Geodesic Graph Neural Network for Efficient Graph Representation Learning

Anonymous Author(s)
Affiliation
Address
email

Abstract

Recently, Graph Neural Networks (GNNs) have been applied to graph learning tasks and achieved state-of-the-art results. However, many competitive methods employ preprocessing on the target nodes, such as subgraph extraction and customized labeling, to capture some information which is hard to be learned by normal GNNs. Such operations are time-consuming and do not scale to large graphs. In this paper, we propose an efficient GNN framework called Geodesic GNN (GD-GNN). It injects conditional relationship between nodes into the model without labeling. Specifically, we view the shortest paths between two nodes as the spatial graph context of the neighborhood around them. The GNN embeddings of nodes on the shortest paths are used to generate geodesic representations. Conditioned on the geodesic representations, GD-GNN is able to generate node, link, and graph representations that carry much richer structural information than plain GNNs. We theoretically prove that GD-GNN is more powerful than plain GNNs, and present experimental results to show that GD-GNN achieves highly competitive performance with state-of-the-art GNN models on link prediction and graph classification tasks while taking significantly less time.

1 Introduction

Graph Neural Network (GNN) is a type of neural networks that learns on relational data. With the emergence of large-scale network data, it can be applied to solve many real-world problems, including recommender systems [43], protein structure modeling [28], and knowledge graph completion [2]. The growing and versatile nature of graph data poses great challenges to GNN algorithms both on their performance and on their efficiency.

GNNs use message passing to propagate features between connected nodes in the graph, and the nodes aggregate their received messages to generate representations that encode the graph structure and feature information around them. These representations can be combined to form multi-node structural representations. Because GNNs are efficient and have great generalizability, they are widely employed in node-level, edge-level and graph-level tasks. A part of the power of GNNs comes from the fact that they resemble the process of 1-dimensional Weisfeiler-Lehman (1-WL) algorithm [20]. The algorithm encodes subtrees rooted from each node through an iterative node coloring process, and if two nodes have the same color, they have the same rooted subtree and should have very similar surrounding graph structures. However, as pointed out by Xu *et al.* [44], GNN's expressive power is also upper-bounded by the 1-WL test. Specifically, GNN is not able to differentiate nodes that have exactly the same subtrees but have different substructures. For example, consider the graph in Figure 1 with two connected components, because all nodes have the same number of degrees, they will have exactly the same rooted subtrees. Nevertheless, the nodes in the left component is clearly different from nodes in the right component, because the left component is a 3-cycle and the right

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

one is a 4-cycle. Such cases can not be discriminated by GNNs or 1-WL test. We refer to this type of GNN as plain GNN or basic GNN.

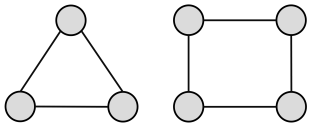


Figure 1: An example where plain GNN fails to distinguish nodes in the graph.

On the other hand, even when a carefully-designed GNN can differentiate all node substructures in a graph, Zhang *et al.* [53] show that the learned node representation still cannot effectively perform structural learning tasks involving multiple nodes, including link predictions and subgraph predictions, because the nodes are not able to learn the relative structural information. We will discuss this with more details in Section 3.1.

These limitations motivate recent research that pushes the limit of GNN's expressiveness. One branch is to use higher-order GNNs that mimic higher-dimensional Weisfeiler-Lehman tests [21, 24]. They extend node-wise message passing to node-tuple-wise and obtain more expressive representations. Another branch is to use labeling methods. By extracting subgraphs around the nodes of interests, they design a set of subgraph-customized labels as additional node features. The labels can be used to distinguish between the nodes of interests and other nodes. The resulting node representations can encode subtructures which plain GNNs cannot, like cycles.

However, the major drawback of these methods is that compared to plain GNNs, their efficiency is significantly compromised. Higher-order GNNs update embeddings over tuples and incur at least cubic complexity [51, 21]. Labeling tricks usually require subgraph extraction on a large graph and running GNNs on mini-batches of subgraphs. Because the subgraphs overlap with each other, the aggregated size of the subgraphs is possibly hundreds times of the large graph. Nowadays GNNs are massive, the increased graph computation cost makes labeling methods inapplicable to real-world problems with millions of nodes.

We observe that a great portion of the labeling methods rely on the geodesics, namely the shortest paths, between nodes. For node representation, Distance-Encoding(DE) [23] labels the target nodes' surrounding nodes with their shortest path distance to the target nodes and achieves higher than 1-WL test expressiveness. Current state-of-the-art methods for link prediction, such as SEAL [49], leverage the labeling tricks in order to capture the shortest path distance along with the number of shortest paths between the two nodes of the link. Our proposed Geodesic-GNN (GD-GNN) model is designed to also capture this characteristic, while addressing the efficiency issue of previous methods.

Specifically, our GD-GNN model learns a function that maps the learning target to a geodesic-augmented target representation (the target can be node, edge, graph, etc). The model consists of two units, a GNN unit and a geodesic pooling unit. The two units are connected to perform end-to-end training. We first apply the GNN to the graph to obtain embeddings of all nodes. Then, for each target, we find the task-specific geodesics and extract the corresponding GNN representations of nodes on the geodesics. The task-specific geodesics have three levels. For node-level tasks, we extract the geodesics between the target node and all of its neighbors. For edge-level tasks, we extract the geodesics between the two target nodes of the edge. For graph-level tasks, we get the geodesic-augmented node representation for each node as in node-level tasks and readout all node representations as the graph representation. Finally, using the geodesic pooling unit, we combine the geodesic representation and target representation to form the final geodesic augmented target representation. Figure 2 summarizes the GD-GNN framework.

All task-specific geodesics are formed by one or multiple **pair-wise** geodesics between two nodes. We propose two methods to extract the pair-wise geodesic information, horizontal or vertical. The horizontal geodesic is obtained by directly extracting one shortest path between the two nodes. The vertical geodesic is obtained by extracting all direct neighbors of the two nodes that are on any of their shortest paths. They focus on the depth and breath respectively, and possess different properties. Horizontal geodesic is good at capturing long distance information not covered by the GNN. Vertical geodesic is *provably more powerful* than plain GNN. Moreover, by incorporating the information of the subgraph induced by the vertical geodesic nodes, we can use GD-GNN to distinguish some *distance-regular graphs*, a type of graph that many current most powerful GNNs cannot distinguish.

During inference, the GNN unit is applied **only once** to the graph, after which we feed nodes and geodesic information to the efficient geodesic pooling unit. This reduces the number of GNN runs from the query number (or number of nodes for graph-level tasks) to a constant of one, which largely reduces the computation cost. *Compared to labeling methods, GD-GNN only requires one GNN run*

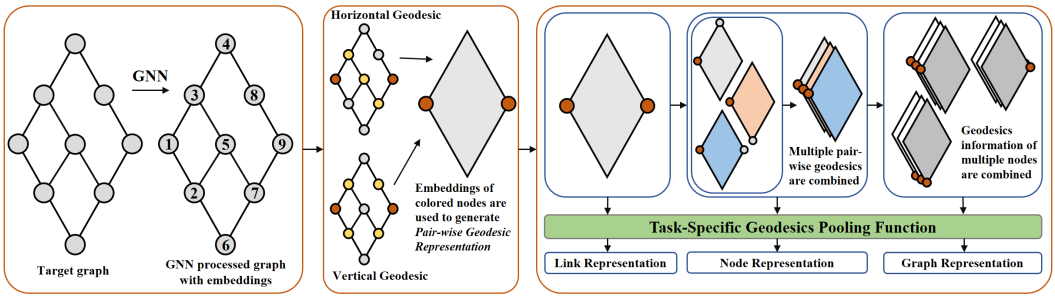


Figure 2: The overall pipeline of GD-GNN. GNN is applied once to generate node embeddings. We then use horizontal or vertical geodesic to extract pair-wise geodesic representation. One or more such representations are collected to generate task-specific representation.

and is much more efficient. Compared to plain GNN methods, GD-GNN uses geodesic information and thus is more powerful. We conducted experiment on link prediction and graph classification across datasets with different scales and show that GD-GNN is able to significantly improve the performance of basic GNNs and achieves very competitive results compared to state-of-the-art methods. We also demonstrate GD-GNN's runtime superiority over other more-expressive GNNs.

2 Preliminaries

A graph can be denoted by $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{R})$, where $\mathcal{V} = \{v_1, ..., v_n\}$ is the node set, $\mathcal{E} \subseteq \{(v_i, r, v_j) | v_i, v_j \in \mathcal{V}, r \in \mathcal{R}\}$ is the edge set and $\mathcal{R}$ is the set of edge types in the graph. When $|\mathcal{R}| = 1$, the graph can be referred to as homogeneous graph. When $|\mathcal{R}| > 1$, the graph can be referred to as heterogeneous graph. Knowledge graphs are usually heterogeneous graphs. The nodes can be associated with features $\mathcal{X} = \{\boldsymbol{x}_v | \forall v \in \mathcal{V}\}$. For simplicity, we use homogeneous graphs to illustrate our ideas and methods.

We follow the definition of message passing GNNs in [44] with MESSAGE and AGGREGATE functions. Such GNNs iteratively update the node embeddings by receiving and combining the neighbor nodes' embeddings. The k-th layer of a GNN can be expressed as:

$$\boldsymbol{m}_u^{(k)} = \text{MESSAGE}^{(K)}(\boldsymbol{h}_u^{(k-1)}), \quad \boldsymbol{h}_v^{(k)} = \text{AGGREGATE}(\{\boldsymbol{m}_u^{(k)}, u \in \mathcal{N}(v)\}, \boldsymbol{h}_v^{(k-1)}) \tag{1}$$

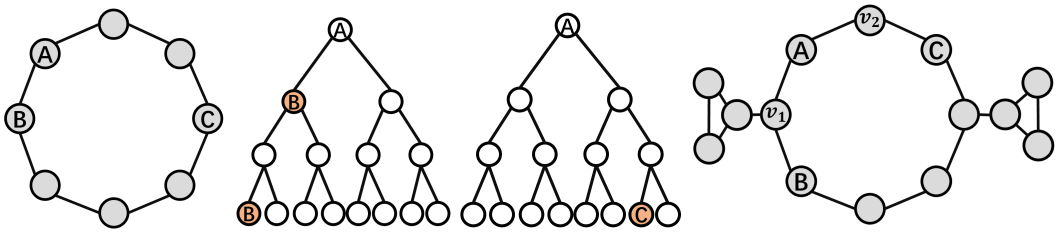
where $\boldsymbol{h}_v^{(k)}$ is the node representation after k iterations, $\boldsymbol{h}_v^{(0)} = \boldsymbol{x}_v$, $\mathcal{N}(v)$ is the set of direct neighbors of v, $\boldsymbol{m}_u$ is the message embedding. Different GNNs vary mainly by the design choice of the MESSAGE and AGGREGATE functions. In this paper, we use $\boldsymbol{h}_v$ to represent the final node representation of v from the plain GNN. Throughout this paper, we use bold letters to represent vectors, capital letters to represent sets, and $\mathcal{N}^k(v)$ to refer to nodes within k-hop neighborhood of v.

3 Geodesic Graph Neural Network

Geodesic is conventionally defined as the shortest path between two nodes and indeed in homogeneous graph it encodes only the distance. However, we found that when combined with GNNs and node representations, geodesics express information much richer than shortest path distance alone. In this section, we first discuss the intuition behind GD-GNN and its effectiveness. Then, we describe the GD-GNN framework and how its GNN unit and geodesic pooling unit are combined to make predictions. Finally, we provide theoretical results of the better expressiveness of GD-GNN.

3.1 Intuition and Overview

The key weakness of directly using plain GNNs is that the embeddings are generated without conditional information. For node-level tasks, as pointed out in [48], a plain GNN is unaware of the position of the target node in the rooted subtree, and hence the target node is oblivious about the conditional information about itself. For edge-level tasks, Zhang *et al.* [53] show that despite a node-most-expressive-GNN, without node labeling, the embedding of one node in the edge is not able to preserve the conditional information of the other node. Concretely, take link prediction as



(a) (b) (c) (d)

Figure 3: (a) A circular graph where plain GNN cannot distinguish link AB and AC. (b) and (c) are the GNN computation graphs of A if B and C, respectively, are labelled. (d) An example where geodesic representation can distinguish links AB and AC, but distance fails to do so.

an example, in Figure 3a, we have two target links AB and AC. Since AB is already connected by a link and AC is not, they should be represented differently. On the other hand, All nodes in the graph are symmetric and should be assigned the same embeddings using any GNN. Hence, if we simply concatenate the node representations of AB and AC, they will have the same link representations, and cannot be differentiated by any downstream learner. To overcome this problem, previous methods rely on labeling tricks to augment the computation graphs with conditional information. When node B and C are labelled respectively in two separate subgraphs, the resulting computation graphs of A is different as shown in Figure 3b and 3c.

Notice that, for the labeling methods to take effect, subgraph extraction is necessary. However, relabeling the graph and applying a GNN to every target to predict is extremely costly. Hence, we wonder whether we can also incorporate the conditional information between nodes without the customized labeling? Observe that in the example, the computation graph captures the number of layers before node A first receives a message from a labelled node, which is essentially the shortest path distance d between A and the other node. This implies that even when we **only** have node representations from a plain GNN, by adding the distance information (which is 1 and 3 for AB and AC respectively) independent of the computation graph, we can still distinguish the links.

While the previous example demonstrates the effectiveness of a simple shortest path distance metric, we notice that distance alone is not sufficient in many times. For example, in Figure 3d, node B and node C are symmetric and hence should be assigned the same node embedding when the graph is not labelled. However, the links AB and AC are apparently different as v_1 connecting AB connects to another group of nodes whereas v_2 connecting AC does not connect to any other node. If we only use shortest path distance between AB and AC to assist the prediction, we still cannot differentiate them. However, if we inject the **entire geodesic** expressed by the corresponding **node embeddings along the shortest path** to the model, we are able to tell AB and AC apart due to v_1 and v_2's different embeddings.

Consider a naive GNN that outputs all node embeddings as one. Taking the shortest path can be seen as extracting a one-valued vector with a length of the shortest distance between the two nodes. Whereas in the neural setting, the general GNN can model the naive GNN and the nodes on the same vector can have different representations besides one by encoding their own neighborhood. The extension of shortest path to the neural setting grants stronger expressive power to the geodesics. Compared to subgraphs in subgraph GNNs, the geodesic generation process and the GNN process is *decoupled*, because we select the geodesic nodes after all node representations are generated, while one subgraph GNN result is tied to a subgraph. Dwelling on this decoupling and conditioning philosophy, GD-GNN can run the GNN **only once** to generate generic node embeddings, and then use geodesic information to inject conditional information into the model in a much **more efficient** way.

3.2 Pair-wise Geodesic Representation

GD-GNN has different ways to leverage geodesic information for different levels of tasks, but they all have the same basic component, pair-wise geodesic representation. It consists of structural information of nodes on the shortest paths of the pair. Directly extracting all such nodes is time-consuming and sometimes not effective, hence we present two primary ways to learn the geodesic representation, *horizontal* and *vertical*.

3.2.1 Horizontal Geodesic Representation

Horizontal Geodesic Representation is designed to directly capture the distinctiveness as described in the previous example shown in the Figure 3d. It first finds a shortest path between two nodes as the geodesic. If there are multiple paths between them, we randomly choose one. The nodes on the shortest path is denoted by $P^{(H)}_{(u,v)} = \{u, n_1, n_2, ..., n_{d-1}, v\}$ with a length of d. Using $P^{(H)}_{(u,v)}$ we can construct the geodesic representation $\boldsymbol{g}_{(u,v)}$ by a geodesic pooling layer.

$$\boldsymbol{g}^{(H)}_{(u,v)} = R^{(H)}_{gd}(\{\boldsymbol{h}_w, \forall w \in P^{(H)}_{(u,v)}\}) \tag{2}$$

To prevent extremely long geodesics which may cause overfitting, we limit the maximum length to a manageable constant d_{max}. Any pair with distance larger than d_{max} will have infinite distance and a zero geodesic representation.

Because of the high computation cost of GNNs, the number of GNN layers is usually very small. Using horizontal geodesics, we are able to connect nodes that are *far away* and do not coexist in a GNN's limited-height computation graph.

3.2.2 Vertical Geodesics Representation

Horizontal geodesics seem like an intuitive choice to directly connect the pair of nodes by a sequence of adjacent nodes. However, a natural question to ask is whether a single path is enough to represent the conditional information between the pair? An example where horizontal geodesic falls short can be found in Appendix H. Realizing this limitation, we propose vertical geodesic representation that includes all geodesic information in an efficient way.

For a pair of nodes u and v, vertical geodesic extracts the **direct neighbors** of them that are on **any** of their shortest paths. Specifically, let the distance between u and v be $d(u, v)$, we have the nodes on their shortest paths to be $W_{(u,v)} = \{w | d(u, w) + d(w, v) = d(u, v), \forall w \in \mathcal{V}\}$. The geodesics can be represented as,

$$P^{(V)}_{(u,v)} = P^{(V)}_{(u,v),u} \cup P^{(V)}_{(u,v),v}, \quad \text{where } P^{(V)}_{(u,v),i} := W \cap \mathcal{N}(i), i \in \{u, v\}. \tag{3}$$

Here $P^{(V)}_{(u,v),i}$ represents the nodes in the geodesic that are also the direct neighbors of i. For some tasks, we only consider one $P^{(V)}_{(u,v),i}$ as the geodesics instead of both. We use a pooling layer to generate vertical geodesic

$$\boldsymbol{g}^{(V)}_{(u,v)} = R^{(V)}_{gd}(\{\boldsymbol{h}_w, \forall w \in P^{(V)}_{(u,v)}\}). \tag{4}$$

While the vertical geodesic does not explicitly encode a long shortest path, it can still capture short shortest path information due to pooling the embeddings of the nodes in it. Compared with the horizontal geodesic that takes one random shortest path, the vertical geodesic focuses more on breadth by incorporating all direct neighbors on any shortest paths.

Unlike nodes in horizontal geodesic, nodes in vertical geodesic induce a more complex subgraph. Consider only $P^{(V)}_{(u,v),v}$ (we will use P_v here for simplicity) and the subgraph $\mathcal{G}_v$ induced by nodes in P_v. We can have a pooling function which encodes the graph structure of $\mathcal{G}_v$,

$$\boldsymbol{g}^{(V)}_{v} = R^{(G)}_{gd}(\{\boldsymbol{h}_w, \forall w \in P_v\}, \mathcal{G}_v). \tag{5}$$

The pooling layer can be a small plain GNN itself, and the initial embedding of $\mathcal{G}_v$ can take the pooled node embeddings from the main GNN. However, to keep our method efficient, we only add the degree of nodes in the subgraph to the node embeddings. And the vertical geodesic representation becomes,

$$\boldsymbol{g}^{(V)}_{v} = R^{(V)}_{gd}(\{\boldsymbol{h}_w \oplus deg_{\mathcal{G}_v}(w), \forall w \in P_v\}), \tag{6}$$

where $\oplus$ means concatenation. Note that a node's degree in the subgraph is different from that in the main graph. This can be seen as a 1-layer GNN applied onto $\mathcal{G}_v$ with identical initial node embeddings. Even with this simple pooling design, the vertical geodesic is able to help distinguish some distance regular graphs, which we cover in section 3.4.

Because vertical geodesics do not encode distance directly, we concatenate distance to the vertical geodesics as an additional feature. Like in horizontal geodesics, when $d(u, v)$ is greater than d_{max}, we do not generate geodesic representation for the pair, and set their distance to infinity.

3.3 Task specific geodesic representation

Task specific geodesic representation is a collection of pair-wise geodesic representations. Here, we focus on three levels of tasks, node-level, edge-level, and graph-level.

For **node-level** tasks, the goal is to generate a representation $\boldsymbol{z}_v^{(n)}$ for a target node v. We first find v's k-hop neighbor nodes $S_v = \mathcal{N}^k(v)$. Then for each node s_i in S_v, we find the pair-wise geodesic, $\boldsymbol{g}_{v,s_i}$. Note that in node level tasks, vertical geodesic only contains the geodesic nodes close to s_i, $P^{(V)}_{(v,s_i),s_i}$. Then the geodesic representations of the neighborhood is combined by a pooling layer,

$$\boldsymbol{z}_v^{(n)} = R^{(n)}(\boldsymbol{g}_{(v,s_i)}|s_i \in S_v). \tag{7}$$

While the GNN node embeddings of the neighborhood is not conditioned on v, by adding the geodesic information to every neighbor, we inject the conditional information of v to its neighbors. Then the collective information of the neighbors helps distinguish v.

For **edge-level** tasks, the goal is to generate an edge embedding $\boldsymbol{z}^{(e)}_{(u,v)}$ for nodes u and v. Here, we directly use the pair-wise geodesic for the two nodes, $\boldsymbol{z}^{(e)}_{(u,v)} = \boldsymbol{g}_{(u,v)}$. As pointed out by previous works [56, 52], the advantage of this setup is that, when the downstream task is to rank one link against N other links with one same node v, we only need to compute the distance vector from v once and find the geodesic nodes efficiently.

For **graph-level** tasks, we need to generate a graph embedding $\boldsymbol{z}_{\mathcal{G}}^{(g)}$ for $\mathcal{G}=(\mathcal{V},\mathcal{E})$. We use another pooling layer to summarize the geodesic-augmented representation of every node in the graph,

$$\boldsymbol{z}_{\mathcal{G}}^{(g)} = R^{(g)}(\boldsymbol{z}_v|v \in \mathcal{V}). \tag{8}$$

GD-GNN can also be easily generalized to other levels of tasks such as subgraph-level.

3.4 Expressive Power of GD-GNN

Distinguishing power on graph-level tasks is often used to characterize the expressiveness of a GNN model. For GNN models to be more expressive than 1-WL test, we can test whether they are able to distinguish regular graphs. In the following theorem, we show that given an injective $R^{(g)}$, the graph embedding $\boldsymbol{z}_G$ generated based on vertical geodesics can distinguish most regular graphs.

Theorem 1. *Let the graph pooling layer $R^{(g)}$ be injective given input from a countable space. Consider all pairs of n-sized r-regular graphs, where $3 \leq r < \sqrt{2\log n}$, n is the number of nodes in the graph. For any small constant $\epsilon > 0$, there exists a GD-GNN using vertical geodesic representations with $d_{max} = \lceil(\frac{1}{2}+\epsilon)\frac{\log n}{\log(r-1-\epsilon)}\rceil$ which distinguishes almost all $(1-o(1))$ such pairs of graphs.*

We include the proof of Theorem 1 in Appendix A. This theorem implies that GD-GNN with vertical geodesics is more expressive than basic GNNs (and 1-WL) and can discriminate almost all r-regular graphs. Moreover, GD-GNN only needs to search a small neighborhood for geodesics to achieve the higher expressiveness. This guarantees the good efficiency and performance of GD-GNN.

Moreover, when the vertical geodesic pooling layer encodes the subgraph induced by the geodesic nodes (using Equation 6 for geodesic pooling), GD-GNN is able to distinguish some distance regular graphs. Figure 4 shows two distance regular graphs, Shrikhande graph and 4x4 rook's graph. They have the same intersection array and number of nodes. As proved by Li *et al.* [23], distance encoding is not able to distinguish this pair of graphs, because nodes in the two graphs will be assigned the same embedding. However, vertical geodesic is able to discriminate them. Suppose the black nodes are the target nodes, the brown nodes are one distance away from the black nodes and the light blue nodes are two distance away from the black nodes. Consider the light blue nodes in both graphs, they each have a pair of brown common neighbors with a black node. The brown pair is the vertical geodesics of one light blue node. If we count the number of edges in the subgraph formed by each brown pair, we will see that four out of nine pairs form a one-edge graph (in red) in the Shrikhande graph, while zero out of nine pairs form a one-edge graph in the 4x4 rook's graph. Since Equation 6 directly captures this and will give four light blue nodes in Shrikhande graph different pair-wise geodesics with respect to the black node, an injective graph pooling layer will yield different node representations for the two black nodes, and in turn discriminates the two graphs.

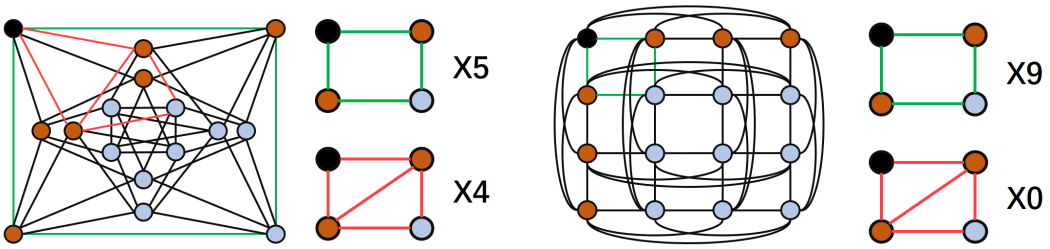


Figure 4: An example of two graphs where GD-GNN with vertical geodesics and subgraph degrees can distinguish, while distance encoding cannot.

Complexity. The main computation advantage of GD-GNN comes from the fact that, for GD-GNN, the number of GNN runs stays as a constant of 1, while subgraph-based methods incur GNN runs which increases linearly with respect to the number of queries. Hence, GD-GNN maintains the good efficiency of basic GNN methods, while injecting the high-expressiveness of subgraph methods into the model. A detailed complexity analysis on different tasks can be found in Appendix B.

Limitation. While a simple geodesic pooling layer satisfies the theoretical expressiveness, if we indeed use a GNN or other more complex layer for pooling as suggested in Section 3.2.2, the pooling will dominate the good amortized complexity of GDGNN. We see this as a possible future direction where we can use a more complex pooling layer to further improve the expressiveness while maintaining the efficiency.

4 Related Works

GNNs beyond 1-WL test. Designing GNNs with stronger representation power is a fundamental task in GNN research. Xu *et al.* [44] first proved that the expressiveness of basic GNNs, such as GCN [18], GraphSAGE [15], and GAT [39], is upper-bounded by the 1-WL test. Following works conducted extensive research to break this limit. They can be sorted into several directions: (1) Earlier works proposed higher-order GNNs that are as expressive as k-WL algorithm, such as k-GNN [21], RingGNN[6], and PPGN[24]. These methods require message passing on node tuples that grows exponentially with respect to k. The increased computation cost makes them not scalable to large graphs. (2) Another type of methods, like GSN [5] and MotifNet [25], incorporates isomorphism counting on certain graph substructures like cycles, triangles, and cliques, similar to earlier kernel-based methods [45, 12]. They essentially augment the models with features undetectable by GNNs. However, a properly-design substructure usually requires expert knowledge which is not available for all data. (3) Some other works use identity or random node labeling within the plain GNN framework. ID-GNN [48] extracts a subgraph around the target node and perform heterogeneous message passing. DE-GNN [23] also requires subgraph extraction but labels nodes in the subgraph with distance to the targets. They both have stronger expressiveness than basic GNNs. Abboud *et al.* [1] shows that GNNs possess universality with random node feature initialization (RNI). Nevertheless, RNI also induces a huge sample space, and GNNs adopting RNI is very hard to train. (4) Some recent works propose to use additional subgraph structures to augment the node embeddings. GraphSNN [42] incorporates properties of overlapping subgraphs between adjacent nodes to be more expressive than 1-WL. Nested-GNN [51] applies GNN to ego-subgraphs of every node and proves that it is able to discriminate regular graphs. Sharing the same idea of Nested-GNN, GNN-AK[54] uses a different subgraph pooling strategy and uses subgraph dropping to improve the model scalability.

Link prediction using GNNs. Earlier works [19, 18, 7, 38] applied basic GNNs to link prediction using an auto-encoder framework, where the GNN serves as the encoder of the nodes, and edges are decoded by their nodes' encoding vectors. Another line of works, including SEAL [49], IGMC [50], and GraIL [35], use GNNs to encode a labelled subgraph to form the link representation. Later work [33, 53] reveals that a good node-level/graph-level representation does not necessarily lead to a good edge-level representation. Zhang *et al.* [53] systematically show that the labeling trick bridges the gap between node-level and edge-level GNN expressiveness. Sharing the spirits of labeling trick, NBFNet [56] extends the generalized Bellman-Ford algorithm to the neural setting, and can encode many traditional path-based methods. NBFNet has a better scalability when there are multiple links sharing one same node. Some other works also propose to use path information between links to enhance the prediction like horizontal geodesic. Jagvaral *et al.* [17] uses CNN and BiLSTM to summarize the

path information, however, their model requires fixed node embedding, while horizontal geodesic is based on graph structure generated from GNN and hence is inductive. Wang *et al.* [41] learns relational path information by training a set of embeddings for all different types of relational path, while horizontal GD-GNN uses GNN to capture the graph structure and pool node embeddings along the geodesic without training relational-path-specific embeddings.

5 Experimental Results

5.1 Link Prediction

Datasets. We use several types of datasets for link prediction. (1) Knowledge Graph (KG) inductive link prediction datasets. Inductive link prediction means that the model is trained on a training graph while tested on a different graph. This requires the model to generalize to unseen entities, which is a property preserved by a limited number of models. We follow the standard inductive split of WN18RR[10] and FB15K237[37] as in [35]. We rank each positive link against 50 randomly sampled negative links with the same head/tail as the positive link, and report Hit@10 ratio. (2) OGB large-scale link prediction dataset [16], including OGBL-COLLAB and OGBL-PPA. We use the official split provided by the OGB team. We rank each positive link against a set of provided negative links in the dataset, and report Hit@N ratio. (3) Results of citation datasets and comparison to other distance-based methods can be found in Appendix D. (4) Full results on the (KG) datasets are included in Appendix C. These datasets vary by graph size and demonstrate GD-GNN's capability across scales.

Baselines. We compare GD-GNN with Rules Mining methods, DRUM [29] and NeuralLP [46], and GNN-based methods, NBFNet [56] and GraIL [35] for the KG tasks. Some other works like [40, 8] incorporate text data in the KG to achieve better performance. As we believe models that employ textual information is beyond the context of graph structural learning, we exclude them in our comparison. For the OGB dataset, we compare GD-GNNs with other GNN-based methods achieving top places on the OGB link prediction leaderboard, including Deep Walk [27], Resource Allocation (RA) [55], SEAL, Deeper GCN [22].

Implementation Details. For link prediction, we use GCN (RGCN for KG) as the basic GNN. For all the datasets, we search the number of GNN layers in $\{2, 3, 4, 5\}$, and the max search distance for geodesic, d_{max}, is same as the GNN layers. We use 32 hidden dimensions for all fully-connected layers in the model. We train 50 epochs with a batch size of 64 for the KG datasets, and we train 50 epochs with a batch size of 4096 for the OGB datasets. We record the test score using the model with the best set of hyperparameters chosen based on the performance on the validation set. The experiment is repeated 5 times and we take the average of the scores.

Table 1: Link prediction results (%) on KG datasets.

Method	FB15K-237		WN18RR	
	v1	v3	v1	v3
DRUM	52.9	52.9	74.4	46.2
NeuralLP	52.9	52.9	74.4	46.2
GraIL	64.2	82.8	82.5	58.4
NBFNet	83.4	95.1	94.8	89.3
GD-GNN-Vert	**85.4**	**98.2**	**95.7**	87.7
GD-GNN-Hor	**85.4**	93.4	93.1	**89.5**

Table 2: Link prediction results (%) on OGB datasets.

	OBGL-COLLAB (H@50)	OGBL-PPA (H@100)
GCN	44.75 ± 1.45	18.67 ± 1.32
DEEP WALK	50.37 ± 0.34	28.88 ± 1.63
RA	-	**49.33** ± 0.00
DeeperGCN	52.73 ± 0.47	-
SEAL	54.37 ± 0.49	48.80 ± 3.16
GD-GNN-Vert	**54.68** ± 0.51	45.61 ± 2.58
GD-GNN-Hor	51.42 ± 0.72	32.17 ± 3.70

Results and discussion. We achieved SOTA performance on the inductive datasets. GD-GNN is able to significantly improve over rule-mining methods and subgraph based method, GraIL. We also outperform NBFNet on most of the datasets. On the OGBL-COLLAB dataset (Table 2), vertical GD-GNN achieved SOTA performance. On the OGBL-PPA dataset, GD-GNN is able to keep a very competitive result. On both OGB datasets, GD-GNN improves significantly over its basic GNN, GCN. Moreover, as shown in Table 3, GD-GNN has running-time superiority over methods achieving comparable performance, and has memory advantage to handle large graphs where NBFNet can not. To make fair running time comparison, we run all models on 32 CPUs and 1 Nvidia GeForce 1080Ti GPU. We also include ablation studies on all components of GD-GNN in Appendix F.

Table 3: Link prediction running time results.

Method	OGBL-COLLAB	OGBL-PPA
GCN	27 sec	17 min
SEAL	90 sec	2 hr 20 min
NBFNet	OOM	OOM
GD-GNN-Vert	36 sec	33 min

Table 4: Graph classification running time results.

Method	OGBG-MOLHIV	OGBG-MOLPCBA
GIN	20 sec	2 min
GIN-AK	70 sec	12 min
Nested-GIN	149 sec	16 min
GD-GNN-Vert	25 sec	2.5 min

5.2 Graph Classification

Datasets. We use two types of datasets for graph classification. (1) TU datasets contain D&D[11], MUTAG[9], PROTEINS[11], PTC_MR[36], ENZYMES[31]. The evaluation metric is Accuracy (%). We follow the evaluation protocol in [51], where we use 10-fold cross-validation to compute the score. (2) OGB datasets[16], including OGBG-MOLHIV and OGBG-MOLPCBA. The task of OGBG-MOLHIV dataset is to predict whether a molecule inhibits HIV virus. We use AUC to evaluate OGBG-MOLHIV dataset. OGBG-MOLPCBA dataset has 128 classification tasks. We follow the evaluation protocol provided by the OGB team and use AP averaged across all classes to evaluate the dataset. (3) Synthetic datasets, including EXP[1] and CSL[26]. These datasets contains graphs that can be only distinguished by model with $>$ 1-WL expressiveness.

Baselines. For the TU dataset, we compare with another plug-and-play GNN framework, NGNN[51]. Both GD-GNN and NGNN use GIN and GCN as their basic GNNs. For the OGB dataset, we use GIN as the basic GNN, we compare GD-GNN with methods achieving top places on the OGB leaderboard, including GIN[44], GSN[5], GIN-AK [54], Nested GNN [51], Directional GSN [3], PF-GNN [13].

Implementation details. We use vertical geodesic in the graph-level tasks because of its theoretical advantage over horizontal GD-GNN. For TU datasets, we search the GNN layers and d_{max} in {2,3,4}. We train on the TU dataset for 100 epochs with a batch size of 64. For OGB datasets, we search the number of GNN layers in {2,3,4,5}, and train for 100 epochs with a batch size of 256. We adopt virtual node technique [14] to increase the connectivity of graphs for the OGBG-MOLPCBA dataset. For synthetic datasets, we search the number of GNN layers in {2,3,4}, and train for 50 epochs with a batch size of 16.

Table 5: Synthetic Results

	EXP	CSL
GIN	50.0	10.0
RNI	99.7	16.0
PF-GNN	100.0	100.0
2-GDGNN	50.0	30.0
3-GDGNN	100.0	60.0
4-GDGNN	100.0	100.0

Table 6: Graph classification results (%) on TU datasets.

	D&D	MUTAG	PROTEINS	PTC_MR	ENZYMES
Avg. #nodes	284.32	17.93	39.06	14.29	32.63
GCN	71.6 ± 2.8	73.4 ± 10.8	71.7 ± 4.7	56.4 ± 7.1	27.3 ± 5.5
GIN	71.6 ± 3.0	74.0 ± 8.8	71.2 ± 5.2	57.0 ± 5.5	30.7 ± 6.3
Nested GCN	76.3 ± 3.8	82.9 ± 11.1	73.3 ± 4.0	57.3 ± 7.7	31.2 ± 6.7
Nested GIN	**77.8** ± 3.9	87.9 ± 8.2	**73.9** ± 5.1	54.1 ± 7.7	29.0 ± 8.0
GD-GCN-Vert	77.6 ± 4.0	88.4 ± 6.6	73.7 ± 3.4	**60.3** ± 4.5	**32.3** ± 5.1
GD-GIN-Vert	**77.8** ± 3.6	**89.4** ± 7.1	73.6 ± 2.5	57.9 ± 3.4	31.6 ± 3.3

Table 7: Graph classification results (%) on OGB datasets.

Method	OGBG-MOLHIV (AUC)		OGBG-MOLPCBA (AP)	
	Validation	Test	Validation	Test
GIN	82.32 ± 0.90	75.58 ± 1.40	27.98 ± 0.25	27.03 ± 0.23
GSN	86.58 ± 0.84	77.99 ± 1.00	-	-
GIN-AK	-	78.22 ± 0.75	30.47 ± 0.07	**29.30** ± 0.44
Nested GIN	83.17 ± 1.99	78.34 ± 1.86	29.15 ± 0.35	28.32 ± 0.41
Directional GSN	84.73 ± 0.96	**80.39** ± 0.90	-	-
PF-GNN	-	80.15 ± 0.68	-	-
GD-GIN-Vert	82.41 ± 1.13	78.43 ± 1.02	29.09 ± 0.38	28.17 ± 0.34

Results and discussions. On the TU dataset (Table 6), GD-GNNs consistently outperform the basic GNNs. Compared to NGNN, GD-GNN is able to bring larger performance increase to the basic GNNs. This shows GD-GNN's enhancing power. On the OGB dataset (Table 7), we can see that GD-GNN's performance is lower than PF-GNN and Directional GSN (so are GIN-AK and NGNN). This might indicate that pairwise conditional information is less effective for graph tasks than for link tasks. Nevertheless, GD-GNN still outperforms the basic GIN a lot, and as GD-GNN is a general framework that can be applied to any basic GNN, the performance might be able to be further improved. On the synthetic data, we achieve comparable results to the universal PF-GNN, which verifies our theory. We include ablation studies of graph-level geodesics in Appendix F.

We've also included experiments on node-level tasks in Appendix E and experiments on the impact of cutoff distance in Appendix G

6 Conclusions

In this paper, we propose a general GNN framework which incorporates geodesic information to generate node, link, and graph representations. Our framework is more powerful than basic GNNs both practically and theoretically and is more efficient than current more-expressive GNNs as demonstrated by the running-time experiments. Our framework has the potential to make GNNs encoding conditional information really applicable to large-scale real-world graphs.

References

[1] Ralph Abboud, Ismail Ilkan Ceylan, Martin Grohe, and Thomas Lukasiewicz. The surprising power of graph neural networks with random node initialization. *arXiv preprint arXiv:2010.01179*, 2020.

[2] Siddhant Arora. A survey on graph neural networks for knowledge graph completion. *arXiv preprint arXiv:2007.12374*, 2020.

[3] Dominique Beaini, Saro Passaro, Vincent Létourneau, Will Hamilton, Gabriele Corso, and Pietro Liò. Directional graph networks. In *International Conference on Machine Learning*, pages 748–758. PMLR, 2021.

[4] Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. *Advances in neural information processing systems*, 26, 2013.

[5] Giorgos Bouritsas, Fabrizio Frasca, Stefanos P Zafeiriou, and Michael Bronstein. Improving graph neural network expressivity via subgraph isomorphism counting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2022.

[6] Zhengdao Chen, Soledad Villar, Lei Chen, and Joan Bruna. On the equivalence between graph isomorphism testing and function approximation with gnns. *Advances in neural information processing systems*, 32, 2019.

[7] Tim R Davidson, Luca Falorsi, Nicola De Cao, Thomas Kipf, and Jakub M Tomczak. Hyper-spherical variational auto-encoders. *arXiv preprint arXiv:1804.00891*, 2018.

[8] Daniel Daza, Michael Cochez, and Paul Groth. Inductive entity representations from text via link prediction. In *Proceedings of the Web Conference 2021*, pages 798–808, 2021.

[9] Asim Kumar Debnath, Rosa L Lopez de Compadre, Gargi Debnath, Alan J Shusterman, and Corwin Hansch. Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. correlation with molecular orbital energies and hydrophobicity. *Journal of medicinal chemistry*, 34(2):786–797, 1991.

[10] Tim Dettmers, Pasquale Minervini, Pontus Stenetorp, and Sebastian Riedel. Convolutional 2d knowledge graph embeddings. In *Thirty-second AAAI conference on artificial intelligence*, 2018.

[11] Paul D. Dobson and Andrew J. Doig. Distinguishing enzyme structures from non-enzymes without alignments. *Journal of Molecular Biology*, 330(4):771–783, 2003.

[12] Simon S Du, Kangcheng Hou, Russ R Salakhutdinov, Barnabas Poczos, Ruosong Wang, and Keyulu Xu. Graph neural tangent kernel: Fusing graph neural networks with graph kernels. *Advances in neural information processing systems*, 32, 2019.

[13] Mohammed Haroon Dupty, Yanfei Dong, and Wee Sun Lee. Pf-gnn: Differentiable particle filtering based approximation of universal graph representations. In *International Conference on Learning Representations*, 2021.

[14] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *International conference on machine learning*, pages 1263–1272. PMLR, 2017.

[15] William L Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 1025–1035, 2017.

[16] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. *Advances in neural information processing systems*, 33:22118–22133, 2020.

[17] Batselem Jagvaral, Wan-Kon Lee, Jae-Seung Roh, Min-Sung Kim, and Young-Tack Park. Path-based reasoning approach for knowledge graph completion using cnn-bilstm with attention mechanism. *Expert Systems with Applications*, 142:112960, 2020.

[18] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[19] Thomas N Kipf and Max Welling. Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308*, 2016.

[20] AA Leman and Boris Weisfeiler. A reduction of a graph to a canonical form and an algebra arising during this reduction. *Nauchno-Technicheskaya Informatsiya*, 2(9):12–16, 1968.

[21] Yujia Li, Chenjie Gu, Thomas Dullien, Oriol Vinyals, and Pushmeet Kohli. Graph matching networks for learning the similarity of graph structured objects. In *International conference on machine learning*, pages 3835–3845. PMLR, 2019.

[22] Guohao Li, Chenxin Xiong, Ali Thabet, and Bernard Ghanem. Deepergcn: All you need to train deeper gcns. *arXiv preprint arXiv:2006.07739*, 2020.

[23] Pan Li, Yanbang Wang, Hongwei Wang, and Jure Leskovec. Distance encoding: Design provably more powerful neural networks for graph representation learning. *Neural Information Processing Systems (NeurIPS)*, 2020.

[24] Haggai Maron, Heli Ben-Hamu, Hadar Serviansky, and Yaron Lipman. Provably powerful graph networks. *Advances in neural information processing systems*, 32, 2019.

[25] Federico Monti, Karl Otness, and Michael M Bronstein. Motifnet: a motif-based graph convolutional network for directed graphs. In *2018 IEEE Data Science Workshop (DSW)*, pages 225–228. IEEE, 2018.

[26] Ryan Murphy, Balasubramaniam Srinivasan, Vinayak Rao, and Bruno Ribeiro. Relational pooling for graph representations. In *International Conference on Machine Learning*, pages 4663–4673. PMLR, 2019.

[27] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. *CoRR*, abs/1403.6652, 2014.

[28] Deep learning in protein structural modeling and design. *Patterns*, 1(9), December 2020.

[29] Ali Sadeghian, Mohammadreza Armandpour, Patrick Ding, and Daisy Zhe Wang. DRUM: end-to-end differentiable rule mining on knowledge graphs. *CoRR*, abs/1911.00055, 2019.

[30] Michael Schlichtkrull, Thomas N. Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks, 2017.

[31] Ida Schomburg, Antje Chang, Christian Ebeling, Marion Gremse, Christian Heldt, Gregor Huhn, and Dietmar Schomburg. Brenda, the enzyme database: updates and major new developments. *Nucleic acids research*, 32(suppl_1):D431–D433, 2004.

[32] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. Collective classification in network data. *AI magazine*, 29(3):93–93, 2008.

[33] Balasubramaniam Srinivasan and Bruno Ribeiro. On the equivalence between positional node embeddings and structural graph representations. In *International Conference on Learning Representations*, 2019.

[34] Zhiqing Sun, Zhi-Hong Deng, Jian-Yun Nie, and Jian Tang. Rotate: Knowledge graph embedding by relational rotation in complex space. In *International Conference on Learning Representations*, 2019.

[35] Komal Teru, Etienne Denis, and Will Hamilton. Inductive relation prediction by subgraph reasoning. In *International Conference on Machine Learning*, pages 9448–9457. PMLR, 2020.

[36] Hannu Toivonen, Ashwin Srinivasan, Ross King, Stefan Kramer, and Christoph Helma. Statistical evaluation of the predictive toxicology challenge 2000-2001. *Bioinformatics (Oxford, England)*, 19:1183–93, 08 2003.

[37] Kristina Toutanova and Danqi Chen. Observed versus latent features for knowledge base and text inference. In *Proceedings of the 3rd workshop on continuous vector space models and their compositionality*, pages 57–66, 2015.

[38] Shikhar Vashishth, Soumya Sanyal, Vikram Nitin, and Partha Talukdar. Composition-based multi-relational graph convolutional networks. In *International Conference on Learning Representations*, 2019.

[39] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations*, 2018.

[40] Bo Wang, Tao Shen, Guodong Long, Tianyi Zhou, Ying Wang, and Yi Chang. Structure-augmented text representation learning for efficient knowledge graph completion. In *Proceedings of the Web Conference 2021*, pages 1737–1748, 2021.

[41] Hongwei Wang, Hongyu Ren, and Jure Leskovec. Relational message passing for knowledge graph completion. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pages 1697–1707, 2021.

[42] Asiri Wijesinghe and Qing Wang. A new perspective on" how graph neural networks go beyond weisfeiler-lehman?". In *International Conference on Learning Representations*, 2021.

[43] Shiwen Wu, Fei Sun, Wentao Zhang, Xu Xie, and Bin Cui. Graph neural networks in recommender systems: a survey. *ACM Computing Surveys (CSUR)*, 2020.

[44] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2018.

[45] Pinar Yanardag and SVN Vishwanathan. Deep graph kernels. In *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1365–1374, 2015.

[46] Fan Yang, Zhilin Yang, and William W Cohen. Differentiable learning of logical rules for knowledge base reasoning. In *NIPS*, 2017.

[47] Jiaxuan You, Rex Ying, and Jure Leskovec. Position-aware graph neural networks. In *International conference on machine learning*, pages 7134–7143. PMLR, 2019.

[48] Jiaxuan You, Jonathan Gomes-Selman, Rex Ying, and Jure Leskovec. Identity-aware graph neural networks. *CoRR*, abs/2101.10320, 2021.

[49] Muhan Zhang and Yixin Chen. Link prediction based on graph neural networks. *Advances in Neural Information Processing Systems*, 31:5165–5175, 2018.

[50] Muhan Zhang and Yixin Chen. Inductive matrix completion based on graph neural networks. In *International Conference on Learning Representations*, 2020.

[51] Muhan Zhang and Pan Li. Nested graph neural networks. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021.

[52] Yongqi Zhang and Quanming Yao. Knowledge graph reasoning with relational digraph. In *Proceedings of the ACM Web Conference 2022*, pages 912–924, 2022.

[53] Muhan Zhang, Pan Li, Yinglong Xia, Kai Wang, and Long Jin. Labeling trick: A theory of using graph neural networks for multi-node representation learning. *Advances in Neural Information Processing Systems*, 34, 2021.

[54] Lingxiao Zhao, Wei Jin, Leman Akoglu, and Neil Shah. From stars to subgraphs: Uplifting any gnn with local structure awareness. *arXiv preprint arXiv:2110.03753*, 2021.

[55] Tao Zhou, Linyuan Lü, and Yi-Cheng Zhang. Predicting missing links via local information. *The European Physical Journal B*, 71(4):623–630, 2009.

[56] Zhaocheng Zhu, Zuobai Zhang, Louis-Pascal Xhonneux, and Jian Tang. Neural bellman-ford networks: A general graph neural network framework for link prediction. *arXiv preprint arXiv:2106.06935*, 2021.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes] We faithfully summarize the paper in the abstract.
   (b) Did you describe the limitations of your work? [Yes] As described in the Section 5 and Appendix C.
   (c) Did you discuss any potential negative societal impacts of your work? [No] Our work only involves existing datasets and solves general graph learning problem without negative societal impact.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes] Assumption is described in the Theorem 1
   (b) Did you include complete proofs of all theoretical results? [Yes] In Appendix A
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes]
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] As described in the implementation details of Section 5
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] As shown in the experiment result tables.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] In the discussion of Section 5
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] The datasets and model source code repositories are all cited.
   (b) Did you mention the license of the assets? [N/A]
   (c) Did you include any new assets either in the supplemental material or as a URL? [No]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [No] The datasets and source code are open source information allowed for research use.
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [No] The data we are using is known to be de-identified.
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

Appendix

A Proof of Theorem 1

The proof of Theorem 1 generally follows the proof of Nested-GNN's expressiveness[51]. Let $Q^k_{v,\mathcal{G}}$ be the set of nodes in $\mathcal{G}$ that are exactly k distance away from node v. We can have the following definition,

Definition A.1. *The edge configuration bewteen $Q^k_{v,\mathcal{G}}$ and $Q^{k+1}_{v,\mathcal{G}}$ is a list $C^k_{v,\mathcal{G}} = (a^{1,k}_{v,\mathcal{G}}, a^{2,k}_{v,\mathcal{G}}, ...)$ where $a^{i,k}_{v,\mathcal{G}}$ denotes the number of nodes in $Q^{k+1}_{v,\mathcal{G}}$ of which each has exactly i edges from $Q^k_{v,\mathcal{G}}$.*

The proof leverages the fact that nodes in randomly sampled regular graphs are very likely to have different edge configuration within their small neighborhood, and GD-GNN directly captures this distinction to discriminate the graphs. Formally, we have the following excerpted Lemma from [51],

Lemma 2. *For two graphs $\mathcal{G}^{(1)} = (\mathcal{V}^{(1)}, \mathcal{E}^{(1)})$ and $\mathcal{G}^{(2)} = (\mathcal{V}^{(2)}, \mathcal{E}^{(2)})$ that are uniformly independently sampled from all n-node r-regular graphs, where $3 \leq r < \sqrt{2\log n}$, we pick any two nodes, each from one graph, denoted by v_1 and v_2 respectively. Then there is at least one $i \in (\frac{1}{2}\frac{\log n}{\log(r-1-\epsilon)}, (\frac{1}{2}+\epsilon)\frac{\log n}{\log(r-1-\epsilon)})$ with probability $1 - o(n^{-1})$ such that $C^i_{v_1,\mathcal{G}^{(1)}} \neq C^i_{v_2,\mathcal{G}^{(2)}}$.*

Proof. To prove Theorem 1, we first assume, without loss of generality, that the hidden dimension of GD-GNN is one. We know plain message passing GNN will generate the same embeddings for every node in the regular graph, because all nodes have the same degree/subtree. Hence, we can normalize all the node representations to a constant of one. For simplicity, we also reduce vertical geodesic pooling function $R^{(V)}_{gd}$ in Equation 4 to a simple sum of the representations of the geodesic nodes.

Let $d_{max} = \lceil(\frac{1}{2}+\epsilon)\frac{\log n}{\log(r-1-\epsilon)}\rceil$, the maximum distance in which GD-GNN searches for a target node's neighbors. We then consider node v_i and it's d_{max}-hop rooted subgraph $\mathcal{G}^{d_{max}}_{v_i}$, the vertical geodesic pooling process is essentially counting the number of geodesic nodes of each node v_j in $\mathcal{G}_{v_i}$. And the pair-wise vertical geodesics between the v_i and v_j becomes,

$$\boldsymbol{g}^{(V)}_{v_i,v_j} = |\{P^{(V)}_{v_j}\}| \oplus d(v_i, v_j) \tag{9}$$

where $d(v_i, v_j)$ is the distance between v_i and v_j, and hence $v_j \in Q^{d(v_i,v_j)}_{v_i,\mathcal{G}_{v_i}}$ and $\boldsymbol{g}^{(V)}_{v_i,v_j}$ records the number of edges of v_j that connects to nodes in $Q^{d(v_i,v_j)-1}_{v_i,\mathcal{G}_{v_i}}$, and its distance to v_i. We also have the node-level pooling function to be a distance sorted bin count function on the set of pair-wise vertical geodesic representations. It first sorts the geodesic representations by their second dimension (distance), and perform bin count on their first dimension (number of geodesic nodes) separately for each distance. Then GD-GNN produces an mapping of $\mathcal{G}_{v_i}$ to its exact set of edge configurations, $\mathcal{C}_{v_i} = (C^1_{v_i,\mathcal{G}_{v_i}}, C^2_{v_i,\mathcal{G}_{v_i}}, ...)$.

Let $\mathcal{G}^{(1)} = (\mathcal{V}^{(1)}, \mathcal{E}^{(1)})$ and $\mathcal{G}^{(2)} = (\mathcal{V}^{(2)}, \mathcal{E}^{(2)})$ be two graphs uniformly independently sampled from all n-node r-regular graphs. We consider two nodes $v_i \in \mathcal{V}^{(1)}$ and $v_j \in \mathcal{V}^{(2)}$. Because $d_{max} = \lceil(\frac{1}{2}+\epsilon)\frac{\log n}{\log(r-1-\epsilon)}\rceil$, the sets of edge configurations of the rooted subgraphs, $\mathcal{C}_{v_i}$ and $\mathcal{C}_{v_j}$ are different with probability $1 - o(n^{-1})$, according to Lemma 2. In such case, GD-GNN also maps their rooted subgraphs to different edge configurations, and hence the node-level geodesic representation $\boldsymbol{z}^{(n)}_{v_i}$ and $\boldsymbol{z}^{(n)}_{v_j}$ generated by GN-GNN will be different with the same probability.

Consider v_i and all node $v_j \in \mathcal{V}^{(2)}$, by union bound, $\boldsymbol{z}_{v_i} \notin \{\boldsymbol{z}_{v_j} | v_j \in \mathcal{V}^{(2)}\}$ with a probability of $1 - o(1)$. □

B Complexity Analysis

The complexity of GD-GNN differs by tasks. On almost all tasks, GD-GNN demonstrates superiority in efficiency. The complexity of GD-GNN is formed by two parts, the geodesic extraction part and the GNN part. We first note that the complexity of geodesic extraction and GNN cannot be directly compared as their computation units are different, geodesic extraction operates on unit integers

while GNN operates on d-dimensional vectors that are potentially very large, and running a geodesic extraction on a graph takes significantly less amount of time compared to running a GNN on the same graph. Meanwhile, we can always pre-compute geodesics which allows fast extraction during inference. Hence, here we present complexity analysis of GD-GNN on the GNN part compared to other methods.

B.1 Graph and node level tasks

The analysis of graph and node level tasks is similar as graph representation is generated by taking all node representation. Specifically, consider the problem where we need to infer k nodes in a graph with $|V|$ nodes and $|E|$ edges. In worst-case, the complexity for a T-layer subgraph-based GNN is $O(kT|E|)$, because it applies a T-layer GNN on k nodes' subgraphs, where each subgraph contains $|E|$ edges in the worst case (the full graph). GD-GNN's worst-case complexity is $O(T|E| + k|E|)$, $O(T|E|)$ is for applying the GNN once to the full graph, and the number of nodes in a vertical geodesic is bounded by the number of edges and hence the pooling takes $O(k|E|)$. Hence, GD-GNN is more efficient than subgraph GNNs. If we consider GNN's layer complexity w.r.t. the embedding dimension d (mostly quadratic due to linear transformation/MLP over node embeddings, e.g. GCN and GIN), we will have $O(kT|E|d^2)$ for subgraph GNNs, but $O(T|E|d^2 + k|E|d)$ for GD-GNN (the geodesic pooling is linear to d). Thus, the efficiency advantage of GD-GNN is more significant as shown in our experiments. For graphs, the comparison becomes $O(|V|T|E|)$ versus $O(T|E|+|V||E|) = O(|V||E|)$, this shows that GD-GNN helps amortize the expensive computation cost, T versus 1, of applying a GNN.

B.2 Edge level tasks

For edge level tasks, we consider the worst-case scenario where we try to predict k links on a graph with $|V|$ nodes and $|E|$ edges. The k links do not share any common nodes. GDGNN takes $O(T|E| + k|V|)$ for $O(T|E|)$ GNN on the full graph and k geodesic pooling, where the number of nodes in a link-level geodesic is bounded by the number of nodes $O(|V|)$ in the graph. Subgraph GNN methods takes $O(kT|E|)$ for applying T-layer GNN onto k links' subgraphs (worst case $O(|E|)$ edges). NBFNet shares the same complexity $O(kT|E|)$ because the links do not share common nodes, and the GNN results can not be shared across links as suggested by the authors of NBFNet. Thus, k runs of GNN is necessary for NBFNet. Then, comparing the two complexity, we can see that when $O(T|E|) > O(k|V|)$, subgraph method's complexity $O(kT|E|)$ grows linearly with respect to the number of queries, while GDGNN's complexity $O(T|E|)$ does not. The practical running time of $O(T|E|)$ and $O(k|V|)$ are not on the same level as the GNN is actually quadratic to the embedding dimension d while the geodesic pooling is only linear to d. Therefore, in GD-GNN, the more expensive GNN layer complexity $O(Te)$ does not increase linearly w.r.t. query number k, which significantly saves the computational time compared to NBFNet or SEAL.

C Knowledge graph link prediction

For knowledge graphs, we use RGCN[30] as our base GNN. We evaluate GD-GNN on FB15K237[37] and WN18RR[10] knowledge graphs for both inductive and transductive link prediction tasks. We use the standard split in [37] for transductive link prediction, and follow [35] for inductive link prediction. We compare with Rules Mining methods, DRUM [29] and NeuralLP [46], and GNN-based methods, NBFNet [56] and GraIL [35], for the inductive link prediction tasks. For transductive tasks, we additionally compared to general knowledge graph link prediction methods that are not inductive, including TransE [4], RotatE [34], and RGCN [30]. For inductive dataset, we randomly sample 50 negative links that have either the same head or tail as the positive link, we report the H@10 against the 50 negative links. For the transductive dataset, we follow the filtered ranking protocol [4], where we rank a positive link against all negative links that have the same head (or tail) as the positive links. We report H@N and Mean Reciprocal Rank (MRR) for the transductive setting.

We achieved SOTA performance on the inductive datasets. GD-GNN is able to significantly improve over rule-mining methods and subgraph based method, GraIL. We also outperform NBFNet on most of the datasets. We note that horizontal geodesic is able to exceed the performance of vertical geodesic on some datasets. We suspect this is because horizontal geodesics implicitly capture the inter-node translation process as in the embedding methods, and this piece of information is more important

Table 8: Knowledge graph inductive link prediction results.

Method	FB15K-237				WN18RR			
	v1	v2	v3	v4	v1	v2	v3	v4
DRUM	52.9	58.7	52.9	55.9	74.4	68.9	46.2	67.1
NeuralLP	52.9	58.9	52.9	55.9	74.4	68.9	46.2	67.1
GraIL	64.2	81.2	82.8	89.3	82.5	78.7	58.4	73.4
NBFNet	83.4	94.9	95.1	96.0	94.8	**90.5**	89.3	**89.0**
GD-GNN-Vert	**85.4**	95.6	**98.2**	**97.8**	**95.7**	89.8	87.7	85.6
GD-GNN-Hor	**85.4**	**96.7**	93.4	**97.7**	93.1	88.0	**89.5**	88.5

Table 9: Knowledge graph transductive link prediction results.

Method	FB15K-237				WN18RR			
	MRR	H@1	H@3	H@10	MRR	H@1	H@3	H@10
DRUM	34.3	25.5	37.8	51.6	48.6	42.5	51.3	58.6
NeuralLP	24.0	-	-	36.2	43.5	37.1	43.4	56.6
TransE	29.4	-	-	46.5	22.6	-	-	50.1
RotatE	33.8	24.1	37.5	55.3	47.6	42.8	49.2	57.1
RGCN	27.3	18.2	30.3	45.6	40.2	34.5	43.7	49.4
NBFNet	**41.5**	**32.1**	**45.4**	**59.9**	**55.1**	**49.7**	**57.3**	**66.6**
GD-GNN-Vert	23.2	15.8	26.4	45.1	46.2	39.3	48.6	59.1
GD-GNN-Hor	25.1	16.2	28.7	44.9	43.2	35.2	47.2	58.0

than graph structure in the knowledge graph setting. We also acknowledge that GD-GNN does not perform as well in the transductive setting. However, GD-GNN is still able to greatly improve the performance of RGCN on WN18RR dataset, and RGCN requires trainable node embeddings while GD-GNN does not. We suspect the performance discrepancy between the homogeneous setting and the knowledge graph setting is due to the increased number of edge types. WN18RR dataset has 11 types of relations, FB15K237 has 237. Note that GNN methods like NBFNet[56] train a set of weights for every target relation type, meaning that their message passing is conditioned on the relation, while GD-GNN only uses one set of weights. We notice that GD-GNN is able to outperform some embedding methods on WN18RR dataset, while not performing as good on FB15K237, possibly due to the fact that FB15K237 dataset contains much more relations than WN18RR dataset. This aligns with our hypothesis. A potential solution is to also train multiple GD-GNNs that each handles one target relation type, note that we still only need to run each relation-specific GD-GNN once, and keep the good amortized property of GD-GNN. We leave this to future work.

D More link prediction results

Citation datasets[32] include Cora, Citeseer, and Pubmed datasets. We follow the experimental setting in [56] and use 85:5:10 for the train/valid/test links split. The evaluation metric is ROC-AUC (AUC) and Average Precision (AP) scores. We compare GD-GNN with popular link prediction methods VGAE [19], SEAL[49], NBFNet[56] for the citation datasets. Note that for citation data, we do not use the pre-trained node embeddings from the publications' content, incorporating such information only increases the power of GD-GNN.

We use GCN as the basic GNN. We search the number of GNN layers in $\{2, 3, 4, 5\}$, and the max search distance for geodesic, d_{max}, is same as the GNN layers. We use 32 hidden dimensions for all fully-connected layers in the model. We train 100 epochs with a batch size of 64 for the citation datasets.

On the citation dataset (Table 10), GD-GNN achieved very competitive results compared to the SOTA method NBFNet, and GD-GNN is more efficient than NBFNet as demonstrated in Table 3.

Table 10: Link prediction results (%) on Citation dataset.

Method	Cora		Citeseer		PubMed	
	AUC	AP	AUC	AP	AUC	AP
VGAE	91.40	92.60	90.80	92.00	94.40	94.70
SEAL	93.32	94.21	90.52	92.43	97.78	97.90
NBFNet	**95.61**	**96.17**	**92.28**	**93.74**	**98.30**	**98.15**
GD-GNN-Vert	94.93	**96.01**	91.72	93.29	98.06	**98.14**
GD-GNN-Hor	94.30	95.22	91.18	92.34	97.74	97.76

We also includes C.elegans, NS, and PB dataset adopted by DE-GNN [23]. We follow its experiment setup to split the edges into 8:1:1 train/test/valid splits, and repeat experiment 20 times and report average AUC and 95% confidence range. For these datasets, we compare with other methods that also rely on distance encoding, including DE-GNN, PGNN[47], SEAL, and base GNN, GIN.

Table 11: Link prediction results compared to other distance-related methods.

Method	C.elegans	NS	PB
GIN	75.58 ± 0.59	87.75 ± 0.56	91.11 ± 0.52
PGNN	78.20 ± 0.33	94.88 ± 0.77	89.72 ± 0.32
SEAL	88.26 ± 0.56	98.55 ± 0.32	94.18 ± 0.57
DE-GNN	$\mathbf{90.05} \pm 0.26$	$\mathbf{99.43} \pm 0.63$	94.95 ± 0.37
GDGNN-hor	89.83 ± 0.70	98.65 ± 0.48	$\mathbf{96.14} \pm 0.73$
GDGNN-ver	87.88 ± 0.42	98.10 ± 0.26	94.43 ± 0.39

From the experiment, we can see that GD-GNN achieves very competitive results compared to distence-encoding GNN and SEAL, and outperforms PGNN [8] significantly on all datasets. We suspect the key reason is that PGNN relies on relative positions to the anchors, but the anchors that PGNN randomly chooses are not necessarily representative to all links. In contrast, the geodesic information is directly associated with each link.

E Node Classification

We use airport dataset, Brazil-Airport, Europe-Airport, and USA-Airport for our node classification experiments. The task is to predict passenger flow level solely from the flight traffic network, where graph structure matters and more expressive GNNs are useful. We compare to GCN, GIN and DE-GNN that also relies on distance information. We follow DE-GNN and split the dataset with train/test/valid ratio of 8:1:1. We run experiment using verticala GDGNN with independent random initialization 20 times and report average accuracy and 95% confidence range.

Table 12: Accuracy on node experiment(%)

Method	Bra.-Airports	Eur.-Airports	USA-Airports
GCN	64.55 ± 4.18	52.07 ± 2.79	56.58 ± 1.11
GIN	72.83 ± 3.57	53.84 ± 3.94	57.43 ± 2.13
DE-GNN	75.37 ± 3.25	$\mathbf{58.41} \pm 3.20$	64.16 ± 1.70
GDGNN	$\mathbf{78.61} \pm 2.20$	53.97 ± 3.50	$\mathbf{64.36} \pm 1.62$

DE-GNN has several variants and we take their best result here. From Table 12, we can see that GD-GNN achieves very competitive results compared to distance-encoded GNN, and significantly improves the performance of base GIN on two datasets, which experimentally verifies our claim that node-level geodesic is more expressive than plain GNNs.

F Ablation study

GD-GNN has many components, and here we present ablation study of GD-GNN to demonstrate the effect of each component. For link prediction dataset, we consider 4 settings, plain GCN (GCN), GCN and distance between the two target nodes of the link (GCN+Dist), GD-GNN-Vertical without geodesic degree (GD-GNN-Vert), and GD-GNN-Vertical with geodesics node degree (GD-GNN-Vert-Deg). Note that the results we report in Section 5 include geodesic node degree.

Table 13: Ablation study on link prediction datasets.

Method	Cora (AUC)	OGBL-COLLAB (H@50)	OGBL-PPA (H@100)
GCN	81.79	44.75	18.67
GCN+DIST	92.75	51.82	21.48
GD-GNN-Vert	94.13	54.19	43.91
GD-GNN-Vert-Deg	94.26	54.68	45.61

From the experiment results (Table 13), we can see that distance is already able to assist link prediction, especially on sparser datasets like Cora and OGBL-COLLAB. However, on OGBL-PPA, where the connectivity of the graph is much larger, distance does not help, whereas geodesic representation is able to significantly improve the expressive power of basic GNNs. Also, when the graph is sparse, it is rather unlikely for the geodesic nodes to form subgraphs, meaning that most of nodes on the vertical geodesic have a degree of zero, and hence geodesic degree does not improve much. Meanwhile, in OBGL-PPA, geodesic degree is able to increase the performance of GD-GNN, which aligns with the intuition and example we present.

For horizontal geodesics, we present results where part of the horizontal geodesic is used.

Table 14: Partial horizontal geodesic results.

Method	Cora	OGBL-COLLAB
4-GDGNN-Hor	94.40	52.13
4-GDGNN-Partial	92.73	52.06
5-GDGNN-Hor	94.37	52.18
5-GDGNN-Partial	92.68	51.42

Partial represents geodesics where only the head/tail, head/tail's direct neighbors on the shortest path are used to form the geodesic. N-GDGNN means the cutoff distance is N. We do not compare with 2-3 distance because Partial and horizontal geodesic will be exactly the same in that case. We add distance to both Partial and hortizontal geodesics to make fair comparison. We can see that without the full horizontal geodesic, the performance on Cora dataset indeed dropped by 1.5%. While the difference is not as significant, we still see that horizontal geodesics outperforms Partial on OGBL-COLLAB dataset. This shows that while being connected within some distance is already a good indicator of the likelihood of the link, incorporating the node structure information can still improve the representation of the link.

For graph classification tasks, we study two simpler version of GD-GNN. We reduce the vertical pair-wise geodesic function (Equation 4) to be a pooling function only on the embeddings of the pair of nodes. Then, the node-level geodesic is essentially combining the node embedding and its k-hop neighbor embedding, we refer to this as Nei. Another version is we append the pair-wise distance to the neighbor embeddings, we refer to this as Dist. Note that vertical geodesic takes part of the direct neighbors of a neighbor node, we study the variant where all neighbors are used as the geodesic, we call this FullNei.

From Table 15, we can see that the neighbor embedding alone is not very effective, and we need to assist it with pair-wise distance, a explicit form of geodesic to improve the performance. By incorporating the geodesic representation, we are able to consistently improve over the 'Nei+Dist' version, which aligns with Theorem 1. Geodesic degree does not give significant improvement on TU dataset possibly due to its high testing variance, but we can still see notable improvement on OGBG-MOLHIV dataset. For FullNei, we see that it does not bring much performance increase onto

Table 15: Ablation study on graph classification datasets.

Method	MUTAG (Accuracy)	PROTEINS (Accuracy)	OGBG-MOLHIV (AUC)
GIN	74.0	71.2	75.5
GIN+Nei	81.2	71.9	76.2
GIN+Nei+Dist	88.1	71.8	76.0
GD-GNN-Vert-FullNei	88.1	71.9	76.4
GD-GNN-Vert	89.0	73.3	77.6
GD-GNN-Vert-Deg	89.4	73.6	78.4

GIN+Nei+Dist compared to GD-GNN-Vert. This shows that FullNei essentially weighs in as an extra layer of GNN which can be covered by hyperparameter search.

G Impact of different cutoff distances

A non-negligible hyperparameter in our model is the cutoff distance. In this section, we present results on the impact of different cutoff distances.

Table 16: Performance on link and graph datasets with different cutoff distances (%).

Method	Cora(L)	OGBL-PPA(L)	OGBG-MOLHIV(G)
1-GDGNN	82.46	21.79	75.68
2-GDGNN	91.80	43.57	78.13
3-GDGNN	93.22	45.61	78.42
4-GDGNN	94.93	44.98	78.36

(L) represents link prediction task, and (G) represents graph classification task. N-GDGNN means GNN with different max-cutoff distance, we use vertical geodesics with 3-layers of base GNN. From Table 16, we see that for link prediction task (cora in particular), as the cutoff distance increases, the performance increases, this is because more links can be connected by geodesic extraction, and distance itself is already a good indicator of the likelihood of the link. However, disconnected nodes are not always predicted negative. When disconnected, the link will have zero geodesic representation, but **still have meaningful node representations from the base GNN**. In such case, GD-GNN degenerates embeddings similar to models like VGAE, which is still able to statistically learn the probability of a link based on the node structure around the two nodes of the link. The choice of cutoff distance is really data-dependent, as we can see in the results of OGBL-PPA and OGBG-MOLHIV, 4-GDGNN is worse than 3-GDGNN, and the actual number can be determined by hyperparameter tuning. In general, cutoff distance resembles the max number of hops in subgraph extraction process.

H Limitation of Horizontal Geodesics

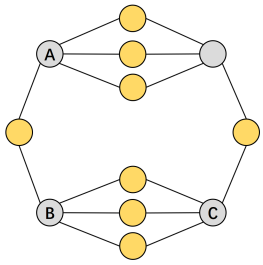


Figure 5

Figure 5 shows an example where vertical geodesic is more expressive than horizontal geodesic. The gray nodes are symmetric and hence will be assigned the same embedding by a 1-WL GNN, the yellow nodes, because they all connect to two gray nodes, will also be assigned same embedding by a 1-WL GNN. In this case, both horizontal geodesics between AB and BC are gray-yellow-gray nodes, hence we can not differentiate the two links. Meanwhile, the vertical geodesic of AB is one yellow node, while the vertical geodesic of BC is three yellow nodes, and hence the two links will have different representation after summarizing the geodesics information.