An Efficient Algorithm for 1-Dimensional (Persistent) Path Homology

Tamal K. Dey
Department of Computer Science and Engineering,
The Ohio State University, Columbus, OH, 43210, USA
tamaldey@cse.ohio-state.edu

Tianqi Li
Department of Computer Science and Engineering,
The Ohio State University, Columbus, OH, 43210, USA
li.6108@osu.edu

Yusu Wang
Department of Computer Science and Engineering,
The Ohio State University, Columbus, OH, 43210, USA
yusu@cse.ohio-state.edu

Abstract

This paper focuses on developing an efficient algorithm for analyzing a directed network (graph) from a topological viewpoint. A prevalent technique for such topological analysis involves computation of homology groups and their persistence. These concepts are well suited for spaces that are not directed. As a result, one needs a concept of homology that accommodates orientations in input space. Path-homology developed for directed graphs by Grigoryan, Lin, Muranov and Yau has been effectively adapted for this purpose recently by Chowdhury and Mémoli. They also give an algorithm to compute this path-homology. Our main contribution in this paper is an algorithm that computes this path-homology and its persistence more efficiently for the 1-dimensional (H_1) case. In developing such an algorithm, we discover various structures and their efficient computations that aid computing the 1-dimensional path-homology. We implement our algorithm and present some preliminary experimental results.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases computational topology, directed graph, path homology, persistent path homology

Digital Object Identifier 10.4230/LIPIcs.SoCG.2020.36

Related Version A full version of this paper is available at `https://arxiv.org/abs/2001.09549`.

Funding This work is in part supported by National Science Foundation under grants CCF1740761, DMS-1547357, and RI-1815697.

1 Introduction

When it comes to graphs, traditional topological data analysis has focused mostly on undirected ones. However, applications in social networks [1, 15], brain networks [16], and others require processing directed graphs. Consequently, topological data analysis for these applications needs to be adapted accordingly to account for directedness. Recently, some work [4, 14] have initiated to address this important but so far neglected issue.

Since topological data analysis uses persistent homology as a main tool, one needs a notion of homology for directed graphs. Of course, one can forget the directedness and consider the underlying undirected graph as a simplicial 1-complex and use a standard persistent homology pipeline for the analysis. However, this is less than desirable because the important information involving directions is lost. Currently, there are two main approaches that have been proposed for dealing with directed graphs. One uses directed clique complexes [8, 14] and the other uses the concept of path homology [10]. In the first approach, a k-clique in the input directed graph is turned into a $(k-1)$-simplex if the clique has a single source and a single sink. The resulting simplicial complex is subsequently analyzed with the usual persistent homology pipeline. One issue with this approach is that there could be very few cliques with the required condition and thus accommodating only a very few higher dimensional simplices. In the worst case, only the undirected graph can be returned as the directed clique complex if each 3-clique is a directed cycle. The second approach based on path homology alleviates this deficiency. Furthermore, certain natural functorial properties, such as Künneth formula, do not hold for the clique complex [10].

The path homology, originally proposed by Grigoryan, Lin, Muranov and Yau in 2012 [10] and later studied by [5, 11, 12], has several properties that make it a richer mathematical structure. For example, there is a concept of homotopy under which the path homology is preserved; it accommodates Künneth formula; and the path homology theory is dual to the cohomology theory of digraphs introduced in [12]. Furthermore, persistent path homology developed in [5] is shown to respect a stability property for its persistent diagrams.

To use path homologies effectively in practice, one needs efficient algorithms to compute them. In particular, we are interested in developing efficient algorithms for computing 1-dimensional path homology and its persistent version because even for this case the current state of the art is far from satisfactory: Given a directed graph G with n vertices, the most efficient algorithm proposed in [5] has a time complexity $O(n^9)$ (more precisely, their algorithm takes $O(n^{3+3d})$ to compute the $(d-1)$-dimensional persistent path-homology).

The main contribution of this paper is stated in Theorem 1. The reduced time complexity of our algorithm can be attributed to the fact that we compute the boundary groups more efficiently. In particular, it turns out that for 1-dimensional path homology, the boundary group is determined by bigons, certain triangles, and certain quadrangles in the input directed graph. The bigons and triangles can be determined relatively easily. It is the boundary quadrangles whose computation and size determine the time complexity. The authors in [5] compute a basis of these boundary quadrangles by constructing a certain generating set for the 2-dimensional chain group by a nice column reduction algorithm (being different from the standard simplicial homology, it is non-trivial to do reduction for path homology). We take advantage of the concept of *arboricity* and related results in graph theory, together with other efficient strategies, to enumerate a much smaller set generating the boundary quadrangles. Computing the cycle and boundary groups efficiently both for non-persistent and persistent homology groups is the key to our improved time complexity.

▶ **Theorem 1.** *Given a directed graph G with n vertices and m edges, set $r = \min\{\mathsf{a}(G)m, \sum_{(u,v)\in E}(d_{in}(u) + d_{out}(v))\}$, where $\mathsf{a}(G)$ is the so-called arboricity of G (with $\mathsf{a}(G) = O(n)$), and $d_{in}(u)$ and $d_{out}(u)$ are the in-degree and out-degree of u, respectively. There is an $O(rm^{\omega-1} + m\alpha(n))$ time algorithm for computing the* 1-dimensional persistent path homology *for G where $\omega < 2.373$ is the exponent for matrix multiplication*[1]*, and $\alpha(\cdot)$ is the inverse Ackermann function.*

This also gives an $O(rm^{\omega-1} + m\alpha(n))$ time algorithm for computing the 1-dimensional path homology H_1 of G.

In particular, for a planar graph G, $\mathsf{a}(G) = O(1)$ and the time complexity becomes $O(n^{\omega})$.

[1] That is, the fastest algorithm to multiply two $r \times r$ matrices takes time $O(r^{\omega})$.

The *arboricity* $\mathsf{a}(G)$ of a graph G mentioned in Theorem 1 denotes the minimum number of edge-disjoint spanning forests into which G can be decomposed [13]. It is known that in general, $\mathsf{a}(G) = O(n)$, but it can be much smaller. For example, $\mathsf{a}(G) = O(1)$ for planar graphs and $\mathsf{a}(G) = O(g)$ for a graph embedded on a genus-g surface [3]. Hence, for planar graphs, we can compute 1-dimensional persistent path homology in $O(n^\omega)$ time whereas the algorithm in [5] takes $O(n^5)$ time[2].

In the full version, we develop an algorithm to compute 1-dimensional *minimal path homology basis* [7, 9], and also show experiments demonstrating the efficiency of our new algorithms.

Organization of the paper. After characterizing the 1-dimensional path homology group H_1 in Section 3, we first propose a simple algorithm to compute it. In Section 4, we consider its persistent version and present an improved and more efficient algorithm.

2 Background

We briefly introduce some necessary background for path homology. Interested readers can refer to [10] for more details. The original definition can be applied to structures beyond directed graphs; but for simplicity, we use directed graphs to introduce the notations.

Given a directed graph $G = (V, E)$, we denote (u, v) as the directed edge from u to v. A *self-loop* is defined to be the edge (u, u) from u to itself. Throughout this paper, we assume that G does not have self-loops. We also assume that G does not have multi-edges, i.e. for every ordered pair u, v, there is at most one directed edge from u to v. For notational simplicity, we sometimes use index i to refer to vertex $v_i \in V = \{v_1, \dots, v_n\}$.

Let $\mathbb{F}$ be a field with 0 and 1 being the additive and multiplicative identities respectively. We use $-a$ to denote the additive inverse of a in $\mathbb{F}$. An *elementary d-path* on V is simply a sequence $i_0, i_1, \cdots, i_d$ of $d+1$ vertices in V. We denote this path by $e_{i_0,i_1,\cdots,i_d}$. Let $\Lambda_d = \Lambda_d(G, \mathbb{F})$ denote the $\mathbb{F}$-linear space of all linear combinations of elementary d-paths with coefficients from $\mathbb{F}$. It is easy to check that the set $\{e_{i_0,\cdots,i_d} \mid i_0, \cdots, i_d \in V\}$ is a basis for Λ_d. Each element p of Λ_d is called a *d-path*, and it can be written as

$$p = \sum\nolimits_{i_0,\cdots,i_d \in V} a_{i_0\cdots i_d} e_{i_0\cdots i_d}, \text{ where } a_{i_0\cdots i_d} \in \mathbb{F}.$$

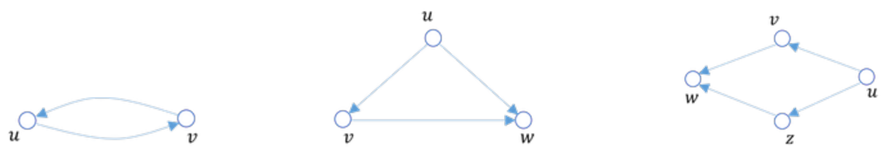
Similar to simplicial complexes, there is a well-defined *boundary operator* $\partial : \Lambda_d \to \Lambda_{d-1}$:

$$\partial e_{i_0\cdots i_d} = \sum_{i_0,\cdots,i_d \in V} (-1)^j e_{i_0\cdots \hat{i}_j \cdots i_d},$$

where $\hat{i}_k$ means the omission of index i_k. The boundary of a path $p = \sum_{i_0,\cdots,i_d \in V} a_{i_0\cdots i_d} \cdot e_{i_0\cdots i_d}$, is thus $\partial p = \sum_{i_0,\cdots,i_d \in V} a_{i_0\cdots i_d} \cdot \partial e_{i_0\cdots i_d}$. We set $\Lambda_{-1} = 0$ and note that Λ_0 is the set of $\mathbb{F}$-linear combinations of vertices in V. Lemma 2.4 in [10] shows that $\partial^2 = 0$.

Next, we restrict to real paths in directed graphs. Specifically, given a directed graph $G = (V, E)$, call an elementary d-path $e_{i_0,\cdots,i_d}$ *allowed* if there is an edge from i_k to i_{k+1} for all k. Define $\mathcal{A}_d$ as the space of all allowed d-paths, that is, $\mathcal{A}_d := \text{span}\{e_{i_0\cdots i_d} : e_{i_0\cdots i_d} \text{ is allowed}\}$. An elementary d-path $i_0 \cdots i_d$ is called *regular* if $i_k \neq i_{k+1}$ for all k, and is *irregular* otherwise. Clearly, every allowed path is regular since there is no self-loop. However,

[2] The original time complexity stated in the paper is $O(n^9)$ for 1-dimensional case. However, one can improve it to $O(n^5)$ by a more refined analysis for planar graphs.



(a) Bigon. **(b)** Boundary triangle. **(c)** Boundary quadrangle.

Figure 1 Examples of 1-boundaries.

the boundary map ∂ on Λ_d may create a term resulting into an irregular path. For example, $\partial e_{uvu} = e_{vu} - e_{uu} + e_{uv}$ is irregular because of the term e_{uu}. To deal with this case, the term containing consecutive repeated vertices is identified with 0 [10]. Thus, for the previous example, we get $\partial e_{uvu} = e_{vu} - 0 + e_{uv} = e_{vu} + e_{uv}$. The boundary map ∂ on $\mathcal{A}_d$ is taken to be the boundary map for Λ_d restricted on $\mathcal{A}_d$ with this modification: where all terms with consecutive repeated vertices created by the boundary map ∂ are replaced with 0's.

Unfortunately, after restricting to the space of allowed paths $\mathcal{A}_*$, the inclusion that $\partial\mathcal{A}_d \subset \mathcal{A}_{d-1}$ may not hold any more; that is, the boundary of an allowed d-path is not necessarily an allowed $(d-1)$-path. To this end, we adopt a stronger notion of allowed paths: an allowed path p is *∂-invariant* if ∂p is also allowed. Let $\Omega_d := \{p \in \mathcal{A}_d \mid \partial p \in \mathcal{A}_{d-1}\}$ be the space generated by all ∂-invariant paths. We then have $\partial\Omega_d \subset \Omega_{d-1}$ (as $\partial^2 = 0$). This gives rise to the following *chain complex of ∂-invariant allowed paths*:

$$\cdots\Omega_d \xrightarrow{\partial} \Omega_{d-1} \xrightarrow{\partial} \cdots\Omega_d \xrightarrow{\partial} \Omega_0 \xrightarrow{\partial} 0.$$

We can now define the homology groups of this chain complex. The d-th cycle group is defined as $\mathsf{Z}_d = \operatorname{Ker}\partial|_{\Omega_d}$, and elements in Z_d are called d-cycles. The d-th boundary group is defined as $\mathsf{B}_d = \operatorname{Im}\partial|_{\Omega_{d+1}}$, with elements of B_d being called d-boundary cycles (or simply d-boundaries). The resulting *d-dimensional path homology group* is $\mathsf{H}_d(G,\mathbb{F}) = \mathsf{Z}_d/\mathsf{B}_d$.

2.1 Examples of 1-boundaries

Below we give three examples of 1-boundaries; see Figure 1.

Bi-gon. A *bi-gon* is a 1-cycle $e_{uv} + e_{vu}$ consisting of two edges (u,v) and (v,u) from E; see Figure 1(a). Consider the 2-path e_{uvu}. We have that its boundary is $\partial(e_{uvu}) = e_{vu} - e_{uu} + e_{uv} = e_{vu} + e_{uv}$. Since both e_{vu} and e_{uv} are allowed 1-paths, it follows that any bi-gon $e_{vu} + e_{uv}$ of G is necessarily a 1-boundary.

Boundary triangle. Consider the 1-cycle $C = e_{vw} - e_{uw} + e_{uv}$ of G (it is easy to check that $\partial C = 0$). Now consider the 2-path e_{uvw}: its boundary is then $\partial(e_{uvw}) = e_{vw} - e_{uw} + e_{uv} = C$. Note that every summand in the boundary is allowed. Thus C is a 1-boundary. We call any triangle in G isomorphic to C a *boundary triangle*. Note that a boundary triangle always has one sink and one source; see the source u and sink w in Figure 1(b). In what follows, we use $(u, w \mid v)$ to denote a boundary triangle where u is the source and w is the sink.

Boundary quadrangle. Consider the 1-cycle $C = e_{uv} + e_{vw} - e_{uz} - e_{zw}$ from G. It is easy to check that C is the boundary of the 2-path $e_{uvw} - e_{uzw}$, as $\partial(e_{uvw} - e_{uzw}) = e_{vw} - e_{uw} + e_{uv} - (e_{zw} - e_{uw} + e_{uz}) = e_{vw} + e_{uv} - e_{zw} - e_{uz} = C$. We call any quadrangle isomorphic to C a *boundary quadrangle*.

In the remainder of the paper, we use $R(u, v, w, z)$ to represent a quadrangle; i.e, a 1-cycle consisting of 4 edges whose *undirected version* has the form $(u, v) + (v, w) + (w, z) + (z, u)$. (Note that a quadrangle may not be a boundary quadrangle). We denote a boundary quadrangle $e_{uv} + e_{vw} - e_{uz} - e_{zw}$ by $\{u, w \mid v, z\}$, where u and w are the source and sink of this boundary quadrangle respectively.

3 Computing 1-dimensional path homology H_1

Note that the 1-dimensional ∂-invariant path space $\Omega_1 = \Omega_1(G)$ is the space generated by all edges [10] because the boundary of every edge is allowed by definition.

Now consider the 1-cycle group $\mathsf{Z}_1 \subseteq \Omega_1$; that is, Z_1 is the kernel of ∂ applied to Ω_1. We show below that a basis of Z_1 can be computed by considering a spanning tree of the undirected version of G, which denoted by G_u. This is well known when $\mathbb{F}$ is $\mathbb{Z}_2$. It is easy to see that this spanning tree based construction also works for arbitrary field $\mathbb{F}$.

Specifically, let T be a rooted spanning tree of G_u with root r, and $\bar{T} := G_u \setminus T$. For every edge $e = (v_1, v_2) \in \bar{T}$, let c_e be the 1-cycle (under $\mathbb{Z}_2$) obtained by summing e and all edges on the paths π_1 and π_2 between v_1 and r, and v_2 and r respectively. The cycles $\{c_e, e \in \bar{T}\}$ form a basis of 1-cycle group of G_u under $\mathbb{Z}_2$ coefficient. Now for every such cycle c_e in G_u, we also have a cycle in $\Omega_1(G)$ containing same edges with c_e which are assigned a coefficient 1 or -1 depending on their orientations in G. We call this 1-cycle in $\Omega_1(G)$ also c_e. Then we have the following proposition, whose proof is in the full version.

▶ **Proposition 2.** *The cycles $\{c_e | e \in \bar{T}\}$ in $\Omega_1(G)$ form a basis for Z_1 under any coefficient field $\mathbb{F}$.*

Now, we show a relation between 1-dimensional homology, cycles, bigons, triangles and quadrangles. Recall that bi-gons, boundary triangles and boundary quadrangles are specific types of 1-dimensional boundaries with two, three or four vertices, respectively; see Section 2.1. The following theorem is similar to Proposition 2.9 from [11], where the statement there is under coefficient ring $\mathbb{Z}$. For completeness, we include the (rather similar) proof for our case in the full version.

▶ **Theorem 3.** *Let $G = (V, E)$ be a directed graph. Let Q denote the space generated by all boundary triangles, boundary quadrangles and bi-gons in G. Then we have $\mathsf{B}_1 = \mathsf{Q}$.*

▶ **Corollary 4.** *The 1-dimensional path homology group satisfies that $\mathsf{H}_1 = \mathsf{Z}_1/\mathsf{Q}$.*

3.1 A simple algorithm

Theorem 3 and Corollary 4 provide us a simple framework to compute H_1. Below we only focus on the computation of the rank of H_1; but the algorithm can easily be modified to output a basis for H_1 as well. Later in Section 4, we will develop a more efficient and sophisticated algorithm for the 1-dimensional *persistent* path homology H_1, which as a by-product, also gives a more efficient algorithm to compute H_1.

In the remaining of this paper, we represent each cycle in Z_1 with a vector. Assume all edges are indexed from 1 to m as $e_1, \cdots, e_m$ where m is the number of edges. Then, each 1-cycle C is an m-dimensional vector, where $C[i] \in \mathbb{F}$ records the coefficient for edge e_i in C.

(Step 1): cycle group Z_1. By Proposition 2, $rank(\mathsf{Z}_1) = |E| - |V| + 1$ for directed graph $G = (V, E)$. The computation of the rank takes $O(1)$ time (or $O(|V|^2)$ time if we need to output a basis of it explicitly).

Algorithm 1 A simple first algorithm to compute rank of H_1.

```
1: procedure CompH1-Simple($G$, $t$)
2:     (Step 1): Compute rank of 1-cycle group $\mathsf{Z}_1$
3:     (Step 2): Compute rank of 1-boundary group $\mathsf{B}_1$
4:         (Step 2.a) Compute a generating set $\mathsf{C}$ of 1-boundary cycles that generates $\mathsf{B}_1$
5:         (Step 2.b) From $\mathsf{C}$ compute a basis for $\mathsf{B}_1$
6:     Return $rank(\mathsf{H}_1) = rank(\mathsf{Z}_1) - rank(\mathsf{B}_1)$.
7: end procedure
```

(Step 2): boundary group B_1. Note that by Theorem 3, we can compute the set of all bigons, boundary triangles and boundary quadrangles as a *generating set* C of 1-boundary cycles (meaning that it generates the boundary group B_1) for (Step 2.a). However, such a set C could have size $\Omega(n^2)$ even for a planar graph, where $n = |V|$; see Figure 2. (For a general graph, the number of boundary quadrangles could be $\Theta(n^4)$.)

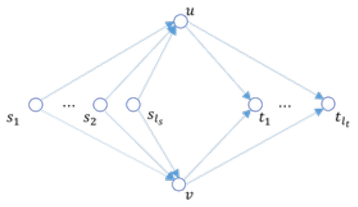


Figure 2 There are n vertices but $l_s \cdot l_t = \Theta(n^2)$ quadrangles, $l_s = \lfloor (n-2)/2 \rfloor$ and $l_t = \lceil (n-2)/2 \rceil$.

To make (Step 2.b) efficient, we wish to have a generating set C of 1-boundary cycles with *small cardinality*. To this end, we leverage a classical result of [3] to reduce the size of C.

Given an undirected graph G, its *arboricity* $\mathsf{a}(G)$ is the minimum number of edge-disjoint spanning forests which G can be decomposed into [13]. An alternative definition is

$$\mathsf{a}(G) = \max_{H \text{ is a subgraph of } G} \frac{|E(H)|}{|V(H)| - 1}.$$

From this definition, it is easy to see (and well-known, see e.g, [3]) that:

▶ **Observation 3.1.**
(1) *If G is a planar graph, or a graph with bounded vertex degrees, then* $\mathsf{a}(G) = O(1)$.
(2) *If G is a graph embedded on a genus g surface, then* $\mathsf{a}(G) = O(g)$.
(3) *In general, if G does not contain self-loops, then* $\mathsf{a}(G) = O(n)$.

We will leverage some classical results from [3]. First, to represent quadrangles, we use the following *triple-list* representation [13] : a triple-list $(u, v, \{w_1, w_2, \cdots, w_l\})$ means that for each i, w_i is adjacent to both u and v, where we say u' and v' are adjacent if either (u', v') or (v', u') are in E (i.e, u' and v' are adjacent when disregarding directions). Given such a triple-list $\xi = (u, v, \{w_1, w_2, \cdots, w_l\})$, it is easy to see that u, w_i, v, w_j form the *consecutive vertices* of a quadrangle in the undirected version of graph G; and we also say that the undirected quadrangle $R(u, w_i, v, w_j)$ is *covered* by this triple-list. We say that a vertex z is in a triple-list $(u, v, \{w_1, w_2, \cdots, w_l\})$ if it is in the set $\{w_1, w_2, \cdots, w_l\}$.

The size of a triple-list is the total number of vertices contained in it. This triple-list ξ thus represents $\Theta(l^2)$ number of undirected quadrangles in G succinctly with $\Theta(l)$ size.

▶ **Proposition 5** ([3]).

(1) *Let G be a connected undirected graph with n vertices and m edges. There is an algorithm listing all the triangles in G in $O(\mathsf{a}(G)m)$ time.*

(2) *There is an algorithm to compute a set of triple-lists which covers all quadrangles in a connected graph G in $O(\mathsf{a}(G)m)$ time. The total size complexity of all triple-lists is $O(\mathsf{a}(G)m)$.*

Using the above result, we can have the following theorem, with proof in the full version.

▶ **Theorem 6.** *Let $G = (V, E)$ be a directed graph with n vertices and m edges. We can compute a generating set C of 1-boundary cycles for B_1 with cardinality $O(\mathsf{a}(G)m)$ in time $O(\mathsf{a}(G)m)$.*

It then follows from Theorem 3 that (Step 2.a) can be implemented in $O(\mathsf{a}(G)m)$ time, producing a generating set of cardinality $O(\mathsf{a}(G)m)$. Finally, representing each boundary cycle in C as a vector of dimension $m = |E|$, we can then compute the rank of cycles in C in $O(|\mathsf{C}|m^{\omega-1}) = O(\mathsf{a}(G)m^{\omega})$,where $\omega < 2.373$ is the exponent for matrix multiplication [2].

Putting everything together, we have that

▶ **Theorem 7.** *Given a directed graph $G = (V, E)$ with $n = |V|$ and $m = |E|$, Algorithm 1 computes the rank of the 1-dimensional path homology group H_1 in $O(\mathsf{a}(G)m^{\omega})$ time.*

The algorithm can be extended to compute a basis for H_1 with the same time complexity.

For example, by Observation 3.1, if G is a planar graph, then we can compute H_1 in $O(n^{\omega})$. For a graph G embedded on a genus g surface, H_1 can be computed in $O(gn^{\omega})$ time. In contrast, we note that the algorithm of [5] takes $O(n^5)$ time for planar graphs.

4 Computing persistent path homology H_1

The concept of arboricity used in the previous section does not consider edge directions. Indeed, our algorithm to compute a generating set C as given in the proof of Theorem 6 first computes a (succinct) representation of all quadrangles, whether they contribute to boundary quadrangles or not. On the other hand, as Figure 3 illustrates, a graph G can have no boundary quadrangle despite the fact that the graph is dense (with $\Theta(n^2)$ edges and thus $\mathsf{a}(G) = \Theta(n)$ arboricity). Another way to view this is that the example has no allowed 2-path, and thus no ∂-invariant 2-paths and consequently no 1-boundary cycles. Our algorithm will be more efficient if it can also respect the number of allowed elementary 2-paths.

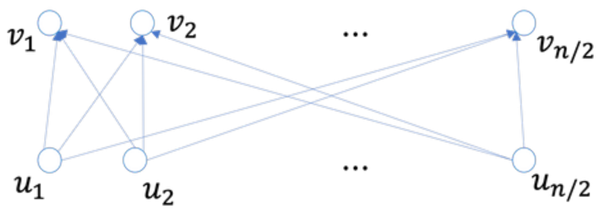


Figure 3 A dense graph with no boundary quadrangle.

In fact, a more standard and natural way to compute a basis for the 1-boundary group proceeds by taking the boundary of ∂-invariant 2-paths. The complication is that unlike in the simplicial homology case, it is not immediately evident how to compute a basis for Ω_2 (the space of ∂-invariant 2-paths). Nevertheless, Chowdhury and Mémoli presented an elegant algorithm to show that a basis for B_1 (and H_1) can still be computed using careful column-based matrix reductions [5]. The time complexity of their algorithm is $O((\sum_{(u,v)\in E}(d_{in}(u) + d_{out}(v)))mn^2)$ which depends on the number of elementary 2-paths[3].

In this section, we present an algorithm that can take advantage of both of the previous approaches (the algorithm of [5] and Algorithm 1). Similar to [5], we will now consider the persistent path homology setting, where we will add directed edges in G one by one incrementally. Hence our algorithm can compute the *persistent* H_1 w.r.t. a filtration. However different from [5], instead of reducing a matrix with columns corresponding to all elementary allowed 2-paths, we will follow a similar idea as in Algorithm 1 and add a generating set of boundary cycles each time we consider a new directed edge.

4.1 Persistent path homology

We now introduce the definition of the persistent path homology [5]. The *persistent vector space* is a family of vector spaces together with linear maps $\{U^\delta \xrightarrow{\mu_{\delta,\delta'}} U^{\delta'}_{\delta\le\delta'\in\mathbb{R}}\}$ so that: (1) $\mu_{\delta,\delta}$ is the identity for every $\delta \in \mathbb{R}$; and (2) $\mu_{\delta,\delta''} = \mu_{\delta,\delta'} \circ \mu_{\delta',\delta''}$ for each $\delta \le \delta' \le \delta'' \in \mathbb{R}$.

Let $G = (V, E, w)$ be a weighted directed graph where V is the vertex set, E is the edge set, and w is the weight function $w : E \to \mathbb{R}^+$. For every $\delta \in \mathbb{R}^+$, a directed graph G^δ can be constructed as $G^\delta = (V^\delta = V, E^\delta = \{e \in E : w(e) \le \delta\})$. This gives rise to a filtration of graphs $\{G^\delta \hookrightarrow G^{\delta'}\}_{\delta\le\delta'\in\mathbb{R}}$ using the natural inclusion map $i_{\delta,\delta'} : G^\delta \hookrightarrow G^{\delta'}$.

▶ **Definition 8** ([5]). *The* 1*-dimensional persistent path homology of a weighted directed graph* $G = (V, E, w)$ *is defined as the persistent vector space* $\mathbb{H}_1 := \{\mathsf{H}_1(G^\delta) \xrightarrow{i_{\delta,\delta'}} \mathsf{H}_1(G^{\delta'})\}_{\delta\le\delta'\in\mathbb{R}}$*. The* 1*-dimensional path persistence diagram* $Dg(G)$ *of* G *is the persistence diagram of* $\mathbb{H}_1$*.*

To compute the path homology $\mathsf{H}_1(G)$ of an unweighted directed graph $G = (V, E)$, we can order edges in E arbitrarily with the index of an edge in this order being its weight. The rank of $\mathsf{H}_1(G)$ can then be retrieved from the 1-dimensional persistent homology group induced by this filtration by considering only those homology classes that "never die".

4.2 A more efficient algorithm

In what follows, to simplify presentation, we assume that we are given a directed graph $G = (V, E)$, where edges are already sorted $e_1, \dots, e_m$ in increasing order of their weights. Let $G^{(i)} = (V, E^{(i)} = \{e_1, \dots, e_i\})$ denote the subgraph of G spanned by the edges $e_1, \dots, e_i$; and set $G^{(0)} = (V, \varnothing)$. We now present an algorithm to compute the 1-dimensional persistent path homology induced by the nesting sequence $G^{(0)} \subseteq G^{(1)} \subseteq \cdots G^{(m)}$. In particular, in Algorithm 2, as we insert each new edge e_s, moving from $G^{(s-1)}$ to $G^{(s)}$, we maintain a basis for $\mathsf{Z}_1(s) := \mathsf{Z}_1(G^{(s)})$ and for $\mathsf{B}_1(s) := \mathsf{B}_1(G^{(s)})$, updated from $\mathsf{Z}_1(s-1)$ and $\mathsf{B}_1(s-1)$ and output new persistent pairs. On a high level, this algorithm follows the standard procedure in [6]; details are in the full version.

4.2.1 Procedure GenSet(s)

Note that $G^{(s)}$ is obtained from $G^{(s-1)}$ by inserting a new edge $e_s = (u, v)$ to $G^{(s-1)}$. At this point, we have already maintained a basis B for $\mathsf{B}_1(G^{(s-1)})$. Our goal is to compute a set of generating boundary cycles C_s such that $B \cup \mathsf{C}_s$ contains a basis for $\mathsf{B}_1(G^{(s)})$.

[3] The time complexity given in the paper [5] assumes that the input directed graph is complete, and takes $O(n^9)$ to compute H_1. However, a more refined analysis of their time complexity shows that it can be improved to $O((\sum_{(u,v)\in E} d_{in}(u) + d_{out}(v))mn^2)$.

Algorithm 2 Compute 1-D persistent path homology for a directed graph $G = (V, E)$.

```
1: procedure Persistence($G$)
2:     Order the edges in non-decreasing order of their weights: $e_1, \dots, e_m$.
3:     Set $G^{(0)} = (V, \varnothing)$, current basis for 1-boundary group is $B = \varnothing$.
4:     for $s = 1$ to $m$ do
5:         Call GenSet($s$) to compute a generating set $\mathsf{C}_s$ containing a basis for newly
           generated 1-boundary cycles moving from $G^{(s-1)}$ to $G^{(s)}$.
6:         Call FindPairs($s$) to output new persistent pairs, and update the boundary basis
           $B$ for $G^{(s)}$.
7:     end for
8: end procedure
```

We first inspect the effect of adding edge $e_s = (u, v)$ to $G^{(s)}$. Two cases can happen:

Case-A: The endpoints u and v are in different connected components in (the undirected version of) $G^{(s-1)}$, and after adding e_s, those two components are merged into a single one in $G^{(s)}$. In this case, no cycle is created, nor does the boundary group change. Thus $\mathsf{Z}_1(G^{(s-1)}) = \mathsf{Z}_1(G^{(s)})$ and $\mathsf{B}_1(G^{(s-1)}) = \mathsf{B}_1(G^{(s)})$. We say that edge e_s is *negative* in this case (as it kills in H_0).
The algorithm maintains the set of negative edges seen so far, which is known to form a spanning forest T_s of V. (Here, we abuse the notation slightly and say that a set of directed edges span a tree for a set of vertices if they do so when directions are ignored.) The algorithm maintains T_s via a union-find data structure.

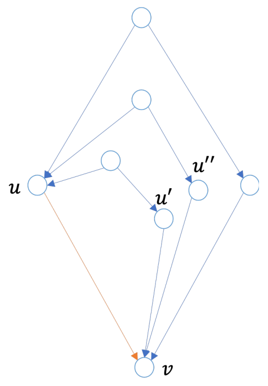


Figure 4 The insertion of edge (u, v) increases the rank of the boundary group by 3.

Case-B: The endpoints u and v are already in the same connected component in $G^{(s-1)}$. After adding this edge e_s, new cycles are created in $G^{(s)}$. Hence e_s is *positive* in this case (as it creates an element in Z_1; although different from the standard simplicial homology, it may not necessarily create an element in H_1 as we will see later).
Whether e_s is positive or negative can be easily determined by performing two Find operations in the union-find data structure representing T_{s-1}. A $\mathsf{Union}(u, v)$ operation is performed to update T_{s-1} to T_s if e_s is negative.

We now describe how to handle (Case-B). After adding edge e_s, multiple cycles containing e_s can be created in $G^{(s)}$. Nevertheless, by Proposition 2, the dimension of Z_1 increases only by 1. On the other hand, the addition of e_s may create new boundary cycles. Interestingly, it could increase the rank of B_1 by more than 1. See Figure 4 for an example where $rank(\mathsf{B}_1)$ increases by 3; and note that this number can be made arbitrarily large.

As mentioned earlier, in this case, we wish to compute a set of generating boundary cycles C_s such that $B \cup \mathsf{C}_s$ contains a basis for $\mathsf{B}_1(G^{(s)})$.

Similar to Algorithm 1, using Theorem 3, we choose some bigons, boundary triangles and boundary quadrangles and add them to C_s. In particular, since C_s only accounts for the newly created boundary cycles, we only need to consider bigons, boundary triangles and boundary quadrangles that contain e_s. We now describe the construction of C_s, which is initialized to be $\varnothing$.

(i) *Bigons.* At most one bigon can be created after adding e_s (namely, the one that contains e_s). We add it to C_s if this bigon exists.

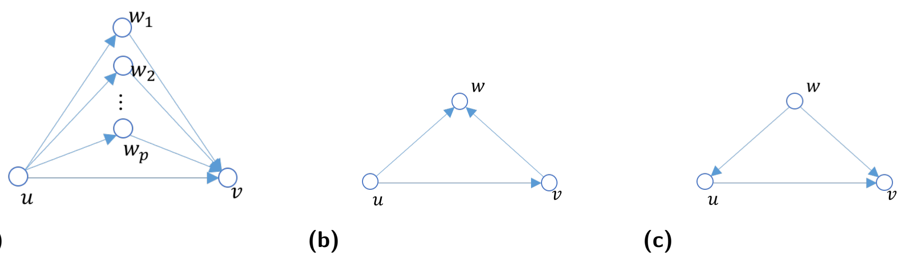


Figure 5 Three types of boundary triangles incident to $e_s = (u, v)$.

(ii) *Boundary triangles.* There could be three types of newly created boundary triangles containing $e_s = (u, v)$. The first case is when u is the source and v is the sink; see Figure 5(a). In this case multiple 2-paths may exist from u to v, $e_{uw_1v}, e_{uw_2v}, \cdots e_{uw_pv}$, forming multiple boundary triangles of this type containing e_s. However, we only need to add *one* triangle of them into C_s, say $(u, v \mid w_1)$ since every other triangle $(u, v \mid w_j)$ can be written as a linear combination of $(u, v \mid w_1)$ and an existing boundary quadrangle $(u, v \mid w_1, w_j)$ in $G^{(s-1)}$.
For the second case (see Figure 5(b)) where u is the source but v is not the sink, we include all such boundary triangles to C_s. We also add all boundary triangles of the last type in which v is the sink but u is not the source to C_s; see Figure 5(c). It is easy to see that $\mathsf{C}_s \cup B$ can generate all new boundary triangles containing $e_s = (u, v)$.

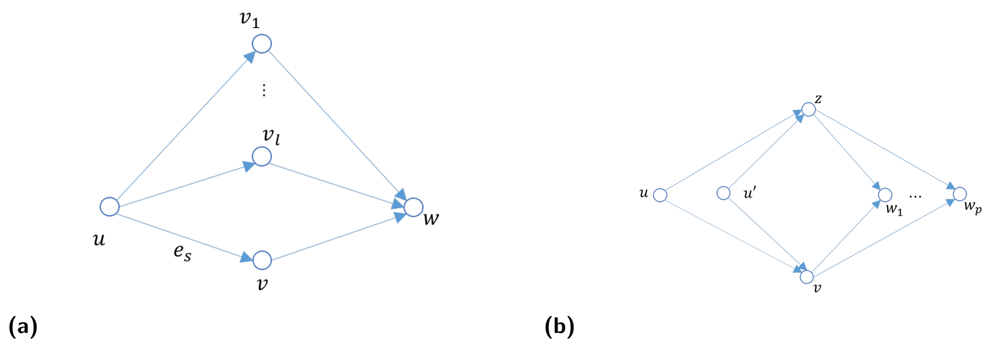


Figure 6 (a) Examples of new boundary quadrangles with u being the source. (b) Not all boundary quadrangles in M will be added to the generating set C_s.

(iii) *Boundary quadrangles.* Given an edge $e_s = (u, v)$, there are two types of the boundary quadrangles incident to it: one has u as the source; the other has v as the sink. We focus on the first case; see Figure 6(a). The second case can be handled symmetrically.

In particular, we will first compute a set M and then select a subset of quadrangles from M for adding to C_s.

Specifically, take any successor w of v, that is, there is an edge $(v, w) \in G^{(s-1)}$ forming an allowed 2-path e_{uvw} in $G^{(s)}$. Before introducing the edge e_s, there may be multiple allowed 2-paths $e_{uv_1w}, e_{uv_2w}, \cdots, e_{uv_lw}$ in $G^{(s-1)}$; see Figure 6 (a). For each such 2-path $e_{uv_kw}, 1 \leq k \leq l$, a new boundary quadrangle $(u, w \mid v, v_k)$ containing $e_s = (u, v)$ will be created. However, among all such 2-paths $e_{uv_1w}, \cdots, e_{uv_lw}$, we will pick just one 2-path, say e_{uv_1w} and only add the quadrangle $(u, w \mid v, v_1)$ formed by e_{uvw} and e_{uv_1w} to M. Observe that any other boundary quadrangle containing 2-path e_{uvw}, say $(u, w \mid v, v_k)$, can be written as a linear combination of the quadrangle $(u, w \mid v, v_1)$ and boundary quadrangle $(u, w \mid v_k, v_1)$ which is already in $G^{(s-1)}$ (and in the span of B which is a basis for $\mathsf{B}_1(G^{(s-1)})$). In other words, $(u, w \mid v, v_1) \cup B$ generates any other boundary quadrangle containing 2-path e_{uvw}.

We perform this for each successor w of v. Hence this step adds at most $d_{out}^{G^{(s-1)}}(v)$ number of boundary quadrangles to the set M.

Not all quadrangles in M will be added to C_s. In particular, suppose we have p quadrangles $A = \{(u, w_j \mid v, z) : 1 \leq j \leq p\} \subseteq M$ incident to the newly inserted edge $e_s = (u, v)$ as well as another vertex z, i.e. there are edges $(u, z), (w_j, z)$ and (v, w_j), $1 \leq j \leq p$; see Figure 6 (b). If there does not exist any other vertex u' such that edges $(u', z), (u', v) \in G^{(s-1)}$, then we add *all* quadrangles in A to C_s. If this is not the case, let u' be another vertex such that (u', z) and (u', v) are already in $G^{(s-1)}$; see Figure 6 (b). In this case, we only add *one* quadrangle from set A, say, $(u, w_1 | v, z)$ to the generating set C_s.

It is easy to check that any other quadrangle $(u, w_j \mid v, z)$, $1 < j \leq p$, can be written as the combination of $(u, w_1 \mid v, z)$, $(u', w_1 \mid v, z)$ and $(u', w_j \mid v, z)$. As the latter two quadrangles are boundary quadrangles from $G^{(s-1)}$, they can already be generated by B. The entire process takes time $O(|M|) = O(d_{out}^{G^{(s-1)}}(v))$. It is also easy to see that $B \cup \mathsf{C}_s$ can generate any boundary quadrangle containing $e_s = (u, v)$ and with u being its source.

The case when v is the sink of a boundary quadrangle is handled symmetrically in time $O(d_{in}^{G^{(s-1)}}(u))$. Hence the total time to compute a generating set C_s is $O(d_{in}^{G^{(s-1)}}(u) + d_{out}^{G^{(s)}}(v))$ when inserting a single edge $e_s = (u, v)$.

4.3 Analysis of Algorithm 2

Correctness of the algorithm. Notice that the invariant that B is a basis for $G^{(s)}$ at the end of the for-loop (line-7 of Algorithm 2) is maintained. Furthermore, B is always in reduced form which is maintained via left-to-right column additions only. Hence the algorithm computes the 1-dimensional persistent path homology correctly [6].

Time complexity analysis. The remainder of this section is devoted to determining the time complexity of Algorithm 2. Specifically, we first show the following theorem.

▶ **Theorem 9.** *Across all stages $s \in [1, m]$, the total cardinality of the generating set $\mathsf{C} = \cup_s \mathsf{C}_s$ is $O(\min\{\mathsf{a}(G)m, \sum_{(u,v)\in E}(d_{in}(u) + d_{out}(v))\})$. The total time taken by procedure* NewBasis*(s) for all $s \in [1, m]$ is $O(m + \sum_{(u,v)\in E}(d_{in}(u) + d_{out}(v))\})$.*

Proof. We will count separately the number of bigons, boundary triangles, and boundary quadrangles added to any C_s. Set $r = \min\{\mathsf{a}(G)m, \sum_{(u,v)\in E}(d_{in}(u) + d_{out}(v))\}$.

(i) *Bigons:* First, it is easy to see that for each edge $e_s = (u, v)$ with $s \in [1, m]$, at most one bigon (incident to e_s) is added. Besides, if $d_{out}(v) = 0$, there is no bigon incident to e_s. Hence the total number ever added to C is $O(\min\{m, \sum_{(u,v)\in E}(d_{out}(v))\}) = O(r)$ and it takes $O(m)$ time to compute them.

(ii) *Boundary triangles:* For boundary triangles, we know from Proposition 5 that there are altogether $O(\mathsf{a}(G)m)$ triangles (thus at most $O(\mathsf{a}(G)m)$ boundary triangles) in a graph G and they can all be enumerated in $O(\mathsf{a}(G)m)$ time. Obviously, the number of boundary triangles ever added to C is at most $O(\mathsf{a}(G)m)$.
We now argue that the number of boundary triangles added to C is also bounded by $O(\sum_{(u,v)\in E}(d_{in}(u)+d_{out}(v)))$. Note that, for every 2-path, at most one boundary triangle is added to the set. Since the number of 2-paths is indeed $\Theta(\sum_{(u,v)\in E}(d_{in}(u)+d_{out}(v)))$, the number of triangles we add is $O(\sum_{(u,v)\in E}(d_{in}(u)+d_{out}(v)))$. Recall that there are three cases for boundary triangles added; see Figure 5. The time spent for the first case for every s is $O(1)$by recording any 2-path e_{uwv}, and $O(d_{in}(u)+d_{out}(v))$ for the last two cases. Thus the total time spent at adding boundary triangles incident to e_s and identifying triangles to be added to C_s for all $s\in[1,m]$ takes $O(m+\sum_{(u,v)\in E}(d_{in}(u)+d_{out}(v)))$ time.

(iii) *Boundary quadrangles:* The situation here is somewhat opposite to that of the boundary triangles: Specifically, it is easy to see that this step accesses at most $O(d_{in}(u)+d_{out}(v))$ boundary quadrangles when handling edge $e_s=(u,v)$. Hence the number of boundary quadrangles it can add to C_s is at most $O(d_{in}(u)+d_{out}(v))$. The total number of boundary quadrangles ever added to C is thus bounded by $O(\sum_{(u,v)\in E}(d_{in}(u)+d_{out}(v)))$.
We now prove that the number of boundary quadrangles ever added to C is also bounded by $O(\mathsf{a}(G)m)$. We use the existence of a succinct representation of all quadrangles as specified in Proposition 5 to help us argue this upper bound. Notice that our algorithm **does not** compute this representation. It is only used to provide this complexity analysis.

Specifically, by Proposition 5, we can compute a list L of triple-lists with $O(\mathsf{a}(G)m)$ total size complexity, which generates all undirected quadrangles. Following the proof of Theorem 6(see the full version), we can further refine this list, where each triple-list $\xi\in L$ further gives rise to three lists that are of type-1, 2, or 3. Let $\widehat{L}$ denote this refinement of L, consisting of lists of type-1, 2 or 3. From the proof of Theorem 6, we know that the total size complexity for all lists in $\widehat{L}$ is still $O(\mathsf{a}(G)m)$. This also implies that the cardinality of $\widehat{L}$ is bounded by $|\widehat{L}|=O(\mathsf{a}(G)m)$.

We now denote by $\mathcal{R}$ the set of all boundary quadrangles ever added to $\mathsf{C}=\cup_s\mathsf{C}_s$ by Algorithm 2. Furthermore, let

$$\mathsf{P}:=\{(\xi,w)\mid \xi\in\widehat{L}, w\in\xi\}.$$

Below we show that we can find an injective map $\pi:\mathcal{R}\to\mathsf{P}$. But first, note that $|\mathsf{P}|$ is proportional to the total size complexity of $\widehat{L}$ and thus is bounded by $O(\mathsf{a}(G)m)$.

We now establish the injective map $\pi:\mathcal{R}\to\mathsf{P}$. Specifically, we process each boundary quadrangle *in the order* that they are added to C. Consider a boundary quadrangle $R=R(u,v,w,z)$ added to C_s while processing edge $e_s=(u,v)$. There are two cases: The first is that R is of the form $(u,w\mid v,z)$ in which u is the source of this quadrangle. The second is that it has the form $(w,v\mid u,z)$ in which v is the sink. We describe the map $\pi(R)$ for the first case, and the second one can be analyzed symmetrically.

By construction of L, there is at least one triple-list $\xi\in L$ covering $R=(u,w\mid v,z)$. There are three possibilities:

Case-a: The triple-list ξ is of the form $\xi=\xi_{uw}=(u,w,\{\cdots\})$. In this case, the boundary quadrangle $R=(u,w\mid v,z)$ is in a type-1 list $\xi^{(1)}_{uw}=(u,w,S)\in\widehat{L}$, and both $v,z\in S$. We now claim that the pair $(\xi^{(1)}_{uw},v)\in\mathsf{P}$ has not yet been mapped (i.e, there is no $R'\in C$ with

$\pi(R') = (\xi^{(1)}_{uw}, v)$ yet), and we can thus set $\pi(R) = (\xi^{(1)}_{uw}, v) \in \mathsf{P}$. Suppose on the contrary there already exists $R' \in C$ that we processed earlier than R with $\pi(R') = (\xi^{(1)}_{uw}, v)$. In that case, $R' = (u, w \mid v, z')$ must contain the 2-path e_{uvw} as well. Since R' is processed earlier than R, and edge $e_s = (u, v)$ is the most recent edge added, R' must be added when we process e_s as well (as R' contains e_s). However, Algorithm 2 in this case only adds *one* quadrangle containing the 2-path e_{uvw}, meaning that R' cannot exist (as otherwise, we would not have added R to C_s; recall Figure 6 (a)). Hence, the map π so far remains injective.

Case-b: The triple-list ξ is of the form $\xi = \xi_{wu} = (w, u, \{\cdots\})$. In this case, this quadrangle is covered by the type-2 list $\xi^{(2)}_{wu} \in \widehat{L}$. We handle this in a manner symmetric to (Case-a) and map $\pi(R) = (\xi^{(2)}_{wu}, v)$.

Case-c: The last case is that R is generated by triple-list ξ of the form $\xi_{vz} = (v, z, \{\cdots\})$. In this case, the quadrangle $R = (u, w \mid v, z)$ will be covered by the type-3 list $\xi^{(3)}_{vz} = (v, z, S_1, S_2)$ with $u \in S_1$ and $w \in S_2$; see Figure 7. We now argue that at least one of $(\xi^{(3)}_{vz}, u)$ and $(\xi^{(3)}_{vz}, w)$ has *not* been mapped under π yet.

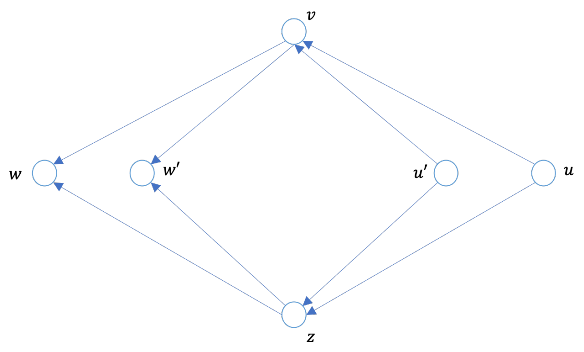


Figure 7 At least one of $(\xi^{(3)}_{vz}, u)$ and $(\xi^{(3)}_{vz}, w)$ has not been mapped yet.

Suppose this is not the case and we already have both $\pi(Q_1) = (\xi^{(3)}_{vz}, u)$ and $\pi(Q_2) = (\xi^{(3)}_{vz}, w)$. Then Q_1 is necessarily of the form $(u, w' \mid v, z)$ and Q_2 is of the form $(u', w \mid v, z)$; and both Q_1 and Q_2 are processed before R. See Figure 7. Furthermore, Q_1 is only added when we process edge e_s. However, in this case, once Q_1 is added, Algorithm 2 will not add further quadrangle containing edges (u, v) and (u, z) (recall the handling of Figure 6 (b)). Hence R cannot be added to C_s in this case.

In other words, it cannot be that both Q_1 and Q_2 already exist, and hence we can set $\pi(R)$ to be one of $(\xi^{(3)}_{vz}, u)$ and $(\xi^{(3)}_{vz}, w)$ that is not yet mapped. Consequently, the map π we construct remains injective.

We process all quadrangles in C in order. The final $\pi : \mathcal{R} \to \mathsf{P}$ is injective, meaning that $|\mathcal{R}| \le |\mathsf{P}|$ and thus $|\mathcal{R}| = O(\mathsf{a}(G)m)$.

Putting everything together, we have that the total number of boundary quadrangles added to C is bounded by $O(\min\{\mathsf{a}(G)m, \sum_{(u,v)\in E}(d_{in}(u) + d_{out}(v))\})$.

Finally, Algorithm 2 spends $O(m + \sum_{(u,v)\in E}(d_{in}(u) + d_{out}(v)))$ time to handle both cases in Figure 6. The theorem then follows. ◀

Combined with some standard matrix operations, the above theorem gives Theorem 1. The details of the proof are given in the full version of the paper.

▶ Remark 10. We note that neither term in $r = \min\{\mathsf{a}(G)m, \sum_{(u,v)\in E}(d_{in}(u) + d_{out}(v))\}$ always dominates. In particular, it is easy to find examples where one term is significantly smaller (asymptotically) than the other. For example, for any planar graph G, $\mathsf{a}(G)m = O(n)$. However, it is easy to have a planar graph where the second term $\sum_{(u,v)\in E}(d_{in}(u)+d_{out}(v)) = \Omega(n^2)$; see e.g, Figure 2.

On the other hand, it is also easy to have a graph G where $\sum_{(u,v)\in E}(d_{in}(u) + d_{out}(v)) = O(1)$ yet $\mathsf{a}(G)m = \Theta(n^3)$. Indeed, consider the bipartite graph in Figure 3, where for each edge $(u, v) \in E$, $d_{in}(u) + d_{out}(v) = 0$. However, this graph has $\mathsf{a}(G) = \Theta(n)$, $m = \Theta(n^2)$ and thus $\mathsf{a}(G)m = \Theta(n^3)$.

▶ Remark 11. We note that the time complexity of the algorithm proposed by Chowdhury and Mémoli in [5] to compute the $(d-1)$-dimensional persistence path homology takes $O(n^{3+3d})$ time. However, for the case $d = 2$, a more refined analysis shows that in fact, their algorithm takes only $O((\sum_{(u,v)\in E}(d_{in}(u) + d_{out}(v)))mn^2)$ time.

Compared with our algorithm, which takes time $O(rm^{\omega-1})$ with $r = \min\{\mathsf{a}(G)m, \sum_{(u,v)\in E}(d_{in}(u) + d_{out}(v))\}$ and $\omega < 2.373$, observe that our algorithm can be significantly faster (when $\mathsf{a}(G)m$ is much smaller than $\sum_{(u,v)\in E}(d_{in}(u)+d_{out}(v))$. For example, for planar graphs, our algorithm takes $O(n^\omega)$ time, whereas the algorithm of [5] takes $O(n^5)$ time.

Finally, in the full version of the paper, we extend our algorithm to compute the so-called *minimal path homology basis* efficiently, and provide some preliminary experimental results, including showing the efficiency of our algorithm compared to the previous best algorithm over several datasets.

5 Concluding remarks

A natural question is whether it is possible to have a more efficient algorithm for computing (persistent) path homology of higher dimensions improving the work of [5]. Another question is whether we can compute a minimal path homology basis faster improving our current time bound $O(m^\omega n)$ (see the full version of the paper).

References

1 Wei Chen, Yajun Wang, and Siyu Yang. Efficient influence maximization in social networks. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 199–208. ACM, 2009.

2 Ho Yee Cheung, Tsz Chiu Kwok, and Lap Chi Lau. Fast matrix rank algorithms and applications. *Journal of the ACM (JACM)*, 60(5):31, 2013.

3 Norishige Chiba and Takao Nishizeki. Arboricity and subgraph listing algorithms. *SIAM Journal on Computing*, 14(1):210–223, 1985.

4 Samir Chowdhury and Facundo Mémoli. A functorial dowker theorem and persistent homology of asymmetric networks. *Journal of Applied and Computational Topology*, 2(1-2):115–175, 2018.

5 Samir Chowdhury and Facundo Mémoli. Persistent path homology of directed networks. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1152–1169. SIAM, 2018.

6 David Cohen-Steiner, Herbert Edelsbrunner, and Dmitriy Morozov. Vines and vineyards by updating persistence in linear time. In *Proceedings of the twenty-second annual symposium on Computational geometry*, pages 119–126. ACM, 2006.

7 Tamal K Dey, Tianqi Li, and Yusu Wang. Efficient algorithms for computing a minimal homology basis. In *Latin American Symposium on Theoretical Informatics*, pages 376–398, 2018.

8 Pawel Dlotko, Kathryn Hess, Ran Levi, Max Nolte, Michael Reimann, Martina Scolamiero, Katharine Turner, Eilif Muller, and Henry Markram. Topological analysis of the connectome of digital reconstructions of neural microcircuits. *arXiv preprint arXiv:1601.01580*, 2016.

9 Jeff Erickson and Kim Whittlesey. Greedy optimal homotopy and homology generators. In *Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1038–1046. Society for Industrial and Applied Mathematics, 2005.

10 Alexander Grigor'yan, Yong Lin, Yuri Muranov, and Shing-Tung Yau. Homologies of path complexes and digraphs. *arXiv preprint arXiv:1207.2834*, 2012.

11 Alexander Grigor'yan, Yong Lin, Yuri Muranov, and Shing-Tung Yau. Homotopy theory for digraphs. *arXiv preprint arXiv:1407.0234*, 2014.

12 Alexander Grigor'yan, Yong Lin, Yuri Muranov, and Shing-Tung Yau. Cohomology of digraphs and (undirected) graphs. *Asian J. Math*, 19(5):887–931, 2015.

13 F. Harary. *Graph Theory*. Addison Wesley series in mathematics. Addison-Wesley, 1971. URL: `https://books.google.com/books?id=q8OWtwEACAAJ`.

14 Paolo Masulli and Alessandro EP Villa. The topology of the directed clique complex as a network invariant. *SpringerPlus*, 5(1):388, 2016.

15 Ron Milo, Shai Shen-Orr, Shalev Itzkovitz, Nadav Kashtan, Dmitri Chklovskii, and Uri Alon. Network motifs: simple building blocks of complex networks. *Science*, 298(5594):824–827, 2002.

16 Lav R Varshney, Beth L Chen, Eric Paniagua, David H Hall, and Dmitri B Chklovskii. Structural properties of the caenorhabditis elegans neuronal network. *PLoS computational biology*, 7(2):e1001066, 2011.