v, we have $\Theta(\log n)$ edge arcs added to the stack. All operations in the while loop are constant time, except splitting the edge arc object in Line 13, which uses one diagram and takes $\Theta(\log n + d + \Pi)$ time.

The complexity of Algorithm 2 is dominated by the complexity of the while loop: the algorithm uses $\Theta(\deg(v) \log n)$ augmented persistence diagrams and takes takes $\Theta((\deg(v) \log n)(\log n + d + \Pi))$ time.

To prove correctness of this algorithm, we state the loop invariant for the while loop:

(i) For $(v, v') \in E$:

- If $\measuredangle(v' - v) > E_v.\alpha_1$, then (v, v') is either in E_v or in_v.
- If $\measuredangle(v' - v) > E_v.\alpha_1$, then v' is in *verts* for some edge arc in *eastack*
- $\measuredangle(v' - v) \neq E_v.\alpha_1$

(ii) The edge arc stack is clockwise-ordered.

This loop invariant ensures that the call to Algorithm 1 in Line 13 has valid input and that all outgoing edges are found when the algorithm terminates. □

A.3 Proof of Theorem 9

Theorem 9 (Edge Reconstruction) *Let (V, E) be a graph GP-immersed in $\mathbb{R}^d$. Given V, Algorithm 3 reconstructs E using $\Theta(m \log n)$ augmented persistence diagrams in $\Theta(n^2 + m \log n(\log n + d + \Pi))$ time.*

Proof. We first analyze the runtime and diagram count for Algorithm 3 by walking through the algorithm line-by-line. In Line 1, we ask the oracle for the diagram in direction $-e_2$, which takes $\Theta(\Pi)$ time. In [2, Theorem 14 (Edge Reconstruction)], simultaneously find the cyclic ordering around all vertices in $\Theta(n^2)$ time by Lemmas 1 and 2 of [13]. In Line 3, we do that as well; however, we do not store vertices that are above v in the array $vertsabove[v]$, and thus this line takes $\Theta(n^2)$ time. We note that such a cyclic ordering exists around each vertex by Assumption 1projectedindep. Once we have *vertsabove*, to find the minimum angle defined by any three vertices of V, we check all angles between vectors $vertsabove[v][i] - v$ and $vertsabove[v][i+1] - v$ in Line 4 in $\Theta(n + m)$ time.

The for loop in Lines 5–8 is repeated n times, once for each vertex in V. To determine the order of processing the vertices in V, we follow the births in $\widehat{\mathcal{D}}_0$, in decreasing order (since $\widehat{\mathcal{D}}_0$ is the lower-star filtration in direction $-e_2$).

Thus, finding the order takes $\Theta(n)$ time. In each iteration, we compute the incoming edges (those whose other vertex is below v) in Line 6 followed by all outgoing edges (those whose other vertex is above v) in Line 7. By Assumption 1(iii), every edge is either incoming or outgoing with respect to direction e_2. Thus, all edges incident to v are in E once E is updated in Line 6. By Theorem 8, when processing vertex v, the call to Algorithm 2 on Line 7 takes $\Theta((\deg(v)\log n)(\log n + d + \Pi))$ time and uses $\Theta(\deg(v)\log n)$ diagrams. Summing over all vertices, we see that the loop in Lines 5–8 takes

$$\sum_{v\in V} \Theta((\deg(v)\log n)(\log n + d + \Pi)) = \Theta(m \log n(\log n + d + \Pi))$$

time and uses $\sum_{v\in V} \Theta(\deg(v)\log n) = \Theta(m\log n)$ augmented persistence diagrams.

In total, Algorithm 3 takes $\Theta(\Pi) + \Theta(n^2) + \Theta(n+m) + \Theta(n) + \Theta(m\log n(\log n + d + \Pi)) = \Theta(n^2 + m\log n(\log n + d + \Pi))$ time and uses $\Theta(1) + \Theta(m\log n) = \Theta(m\log n)$ diagrams.

Next, we prove the correctness of Algorithm 3 (i.e., that all edges are found). In order to process vertices in order of their heights in the e_2 direction, we first sort them in Line 5. For $1 \le j \le n$, let v_j be the j^{th} vertex in this ordering. To show that Algorithm 3 finds all edges in E, we consider the loop invariant (LI): when we process v_j, all edges with maximum vertex height equal or less than the height of v_j are known. The LI is trivially true for v_1. We now assume that it is true for iteration j, and show that it must be true for iteration $j+1$. By assumption, all edges (v_i, v_j) with $1 \le i < j$ are known, and so by Theorem 8, Algorithm 2 finds all edges (v_k, v_j), where $k > j$, and we add them to the edge set E. Note that, by assumption, all edges (v_x, v_i) for $1 \le i \le j$ are also already known, and so the invariant is maintained. Thus, after the loop terminates, all edges are found. □

B Basis

In Assumption 1(iii), we assume all vertices of the underlying graph are unique with respect to the first basis direction e_2. In this appendix, we provide details of how to *find* a basis where all vertices have a unique height with respect to the second basis direction. I.e., this appendix allows us to remove one general position assumption by showing it can be satisfied deterministically, at an added cost of $\Theta(|P|\log|P| + d + \Pi)$ time.

Lemma 13 (Creation of Orthonormal Basis) *Given a point set $P \subset \mathbb{R}^d$ satisfying Assumption 1(i) and Assumption 1(ii), we can use two diagrams and $\Theta(|P|\log|P| + d + \Pi)$ time to create the orthonormal basis $\{b_1, b_2, e_3, e_4, \ldots, e_d\}$ so that all points of P have a unique height in direction b_2.*

Proof. Algorithm 6 (`Tilt`) of [7] takes diagrams from two linearly independent directions $s, s' \in \mathbb{S}^{d-1}$, the point set P, and returns a direction s_* in $\Theta(|P|\log|P| + d + \Pi)$ time[4] so that the following properties holds for all $p_1, p_2 \in P$:

(i) If p_1 is strictly above (below) p_2 with respect to direction s, then p_1 is strictly above (below, respectively) p_2 with respect to direction s_*.

(ii) If p_1 and p_2 are at the same height with respect to direction s and p_1 is strictly above (below) p_2 with respect to direction s', then p_1 is strictly above (respectively, below) p_2 with respect to direction s_*.

(iii) If p_1 is is at the same height as p_2 with respect to both directions s and s', then p_1 and p_2 are at the same height with respect to direction s_*.

A proof of correctness is given in [7, Lemma 32 (Tilt)].

We start with the standard basis for $\mathbb{R}^d$, $\{e_1, e_2, \ldots, e_d\}$, and we replace the first two basis elements as follows. Let b_2 be the direction obtain by using `Tilt` with $s = e_1$, $s' = e_2$, and $P = P$.

By Assumption 1(ii), no three points of P are colinear when projected onto the first two coordinates. In particular, this means no two points share the same heights in both the e_1 and e_2 directions. Then, by Statements (i)-(ii) above, the direction b_2 must order all vertices of P uniquely. Using only the first two coordinates of b_2 and e_1, we then perform Gram Schmidt orthanormalization to compute the first two coordinates of b_1. More precisely, letting $b_i^{(j)}$ denote the jth coordinate of b_i, we compute

$$\begin{pmatrix} b_1^{(1)} \\ b_1^{(2)} \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix} - \frac{\left\langle \begin{pmatrix} b_2^{(1)} & b_2^{(2)} \end{pmatrix}^T, \begin{pmatrix} 1 & 0 \end{pmatrix}^T \right\rangle}{\left|\left| \begin{pmatrix} b_2^{(1)} & b_2^{(2)} \end{pmatrix}^T \right|\right|^2} \begin{pmatrix} b_2^{(1)} \\ b_2^{(2)} \end{pmatrix} \quad (3)$$

We then set $b_1^{(j)} = 0$ for $2 < j \le d$, so that $b_1 \in \text{span}\{e_1, e_2\}$, $b_2 \perp b_1$, and $||b_1|| = 1$. Only considering the first two coordinates of b_2 and e_1 means this process takes constant time. The remaining e_i for $2 \le i \le d$ can be used to fill the basis.

Finally, we have a basis satisfying all assumptions of Assumption 1, namely, $\{b_1, b_2, e_3, e_4, \ldots, e_d\}$. □

[4] While [7] does not account for diagram computation time, there are two diagrams used in this process, hence our addition of $\Theta(\Pi)$ to the total runtime.