Link Prediction with Persistent Homology: An Interactive View – Supplementary Materials –

Zuoyu Yan [1] **Tengfei Ma** [2] **Liangcai Gao** [1] **Zhi Tang** [1] **Chao Chen** [3]

In this supplemental material, we provide (1) additional details for persistence image; (2) additional examples to illustrate the matrix reduction algorithm for extended persistent homology; (3) a complete proof of the correctness of the proposed faster algorithm; and (4) additional experimental details and qualitative results.

1. Persistence Image

In recent years, efforts have been made to map persistence diagrams into representations valuable to machine learning tasks. Persistence Image (Adams et al., 2017) is one such approach to convert persistence diagrams to vectors, which will be used in our model. Let $T : \mathbf{R}^2 \to \mathbf{R}^2$ be a linear transformation $T(x, y) = (x, y - x)$ of persistence points. Given a persistence diagram D, $T(D) = \{T(d)|d \in D\}$ is the transformed diagram. For any $z \in \mathbf{R}^2$, $\phi_u(z) = \frac{1}{2\pi\sigma^2}e^{-\frac{||z-u||^2}{2\sigma^2}}$ is the 2D Gaussian function with mean u and standard deviation σ.

Let $\alpha : \mathbf{R}^2 \to \mathbf{R}$ be a non-negative weight function for the persistence plane $\mathbf{R}^2$. Given a persistence diagram DgX, its persistence surface is defined as: $\rho_D(z) = \sum_{u \in T(D)} \alpha(u)\phi_u(z)$. Fix a grid in the plane with n pixels, the persistence image is the collection of pixels $PI_D = \{PI_D[p]\} \in \mathbf{R}^n$ where $PI_D[p] = \int\int_p \rho_D(x, y)dxdy$, thus can be directly used in machine learning tasks. The stability of persistence image under perturbation has been proven in (Adams et al., 2017). In our setting, α is a piecewise linear weighting function:

$$\alpha(x, y) = \begin{cases} 0 & \text{if } y \leq 0 \\ y & \text{if } 0 < y \leq 1 \\ 1 & \text{if } y > 1 \end{cases}.$$

2. Examples of the Matrix Reduction Algorithm

The reduction matrix M for Figure 1 is shown in Figure 2. Recall that M is a binary valued matrix to encode the adjacency relationship between nodes and edges. M is a $2m \times 2m$ matrix consisting of four $m \times m$ matrices: $M = \begin{bmatrix} A & P \\ 0 & D \end{bmatrix}$. Every column or row of M corresponds to a simplex. In particular, the first m columns of M correspond to the ascending sequence of simplices $\kappa_1, ..., \kappa_m$. The last m columns of M correspond to the descending sequence of simplices $\lambda_1, ..., \lambda_m$. The setting is the same for the rows of M. Matrix A encodes the relationship between all the simplices in the ascending sequence. Similar to the incidence matrix of a graph, $A[i, j] = 1$ iff κ_i is the boundary of κ_j, i.e., κ_i is a node adjacent to the edge κ_j.

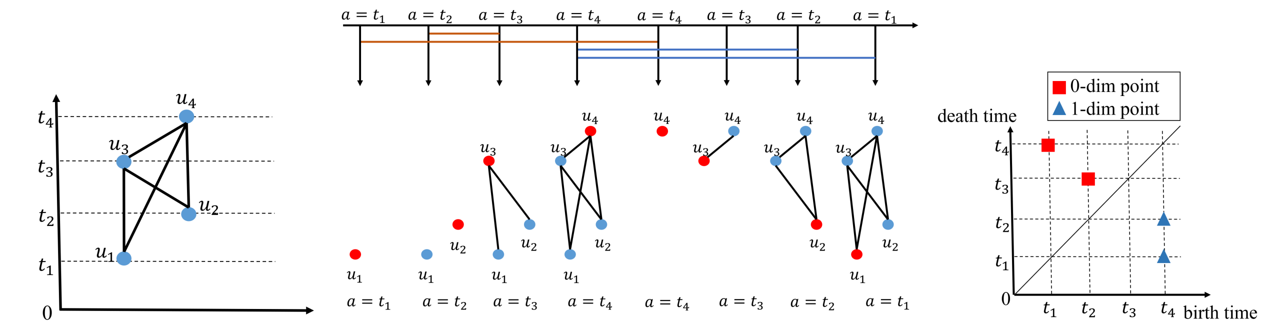
D is defined similarly, except that it encodes the relationship of the simplices in the descending sequence, i.e., $D[i, j] = 1$ iff λ_i is the boundary of λ_j. P stores the permutation that connects the two sequences of simplices, i.e., $P[i, j] = 1$ iff κ_i and λ_j denote the same simplex. 0 is a zero-valued $m \times m$ matrix.

Algorithm 1 reduces the columns of matrix M from left to right. If we allow certain flexibility in the reduction ordering, we can separate the algorithm into 3 phases: the reduction of matrix A (Phase 1), the reduction of matrix D (Phase 2) and the reduction of matrix P (Phase 3) (Cohen-Steiner et al., 2009). We define a simplex as positive if its corresponding column is zero after reduction, and negative if its corresponding column is not zero after reduction. In our setting, all the nodes, as well as edges that give rise to a loop are positive simplices, edges that destroy a connected component are negative simplices. All the simplices are either negative or positive (Edelsbrunner et al., 2000; Edelsbrunner & Harer, 2010). Notice that the positive and negative edges in the ascending filtration are not the same as the positive and negative edges in the descending filtration.

In the following paragraph, we will introduce the whole process of the matrix reduction algorithm by introducing the 3 phases successively.

[1]Wangxuan Institute of Computer Technology, Peking University, Beijing, China [2]T. J. Watson Research Center, IBM, New York, USA [3]Department of Biomedical Informatics, Stony Brook University, New York, USA. Correspondence to: Chao Chen <chao.chen.1@stonybrook.edu>, Liangcai Gao <glc@pku.edu.cn>.

Proceedings of the 38th International Conference on Machine Learning, PMLR 139, 2021. Copyright 2021 by the author(s).



(a) (b) (c)

Figure 1. An illustration of extended persistent homology. (a) We plot the input graph with a given filter function. The filter value for each node is $f(u_1) = t_1$, $f(u_2) = t_2$, $f(u_3) = t_3$, $f(u_4) = t_4$. (b) The ascending and descending filtrations of the input graph. The bars of brown and blue colors correspond to the life spans of connected components and loops respectively. The first four figures are the ascending filtration, while the last four figures denote the descending filtration. In the ascending filtration, $f(uv) = max(f(u), f(v))$, while in the descending filtration, $f(uv) = min(f(u), f(v))$. (c) In the resulting extended persistence diagram, red and blue markers correspond to 0-dimensional and 1-dimensional topological structures. There are two blue markers, corresponding to two loops (u_1u_3, u_3u_4, u_4u_1), (u_2u_3, u_3u_4, u_4u_2). The range of filter function f for these two loops are $[t_1, t_4]$, $[t_2, t_4]$ respectively. These ranges are encoded as the coordinates of the blue markers.

$$\begin{array}{cccccccccc}
A & u_1 & u_2 & u_3 & u_1u_3 & u_2u_3 & u_4 & u_1u_4 & u_2u_4 & u_3u_4 \\
u_1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\
u_2 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\
u_3 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 \\
u_1u_3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
u_2u_3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
u_4 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\
u_1u_4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
u_2u_4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
u_3u_4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{array}$$

(a)

$$\begin{array}{cccccccccc}
P & u_4 & u_3 & u_4u_3 & u_2 & u_4u_2 & u_3u_2 & u_1 & u_4u_1 & u_3u_1 \\
u_1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
u_2 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
u_3 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
u_1u_3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
u_2u_3 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
u_4 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
u_1u_4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
u_2u_4 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
u_3u_4 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0
\end{array}$$

(b)

$$\begin{array}{cccccccccc}
0 & u_1 & u_2 & u_3 & u_1u_3 & u_2u_3 & u_4 & u_1u_4 & u_2u_4 & u_3u_4 \\
u_4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
u_3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
u_4u_3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
u_2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
u_4u_2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
u_3u_2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
u_1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
u_4u_1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
u_3u_1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{array}$$

(c)

$$\begin{array}{cccccccccc}
D & u_4 & u_3 & u_4u_3 & u_2 & u_4u_2 & u_3u_2 & u_1 & u_4u_1 & u_3u_1 \\
u_4 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\
u_3 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\
u_4u_3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
u_2 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\
u_4u_2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
u_3u_2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
u_1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\
u_4u_1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
u_3u_1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{array}$$

(d)

Figure 2. Reduction matrix M for Figure 1

2.1. Phase 1

Phase 1 is the matrix reduction for A. All the columns of nodes and all the rows of edges are all zero in A, therefore they will have no impact on the matrix reduction algorithm. After deleting these rows and columns, matrix A is shown below:

$$\begin{array}{cccccc} A & u_1u_3 & u_2u_3 & u_1u_4 & u_2u_4 & u_3u_4 \\ u_1 & 1 & 0 & 1 & 0 & 0 \\ u_2 & 0 & 1 & 0 & 1 & 0 \\ u_3 & 1 & 1 & 0 & 0 & 1 \\ u_4 & 0 & 0 & 1 & 1 & 1 \end{array}$$

For simplicity, we define $low_A(e_i)$ as the maximum row of node v_j for which $A[v_j, e_i] = 1$. Notice that low_A is originally defined for the row index, and we replace the index with the simplex it represents. i.e., we replace $A[1, 1] = 1$ with $A[u_1, u_1u_3] = 1$, and we replace $low_A(1) = 3$ with $low_A(u_1u_3) = u_3$.

From left to right, we can find that $low_A(u_1u_3) = u_3$, and $low_A(u_2u_3) = u_3 = low_A(u_1u_3)$, thus we add column of u_1u_3 to column of u_2u_3: $[1, 0, 1, 0] + [0, 1, 1, 0] = [1, 1, 0, 0]$. Notice that "add" here means the mod-2 sum of the two binary vectors. Thus $low_A(u_2u_3) = u_2$.

Then we can find that $low_A(u_1u_4) = u_4$, $low_A(u_2u_4) = u_4 = low_A(u_1u_4)$, we add the column of u_1u_4 to the column of u_2u_4: $[1, 0, 0, 1] + [0, 1, 0, 1] = [1, 1, 0, 0]$, thus $low_A(u_2u_4) = u_2 = low_A(u_2u_3)$. Again we add the column of u_2u_3 to the column of u_2u_4: $[1, 1, 0, 0] + [1, 1, 0, 0] = [0, 0, 0, 0]$. Notice here the column value for u_2u_3 is the value after matrix reduction. Therefore u_2u_4 is not paired.

Similarly, we add the column of u_1u_4 and column of u_1u_3 to column of u_3u_4 and get $[0, 0, 0, 0]$. Then u_3u_4 is not paired. After Phase 1, matrix A is shown below. And we can obtain the persistence pair: (u_3, u_1u_3), (u_2, u_2u_3), (u_4, u_1u_4).

$$\begin{array}{cccccc} A & u_1u_3 & u_2u_3 & u_1u_4 & u_2u_4 & u_3u_4 \\ u_1 & 1 & 1 & 1 & 0 & 0 \\ u_2 & 0 & 1 & 0 & 0 & 0 \\ u_3 & 1 & 0 & 0 & 0 & 0 \\ u_4 & 0 & 0 & 1 & 0 & 0 \end{array}$$

Recall that A encodes the relationship between all the simplices in the ascending sequence. In the ascending sequence, the filter function for an edge u_iu_j is defined as $f_a(u_iu_j) = max(f(u_i), f(u_j))$. Thus we can infer the persistence points from the persistence pairs: (t_3, t_3), (t_2, t_3), (t_4, t_4). We remove the persistence points whose birth time and death time are the same, and get the final persistence point: (t_2, t_3).

Recall that a simplex is defined as positive if its corresponding column is zero after reduction, and negative if its corresponding column is not zero after reduction. So the positive edges in the ascending sequence are u_2u_4 and u_3u_4, the negative edges in the ascending sequence are u_1u_3, u_2u_3, and u_1u_4.

2.2. Phase 2

Phase 2 is the matrix reduction for D. Notice that we define low_D similar to low_A, and Phase 2 influences not only D but also P (Cohen-Steiner et al., 2009). All the columns of nodes and all the rows of edges are all zero in D. In P, all the columns of nodes and all the rows of nodes have no impact to the reduction process and pairing for 1-dimensional topology. Therefore for simplicity, we delete these rows and edges and obtain the condensed matrices P and D:

$$\begin{array}{cccccc} P & u_4u_3 & u_4u_2 & u_3u_2 & u_4u_1 & u_3u_1 \\ u_1u_3 & 0 & 0 & 0 & 0 & 1 \\ u_2u_3 & 0 & 0 & 1 & 0 & 0 \\ u_1u_4 & 0 & 0 & 0 & 1 & 0 \\ u_2u_4 & 0 & 1 & 0 & 0 & 0 \\ u_3u_4 & 1 & 0 & 0 & 0 & 0 \end{array}$$

$$\begin{array}{cccccc} D & u_4u_3 & u_4u_2 & u_3u_2 & u_4u_1 & u_3u_1 \\ u_4 & 1 & 1 & 0 & 1 & 0 \\ u_3 & 1 & 0 & 1 & 0 & 1 \\ u_2 & 0 & 1 & 1 & 0 & 0 \\ u_1 & 0 & 0 & 0 & 1 & 1 \end{array}$$

From left to right, $low_D(u_4u_3) = u_3$, $low_D(u_4u_2) = u_2$, $low_D(u_3u_2) = u_2 = low_D(u_4u_2)$. We add the column of u_4u_2 to the column of u_3u_2: $[1, 0, 1, 0] + [0, 1, 1, 0] = [1, 1, 0, 0]$. Then $low_D(u_3u_2) = u_3 = low_D(u_4u_3)$, and we add column of u_4u_3 to column of u_3u_2: $[1, 1, 0, 0] + [1, 1, 0, 0] = [0, 0, 0, 0]$. Therefore u_3u_2 is not paired. Notice that when we add column of u_4u_2 and column of u_4u_3 to column of u_3u_2 in D, we are also adding these columns in P. Consequently the column of u_3u_2 in P will become: $[0, 0, 0, 1, 0] + [0, 0, 0, 0, 1] + [0, 1, 0, 0, 0] = [0, 1, 0, 1, 1]$.

Then we continue the matrix reduction algorithm, $low_D(u_4u_1) = u_1$, and $low_D(u_3u_1) = u_1 = low_D(u_4u_1)$. Then we add column of u_4u_1 to column of u_3u_1: $[1, 0, 0, 1] + [0, 1, 0, 1] = [1, 1, 0, 0]$, $low_D(u_3u_1) = u_3 = low_D(u_4u_3)$. Again we add column of u_4u_3 to column of u_3u_1: $[1, 1, 0, 0] + [1, 1, 0, 0] = [0, 0, 0, 0]$. Therefore u_3u_1 is not paired, and column of u_3u_1 in P becomes $[1, 0, 1, 0, 1]$. Matrix D and P will therefore become:

$$\begin{array}{cccccc} P & u_4u_3 & u_4u_2 & u_3u_2 & u_4u_1 & u_3u_1 \\ u_1u_3 & 0 & 0 & 0 & 0 & 1 \\ u_2u_3 & 0 & 0 & 1 & 0 & 0 \\ u_1u_4 & 0 & 0 & 0 & 1 & 1 \\ u_2u_4 & 0 & 1 & 1 & 0 & 0 \\ u_3u_4 & 1 & 0 & 1 & 0 & 1 \end{array}$$

$$\begin{array}{cccccc} D & u_4u_3 & u_4u_2 & u_3u_2 & u_4u_1 & u_3u_1 \\ u_4 & 1 & 1 & 0 & 1 & 0 \\ u_3 & 1 & 0 & 0 & 0 & 0 \\ u_2 & 0 & 1 & 0 & 0 & 0 \\ u_1 & 0 & 0 & 0 & 1 & 0 \end{array}$$

Similar to the process to discover the persistence point, negative edges and positive edges in Phase 1. We can find that all the persistence pairs appear and disappear at the same time, thus there is no persistence point. The positive edges in the descending sequence are u_3u_2 and u_3u_1. The negative edges in the descending sequence are u_4u_3, u_4u_2, and u_4u_1.

Notice that before Phase 3, all the persistence points are all 0-dimensional persistence points. In other words, they just record the birth and death of connected components. 1-dimensional extended persistence pair will be discussed in Phase 3, where positive edges in the descending filtration will each be paired with an edge in the ascending filtration, thus the saliency of loops can be measured.

2.3. Phase 3

Phase 3 is the reduction process of P. Only positive descending edges (edges whose column in D is 0) will be reduced, thus the value in D will not be influenced. Similar to the definition of low_A in Phase 1, we define $low_P(e_i)$ as the maximum row of edge e_j for which $P[e_j, e_i] = 1$.

From left to right, we only consider the positive descending edges. We find that $low_P(u_3u_2) = u_3u_4$, $low_P(u_3u_1) = u_3u_4 = low_P(u_3u_2)$. We add column of u_3u_2 to column of u_3u_1: $[0, 1, 0, 1, 1] + [1, 0, 1, 0, 1] = [1, 1, 1, 1, 0]$.

As a consequence, we can get matrix P and D after Phase3:

$$\begin{array}{cccccc} P & u_4u_3 & u_4u_2 & u_3u_2 & u_4u_1 & u_3u_1 \\ u_1u_3 & 0 & 0 & 0 & 0 & 1 \\ u_2u_3 & 0 & 0 & 1 & 0 & 1 \\ u_1u_4 & 0 & 0 & 0 & 1 & 1 \\ u_2u_4 & 0 & 1 & 1 & 0 & 1 \\ u_3u_4 & 1 & 0 & 1 & 0 & 0 \\ D & u_4u_3 & u_4u_2 & u_3u_2 & u_4u_1 & u_3u_1 \\ u_4 & 1 & 1 & 0 & 1 & 0 \\ u_3 & 1 & 0 & 0 & 0 & 0 \\ u_2 & 0 & 1 & 0 & 0 & 0 \\ u_1 & 0 & 0 & 0 & 1 & 0 \end{array}$$

And the extended persistence pair is (u_3u_4, u_3u_2), (u_2u_4, u_3u_1). Notice that for a extended persistence pair, the latter edge is the positive edge in the descending filtration, and the former edge is its paired edge in the ascending filtration. i.e, in the extended persistence pair (u_3u_4, u_3u_2), u_3u_2 is the positive edge in the descending filtration, while u_3u_4 is the paired edge in the ascending filtration. Recall that for a certain edge, its filter value in the ascending sequence is defined as the maximum value of the filter value of its nodes: $f_a(u_3u_4) = max(f(u_3), f(u_4)) = max(t_3, t_4) = t_4$. Similarly, $f_a(u_2u_4) = t_4$. And for a certain edge in the descending filtration, its filter value is defined as the minimum value of the filter value of its nodes: $f_d(u_3u_2) = min(f(u_3), f(u_2)) = min(t_3, t_2) = t_2$. Similarly, $f_d(u_3u_1) = t_1$. As a consequence, the extended persistence point are (t_4, t_2) and (t_4, t_1) respectively.

After the whole matrix reduction algorithm, we can get the ordinary persistence diagram for 0-dimensional topological structures (connected components): $[(t_2, t_3)]$ and the extended persistence diagram for 1-dimensional topological structures (loops): $[(t_4, t_2), (t_4, t_1)]$. To get the 0-dimensional extended persistence diagram, the birth and death of the whole connected component is also recorded, that is, the minimum and the maximum filter value (t_1, t_4). So the 0-dimensional extended persistence diagram is $[(t_2, t_3), (t_1, t_4)]$, as shown in Figure 1 (c).

3. Correctness of the Faster Algorithm

In this section, we provide complete proof of the correctness of the faster algorithm. For convenience, we restate the matrix reduction algorithms Alg. 1 and the proposed faster algorithm Alg. 2. We also restate the main theorem 1.

Theorem 1. *Algorithm 2 outputs the same extended persistence diagram as Algorithm 1.*

To prove Theorem 1, the core of our proof is to show that for each positive descending edge, its corresponding pair found by the new algorithm is equivalent to the pair resulting from the matrix reduction algorithm. We prove this by induction. As we go through all positive descending edges in the descending filtration, we show that for each positive edge, which creates a new loop, its extended persistence pair from our algorithm is the same as the reduction result.

In Lemma 2, we prove that after reducing the D part of the column of a given edge e_i, i.e., when $low_M(e_i) \leq m$, the remaining entries of e_i in P constitute a unique loop in $\{e_i\} \cup \mathcal{T}$.

Algorithm 1 Matrix Reduction

```
1:  **Input:** filter funtion $f$, graph $G$
2:  Persistence Diagram $PD = \{\}$
3:  $M$ = build reduction matrix($f, G$)
4:  **for** $j = 1$ **to** $2m$ **do**
5:      **while** $\exists k < j$ with $low_M(k) = low_M(j)$ **do**
6:          add column $k$ to column $j$
7:      **end while**
8:      add $(f(low_M(j)), f(j))$ to $PD$
9:  **end for**
10: **Output:** Persistence Diagram $PD$
```

Algorithm 2 A Faster Algorithm for Extended Persistence Diagram

```
Input: filter funtion $f$ of descending filtration, graph $G$, filter function $f_a$ of ascending filtration
0-dim PD, $E_{pos}$, $E_{neg}$ = Union-Find($G, f$)
Tree $\mathcal{T}$ = $E_{neg}$ + all the nodes
1-dim PD = {}
for $e_j$ in $E_{pos}$ do
    assume $e_j = uv$, $Path_u \subseteq \mathcal{T}$ is the path from $u$ to $r$ within the tree $\mathcal{T}$, $Path_v \subseteq \mathcal{T}$ is the path from $v$ to $r$
    $Loop = Path_u \cup Path_v - Path_u \cap Path_v$ + $e_j$
    add $(max_{e\in Loop} f_a(e), f(e_j))$ to 1-dim PD
    $e_k = argmax_{e\in Loop} f_a(e)$
    $\mathcal{T} = \mathcal{T} - \{e_k\} + \{e_j\}$
end for
Output: 0-dim PD, 1-dim PD
```

In Lemma 3 and Lemma 4, we prove that: (1) The lowest entry of the reduced column, e_j, is not paired by any other previous edges, and thus will be paired with e_i. Indeed, it is the last edge in the loop w.r.t. the ascending ordering. (2) The updating of the tree $\mathcal{T} = \mathcal{T} - \{e_j\} + \{e_i\}$ is equivalent to adding the reduced column of e_i to all descending columns with nonzero entry at row of e_j. Although this will affect the final reduced matrix, it will not change the resulting simplex pairings.

In Lemma 5, we further prove inductively that in Phase 3, the remaining entries of e_i in P constitute a unique loop in $\{e_i\} \cup \mathcal{T}_{i-1}$. Here $\mathcal{T}_i$ is the tree after updating the first i positive edges.

Finally, in Lemma 6, we prove that the highest filter value and the lowest value exactly form the persistence point of the loop.

Lemma 2. *After Phase 2 (the reduction of descending matrix D) and before Phase 3 (the reduction of permutation matrix P), for a positive edge λ_j, the set $\{\kappa_i | P[i, j] = 1\}$ stores the loop that λ_j and some of the former negative edges form. Besides, it is the loop that λ_j gives birth to.*

Proof. Denote the reduced matrix and its submatrices after Algorithm 1 by $\overline{M} = \begin{bmatrix} \overline{A} & \overline{P} \\ 0 & \overline{D} \end{bmatrix}$. As is shown in (Edelsbrunner & Harer, 2010), matrix reduction algorithm can be interpreted as computing the reduced matrix i.e., $\overline{A} = AV_1$, $\overline{D} = DV_2$, $\overline{P} = PV_2V_3$, where V_1, V_2 and V_3 are invertible and upper-triangular matrices.

For a positive edge λ_j, we observe that after Phase 2, its column in D is set to zero. Here $\overline{D}[:, j]$ is denoted as the j-th column of matrix $\overline{D}$, and we have $DV_2[:, j] = \overline{D}[:, j] = 0$. According to the definition of loop, the boundary of λ_j is finally reduced to zero, so the set $\{\lambda_i | V_2[i, j] = 1\}$ contains all the edges that form the loop which λ_j gives birth to. Considering that (1) V_2 is upper-triangular, thus only columns of former edges will be added to the column of λ_j. (2) All the columns of former positive edges in D have been reduced to zero, thus λ_j will not be reduced by positive edges. $\{\lambda_i | V_2[i, j] = 1\}$ contains the loop that λ_j and some of the former negative edges form.

In fact, after Phase 2, P represents the matrix PV_2. Recall that P stores the permutation that connects the two sequences of simplices, i.e., $P[i; j] = 1$ iff κ_i and λ_j denote the same simplex. Thus $\{\kappa_i | PV_2[i, j] = 1\}$ stores the same component as $\{\lambda_i | V_2[i, j] = 1\}$, that is the loop λ_j and some of the negative edges form. It is the loop that λ_j gives rise to. □

In Algorithm 2, we first add all the negative edges and all the nodes to form the original tree $\mathcal{T}$. If we add a positive edge λ_j to $\mathcal{T}$, then a loop will appear. Considering that in the graph $\mathcal{T}$ + $\{\lambda_j\}$, there is only one loop that λ_j gives birth to, and it consists of λ_j and some of the negative edges that appear before λ_j in the descending sequence. Therefore, it is exactly the loop that set $\{\kappa_i | P[i, j] = 1\}$ consists of after Phase 2.

In the following lemmas, we try to show that: In Algorithm 2, replacing the paired edge e_k with the positive edge e_j every step has the same result with the same step in Phase 3 (matrix reduction for P).

Lemma 3. *In Algorithm 2, the process of replacing the positive edge e_j with its paired edge e_k is equivalent to adding the column of e_j to all the columns whose row of e_k is one.*

Proof. If we add the column of e_j to all the later columns whose row of e_k is "one", all the rows of e_k in the later columns will be zero. i.e., assume $P[e_k, e_l] = 1$ [1], then after adding column of e_j with column of e_l, $P[e_k, e_l] = 0$. The set $\{e_i | P[e_i, e_l] = 1\}$ contains the loop that appear before e_l (without e_k) and e_j, e_l.

Here we denote $\mathcal{T}_{e_j} = \mathcal{T} - \{e_k\} + \{e_j\}$. In the graph $\mathcal{T}_{e_j}$ + $\{e_l\}$, there is only a loop. Therefore, it is exactly the loop that set $\{e_i | P[e_i, e_l]\}$ consists of.

For the former columns whose row of e_k is "one", i.e., $P[e_k, e_l] = 1$. This means $low_P(e_l) > e_k$ [2], then adding column e_j to e_l will not change the extended persistence pair because $low_P(e_l)$ will not be affected by previous edges and thus remain the same. □

[1]Here, for simplicity, we use e_k and e_l to represent their indices.

[2]If $low_P(e_l) = e_k$, e_k will be paired in the former columns, thus dissatisfies the assumption that $low_P(e_j) = e_k$.

Lemma 4. *Adding the column of e_j to all the columns whose row of e_k is one in Lemma 3 has the same extended pair as the matrix reduction algorithm in Phase 3.*

Proof. Assume $P[e_k, e_l] = 1$, for simplicity, we define $low_P(e_l)$ as the maximum row index e_i for which $P[e_i, e_l] = 1$. If $low_P(e_l) = e_k$, then in the matrix reduction algorithm, we should add column e_j to e_l, thus the two algorithms are exactly the same. If $low_P(e_l) \neq e_k$, which means $low_P(e_l) > e_k$, then adding column e_j to e_l will not change the extended persistence pair because $low_P(e_l)$ will not be affected by previous edges and thus remain the same. As a consequence, adding the column of e_j to all the later columns whose row of e_k is one in Lemma 3 has the same extended pair with the matrix reduction algorithm in Phase 3. Together with Lemma 3, adding the column of e_j to all the columns whose row of e_k is one in Lemma 3 has the same extended pair with the matrix reduction algorithm in Phase 3. □

From Lemma 3 and Lemma 4, we manage to prove that in Algorithm 2, replacing the paired edge e_k with the corresponding positive edge e_j is equivalent to Phase 3 (matrix reduction for P). Combining it with Lemma 2, we can prove that for every positive edge, to update $\mathcal{T}$ in the faster algorithm leads to the same extended pair with the matrix reduction algorithm. We have proved that in a single step, the two algorithms are equivalent. Then we should prove inductively that the whole process of the Algorithm 2 is equivalent to the matrix reduction Algorithm 1.

Lemma 5. *In Algorithm 2, the process to update the tree $\mathcal{T}$ is equivalent to the matrix reduction process in Phase 3.*

Proof. First, in the original lemmas, we have proved that adding a positive edge to the original tree $\mathcal{T}$ and updating the tree leads to the same result as the matrix reduction algorithm.

We then assume that after adding the first $j-1$ positive edges, the process of updating the tree can output the same results as the matrix reduction algorithm. Denote the tree after updating the first $j-1$ positive edges as $\mathcal{T}_{j-1}$.

When adding the j-th positive edge e_j. Similar to the prove in Lemma 2, we can prove that the set $\{e_i | P[e_i, e_j] = 1\}$ stores the loop that e_j and $\mathcal{T}_{j-1}$ form. And similar to the prove in Lemma 3 and Lemma 4, we can prove that replacing the paired edge e_k with the newly added positive edge e_j leads to the same extended persistence pair with the matrix reduction algorithm in Phase 3. As a consequence, the process of updating the tree to $\mathcal{T}_j$ can lead to the same results as the reduction algorithm. Then Lemma 5 is proved inductively. □

In the above Lemmas, we have proven that to update the tree $\mathcal{T}$ in the faster algorithm output the same result as the matrix reduction algorithm. Then we should confirm that the extended point given by Algorithm 2 is correct.

Lemma 6. *In a loop, the highest filter value and the lowest filter value form its extended persistence point.*

Proof. For a positive edge e_j, the edge it pairs in matrix P is the lowest one in its column, representing the latest one in the ascending filtration. Notice that in the ascending filtration, we define the filter value of an edge $f_a(uv) = max(f(u), f(v))$, thus it has the biggest filter value in the loop. e_j is the latest born edge in the descending filtration, Recall that we define the filter value of an edge in the descending filtration as $f_d(uv) = min(f(u), f(v))$, thus it contains the lowest filter value in the loop. As a result, the extended persistence point will be the highest and lowest filter value of the loop. □

From Lemma 6, we manage to prove that the persistence point of the positive edge e_j is exact the value provided in Algorithm 2. As a consequence, we can justify Theorem 1: Algorithm 2 outputs the same extended persistence diagram as Algorithm 1.

4. Experiment

4.1. Details of Real-World Datasets

The real-world datasets in this paper include:

1. Citation network: PubMed (Sen et al., 2008) is a standard benchmark describing citation network where nodes denote scientific papers and edges are citations between them.
2. Amazon networks: Photo and Computers (Shchur et al., 2018) are datasets related to Amazon shopping records where nodes represent products and edges imply that two products are frequently brought together.
3. PPI networks: 24 Protein-protein interaction networks(Zitnik & Leskovec, 2017) where nodes denote protein and edges represent the interaction between proteins. Each graph has 3000 nodes with average degree 28.8. The dimension of node feature vector is 50.

The detailed statistics of these data is shown in Table 1. Because PPI networks contain multiple graphs, we do not add it in the Table.

4.2. Details of Experiment Setting

Data split. We follow the experimental setting from (Chami et al., 2019; Zhu et al., 2020) and use 5% (resp. 10%)

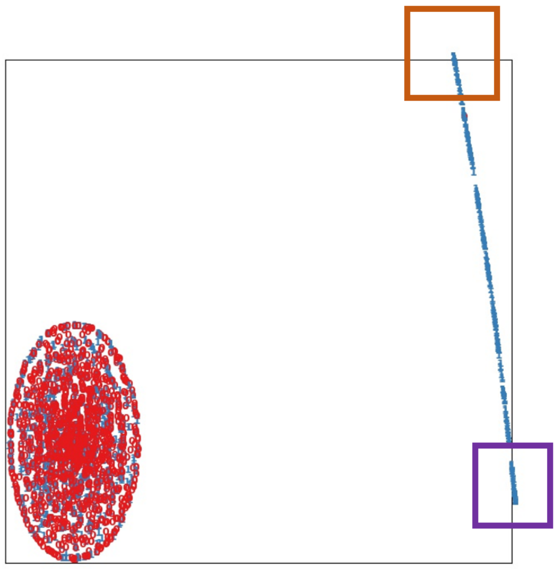


Figure 3. The t-distributed stochastic neighbor embedding (t-SNE) projection of the persistence images on PubMed. In the figure, the red and blue marks denote the persistence images generated by the negative edge and positive edges respectively.

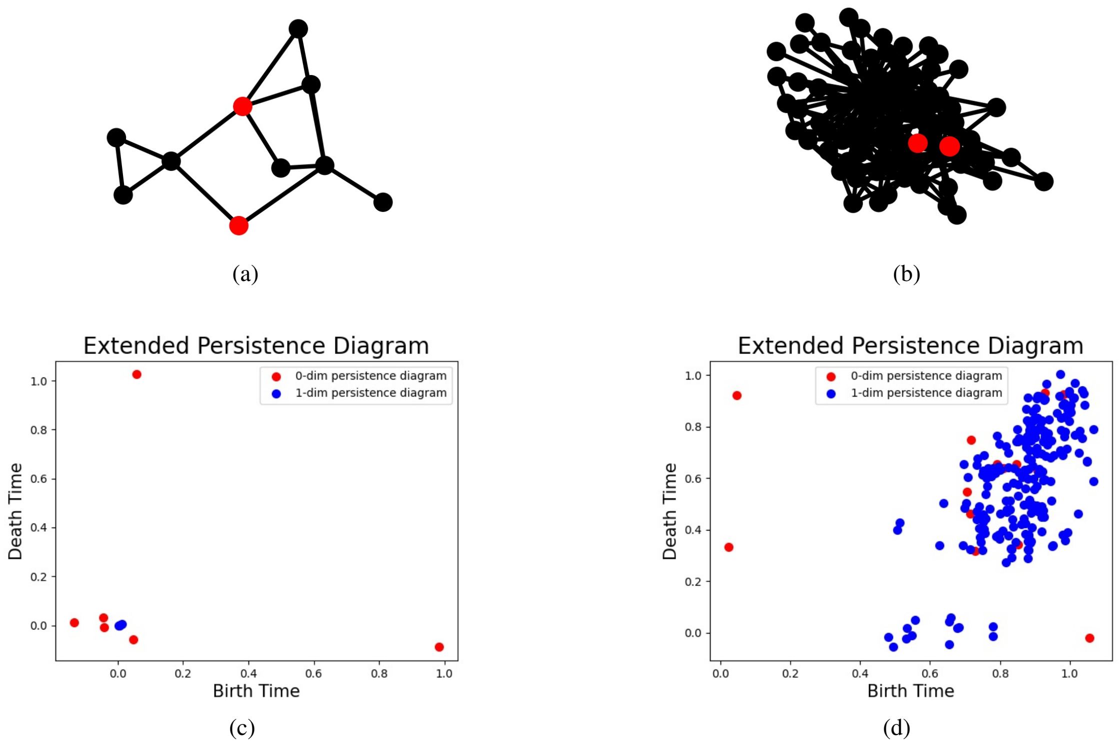


Figure 4. We sample two positive edges from the visualization of PubMed in Figure 3. We draw their subgraphs and diagrams. (a) and (c) represent the sample in the brown box. (b) and (d) represent the sample in the purple box.

Table 1. Statistics of benchmark datasets

DATASET	FEATURES	NODES	EDGES	EDGE DENSITY
PUBMED	500	19717	44338	0.0002
PHOTO	745	7487	119043	0.0042
COMPUTERS	767	13381	245779	0.0027

of existing links as positive samples of the validation set (resp. test set). And equal number of non-existent links are sampled as negative samples of the validation and test set. The remaining 85% existing links are used as the positive training set. In every epoch, we randomly choose the same number of remaining non-existent links as the negative training set. We report the results on test set when the models achieve the best performance on the validation set.

Training setting. On synthetic experiments, we run all the methods on each graph 10 times and report the mean average area under the ROC curve (ROCAUC) scores as the result. On real-word benchmarks, we run all the methods on each graph 50 times and report the mean and standard deviation of ROCAUC scores as the result. All methods use the following training strategy: the same training epochs (2000), and the same early stopping on validation set with 200 patience epochs. The only exception is SEAL; due to its slow training speed and fast convergence, we only train 200 epochs.

Following (Chami et al., 2019; Zhu et al., 2020), during training, we remove positive validation and test edges from the graph. Cross Entropy Loss is chosen as the loss function and Adam is adopted as the optimizer with the learning rate set to 0.01 and weight-decay set to 0. For fairness, we set the number of node embeddings of the hidden layer and the final layer to be the same (100 and 16) for all networks. The backbone GNN in our model is a classic 2-layer GCN with one hidden and one output layer. All persistence images in the experiments are 25-dimension. All the activation function used in the graph neural networks is RELU and all the activation function used in Fermi-Dirac decoder (needed in TLC-GNN) is Leakyrelu with negative slope set to 0.2.

Details on evaluation of algorithm efficiency. To evaluate the efficiency of the proposed faster algorithm (Section 5.3 in the main paper), we use the following setting. For the sparse graph PubMed, we compute 0-dimensional and 1-dimensional extended persistence diagrams on all the existing edges. No edges are removed. For large and dense graphs like Photo and Computer, we compute 0-dimensional and 1-dimensional extended persistence diagrams on the first 1000 edges in the default edge list. We run the algorithms on each graph 10 times and report the average seconds per edge as the result. We use a cluster with two Intel Xeon Gold 5128 processors and 192GB RAM to run the two algorithms without multi-threading.

Table 2. Experimental results(s) on the chosen of hop distance

k	1	2
PubMed	96.79	**97.03**
PPI	83.92	**84.11**

4.3. Further experiments

In this paragraph, we add experiments to evaluate the effect of k (the hop distance to form the enclosing subgraph).

Considering that on large and dense graphs such as Photo and Computers, it costs immensely to compute the persistence image when k is 2, and on all the datasets, computing persistence image when k is larger than 3 takes immense computational cost, we evaluate the effect of k on PubMed and a sampled graph in PPI. As shown in Table 2, $k = 2$ is generally a good choice in these two datasets. However, it cost much more to compute the persistence images in PPI networks when k = 2, so we finally set k = 1 in PPI datasets.

4.4. Visualization of Extended Persistence

We provide qualitative examples to further illustrate how topological features can help differentiate edges/non-edges.

We use the t-distributed stochastic neighbor embedding (t-SNE) (Van der Maaten & Hinton, 2008) to map the 25-dim persistence images of samples to a 2D plane, as shown in Figure 3. The persistence images here are created using the Ollivier-Ricci curvature (Ni et al., 2018) as the filter function. For each graph, we randomly choose 1500 positive edges and 1500 negative edges. Figure 3 shows the t-SNE results of PubMed. The red and blue marks represent negative and positive edges respectively. Despite some exceptions, negative edges and positive edges are well separated in terms of persistent homology features.

To further understand the data, we choose 2 positive samples from the t-SNE plot of PubMed (Figure 3) and draw their local enclosing graphs and persistence diagrams. In the t-SNE plot, positive samples form an elongated linear structure. We intentionally sample the two samples from the two ends of the structure. One from the center of the brown box. The other from the center of the purple box. The local enclosing graph and diagram of the first sample is drawn in Figure 4 (a) (c). The graph and diagram of the second sample is drawn in Figure 4 (b) (d).

In the graph, the red nodes denote the target nodes, while the other nodes are black. In the persistence diagram, the red and blue markers represent the 0-dimensional and 1-dimensional persistence points respectively. Notice that we add a random jitter to each persistence point so that we can observe the overlapped persistence points.

For the sample in the brown square, we observe that in its enclosing subgraph, there are several loops passing the target nodes. They correspond to 1D persistence points with death time zero in the diagram. The only loop that does not pass the target nodes has the same birth and death time, thus is not shown in the persistence diagram.

For the sample in the purple square, we observe that there exist many loops in the generated subgraph, and the distribution of the 1-dimensional persistence points mainly concentrate on the top right of the diagram. In addition, more 0-dimensional extended persistence points whose birth time is smaller than its death time appear. We observe (1) the density of 1-dimensional extended persistence points gradually increase from the bottom to the upper-right of the diagram. (2) more 0-dimensional extended persistence points from the ascending filtration appear.

Discussion. From Figure 3 and Figure 4, we observe the following phenomena. Persistence images effectively differentiate positive and negative edges in all graphs. While almost all negative samples form a tight cluster, positive samples form clusters like pieces of 1-manifolds. This makes us wonder whether these clusters can be parameterized by a latent parameter. The selected two samples further suggest the possibility of this hypothesis. The brown and purple samples represent the two extreme of the positive cluster in PubMed. The share common characteristics, e.g., both have rich loops (compared to their number of nodes). Meanwhile, they range from small subgraphs with less loops to dense subgraphs with many loops. To investigate further on these positive sample clusters is an interesting research direction in the future.

References

Adams, H., Emerson, T., Kirby, M., Neville, R., Peterson, C., Shipman, P., Chepushtanova, S., Hanson, E., Motta, F., and Ziegelmeier, L. Persistence images: A stable vector representation of persistent homology. *The Journal of Machine Learning Research*, 18(1):218–252, 2017.

Chami, I., Ying, Z., Ré, C., and Leskovec, J. Hyperbolic graph convolutional neural networks. In *Advances in neural information processing systems*, pp. 4868–4879, 2019.

Cohen-Steiner, D., Edelsbrunner, H., and Harer, J. Extending persistence using poincaré and lefschetz duality. *Foundations of Computational Mathematics*, 9(1):79–103, 2009.

Edelsbrunner, H. and Harer, J. *Computational topology: an introduction*. American Mathematical Soc., 2010.

Edelsbrunner, H., Letscher, D., and Zomorodian, A. Topological persistence and simplification. In *Proceedings 41st annual symposium on foundations of computer science*, pp. 454–463. IEEE, 2000.

Ni, C.-C., Lin, Y.-Y., Gao, J., and Gu, X. Network alignment by discrete ollivier-ricci flow. In *International Symposium on Graph Drawing and Network Visualization*, pp. 447–462. Springer, 2018.

Sen, P., Namata, G., Bilgic, M., Getoor, L., Galligher, B., and Eliassi-Rad, T. Collective classification in network data. *AI magazine*, 29(3):93–93, 2008.

Shchur, O., Mumme, M., Bojchevski, A., and Günnemann, S. Pitfalls of graph neural network evaluation. *arXiv preprint arXiv:1811.05868*, 2018.

Van der Maaten, L. and Hinton, G. Visualizing data using t-sne. *Journal of machine learning research*, 9(11), 2008.

Zhu, S., Pan, S., Zhou, C., Wu, J., Cao, Y., and Wang, B. Graph geometry interaction learning. *Advances in Neural Information Processing Systems*, 33, 2020.

Zitnik, M. and Leskovec, J. Predicting multicellular function through multi-layer tissue networks. *Bioinformatics*, 33 (14):i190–i198, 2017.