Link Prediction with Persistent Homology: An Interactive View

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

Abstract

Link prediction is an important learning task for graph-structured data. In this paper, we propose a novel topological approach to characterize interactions between two nodes. Our topological feature, based on the extended persistent homology, encodes rich structural information regarding the multi-hop paths connecting nodes. Based on this feature, we propose a graph neural network method that outperforms state-of-the-arts on different benchmarks. As another contribution, we propose a novel algorithm to more efficiently compute the extended persistence diagrams for graphs. This algorithm can be generally applied to accelerate many other topological methods for graph learning tasks.

1. Introduction

Graph-structured data is very common in our life. Learning from graphs is important in various scientific and industrial domains (Zhang et al., 2020; Wu et al., 2020). In this paper, we focus on the link prediction task, i.e., to learn to predict whether an edge exists between two target nodes, conditioned on their attributes and local connectivity (Liben-Nowell & Kleinberg, 2007; Schlichtkrull et al., 2018; Lü & Zhou, 2011). Link prediction is an important step in knowledge discovery in various applications, e.g., recommendation systems (Koren et al., 2009; Adamic & Adar, 2003), knowledge graph completion (Teru et al., 2020), protein-protein interactions (Coulomb et al., 2005), and gene prediction (Nagarajan et al., 2015).

Classic link prediction methods (Barabási & Albert, 1999; Zhou et al., 2009; Brin & Page, 2012) use hand-crafted connectivity features and enforce strong assumptions of the distributions of links, e.g., nodes with similar connectivity features tend to be connected. Better performance has been achieved by comparing the similarity of nodes in the embedding space (Perozzi et al., 2014), which encodes more global connectivity information. In recent years, graph neural networks (GNNs) have achieved state-of-the-art link prediction performance as they exploit graph connectivity information and node attributes in a completely data driven manner. However, even for GNNs, graph connectivity information such as node degrees is beneficial; it provides contextual information for the graph convolutional operations (Kipf & Welling, 2016; Qiu et al., 2018; Ye et al., 2020).

For link prediction, an effective strategy is the direct modeling of the interaction between two target nodes. The path distance of nearby nodes from the target nodes has been shown useful (Zhang & Chen, 2018). However, these distance-based methods mainly focus on the "closeness" between target nodes, but do not explicitly model the "richness of connections" between them. In Figure 1, we show examples with the same distance encoding. But the connections in the right example are much richer than in the left example. It is conceivable that nodes with a wealth of multi-hop connections have a better chance to share an edge.

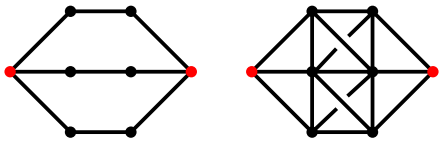

Figure 1. Two example graphs. In both cases, black nodes have the same distance from the red target nodes (either (2,1) or (1,2)). But the richness of the connections is very different. On the left, there are only three connections between the target nodes. On the right, there are many more connections between the targets.

To exploit the richness of connections between nodes, we propose a novel method based on the theory of persistent homology (Edelsbrunner et al., 2000; Edelsbrunner & Harer, 2010), which encodes high-order structural information of the graph via algebraic topology. The theory captures topological structures of arbitrary shape and scale, e.g., connected components and loops, and encodes them in a robust-to-noise manner. To predict whether two given nodes u and v are connected, our method explicitly counts the number of loops within their vicinity. This count essentially measures the complexity of connections between u and v, which can

[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 38^{th} International Conference on Machine Learning, PMLR 139, 2021.

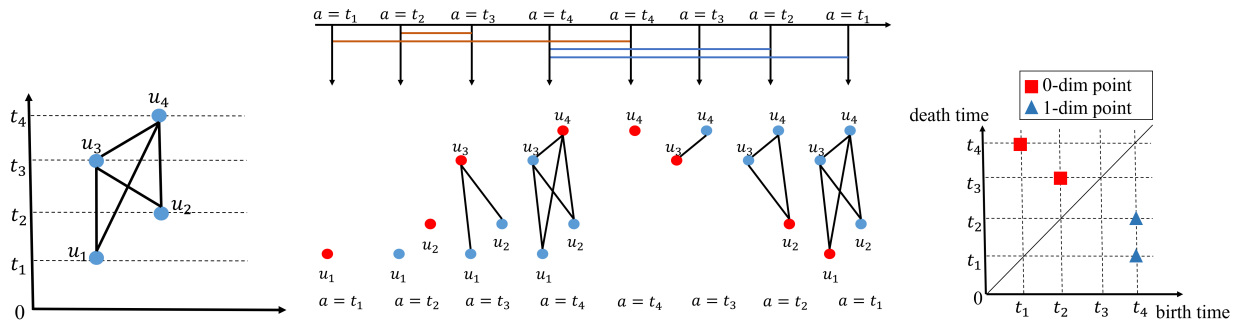

Figure 2. An illustration of extended persistent homology. (a) We plot the input graph with a given filter function. The filter value for each node is $f(u_1) = t_1$, $f(u_2) = t_2$, $f(u_3) = t_3$, $f(u_4) = t_4$. (b) The ascending and descending filtrations of the input graph. The bars of brown and 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.

be indicative of the existence of links. The method not only counts the numbers of loops, but also measures the range of distance from u and v for each loop. See Figure 2 for an illustration. This rich set of structural information is mapped into a topological feature space and is integrated into a graph neural network. As will be shown in experiments, our topological loop-counting graph neural network achieves better performance for link prediction.

Persistent homology has been used for learning with graphs (Zhao & Wang, 2019; Hofer et al., 2020; 2017; Carrière et al., 2020). However, most existing works use it as a global structural feature for the whole graph. These global features, although proven useful for graph classification tasks, cannot describe the interaction between a pair of nodes.

In this paper, we propose a *pairwise topological feature* to capture the richness of the interaction between a specific pair of target nodes. We compute topological information within the vicinity of the target nodes, i.e., the intersection of the k-hop neighborhoods of the nodes. It has been shown that such *local enclosing graph* carries sufficient information for link prediction (Zhang & Chen, 2018). To measure the saliency of topological structures, we also introduce a distance-based filter function to measure the interaction between nodes. These choices ensure our pairwise topological feature to be informative of the interaction between the targets.

We propose *topological loop-counting graph neural network (TLC-GNN)* by injecting the pairwise topological feature into the latent representation of a graph neural network. Our method achieves state-of-the-art performance in link prediction. Another contribution of this paper is on the computational side. To capture the full information of loops, we use the extended persistent homology (Cohen-Steiner et al., 2009). The commonly used algorithm is based on a matrix reduction algorithm that is similar to the Gaussian elimination. It is cubic to the input graph/subgraph size, $O((|V| + |E|)^3)$.[1] Since the computation needs to be executed on all training/validation/testing pairs of nodes, we could significantly benefit from a faster algorithm. In this paper, we propose a novel algorithm for the extended persistent homology, specific for graphs. Instead of the expensive matrix reduction, our algorithm directly operates on the graph and is quadratic to the input graph/subgraph size, $O(|V||E|)$. This algorithm is not application specific and can be applied to other graph learning problems (Zhao & Wang, 2019; Hofer et al., 2020; 2017; Carrière et al., 2020).

In summary, our contribution is three-fold:

- We introduce a pairwise topological feature based on persistent homology to measure the complexity of interaction between nodes. We compute the topological feature specific to the target nodes using a carefully designed filter function and domain of computation.
- We use the pairwise topological feature to enhance the latent representation of a graph neural network and achieve state-of-the-art link prediction results on various benchmarks.
- We propose a general-purpose fast algorithm to compute extended persistent homology on graphs. The time complexity is improved from cubic to quadratic to the input size. It applies to many other persistent-homology-based learning methods for graphs.

[1] In theory, the fastest algorithm for persistence diagram has the same complexity as matrix multiplication (Milosavljević et al., 2011), i.e., $O((|V| + |E|)^\omega)$, in which $\omega = 2.3728596$ (Alman & Williams, 2021).

Outline. In Section 2, we briefly introduce existing works on link prediction and on learning with topological information. In Section 3, we present details of extended persistent homology and our model, TLC-GNN. In Section 4, we introduce a faster algorithm for extended persistent homology and prove its correctness. In Section 5, we evaluate our method on synthetic and real-world benchmarks.

2. Related Work

Link prediction methods. Early works (Barabási & Albert, 1999; Zhou et al., 2009; Brin & Page, 2012; Jeh & Widom, 2002) predict links based on node similarity scores measured within the local neighborhood of the two target nodes. These methods tend to have strong assumptions on the link distribution and do not generalize well. Graph embeddings have been used to encode more global structural information for link prediction (Koren et al., 2009; Airoldi et al., 2008; Perozzi et al., 2014; Tang et al., 2015; Qiu et al., 2018). However, these methods only rely on graph connectivity and do not take full advantage of node attributes.

In recent years, new GNN-based methods have been proposed to jointly leverage graph connectivity and node attributes. Zhang & Chen (2018) show that local enclosing subgraphs contains sufficient information, and propose a GNN to leverage such information. Considering the non-Euclidean nature of graph metrics, one may generalize graph convolution to the hyperbolic space (Chami et al., 2019; Zhu et al., 2020). However, most existing methods use either limited structural information or node embedding to represent edge features. They do not explicitly model the advanced topological information that arises in node interaction.

Learning with topological features. Persistent homology (Edelsbrunner et al., 2000; Edelsbrunner & Harer, 2010) captures structural information from the data using the language of algebraic topology (Munkres, 2018). It captures multi-scale topological structures in a provably robust manner (Cohen-Steiner et al., 2007). Different learning methods for persistent homology have been proposed, such as direct vectorization (Adams et al., 2017), kernel machines (Reininghaus et al., 2015; Kusano et al., 2016; Carriere et al., 2017), convolutional neural networks (Hofer et al., 2017), topological loss (Chen et al., 2019; Hu et al., 2019; Hofer et al., 2019), and generative model (Wang et al., 2020).

For graph-structured data, topological features have been used for node classification (Zhao et al., 2020) and graph classification (Zhao & Wang, 2019; Hofer et al., 2020; Carrière et al., 2020). However, these existing methods cannot model interactions between nodes as desired in link prediction tasks. Bhatia et al. (2018) also use persistent homology for link prediction. But their method only exploits 0-dimensional topology, i.e., whether the target nodes are connected or not within the local neighborhood. This cannot capture the complexity of connection as we intend to model.

3. Link Prediction with Persistent Homology

In this section, we introduce our topological loop-counting graph neural network (TLC-GNN), which computes persistent homology based on the chosen subgraph and incorporates the pairwise topological feature into a graph neural network. The input of the model includes two target nodes and a subgraph encoding their topological information. The output is the probability of whether an edge exists between the two target nodes.

In section 3.1 and 3.2, we will briefly introduce the extended persistent homology and its computation, respectively. In section 3.3, we will illustrate how to combine the topological feature with a standard graph neural network.

3.1. Extended Persistent Homology

In this section, we provide a brief introduction to extended persistent homology and refer the reader to (Cohen-Steiner et al., 2009) for details. In the setting of graphs, the data only contain 0-dimensional (connected components) and 1-dimensional (loops) topological structures[2]. We define simplices to be all elements in the graph, including nodes and edges. The combinatorial relationship between simplices determines the topological structure of the graph, and persistent homology can count the number of these topological structures. Besides, persistent homology measures the saliency of all topological structures in view of a scalar function defined on all simplices, called the filter function. For example, let V, E be the sets of nodes and edges, and X be the union of V and E, namely the set of simplices. The filter function for nodes $f : V \to \mathbf{R}$ can be defined as the sum of the distance to the target nodes. Then we can further define the filter function for edge uv as the maximum value of $f(u)$ and $f(v)$.

Given the filter function for all the simplices in a graph, we can define X_a as the sublevel set of X: $X_a = \{x | f(x) \leq a, x \in X\}$. Here a is a threshold in the filter function f, and X_a is the subset of X whose filter value are not greater than a. As the threshold a increases from $-\infty$ to ∞, we obtain the ascending filtration of X: $\emptyset = X_{-\infty} \subset ... \subset X_{\infty} = X$. An example is shown in the first half of Figure 2 (b).

With the increasing of threshold a, the sublevel set grows from empty to X, and new topological structures gradually appear (born) and disappear (die). For example, two connected components appear when reaching X_{t_1} and X_{t_2} (for simplicity, we replace X_{t_i} with X_i). One of the connected

[2]In graphs, there is no triangle. As a result, all the loops are topological structures (non-bounding cycles).

components disappears in X_3. Besides, two loops appear when reaching X_4.

Extended persistence. However, with the settings above, we find that some structures, such as the whole connected component and all the loops, will never disappear. To address this limitation of ordinary persistent homology, Cohen-Steiner et al. (2009) propose extended persistence by introducing another filtration of the superlevel set $X^a = \{x|f(x) \geq a, x \in X\}$. Let a decrease from ∞ to $-\infty$, and we can obtain the descending filtration of X: $\emptyset = X^{\infty} \subset ... \subset X^{-\infty} = X$[3]. An example descending filtration is shown in the second half of Figure 2 (b). In the descending filtration, the filter function for an edge uv is set differently, i.e., $f(uv) = min(f(u), f(v))$.

For loops and the whole connected component in a graph, the death time can be defined as the filter value in the superlevel set when the structure appears again. For instance, the extended persistence point of the loop $\{ac, cd, da\}$ in Figure 2 is (t_4, t_1). It is born when reaching X_4 in the ascending filtration and dies when reaching X^1 in the descending filtration. It is a bit counter-intuitive that the death time can be smaller than the birth time.

After capturing the birth, death times of all the topological structures, we encode them into a 2-D point set called persistence diagram. Each topological structure corresponds to one persistence point in the diagram. Its x and y coordinates are the birth and death times. In Figure 2 (c), we show an extended persistence diagram.

After obtaining the extended persistence diagram, we can encode it into a vectorized feature called persistence image (Adams et al., 2017). Further details are available in the supplementary material.

3.2. Matrix Reduction Algorithm for Extended Persistence Diagram

In this section, we will introduce the algorithm to compute the extended persistence diagrams. Let m be the total number of simplices ($m = |V| + |E|$). We write $(\kappa_1, \kappa_2, ..., \kappa_m)$ as the ascending sequence of simplices in X, i.e., $f(\kappa_1) < f(\kappa_2) < ... < f(\kappa_m)$[4]. Similarly, we write $(\lambda_1, \lambda_2, ..., \lambda_m)$ as the descending sequence of simplices in X. Every simplex will appear once in the ascending sequence and once in the descending sequence.

To compute the extended persistence diagram, we need a binary valued matrix M to encode the adjacency relationship between nodes and edges. Matrix 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. The matrix A of Figure 2 is shown in Figure 3

	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

Figure 3. The matrix A of Figure 2

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. In the supplementary material, we will provide a complete example of matrix M.

The computation of persistent homology boils down to counting ranks of submatrices of the boundary matrix. This can be achieved by a specific matrix reduction algorithm on M. Algorithm 1 reduces M from left to right. To describe the reduction process, let $low_M(j)$ be the maximum row index i for which $M[i, j] = 1$. For instance, in Figure 3, $low_M(4) = 3$. If column j is zero, then $low_M(j)$ is undefined. M is reduced if $low_M(j) \neq low_M(k)$ for any two non-zero columns $j \neq k$. Notice that "add" here is the mod-2 sum of the binary column vectors. After the matrix reduction is finished, we can record the persistence diagram. Each pair of simplex ids $(low_M(j), j)$ corresponds to an extended persistence pair. By taking the filter function value of the corresponding simplices, we will obtain the corresponding birth and death times of a persistence point. For simplicity, we slightly abuse the notation, that is to denote the birth and death time as $f(low_M(j))$ and $f(j)$ respectively. To better illustrate Algorithm 1, we provide the reduction process of Figure 2 in the supplementary material.

[3]Technically, the superlevel set should be the relative homology groups (X, X^a) in the second half of the filtration.

[4]Without loss of generality, we assume the filter function of all nodes are distinct.

Algorithm 1 Matrix Reduction

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

3.3. TLC-GNN for Link Prediction

We have introduced extended persistent homology, its computation and vectorization. Next, we explain how our method computes topological feature for a specific pair of target nodes, and combine it with GNN for link prediction.

To characterize the interaction between two target nodes, we need to choose a vicinity graph and a corresponding filter function. Given a weighted graph, we can define the sum of the distance from any node v to the target nodes v_1 and v_2 as the filter function, $f(v) = d(v, v_1) + d(v, v_2)$. Note that for each pair of target nodes on which we predict whether a link exists, we create a filter function and then compute the persistent homology as the topological feature.

Our topological feature is only extracted within vicinity of the target nodes. Existing works using persistent homology usually exploit topological information from the whole graph (Zhao & Wang, 2019; Hofer et al., 2020), thus cannot be directly used in our task. Besides, topological structures captured by global persistent homology can be irrelevant to the target nodes if they are far away. As justified in (Zhang & Chen, 2018), local enclosing subgraphs already contain enough information for link prediction. We exploit the intersection of the k-hop neighborhoods of the two target nodes: let $G = (V, E)$ be the whole graph, where V and E are the set of vertices and edges. $V_1 = \{v | d(v, v_1) \leq k\}$ and $V_2 = \{v | d(v, v_2) \leq k\}$ are the k-hop neighborhoods of target nodes v_1 and v_2, $V_{12} = V_1 \cap V_2$ is the intersection. The enclosing subgraph is $G_{12} = (V_{12}, E_{12})$, where $E_{12} = E \cap V_{12}^2$. We compute the persistence image of the subgraphs generated by all possible links, and define $PI(v_1, v_2)$ as the persistence image of the subgraph generated by target nodes v_1 and v_2.

We combine the extracted topological feature $PI(v_1, v_2)$ with GNN-generated node embedding features for link prediction. To obtain node embeddings, we use a classic L-layer graph convolutional network (GCN) (Kipf & Welling, 2016). Messages are passed between nodes to update their feature representations. After an L-layer GCN, the embedding of a certain node can be viewed as a combination of node representation from its L-hop neighborhood. $H^\ell = [h_1^\ell, h_2^\ell, ..., h_{|V|}^\ell]$ where $h_i^\ell \in \mathbf{R}^{d_\ell}$ is the representation of node i in the ℓ-th layer, $\ell = 0, 1, ..., L$, and $|V|$ is the number of nodes. Here, H^0 is the input node features, and H^L is the node embeddings of the final layer.

To compute the representation of all the nodes, in the ℓ-th layer, $H^\ell = \sigma(\hat{D}^{-1/2}\hat{A}\hat{D}^{-1/2}H^{\ell-1}W^\ell)$, where $\hat{A} = A + I$ denotes the adjacency matrix with inserted self-loop and $\hat{D}$ is a diagonal matrix with $\hat{D}[i, i] = \sum_j \hat{A}[i, j]$. σ represents the activation function, and W^ℓ is a learned matrix to encode node embedding from dimension $d_{\ell-1}$ to d_ℓ. The nodewise formulation of node i is given by $h_i^\ell = \sigma(W^\ell \sum_{j \in N(i)} \frac{1}{\sqrt{\hat{d}_j \hat{d}_i}} h_j^{\ell-1})$ where $N(i)$ is the neighborhood of node i including i itself.

For given target nodes u and v, the node embeddings h_u and h_v obtained through the GCN model can be viewed as the node feature, and the persistence image $PI(u, v)$ can be viewed as the edge feature. To combine the two features effectively, we use a modified Fermi-Dirac decoder (Krioukov et al., 2010; Nickel & Kiela, 2017): $(h_u - h_v)^2$ is defined as the distinction between the two nodes. It is then concatenated with $PI(u, v)$, and passed to a two layer Multi-Layer Perceptron (MLP), which outputs a single value. Denote the value after the two layer MLP as distance of the two target nodes: $dist(u, v)$, and the final probability of whether there exists an edge between u and v is $prob(u, v) = \frac{1}{(e^{(dist(u,v)-2)}+1)}$. We then train TLC-GNN by minimizing the cross-entropy loss using negative sampling.

4. A Faster Algorithm for Extended Persistence Diagram

In this section, we propose a new algorithm for extended persistent homology. Recall that Algorithm 1 reduces the matrix M. In the new algorithm, we manage to avoid explicit construction and reduction of the matrix M. Instead, we create and maintain a rooted tree while going through the descending filtration. When processing a new edge, by inspecting its relationship with the current tree, we can find the corresponding persistence pair efficiently. Afterward, we update the tree and continue with the next edge in the descending filtration. In section 4.1, we explain the algorithm in details. In section 4.2, we prove the correctness of the proposed algorithm.

4.1. A Faster Algorithm

Our new algorithm is shown in Algorithm 2. For 0-dim topology, we run existing union-find algorithm (Edelsbrunner & Harer, 2010) for the ascending filtration once and for the descending filtration once. The algorithm is guaranteed

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

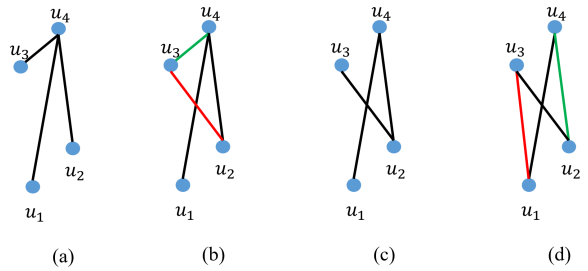

Figure 4. An illustration of Algorithm 2, based on Figure 2.

to be correct. In the following section, we will therefore mainly focus on 1-dimensional extended persistence.

Take Figure 4 as an example, with the Union-Find algorithm (Edelsbrunner et al., 2000; Cohen-Steiner et al., 2006), we can procure the set of positive edges and the set of negative edges in the descending filtration: $E_{pos} = \{u_2u_3, u_1u_3\}$, $E_{neg} = \{u_3u_4, u_2u_4, u_1u_4\}$. Recall that positive edges give rise to new loops, while negative edges do not. And in the descending filtration, every positive edge will be paired with a certain edge in the ascending filtration, and give rise to a certain loop.

In order to find all the persistence pairs, we construct a tree with all the negative edges and nodes. Every time we add a positive edge into the tree, a loop appears. The newly added edge and the edge which appears the latest in the ascending filtration in the loop form the persistence pair. We then delete the paired edge to update the tree iteratively. Updating the tree is essential. When a new edge e is added into the descending filtration, it forms a unique loop with the tree; this loop is exactly the loop with the latest birth during ascending and is coned out due to e. Without tree updating, the loop formed by the tree and e will not be the desired one.

Look at Figure 4, all the negative edges and nodes form (a). The positive edge u_2u_3 is then added, and we can discover the loop it forms: $\{u_3u_4, u_4u_2, u_2u_3\}$. In the loop, the edge that appears the latest in the ascending filtration is u_3u_4. The sequence of simplices in the ascending filtration is shown in Figure 3. Then u_2u_3 is paired with u_3u_4, we can obtain the persistence point (t_4, t_2).

After that, we delete the paired edge u_3u_4 to get (c). Then the positive edge u_1u_3 is added, with the same process, we can discover the loop: $\{u_1u_3, u_3u_2, u_2u_4, u_4u_1\}$, and the paired edge is u_2u_4. The persistence point of the loop is (t_4, t_1), so the final 1-dimensional extended persistence diagram is $\{(t_4, t_2), (t_4, t_1)\}$.

Complexity of Algorithm 2. For every positive edge, the time cost to get the loop and to form the new path are both $O(|V|)$, here $|V|$ denotes the number of nodes, and $|E|$ represents the number of edges. So the time cost among all positive edges is $(|E|-|V|+1)*O(|V|) = O(|V||E|)$. The union-find algorithm will take $O(|V|+|E|)log(|E|+|V|)$. Therefore the final computational cost is $O(|V||E|)$. This is much more efficient than the complexity of the matrix reduction algorithm, $O((|V|+|E|)^3)$.

4.2. The Correctness of Algorithm 2

In this section, we prove the proposed faster algorithm is correct, i.e., it produces the same output as the matrix reduction algorithm. Formally, we state the theorem as follows.

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

We briefly explain the idea of the proof. A complete proof is provided in the supplementary material.

The 0-dimensional persistent homology is computed using known Union-Find algorithm. We only need to prove the correctness for the 1-dimensional topology, i.e., loops. Recall each loop is created by an edge in the ascending filtration and destroyed by an edge in the descending filtration. We plan to show that the pairs of the edges found by our new algorithm are the same as the pairs found by the matrix reduction algorithm. Denote by $\overline{M} = \begin{bmatrix} \overline{A} & \overline{P} \\ 0 & \overline{D} \end{bmatrix}$ the reduced matrix of M and its submatrices.

We classify edges in the descending filtration into negative descending edges and positive descending edges. A negative descending edge e_i destroys a connected component created by a descending node v_j. In the reduced matrix $\overline{M}$, its lowest entry $[v_j, e_i]$ falls into $\overline{D}$. A positive descending edge e_i adds a new loop to the superlevel set. But for extended persistence, it destroys the loop created by an edge

e_j in the ascending filtration. The lowest entry of e_i, $[e_j, e_i]$ falls into $\overline{P}$. See Figure 2 (c) for an illustration of different types of simplex pairs and their corresponding persistence points in the extended diagram.

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 in the reduced algorithm. We prove this by induction. As we go through all positive descending edges in the descending filtration, we show that for edge e_i, the pairing by our algorithm is the same as the reduction result. We prove that:

- After reducing the D part of the column of e_i, i.e., when $low(e_i) \leq m$, 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.
- 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.
- The updating of the tree $\mathcal{T}_i = \mathcal{T}_{i-1} - \{e_j\} + \{e_i\}$ 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.

This proves that our new algorithm has the same output as the existing matrix reduction algorithm. We have shown that the new algorithm is much more efficient in terms of complexity (Section 4.1). We will also validate the benefit empirically in Section 5.

5. Experiments

We evaluate our method on on synthetic and real-world graph datasets. We compare with different SOTA link prediction baselines. Furthermore, we evaluate the efficiency of the proposed faster algorithm for extended persistent homology. Source code is available at https://github.com/pkuyzy/TLC-GNN.

Baseline methods. We compare our framework TLC-GNN with different link prediction methods. We compare with popular GNN models such as **GCN** (Kipf & Welling, 2016) and **GAT** (Veličković et al., 2017). We use them for node embedding and then adopt the Fermi-Dirac decoder (Krioukov et al., 2010; Nickel & Kiela, 2017) to predict whether there is a link between two nodes. We also compare with several SOTA methods. **SEAL** (Zhang & Chen, 2018), which utilize local distances and pre-trained transductive node features for link prediction.**HGCN** (Chami et al., 2019), which introduces hyperbolic space to deal with free-scale graphs. **GIL** (Zhu et al., 2020), which takes advantage of both Euclidean and hyperbolic geometries.

To demonstrate the importance of *pairwise* topological features, we also compare with two baseline methods using node-wise topological features, i.e., topological feature for each node. **PEGN** (Zhao et al., 2020) extracts topological features for each node using its neighborhood. Then the node-wise topological feature is used to re-calibrate the graph convolution. PEGN was originally designed for node classification. Similar to GCN and GAT, we adapt it to link prediction via the Fermi-Dirac decoder. A second baseline, **TLC-GNN (Nodewise)**, is a modification of our method TLC-GNN (Ricci). Instead of the pairwise topological feature for a pair of target nodes, we concatenate their node-wise topological features, inject it into the node embedding, and then use MLP to predict.

For all settings, we randomly split edges into 85/5/10% for training, validation, and test sets. To compute the distance-based filter function, we need a graph metric. We use hop-distance and Ollivier-Ricci curvature (Ni et al., 2018) respectively. The filter function with hop-distance is the same as Double-Radius Node Labeling (DRNL) (Zhang & Chen, 2018). For Ollivier-Ricci curvature, we compute the curvature for all training edges and use them as the edge weights. We add 1 to the curvature of all edges to avoid negative edge weights. With this weighted graph, we define the filter function of each node as the total shortest distance from the two target nodes. In each subgraph, the corresponding target nodes (v_1, v_2) are used as the anchors to compute the filter function: for given node v, $f(v) = d(v, v_1) + d(v, v_2)$, where $d(v, v_1)$ is the shortest path distance between the given node v and the corresponding target node v_1 in the weighted graph. Recall that we need to use the intersection of the k-hop neighborhoods of the two target nodes to compute pairwise topological features. The choice of k is either 1 or 2, depending on the size and density of the graph.

Note that here we use pre-computed weight function to define filter function. It is possible to extend the framework to learn the filter function end-to-end, as in graph classification (Hofer et al., 2020). However, this involves recomputing filter function and persistence diagram for each pair of target nodes for each epoch; it is computationally prohibitive. Alternatively, we may learn better kernels on the persistence diagrams, which do not require recomputing persistence diagrams every epoch (Zhao & Wang, 2019).

5.1. Synthetic Experiments

We generate synthetic data using a graph theoretical model called Stochastic Block Model (SBM) (Holland et al., 1983). To be specific, we create random graphs with 1000 nodes, forming 5 equal-size communities. Edges are randomly sampled with intra-community probability p and inter-community probability q. We randomly create 12 graphs with p ranging in $\{0.05, 0.25, 0.45\}$ and q ranging

Table 1. Mean and standard deviation of ROC-AUC on real-world data. "*": results copied from (Chami et al., 2019; Zhu et al., 2020).

METHOD	PUBMED	PHOTO	COMPUTERS
GCN (KIPF & WELLING, 2016)	89.56±3.660*	91.82±0.000	87.75±0.000
HGCN (CHAMI ET AL., 2019)	96.30±0.000*	95.40±0.000	93.61±0.000
GIL (ZHU ET AL., 2020)	95.49±0.160*	97.11±0.007	95.89±0.010
SEAL (ZHANG & CHEN, 2018)	92.42±0.119	97.83±0.013	96.75±0.015
PEGN (ZHAO ET AL., 2020)	95.82±0.001	96.89±0.001	95.99±0.001
TLC-GNN (NODEWISE)	96.91±0.002	97.91±0.001	97.03±0.001
TLC-GNN (DRNL)	96.89±0.002	97.61±0.003	97.23±0.003
TLC-GNN (RICCI)	**97.03±0.001**	**98.23±0.001**	**97.90±0.001**

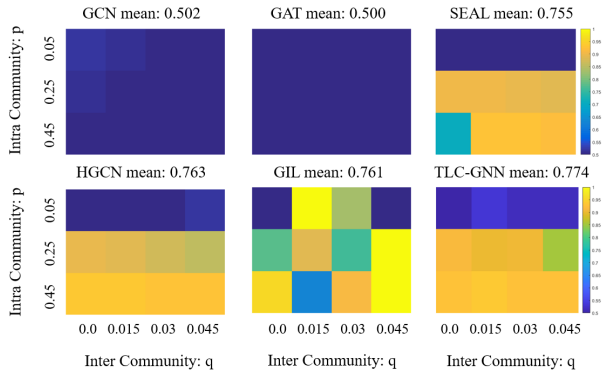

Figure 5. Heatmap of ROC-AUC score for different methods on synthetic data. For each method we generate a heatmap for its performance on 12 synthetic graphs with different (p, q) combinations. We also report in the title of each heatmap the average performance over 12 graphs.

in $\{0.0, 0.015, 0.03, 0.045\}$. Furthermore, we assign each node with a randomly created feature of dimension 100 and use them as the input of all the methods. For pairwise persistence diagrams, we use intersections of 1-hop neighborhoods for all the graphs.

We run the methods on each synthetic graph 10 times and report the mean average area under the ROC curve (ROC-AUC) score as the result. More details can be found in the supplementary material.

Results. Figure 5 reports the results of 6 methods. We observe that TLC-GNN outperforms nearly all the SOTA baselines among all the synthetic graphs. This justifies the benefit of pairwise topological feature. We observe that GIL sometimes performs well, but is unstable overall.

5.2. Real-World Benchmarks

We use a variety of datasets: (a) PubMed (Sen et al., 2008) is a standard benchmark describing citation network. (b) Photo and Computers (Shchur et al., 2018) are graphs related to Amazon shopping records. (c) PPI networks are protein-protein interaction networks(Zitnik & Leskovec, 2017). More details of these datasts can be found in the supplementary material.

Table 2. Computational time (seconds per edge) evaluation.

	PUBMED	PHOTO	COMPUTERS
ALG. 1	0.0068	1.6557	4.6531
ALG. 2	0.0027	1.1176	2.7033

Table 3. Experimental results(s) on PPI datasets

SAMPLES	1	2	3	4	5
GCN	75.21	74.42	77.68	76.22	69,67
GIL	57.69	1.45	34.90	**85.61**	33.65
HGCN		CANNOT	CONVERGE		
SEAL	50.00	64.79	67.14	72.55	50.00
TLC-GNN	**83.92**	**81.21**	**83.95**	83.03	**83.53**

For pairwise persistence diagrams, we compute intersections of 1-hop neighborhoods for PPI, Photo and Computers due to their high density. We use intersections of 2-hop neighborhoods for PubMed. Following prior works (Chami et al., 2019; Zhu et al., 2020), we evaluate link prediction using the ROC-AUC score on the test set. We run the methods on each graph 50 times to get the results.

Results. Table 1 summarizes the performance of all methods. TLC-GNN is superior to others among all the benchmarks. This implies that the high-order topological feature is more effective in these large and dense graphs, which tend to have rich and heterogeneous structures.

For PPI networks, we run experiments on 5 sample graphs. Ollivier-Ricci curvature is adopted as the filter function. The results are shown in Table 3. We observe that TLC-GNN consistently outperforms nearly all the SOTA baselines among all the sampled PPI graphs. Although GIL sometimes performs well, it is unstable overall. This further justifies the benefit of pairwise topological feature.

Ablation study. Comparing with models using nodewise persistent homology (PEGN and TLC-GNN (Nodewise)), TLC-GNN generally performs better. This proves that for link prediction task, the proposed pairwise persistent homology carries crucial structural information to model node interaction. Whereas nodewise topological feature cannot.

5.3. Algorithm Efficiency

To verify the computational benefit of the proposed new algorithm (Algorithm 2), we compare it with the classic matrix reduction algorithm (Algorithm 1) (implemented in the *Dionysus* package (Morozov)). Both implementations are written in python[5]. We compare the two algorithms on all real-world benchmarks. We report the average running time for computing the pairwise topological feature for each edge (in seconds). More details are available in the supplementary material.

Results. As is shown in Table 2, the proposed Algorithm 2 achieves 1.5 to 2.5 times speedup compared with the matrix reduction algorithm (Algorithm 1) on all benchmarks.

6. Conclusion

In this paper, we propose a novel pairwise topological feature for link prediction, based on the theory of persistent homology. We also introduce a GNN model to leverage this topological feature. Experiments show that our approach outperforms state-of-the-arts on various benchmarks, especially on large and dense graphs. Besides, we propose a novel algorithm to more efficiently calculate the extended persistence diagrams on graphs. We verify the correctness and efficiency of the algorithm. The algorithm can be generalized to other graph learning tasks.

Acknowledgements. We thank anonymous reviewers for their constructive feedback. This work is supported by the projects of National Natural Science Foundation of China (No. 61876003) and National Key R&D Program of China (2019YFB1406303), which is also a research achievement of Key Laboratory of Science, Technology and Standard in Press Industry (Key Laboratory of Intelligent Press Media Technology).

References

Adamic, L. A. and Adar, E. Friends and neighbors on the web. *Social networks*, 25(3):211–230, 2003.

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.

Airoldi, E. M., Blei, D. M., Fienberg, S. E., and Xing, E. P. Mixed membership stochastic blockmodels. *Journal of machine learning research*, 9(Sep):1981–2014, 2008.

Alman, J. and Williams, V. V. A refined laser method and faster matrix multiplication. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pp. 522–539. SIAM, 2021.

Barabási, A.-L. and Albert, R. Emergence of scaling in random networks. *science*, 286(5439):509–512, 1999.

Bhatia, S., Chatterjee, B., Nathani, D., and Kaul, M. Understanding and predicting links in graphs: A persistent homology perspective. *arXiv preprint arXiv:1811.04049*, 2018.

Brin, S. and Page, L. Reprint of: The anatomy of a large-scale hypertextual web search engine. *Computer networks*, 56(18):3825–3833, 2012.

Carriere, M., Cuturi, M., and Oudot, S. Sliced wasserstein kernel for persistence diagrams. In *International Conference on Machine Learning*, pp. 664–673. PMLR, 2017.

Carrière, M., Chazal, F., Ike, Y., Lacombe, T., Royer, M., and Umeda, Y. Perslay: a neural network layer for persistence diagrams and new graph topological signatures. In *International Conference on Artificial Intelligence and Statistics*, pp. 2786–2796. PMLR, 2020.

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.

Chen, C., Ni, X., Bai, Q., and Wang, Y. A topological regularizer for classifiers via persistent homology. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pp. 2573–2582. PMLR, 2019.

Cohen-Steiner, D., Edelsbrunner, H., and Morozov, D. Vines and vineyards by updating persistence in linear time. In *Proceedings of the twenty-second annual symposium on Computational geometry*, pp. 119–126, 2006.

Cohen-Steiner, D., Edelsbrunner, H., and Harer, J. Stability of persistence diagrams. *Discrete & computational geometry*, 37(1):103–120, 2007.

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

[5]The codes of Dionysus are transformed into Cython, a C-Extension for Python.

Coulomb, S., Bauer, M., Bernard, D., and Marsolier-Kergoat, M.-C. Gene essentiality and the topology of protein interaction networks. *Proceedings of the Royal Society B: Biological Sciences*, 272(1573):1721–1725, 2005.

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.

Hofer, C., Kwitt, R., Niethammer, M., and Uhl, A. Deep learning with topological signatures. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pp. 1633–1643, 2017.

Hofer, C., Kwitt, R., Niethammer, M., and Dixit, M. Connectivity-optimized representation learning via persistent homology. In *International Conference on Machine Learning*, pp. 2751–2760. PMLR, 2019.

Hofer, C., Graf, F., Rieck, B., Niethammer, M., and Kwitt, R. Graph filtration learning. In *International Conference on Machine Learning*, pp. 4314–4323. PMLR, 2020.

Holland, P. W., Laskey, K. B., and Leinhardt, S. Stochastic blockmodels: First steps. *Social networks*, 5(2):109–137, 1983.

Hu, X., Li, F., Samaras, D., and Chen, C. Topology-preserving deep image segmentation. In *Advances in neural information processing systems*, pp. 5657–5668, 2019.

Jeh, G. and Widom, J. Simrank: a measure of structural-context similarity. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 538–543, 2002.

Kipf, T. N. and Welling, M. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

Koren, Y., Bell, R., and Volinsky, C. Matrix factorization techniques for recommender systems. *Computer*, 42(8): 30–37, 2009.

Krioukov, D., Papadopoulos, F., Kitsak, M., Vahdat, A., and Boguná, M. Hyperbolic geometry of complex networks. *Physical Review E*, 82(3):036106, 2010.

Kusano, G., Hiraoka, Y., and Fukumizu, K. Persistence weighted gaussian kernel for topological data analysis. In *International Conference on Machine Learning*, pp. 2004–2013. PMLR, 2016.

Liben-Nowell, D. and Kleinberg, J. The link-prediction problem for social networks. *Journal of the American society for information science and technology*, 58(7): 1019–1031, 2007.

Lü, L. and Zhou, T. Link prediction in complex networks: A survey. *Physica A: statistical mechanics and its applications*, 390(6):1150–1170, 2011.

Milosavljević, N., Morozov, D., and Skraba, P. Zigzag persistent homology in matrix multiplication time. In *Proceedings of the twenty-seventh Annual Symposium on Computational Geometry*, pp. 216–225, 2011.

Morozov, D. Dinoysus2. https://www.mrzv.org/software/dionysus2/, accessed on 2021/03/01.

Munkres, J. R. *Elements of algebraic topology*. CRC press, 2018.

Nagarajan, M., Wilkins, A. D., Bachman, B. J., Novikov, I. B., Bao, S., Haas, P. J., Terrón-Díaz, M. E., Bhatia, S., Adikesavan, A. K., Labrie, J. J., et al. Predicting future scientific discoveries based on a networked analysis of the past literature. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 2019–2028, 2015.

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.

Nickel, M. and Kiela, D. Poincaré embeddings for learning hierarchical representations. In *Advances in neural information processing systems*, pp. 6338–6347, 2017.

Perozzi, B., Al-Rfou, R., and Skiena, S. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 701–710, 2014.

Qiu, J., Dong, Y., Ma, H., Li, J., Wang, K., and Tang, J. Network embedding as matrix factorization: Unifying deepwalk, line, pte, and node2vec. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*, pp. 459–467, 2018.

Reininghaus, J., Huber, S., Bauer, U., and Kwitt, R. A stable multi-scale kernel for topological machine learning. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4741–4748, 2015.

Schlichtkrull, M., Kipf, T. N., Bloem, P., Van Den Berg, R., Titov, I., and Welling, M. Modeling relational data with graph convolutional networks. In *European semantic web conference*, pp. 593–607. 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.

Tang, J., Qu, M., Wang, M., Zhang, M., Yan, J., and Mei, Q. Line: Large-scale information network embedding. In *Proceedings of the 24th international conference on world wide web*, pp. 1067–1077, 2015.

Teru, K., Denis, E., and Hamilton, W. Inductive relation prediction by subgraph reasoning. In *International Conference on Machine Learning*, pp. 9448–9457. PMLR, 2020.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., and Bengio, Y. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

Wang, F., Liu, H., Samaras, D., and Chen, C. Topogan: A topology-aware generative adversarial network. In *European Conference on Computer Vision*, volume 2, 2020.

Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., and Philip, S. Y. A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems*, 2020.

Ye, Z., Liu, K. S., Ma, T., Gao, J., and Chen, C. Curvature graph network. In *International Conference on Learning Representations*, 2020.

Zhang, M. and Chen, Y. Link prediction based on graph neural networks. In *Advances in Neural Information Processing Systems*, pp. 5165–5175, 2018.

Zhang, Z., Cui, P., and Zhu, W. Deep learning on graphs: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 2020.

Zhao, Q. and Wang, Y. Learning metrics for persistence-based summaries and applications for graph classification. In *Advances in Neural Information Processing Systems*, pp. 9859–9870, 2019.

Zhao, Q., Ye, Z., Chen, C., and Wang, Y. Persistence enhanced graph neural network. In *International Conference on Artificial Intelligence and Statistics*, pp. 2896–2906. PMLR, 2020.

Zhou, T., Lü, L., and Zhang, Y.-C. Predicting missing links via local information. *The European Physical Journal B*, 71(4):623–630, 2009.

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.