A Learned Generalized Geodesic Distance Function-Based Approach for Node Feature Augmentation on Graphs

Amitoz Azad
Singapore Management University
Singapore
amitoz.sudo@gmail.com

Yuan Fang
Singapore Management University
Singapore
yfang@smu.edu.sg

Figure 1: (*Left*) A projection of a black dragon onto a 2D grid graph. The aim is to find a geodesic (*shortest-path*) from the source node (boundary) to the target node on the dragon's projection on the grid graph. (*Center*) The geodesic distance map from the source node. (*Right*) The geodesic distance map is used to find the actual geodesic.

ABSTRACT

Geodesic distances on manifolds have numerous applications in image processing, computer graphics and computer vision. In this work, we introduce an approach called 'LGGD' (*Learned Generalized Geodesic Distances*). This method involves generating node features by learning a generalized geodesic distance function through a training pipeline that incorporates training data, graph topology and the node content features. The strength of this method lies in the proven robustness of the generalized geodesic distances to noise and outliers. Our contributions encompass improved performance in node classification tasks, competitive results with state-of-the-art methods on real-world graph datasets, the demonstration of the learnability of parameters within the generalized geodesic equation on graph, and dynamic inclusion of new labels.

CCS CONCEPTS

• **Computing methodologies** → *Neural networks*; *Supervised learning by classification*; • **Mathematics of computing** → Ordinary differential equations.

KEYWORDS

Graph Neural Network, Geodesic Distance Function, Node Feature Augmentation, Node Classification.

ACM Reference Format:
Amitoz Azad and Yuan Fang. 2024. A Learned Generalized Geodesic Distance Function-Based Approach for Node Feature Augmentation on Graphs. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '24), August 25–29, 2024, Barcelona, Spain.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3637528.3671858

KDD '24, August 25–29, 2024, Barcelona, Spain

ACM ISBN 979-8-4007-0490-1/24/08
https://doi.org/10.1145/3637528.3671858

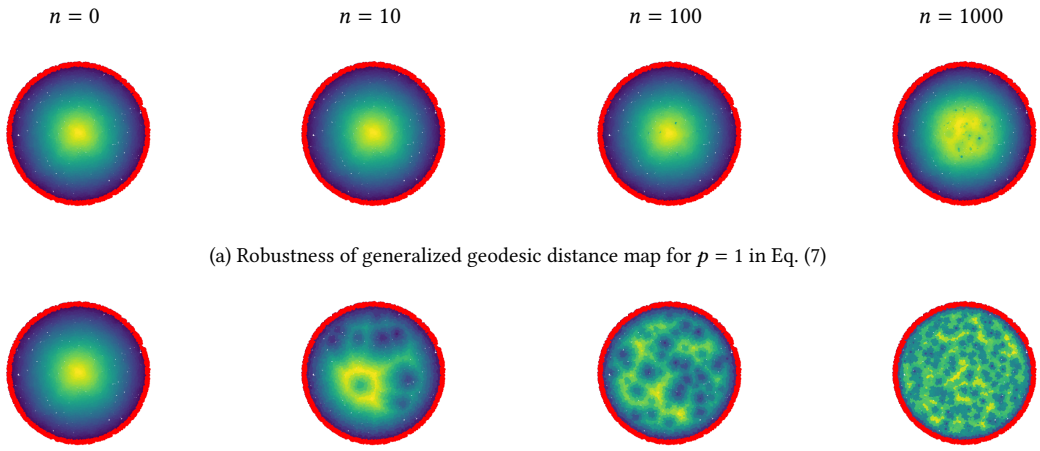
1 INTRODUCTION

In recent times, there has been a growing interest in data augmentation techniques for graphs [39]. The primary motivation behind augmenting graphs is to improve model performance by enhancing the quality of the graph data through some form of denoising. Real-world graphs, which depict the underlying relationships between nodes, often suffer from noise due to various factors such as fake connections [14], arbitrary edge thresholds [34], limited or partial observations [7], adversarial attacks [18], and more. These factors collectively render the graphs suboptimal for graph learning tasks. To address these issues, researchers have been exploring graph structural and node feature augmentation techniques that aim to generate improved graphs [39].

Geodesic distances (Figure 1) has found numerous applications in computer vision, ranging from calculating shortest-path distances on discrete surfaces [16], to shape-from-shading [26], median axis or skeleton extraction [30], graph classification [1], statistical data depth [21], noise removal, and segmentation [20].

Recently the authors in [2] studied a generalized geodesic distance function equation on graphs Eq. (7), which they referred to as the graph p-eikonal equation, earlier proposed in [9, 10]. The authors provided both theoretical and experimental evidence to demonstrate that, unlike the geodesic (shortest-path) distance function on graphs (as can be computed with classic Dijkstra algorithm, Sec. 2.6), the generalized geodesic distance function is provably more robust (less affected by change) when the graph is subjected to the addition of corrupted edges, especially for $p = 1$ in Eq. (7).

Contributions. Motivated by the proven robustness of the generalized geodesic distance function to edge corruptions (see Figure 2) and outliers [2], in this work, we focus on generating node feature vectors using *learned* generalized geodesic distances on a graph



(a) Robustness of generalized geodesic distance map for $p = 1$ in Eq. (7)

(b) Robustness of geodesic (shortest-path) distance map Eq. (9)

Figure 2: The n represents the number of random corrupted edges added to a given graph. The graph construction: 20,000 points (nodes) were randomly sampled from a unit ball in R^2. An ϵ-neighborhood unweighted graph was constructed using these sampled points with $\epsilon = 0.05$. All points within ϵ distance of the boundary of the unit ball are considered boundary nodes. Colors represent the distance from the boundary, with red indicating the boundary where the distance function is zero, and yellow indicating the maximum distance.

for node classification task. This learning of generalized geodesic distances is achieved by formulating the generalized geodesic distance function Eq. (7) as a time-dependent problem Eq. (8). This time-dependent version allows us not only to solve Eq. (7), but it also enables gradient-based learning of the generalized geodesic distance function using node content features (such as bag-of-words for citation networks).

Since the generalized geodesic distance function Eq. (7) only considers the graph topology, the generated node features are robust, but they are purely topological, as it does not consider the original node content features. We propose a hybrid model that learns the generalized geodesic distance function using gradient descent, and generates robust node features which not only consider the graph topology but also take into account the original node content features (Figure 4). Using these learned generalized geodesic distances at different time values (t in Eq. (8)) as node features improves the performance of the backbone model, and makes it competitive with state-of-the-art augmentation methods (Table 1).

We refer to the node feature generated through learning generalized geodesic distances as "LGGD" (*Learned Generalized Geodesic Distances*) (Table 1). To summarize our key contributions:

- We propose a hybrid model in which the generalized geodesic distances are learned using the training data, graph topology and the node features (Figure 4).
- The generation of node features based on these learned generalized geodesic distances improves the performance of various backbone models (Figure 5, top row) and enables them to compete with SOTA methods (Table 1, Row 09).
- The proposed approach allows for the dynamic inclusion of new incoming labels without the need for retraining the backbone GNN (Figure 5, bottom row).
- We show that gradient based learning of the potential function $\rho(x)$ in generalized geodesic distance function Eq. (7) provides a slight boost in the backbone model's performance (Table 1, Row 10).

2 MATHEMATICAL BACKGROUND

In this section, we review briefly some basic definitions and operators on graphs and provide the necessary background to understand the generalized geodesic distance function on graphs. For a more comprehensive mathematical background, refer to prior works [32, 33].

2.1 Notation

A weighted graph, denoted as $G = (V, E, w)$, is defined by a finite set of nodes in V and a finite set of edges in E, where each edge (i, j) connects nodes i and j. The weights of the graph are determined by a weight function $w : V \times V \longrightarrow [0, 1]$, and the set of edges is determined by the non-zero weights: $E = \{(i, j)|w(i, j) \neq 0\}$. If there is no edge between i and j, then $w(i, j) = 0$. We represent the set of nodes neighboring node i as $N(i)$, where $j \in N(i)$ signifies that node j is in the neighborhood of node i, i.e., $N(i) = \{j \in V|(i, j) \in E\}$. In this paper, we consider symmetric graphs, meaning that $w(i, j) = w(j, i)$, and the presence of an edge (i, j) is equivalent to the presence of its reverse (j, i). The degree of a node i, denoted as $\delta(i)$, is computed as the sum of weights for all nodes in its neighborhood: $\delta(i) = \sum_{j \in N(i)} w(i, j)$.

Let $H(V)$ be a Hilbert space comprised of real-valued functions defined on the graph's nodes. A function $f : V \longrightarrow R$ of $H(V)$ characterizes a signal associated with each node, assigning a real value $f(i)$ to every node i in V. Similarly, let $H(E)$ denote a Hilbert space encompassing real-valued functions defined on the edges of the graph.

2.2 Gradient Operators

The graph difference operator $d_w : H(V) \rightarrow H(E)$ is defined as:

$$(d_w f)(i,j) = \sqrt{w(i,j)}(f(j) - f(i)) \tag{1}$$

Using this, one can define the graph gradient vector of a function $f \in H(V)$, at a vertex $i \in V$ as:

$$(\nabla_w f)(i) = [(d_w f)(i,j) : \forall j \in V]^T \tag{2}$$

The L_p norm of the graph gradient is defined as:

$$\|(\nabla_w f)(i)\|_p = \Big[\sum_{j \in V} |(d_w f)(i,j)|^p \Big]^{\frac{1}{p}} \tag{3}$$

Based on the graph difference operator, one can define the directional graph difference operator as follows:

$$\begin{aligned} (d_w^+ f)(i,j) &= \sqrt{w(i,j)}(f(j) - f(i))_+ \\ (d_w^- f)(i,j) &= \sqrt{w(i,j)}(f(j) - f(i))_- \end{aligned} \tag{4}$$

Here $(a)_+ = max\{a, 0\}$ and $(a)_- = -min\{a, 0\}$. Following above, one can come up with directional graph gradient vectors as:

$$\begin{aligned} (\nabla_w^+ f)(i) &= [(d_w^+ f)(i,j) : \forall j \in V]^T \\ (\nabla_w^- f)(i) &= [(d_w^- f)(i,j) : \forall j \in V]^T \end{aligned} \tag{5}$$

One can then define the L_p norm of these directional graph gradients vectors as:

$$\begin{aligned} \|(\nabla_w^+ f)(i)\|_p &= \Big[\sum_{j \in V} |(d_w^+ f)(i,j)|^p \Big]^{\frac{1}{p}} \\ \|(\nabla_w^- f)(i)\|_p &= \Big[\sum_{j \in V} |(d_w^- f)(i,j)|^p \Big]^{\frac{1}{p}} \end{aligned} \tag{6}$$

In this work, we focus exclusively on the negative graph gradient operator $(\nabla_w^- f)(i)$ and its associated L_p norm $\|(\nabla_w^- f)(i)\|_p$ for $p = 1$. This operator is closely linked to the morphological erosion PDE process on graphs [32], and plays a crucial role in defining generalized geodesic distance function on graphs.

2.3 Generalized Geodesic Distance Function

The generalized geodesic distance function equation on graphs as introduced in [2], can be written in this form (see Appendix A.1):

$$\begin{aligned} \rho(x)\|(\nabla_w^- f)(x)\|_p &= 1, \quad x \in V \setminus V_0 \\ f(x) &= 0, \quad x \in V_0 \end{aligned} \tag{7}$$

Here, $f(x)$ represents the generalized geodesic distance function. $\rho(x)$ is the potential function (Sec. 2.7). V_0 represents the boundary nodes (training set) from which the distances have to be calculated (hence $f(x) = 0$ at boundary).

One way to solve the above equation is by employing the fast marching [28] or fast iterative [15] methods. Alternatively, one can solve it numerically by considering a time-dependent version of it:

$$\begin{aligned} \partial_t f(x,t) &= -\rho(x)\|(\nabla_w^- f)(x,t)\|_p + 1, \; x \in V \setminus V_0 \\ f(x,t) &= 0, \qquad\qquad x \in V_0 \\ f(x,0) &= \phi_0(x) \qquad\qquad x \in V \end{aligned} \tag{8}$$

At steady-state ($t \rightarrow \infty$), this equation provides the solution to the generalized geodesic distance function Eq. (7). Note the introduction of an extra variable time t and the initial condition $f(x,0)$. A default choice of initializing is to let the distance be zero on the boundary nodes and infinity (a large positive number) on the rest (*just like the initialization in Dijkstra's algorithm to find the distance map from a source node*). In this work, we utilize this time-dependent version to generate generalized geodesic distance features for every node for different time (t) values. As we will see, this formulation provides us with the capability to incorporate the original node content features, and generate **learned generalized geodesic distances** for different time (t) values. This is achieved by employing backpropagation through an ODE solver [23], letting $f(x,0)$ be an MLP (multi-layer perceptron) function of node content features, and then learning the MLP function through gradient descent.

2.4 Solving Eq. (8) with an ODE Solver

The Eq. (8) can be solved using an ODE solver like Torchdiffeq [5]. Various numerical schemes, consisting of fixed step or adaptive step sizes, can be employed from Torchdiffeq. Although Eq. (8) is a PDE on a graph, it can be viewed as a system of coupled ODEs on the graph. This is because the spatial domain is already discretized, and the spatial derivatives at each node can be viewed as finite differences (similar to the Finite Difference Method).

It must be pointed out that Eq. (8) is a vector-valued equation. Since the training set (boundary nodes) consists of nodes from K different classes, Eq. (8) is solved for each class for every node x, thus providing $f^k(x,t)$ as the solution. Here, $f^k(x,t)$ represents the generalized geodesic distance of node x at time t from the boundary nodes of k^{th} class.

2.5 Learning Eq. (8) with an ODE Solver

Torchdiffeq not only allows us to solve a differential equation numerically using various numerical schemes, but it also enables us to learn the parameters of the differential equation through backpropagation using the adjoint sensitivity method [23].

To learn the parameters of the differential equation, first, a segment of the differential equation needs to be converted into a loss function. This loss function is then minimized using a gradient descent based technique via the adjoint sensitivity method. In the case of Eq. (8), to learn $\rho(x)$ and $\phi_0(x)$, the boundary condition $f(x,t) = 0$ can be employed to construct a loss $L(f(x,t), 0)$ where $x \in V_0$. The approach of converting the boundary condition to a loss function is very similar to the inspiring work done in PINNS [24].

2.6 Connection with Dijkstra

We now explain the connection between Eq. (7), and the celebrated Dijkstra algorithm. The Dijkstra algorithm can be used to find the *geodesic* (shortest-path) distance map on a graph, which then can be further used to find the actual shortest-path between the boundary node and a target node. We call Eq. (7) as *generalized geodesic* distance function, because the *geodesic* (shortest-path) distance function (as can be obtained using Dijkstra) is a special case of Eq. (7), as we will see shortly.

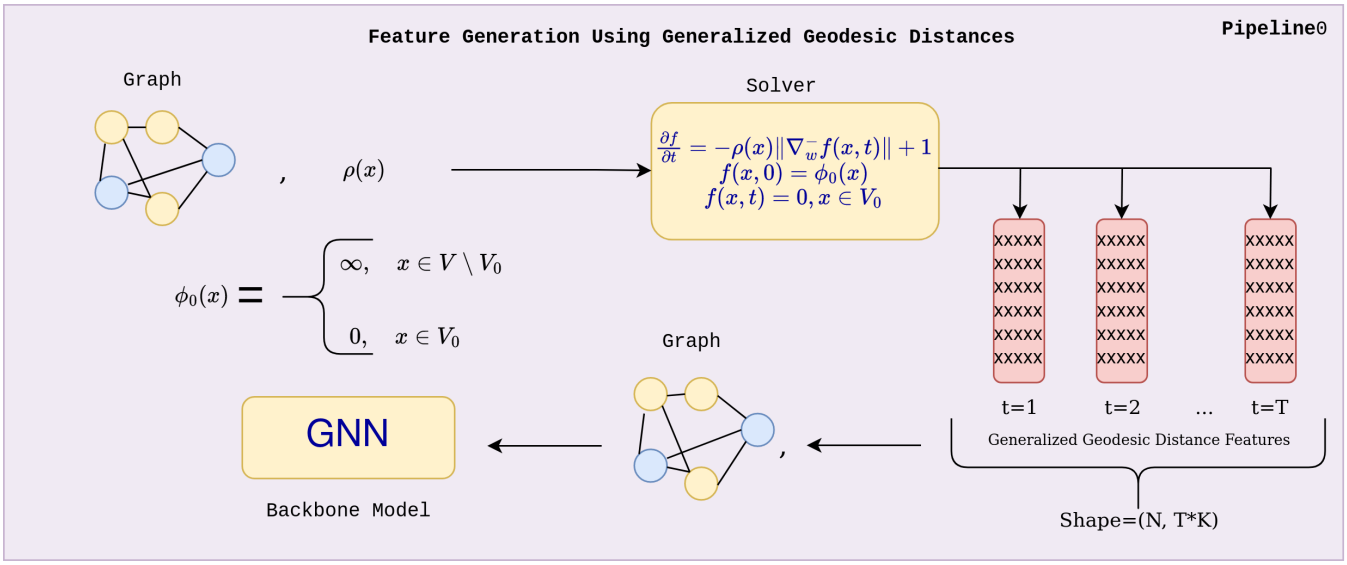


Figure 3: Generalized Geodesic Distances as Features

From dynamic programming perspective, for a unweighted graph, the functional equation for Dijkstra algorithm, satisfies the following *shortest-path* distance function from the boundary set V_0:

$$\begin{aligned} f(i) &= \min_{j \in N(i)} \{f(j) + 1\}, i \in V \setminus V_0 \\ f(i) &= 0, \qquad\qquad\qquad i \in V_0 \end{aligned} \tag{9}$$

This equation can then be solved using direct or successive approximation methods [31]. Often the boundary set consist of a single node. The geodesic map on the dragon's projection on the grid (Figure 1) is calculated from the above equation. Once the solution to the above equation is obtained, it enables the determination of the geodesic (shortest-path) from the boundary node(s) and a target node. The following proposition explains why Eq. (7) is a generalized geodesic distance function.

Proposition 1. *For an unweighted graph with a constant potential function $\rho(x) = 1$, the Eq. (7) with supremum norm (*i.e. *$p = \infty$) yields geodesic (shortest-path) distance function of Eq. (9).*

Refer Appendix A.2 for the proof. The above proposition clearly implies that the space of distance function in Eq. (7) is much larger space which encompasses the geodesic (shortest-path) distance function Eq. (9) as a special case. So in that sense, the former represents a more generalized geodesic distance function on graphs.

2.7 Choosing Potential Function $\rho(x)$

The potential function $\rho(x)$, often plays a crucial role in the generalized geodesic distance function of a graph. For instance, in tasks related to image processing, such as segmentation, it is often contingent on the image gradient at a pixel. This dependency allows distances to be shorter in the smooth regions of an image and longer in the non-smooth regions. In this work, drawing inspiration from [2], we opt to associate the potential function with the local density at a node. By making the potential function dependent on local density, generalized geodesic distances are shortened in denser regions and lengthened in sparser areas. This brings generalized geodesic distances of nodes within a cluster closer together while pushing generalized geodesic distances of nodes in different clusters further apart. We take the node degree, $\delta(x)$, as a measure of density at a node x. And set $\rho(x) = \delta(x)^\alpha$, where α is a hyperparameter searched within the range of -1 to 0. Later in Sec. 4.2, we will see that how gradient based learning of this function results in slight boost in the performance.

3 PROPOSED APPROACHES

In this section we describe our proposed approaches of generating node features using generalized geodesic distance function without and with gradient based learning.

3.1 Generalized Geodesic Distances as Features

This subsection describes the approach to generate generalized geodesic distance as node features with no gradient based learning. To use generalized geodesic distances as node features, we initially start by generating features using Eq. (8), where we consider the training set as the boundary nodes and use the default initial condition in Eq. (8) (Figure 3). So for the initial condition ($f(x,0) = \phi_0(x), x \in V$) we have:

$$\phi_0(x) = \begin{cases} \infty, & x \in V \setminus V_0 \\ 0, & x \in V_0 \end{cases} \tag{10}$$

For every node x, Eq. (8) is solved for K different classes in the boundary nodes (training set) for T different time (t) values. The final feature at each node x is a set $\{f^k(x,t)\}$, where $k \in \{1, 2, ..K\}$ and $t \in \{1, 2, ..T\}$. Here $f^k(x,t)$ represents the generalized geodesic distance of node x at time t from the boundary nodes of k^{th} class. These generalized geodesic distance features are assigned to the nodes (replacing the original node content features) and provided as input to a backbone GNN, along with the graph structure, for the

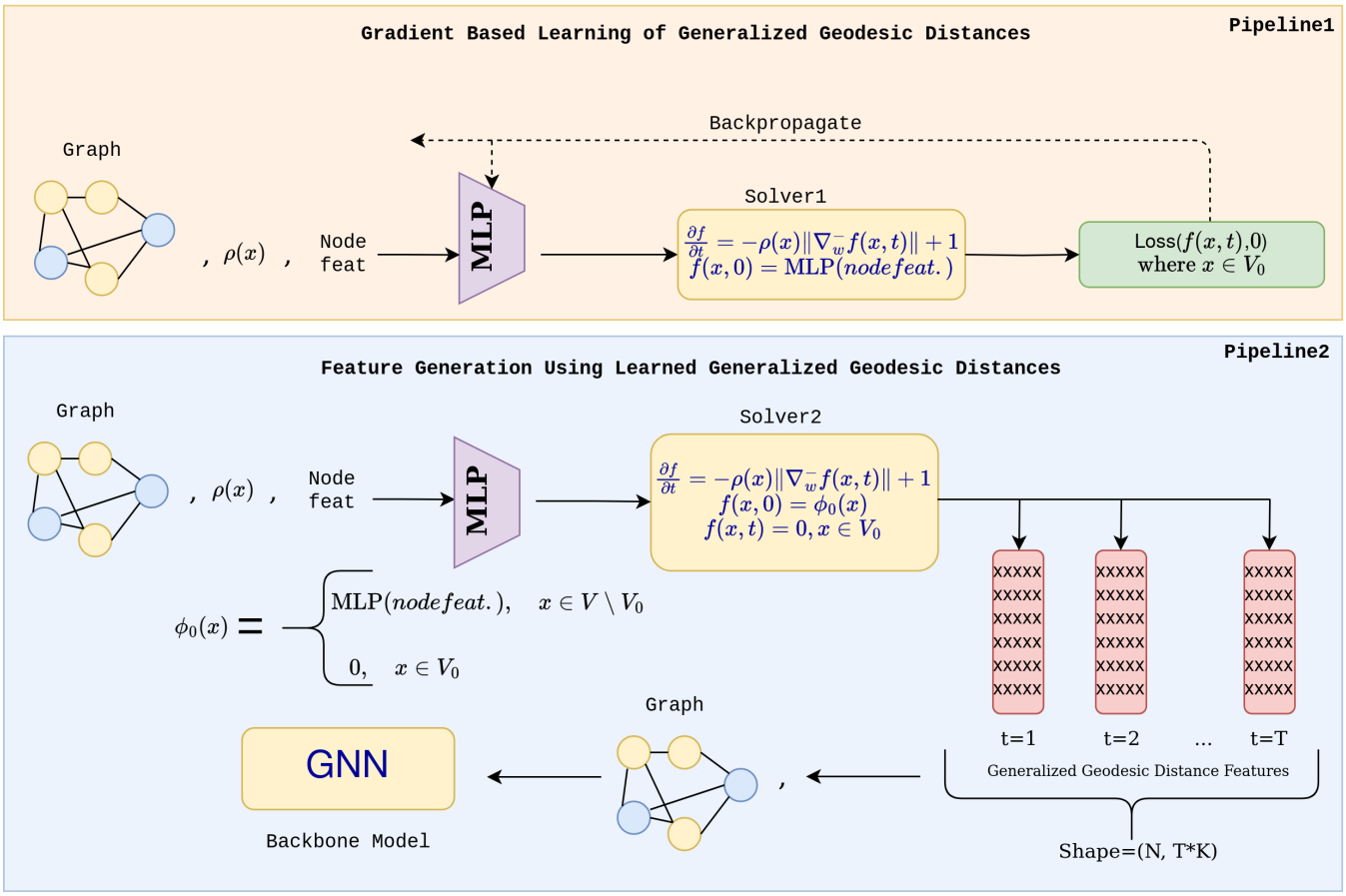


Figure 4: Learned Generalized Geodesic Distances as Features.

node classification task (Figure 3). We refer to the distance features generated using this approach as *GGD*.

3.2 Learned Generalized Geodesic Distances as Features

The previous generalized geodesic distances, treated as node features, are pure topological features as they do not consider the original node content features. This subsection describes the approach to generate the learned generalized geodesic distances. We refer to them as learned generalized geodesic distances (*LGGD*) because they are generated after gradient-based learning of the parameters of Eq. (8), as we will see shortly. Figure 4 depicts the proposed architecture for generating learned generalized geodesic distance features, which also takes the original node content features into account. This architecture can be viewed as a mechanism for converting the original node content features into learned generalized geodesic distance features.

It is essentially a two-step approach involving Pipeline1 and Pipeline2 (Figure 4). Pipeline1 is different from Pipeline0 of the previous case (Figure 3), as now we have converted the boundary condition into a loss function and added an MLP function as the initial condition (Figure 4) in Solver1, which takes into account the original node content features. Unlike Pipeline0, Pipeline1 is not used to generate the distance features; rather, Pipeline1 is tasked with learning the weights of the MLP function and optionally learning the parameters ($\rho(x)$) in Solver1. Node features are input into the MLP, and the output serves as the initial distances $f(x, 0)$, provided to Solver1.

The loss function $L(f(x,t),0)$ within Pipeline1 plays a crucial role in facilitating the learning process. Specifically, it enforces that the self-distances of all nodes on the boundary from the boundary (the training set) should remain zero, as required by the boundary condition in Eq. (8). It's worth noting that the boundary condition should not be directly incorporated into Solver1, as doing so would result in the loss remaining perpetually at zero, hindering the learning process (loss minimization using gradient descent).

Pipeline2 serves as the feature-generating pipeline. It is similar to Pipeline0 (no learning case, Figure 3), as both of them function as feature-generating pipelines. The difference between Pipeline2 and Pipeline0 is that the former uses the learned parameters MLP(*node feat*) and $\rho(x)$ from Pipeline1 to generate the features, whereas the latter uses the default initialization. Observe how the learned MLP function from Pipeline1 is used to construct the initial condition

$f(x, 0) = \phi_0(x)$ of Solver2:

$$\phi_0(x) = \begin{cases} \text{MLP}(nodefeat.), & x \in V \setminus V_0 \\ 0, & x \in V_0 \end{cases} \tag{11}$$

This construction ensures that self-distances of the boundary nodes are zero from the very beginning $t = 0$.

Pipeline2 can be deployed separately once the MLP function and the parameters $\rho(x)$ are learned and saved from Pipeline1. Pipeline2's purpose is to generate learned generalized geodesic distance features for different time steps, which are then concatenated and provided as input to the backbone model for evaluation on the validation and test set. Note that, unlike Solver1, Solver2 explicitly respects the boundary condition specified in Eq. (8).

3.3 Dynamic Inclusion of New Labels

In Pipeline2, after training the backbone model, one can dynamically include the new labels by simply updating the boundary condition ($f(x, t) = 0, x \in V_0$) and initial condition ($f(x, 0) = \phi_0(x), x \in V$) in Eq. (8). Then, one can use the same learned parameters (MLP function and $\rho(x)$ from Pipeline1) to run Solver2 with updated conditions and generate new features for different time steps. These features can subsequently be used as input for the backbone model that has already been trained.

So let V_1 be the set of new incoming labels, the new boundary condition would become: $f(x, t) = 0,\ x \in (V_0 \cup V_1)$. And the new initial condition $f(x, 0)$ would be:

$$\phi_0(x) = \begin{cases} \text{MLP}(nodefeat.), & x \in V \setminus (V_0 \cup V_1) \\ 0, & x \in V_0 \cup V_1 \end{cases} \tag{12}$$

These updates to boundary condition and initial condition are just to ensure that self-distances to all the nodes on the new boundary $(V_0 \cup V_1)$ remain zero for all instances of time t.

4 EXPERIMENTS & RESULTS

In this section, we walk through the research questions pertaining to our proposed models, detail the experiments conducted to answer them, and analyze the results. For all the experiments, we kept $p = 1$ in Eq. (8) as it has been shown to yield the most robust generalized geodesic distances [2].

Software. We employed the PyTorch framework [22] for our work. To execute the time-dependent generalized geodesic distance Eq. (8), we harnessed the combined power of TorchGeometric [12] along with TorchDiffeq [5]. TorchDiffeq, a well-regarded GPU accelerated ODE solver implemented in PyTorch, offers the capability to perform backpropagation through ODEs using the adjoint sensitivity method [23], and it offers a variety of numerical schemes. Throughout all of our experiments, we consistently utilized the Runge-Kutta (RK4) method, adjusting the step size and tolerance values as hyperparameters, which were set using the performance of the backbone model on the validation set.

Datasets. We used the well-known citation graphs, which have been widely employed for evaluating Graph Neural Networks. These graphs include Cora, Citeseer, and Pubmed [27], where each node signifies a document, edges represent citation links, and nodes are associated with sparse bag-of-words feature vectors and class labels. In addition to the citation graphs, we incorporated two additional real-world datasets, namely Amazon Photo and Amazon Computer [29]. In these datasets, nodes represent items, edges signify frequent co-purchases, node features are represented as bag-of-words from product reviews, and the objective is to assign nodes to their respective product categories.

4.1 Main Results

RQ1. How do the generalized geodesic distance features with no learning (Sec. 3.1, Figure 3) obtained from Eq. (8) perform on a GCN backbone for node classification?

Evaluation. For the experiments, we follow a low-resource setting with a train/val set split of 2.5%/2.5%, with the rest constituting the test set. We report the average accuracy over 10 random splits. We utilize the performance of the backbone model on the validation set to search the hyperparameters of the ODE solver. The α in $\rho(x) = \delta(x)^\alpha$ (see Sec. 2.7) was varied in the range 0 to -1 with an interval size of -0.1. In the Runge-Kutta (RK4) numerical scheme the relative tolerance (*rtol*) was varied from 0.001 to 0.05 with an interval size of 0.001. The absolute tolerance (*atol*) in RK4 was always kept as one tenth of the relative tolerance. The *step_size* parameter in Runge-Kutta scheme was kept either 0.1 or 1. The initial distances $f(x, 0)$ in Eq. (8) are set to be zero for the boundary nodes (training set) and a larger positive value, 1e+6, for the remaining nodes. The features are generated for five different time steps, with t varying from 1 to 5 with an interval of 1. For the backbone GCN [17], we employ a hidden layer of size 32, a dropout rate of 0.5, ReLU activation, a fixed learning rate of 0.01, the Adam optimizer, and a weight decay of 1e-6. We train the GCN for 5000 epochs with a patience counter of 100.

Observation. In Table 1, Row 01 displays the performance of the GCN with original node content features, while Row 08 demonstrates the performance of *Generalized Geodesic Distances* (GGD) as the node features input to the same GCN. It is evident that the GCN using original node features outperforms significantly the use of generalized geodesic distances as input. This observation strongly suggests that the original node content features contain valuable information about the nodes, which the GCN in Row 01 effectively leverages. In contrast, the GGD features represent a purely topological approach and does not take into account any node content features. Even though the generalized geodesic distances are known to be robust to noise, the original node content features simply outperform GGD features. These findings prompt our next research question.

RQ2. How do the learned generalized geodesic distance features (Sec. 3.2, Figure 4) perform on a backbone GCN for node classification task? And how do they compare with other graph augmentation methods?

Evaluation. We use the same splits as in RQ1. For the backbone GCN, we retain the settings as before. In the training configuration of Pipeline1, we use 150 epochs and employ the Adam optimizer with a fixed learning rate 0.01 in all of our experiments, along with L2 weight regularization chosen from {0.0005, 0.001, 0.005, 0.01}. The MLP functions used in Pipeline1 had either one or two hidden

Table 1: Test accuracy over different datasets. From Row 02 to 10, the backbone model is the same GCN. OOM stands for out-of-memory.

Model	Cora	Citeseer	Pubmed	Computers	Photo
01 GCN	74.13 ± 2.08	66.08 ± 2.16	79.73 ± 0.71	81.72 ± 1.78	87.57 ± 1.18
02 MixUp	72.72 ± 1.78	64.14 ± 1.75	80.02 ± 0.52	80.76 ± 1.40	88.67 ± 0.80
03 DropEdge	72.28 ± 1.39	65.73 ± 1.83	81.89 ± 0.84	81.45 ± 1.02	88.29 ± 1.27
04 GAug-M	72.14 ± 1.37	66.38 ± 1.29	82.18 ± 1.36	84.82 ± 0.78	91.05 ± 1.21
05 GAug-O	71.30 ± 1.54	67.22 ± 1.06	OOM*	83.03 ± 0.50	90.62 ± 0.30
06 GDC (heat)	77.52 ± 1.74	65.38 ± 1.36	82.16 ± 0.93	80.18 ± 1.31	88.12 ± 2.21
07 GDC (ppr)	78.13 ± 2.13	66.33 ± 1.84	80.86 ± 0.78	82.88 ± 1.14	89.07 ± 2.19
08 GGD	69.95 ± 2.51	43.21 ± 2.44	76.49 ± 0.87	78.89 ± 1.61	85.69 ± 0.92
09 LGGD	80.18 ± 1.53	67.23 ± 1.79	83.24 ± 1.79	85.23 ± 2.18	92.02 ± 2.33
10 LGGD w. $\rho(x)$	**81.56 ± 2.29**	**68.63 ± 1.70**	83.36 ± 1.88	**85.49 ± 1.09**	**92.39 ± 2.11**
11 GPR-GNN	79.45 ± 1.66	67.18 ± 1.84	**84.11 ± 0.38**	82.80 ± 2.01	91.48 ± 1.59
12 GOAL	76.07 ± 1.56	66.57 ± 1.26	81.83 ± 1.28	83.43 ± 1.04	91.65 ± 0.69

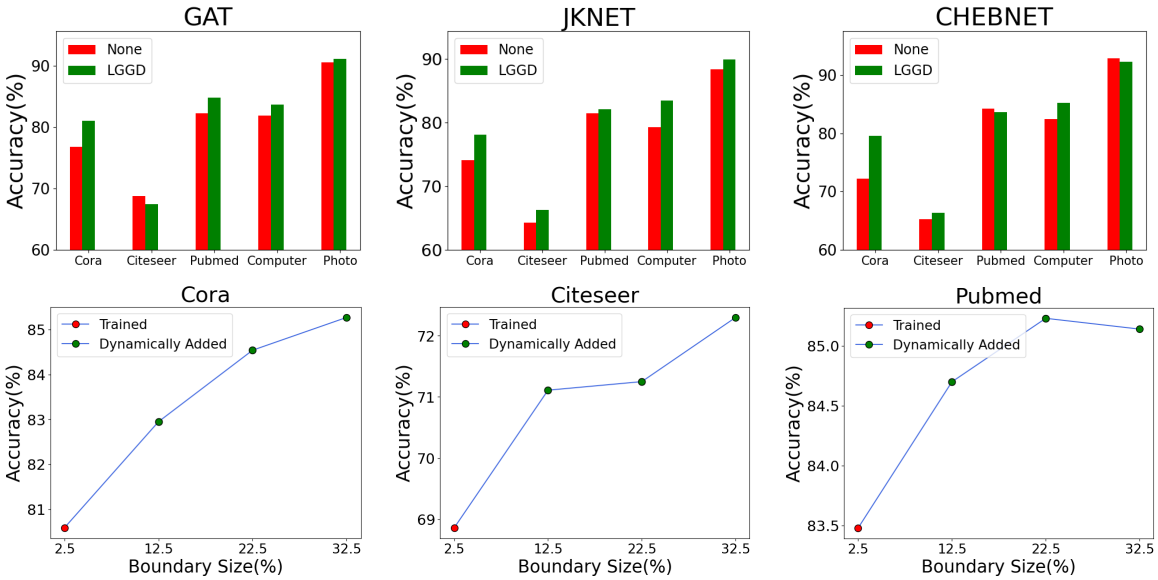


Figure 5: The top row shows the performance of LGGD (*Learned Generalized Geodesic Distances*) features across different datasets for various backbone models. The bottom row demonstrates the ability to incorporate new incoming labels without retraining the backbone model (see Sec. 3.3), as illustrated for three datasets. A green dot represents the results obtained after dynamically adding 10% new labels.

layers with ReLU activations. The hidden layer size was kept in {32, 64, 128, 256, 512, 768}. The dropouts were searched in 0.1 to 0.9 with an interval size 0.1. The chosen loss function was cross-entropy. For Pipeline2, we generate features for five different time steps (t =1 to 5, with an interval of 1). The hyperparameters for Solver1 and Solver2 were kept same. They were searched in the same range as described in the in RQ1. To search for the hyperparameters, we relied on the performance of the backbone model on the validation split. All experiments were conducted using the NVIDIA RTX 3090.

Baselines. We employed various graph augmentation methods for comparision, including MixUp [37], DropEdge [25], GDC [13], and GAug [40] (Row 02 to 07 Table 1). One can find more details about these methods in Sec. 5. To learn about the range of their hyperparameter tuning, refer to Appendix A.3. For all these methods, the backbone model remained a simple GCN with the same setting as mentioned before. In addition to these models, we also utilized two state-of-the-art models for comparison, namely GPRGNN [6] and GOAL [41] (Row 11 & 12, Table 1).

Observation. We observe that the proposed model (Figure 4) significantly enhances the performance of the generalized geodesic distance features, making it competitive with several other methods (Table 1). Row 08 corresponds to 'Generalized Geodesic Distances' (GDD), for which no learning took place. Row 09 corresponds to 'Learned Generalized Geodesic Distances' (LGGD), where a significant improvement in the performance of the backbone model is achieved due to the learning factor by incorporating the node content features.

RQ3. After training the backbone GCN, how does the dynamic inclusion of new labels (Sec. 3.3, Eq. (12)) affect performance over test set?

In the bottom row of Figure 5, one can observe the results of this approach on the three citation networks. We create a split consisting of train/val/nl1/nl2/nl3/test, with percentages of 2.5%, 2.5%, 10%, 10%, 10%, and 65%, respectively. The proposed hybrid model is trained and validated using the 2.5% splits. After training the backbone model, in the Pipeline2, we dynamically add the new labels (nl1, nl2, nl3) to expand the boundary size ($V_0 = \bigcup_{i=0}^{n} V_i$), and update the initial condition according to Eq. (12), and then generate new features using Solver2 and monitor the performance over the test split of the backbone GCN without retraining the backbone GCN. As shown in Figure 5, this approach results in a significant increase in performance on the test set without retraining the backbone GCN.

It is important to note that one can always retrain the backbone model with the incoming new labels, possibly achieving even better performances that increase monotonically. However, the purpose of these experiments is to demonstrate faster predictions without retraining. This has practical potential in scenarios where the backbone model is very large and would require a significant amount of time to retrain. In such cases, new predictions can be made in a fraction of a second by simply generating new features with an updated boundary and initial condition Eq. (12), and then providing them as input to the already trained backbone model for inference.

4.2 Additional Results

RQ4. How does the optional gradient-based learning of the potential function $\rho(x)$ affect the performance of the generated learned generalized geodesic distances features?

In Row 10 of Table 1, we can observe that this change results in a slight performance increase across all datasets. Making it the the top-performing row across several datasets. It is worth noting that Row 09 and Row 10 share the same hyperparameters, and the slight gains are achieved simply by allowing gradient-based learning of the potential function $\rho(x)$.

RQ5. How do the proposed learned generalized geodesic distance features perform for backbone models other than a GCN?

Figure 5 (top row) showcases the performance of the *LGGD* features on three different backbone models: GAT [35], CHEBNET [8], and JKNET [38]. Mean accuracies for the 10 random splits are presented for the same low-resource split setting. We can observe that the learned generalized geodesic distance-based features lead to performance improvements, sometimes quite significant, on most of the datasets for these models. Refer to Appendix A.4 to know the hyperparams of the backbone models.

While Table 1 aims to demonstrate that our method competes with various state-of-the-art structural and feature augmentation methods (using a common backbone GCN), Figure 5 (top row) illustrates how our method enhances the performances across different backbone GNNs. It is essential to note that, for both Table 1 and Figure 5, the hyperparameter configuration of the backbone GNN is consistently maintained (with and without augmentation(s)), allowing us to focus solely on studying the effect of augmentation.

5 RELATED WORK

The field of graph augmentation is vast and rapidly gaining interest within the graph learning community. Here, we will focus on some popular methods for node-level tasks, which essentially make the graph robust to noise by either changing its topology (structural augmentation) or altering its node features.

GraphMix [36] and MixUp [37] are two popular methods for node feature augmentation in semi-supervised learning. Both GraphMix and MixUp employ training a Graph Neural Network (GNN) by interpolating node features and node targets using a convex combination. Both methods draw the parameter λ for the convex combination from a beta distribution. While MixUp involves the mixing of node features and their hidden representations through the message passing within a GNN, GraphMix utilizes a Fully Connected Network (FCN) alongside a GNN, exclusively for feature mixing. The FCN layers and GNN layers share their weights and are jointly trained on a common loss function, combining predictions from the training set FCN layer and GNN layer. Additionally, an unsupervised loss term is incorporated to ensure that the predictions of the GNN on unlabeled nodes match those of the FCN.

DropEdge [25], GAug [40] and GDC [13] are three popular structural augmentation methods for node classification. DropEdge just randomly removes the edges, and redo the normalization on the adjacency matrix before every training epoch. GAug comes in two versions: GAug-M and GAug-O. Both use Graph Autoencoder as the edge prediction module. In GAug-M, an edge prediction module is trained before passing the modified graph to the backbone model. Then, edges with high and low probabilities are added and removed, respectively. In GAug-O, the edge prediction module is trained in combination with the backbone model, using a common loss function that combines node classification loss and edge prediction loss. Training is performed by sparsifying the convex combination of the edge prediction module and the original graph, using differential Bernoulli sampling on this combination. We find it to be slow and memory intensive (Table 1). GDC essentially smooths out the neighborhood by acting as a denoising filter, similar to a Gaussian filter for images. It achieves this by first calculating an influence matrix using methods such as page rank or a heat kernel to make the graph fully connected. Then, it sparsifies the influence matrix using either a *top-k* cutoff or an epsilon cutoff to retain only the edges with maximum influence.

6 CONCLUSION

We proposed a hybrid model in which learned generalized geodesic distances were used as node features to improve the performance of various backbone models for the node classification task. The proposed model allows the dynamic inclusion of new incoming labels without retraining the backbone model.

One limitation of our work is that we did not find much success for heterophilous graph datasets. In fact, most of the structural and node feature augmentation methods work only on homophilous graph datasets. One potential way to overcome this issue is to try negative weights to represent dissimilarity [19]. Alternatively, allowing the potential function to take on negative values could be considered. These approaches will be investigated in the future.

ACKNOWLEDGMENTS

This research project is supported by the Ministry of Education, Singapore, under its Academic Research Fund Tier 2 (Proposal ID: T2EP20122-0041). Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not reflect the views of the Ministry of Education, Singapore.

REFERENCES

[1] Karsten M Borgwardt and Hans-Peter Kriegel. 2005. Shortest-path kernels on graphs. In *Fifth IEEE international conference on data mining (ICDM'05)*. IEEE, 8–pp.
[2] Jeff Calder and Mahmood Ettehad. 2022. Hamilton-Jacobi equations on graphs with applications to semi-supervised learning and data depth. *The Journal of Machine Learning Research* 23, 1 (2022), 14267–14328.
[3] Benjamin Chamberlain, James Rowbottom, Davide Eynard, Francesco Di Giovanni, Xiaowen Dong, and Michael Bronstein. 2021. Beltrami Flow and Neural Diffusion on Graphs. *Advances in Neural Information Processing Systems* 34 (2021).
[4] Benjamin Paul Chamberlain, James Rowbottom, Maria Gorinova, Stefan Webb, Emanuele Rossi, and Michael M Bronstein. 2021. GRAND: Graph Neural Diffusion. In *ICML*.
[5] Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. 2018. Neural ordinary differential equations. In *NeurIPS*.
[6] Eli Chien, Jianhao Peng, Pan Li, and Olgica Milenkovic. 2021. Adaptive universal generalized pagerank graph neural network. In *ICLR*.
[7] Flavio Chierichetti, Alessandro Epasto, Ravi Kumar, Silvio Lattanzi, and Vahab Mirrokni. 2015. Efficient algorithms for public-private social networks. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 139–148.
[8] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. 2016. Convolutional neural networks on graphs with fast localized spectral filtering. *Advances in neural information processing systems* 29 (2016).
[9] Xavier Desquesnes and Abderrahim Elmoataz. 2017. Nonmonotonic front propagation on weighted graphs with applications in image processing and high-dimensional data classification. *IEEE Journal of Selected Topics in Signal Processing* 11, 6 (2017), 897–907.
[10] Xavier Desquesnes, Abderrahim Elmoataz, and Olivier Lézoray. 2013. Eikonal equation adaptation on weighted graphs: fast geometric diffusion process for local and non-local image and data processing. *Journal of Mathematical Imaging and Vision* 46 (2013), 238–257.
[11] Emilien Dupont, Arnaud Doucet, and Yee Whye Teh. 2019. Augmented neural odes. *Advances in neural information processing systems* 32 (2019).
[12] Matthias Fey and Jan Eric Lenssen. 2019. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop*.
[13] Johannes Gasteiger, Stefan Weißenberger, and Stephan Günnemann. 2019. Diffusion improves graph learning. *Advances in neural information processing systems* 32 (2019).
[14] Bryan Hooi, Hyun Ah Song, Alex Beutel, Neil Shah, Kijung Shin, and Christos Faloutsos. 2016. Fraudar: Bounding graph fraud in the face of camouflage. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*. 895–904.
[15] Won-Ki Jeong and Ross T Whitaker. 2008. A fast iterative method for eikonal equations. *SIAM Journal on Scientific Computing* 30, 5 (2008), 2512–2534.
[16] Ron Kimmel and James A Sethian. 1998. Computing geodesic paths on manifolds. *Proceedings of the national academy of Sciences* 95, 15 (1998), 8431–8435.
[17] Thomas N Kipf and Max Welling. 2017. Semi-supervised classification with graph convolutional networks. In *ICLR*.
[18] Srijan Kumar and Neil Shah. 2018. False information on web and social media: A survey. *arXiv preprint arXiv:1804.08559* (2018).
[19] Jeremy Ma, Weiyu Huang, Santiago Segarra, and Alejandro Ribeiro. 2016. Diffusion filtering of graph signals and its use in recommendation systems. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 4563–4567.
[20] Ravikanth Malladi and James A Sethian. 1996. A unified approach to noise removal, image enhancement, and shape recovery. *IEEE Transactions on Image Processing* 5, 11 (1996), 1554–1568.
[21] Martin Molina-Fructuoso and Ryan Murray. 2022. Tukey Depths and Hamilton–Jacobi Differential Equations. *SIAM Journal on Mathematics of Data Science* 4, 2 (2022), 604–633.
[22] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. In *NeurIPS*.
[23] LS Pontryagin, EF Mishchenko, VG Boltyanskiy, and RV Gamkrelidze. 1962. Mathematical theory of optimal processes. (1962).
[24] Maziar Raissi, Paris Perdikaris, and George E Karniadakis. 2019. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *J. Comput. Phys.* (2019).
[25] Yu Rong, Wenbing Huang, Tingyang Xu, and Junzhou Huang. 2020. Dropedge: Towards deep graph convolutional networks on node classification. In *ICLR*.
[26] Elisabeth Rouy and Agnes Tourin. 1992. A viscosity solutions approach to shape-from-shading. *SIAM J. Numer. Anal.* 29, 3 (1992), 867–884.
[27] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. 2008. Collective classification in network data. *AI magazine* (2008).
[28] James A Sethian et al. 1999. *Level set methods and fast marching methods*. Vol. 98. Cambridge Cambridge UP.
[29] Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. 2018. Pitfalls of graph neural network evaluation. *arXiv:1811.05868* (2018).
[30] Kaleem Siddiqi, Sylvain Bouix, Allen Tannenbaum, and Steven W Zucker. 1999. The hamilton-jacobi skeleton. In *Proceedings of the Seventh IEEE International Conference on Computer Vision*, Vol. 2. IEEE, 828–834.
[31] Moshe Sniedovich. 2006. Dijkstra's algorithm revisited: the dynamic programming connexion. *Control and cybernetics* 35, 3 (2006), 599–620.
[32] Vinh-Thong Ta, Abderrahim Elmoataz, and Olivier Lézoray. 2008. Partial difference equations over graphs: Morphological processing of arbitrary discrete data. In *Computer Vision–ECCV 2008: 10th European Conference on Computer Vision, Marseille, France, October 12-18, 2008, Proceedings, Part III 10*. Springer, 668–680.
[33] Vinh-Thong Ta, Abderrahim Elmoataz, and Olivier Lézoray. 2009. Adaptation of eikonal equation over weighted graph. In *Scale Space and Variational Methods in Computer Vision: Second International Conference, SSVM 2009, Voss, Norway, June 1-5, 2009. Proceedings 2*. Springer, 187–199.
[34] Yu-Hang Tang, Dongkun Zhang, and George Em Karniadakis. 2018. An atomistic fingerprint algorithm for learning ab initio molecular force fields. *The Journal of Chemical Physics* 148, 3 (2018).
[35] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2018. Graph attention networks. In *ICLR*.
[36] Vikas Verma, Meng Qu, Kenji Kawaguchi, Alex Lamb, Yoshua Bengio, Juho Kannala, and Jian Tang. 2021. Graphmix: Improved training of gnns for semi-supervised learning. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 35. 10024–10032.
[37] Yiwei Wang, Wei Wang, Yuxuan Liang, Yujun Cai, and Bryan Hooi. 2021. Mixup for node and graph classification. In *Proceedings of the Web Conference 2021*. 3663–3674.
[38] Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. 2018. Representation learning on graphs with jumping knowledge networks. In *International conference on machine learning*. PMLR, 5453–5462.
[39] Tong Zhao, Wei Jin, Yozen Liu, Yingheng Wang, Gang Liu, Stephan Günnemann, Neil Shah, and Meng Jiang. 2022. Graph data augmentation for graph machine learning: A survey. *arXiv preprint arXiv:2202.08871* (2022).
[40] Tong Zhao, Yozen Liu, Leonardo Neves, Oliver Woodford, Meng Jiang, and Neil Shah. 2021. Data augmentation for graph neural networks. In *Proceedings of the aaai conference on artificial intelligence*, Vol. 35. 11015–11023.
[41] Yizhen Zheng, He Zhang, Vincent Lee, Yu Zheng, Xiao Wang, and Shirui Pan. 2023. Finding the Missing-half: Graph Complementary Learning for Homophily-prone and Heterophily-prone Graphs. In *ICML*.

A APPENDICES

A.1 Rewriting Graph p-Eikonal Equation in [2] as Eq. (7)

Let us recall the generalized geodesic distance function equation:

$$\rho(i)\|(\nabla_w^- f)(i)\|_p = 1, \quad i \in V \setminus V_0$$
$$f(i) = 0, \quad i \in V_0$$

Using the definitions of $\|(\nabla_w^- f)(i)\|_p$ and $(d_w^- f)(i,j)$ from Sec 2.2, one obtains the following for the non-boundary nodes $(V \setminus V_0)$:

$$\sum_{j \in V} w(i,j)^{\frac{p}{2}} (f(j) - f(i))_-^p = (\rho(i))^{-p}$$

Using $(a)_- = -min\{a, 0\} = max\{-a, 0\} = (-a)_+$:

$$\sum_{j \in V} w(i,j)^{\frac{p}{2}} (f(i) - f(j))_+^p = (\rho(i))^{-p}$$

By introducing a change of variable $w(i,j)^{\frac{p}{2}} = \tilde{w}(i,j)$ and $\rho(i)^{-p} = \tilde{\rho}(i)$, one obtains the exact graph p-eikonal equation as proposed in [2].

A.2 Proof of Proposition1

Proof. Let us recall that the supremum norm (*aka* infinity norm) for n dimensional vector x is $\|x\|_\infty = max\{|x_1|, |x_2|, ...|x_n|\}$. By using $\|.\|_\infty$ norm in generalized geodesic distance function Eq. (7) becomes:

$$\max_{j \in V}\{|(d_w^- f)(i,j)|\} = (\rho(i))^{-1}, \quad i \in V \setminus V_0$$
$$f(i) = 0, \quad i \in V_0$$

For a unweighted graph with potential function $\rho(i) = 1$, one obtains the following for the non boundary nodes $V \setminus V_0$:

$$\max_{j \in N(i)}\{\max(0, f(i) - f(j))\} = 1$$

Here we used $(a)_- = (-a)_+$, and $w(i,j) = 0$ when $j \notin N(i)$.

One can rewrite the above as:

$$\max_{j \in N(i)}\{\max(-1, f(i) - f(j) - 1)\} = 0$$

Since the R.H.S. is zero, for any valid solution of the above equation, there must be at least one $j \in N(i)$ for which $(f(i) - f(j) - 1)$ is zero, and this corresponds to the maximum element in the set on L.H.S. Therefore, the above equation can be rewritten as:

$$\max_{j \in N(i)}\{(f(i) - f(j) - 1)\} = 0$$
$$f(i) - \min_{j \in N(i)}\{(f(j) + 1)\} = 0.$$

The above equation corresponds exactly to Eq. (9).

A.3 Hyperparam Tuning of Baselines

For MixUp [37], we used random search to uniformly draw α parameter from 1 to 5. The parameter λ is sampled from $Beta(\alpha, \alpha)$, which determines the extent of mixing node features. Regarding DropEdge [25], we adjusted the edge dropout probability from 0 to 0.99 with an interval size of 0.01. In the case of GAug-M [40], we tuned the percentage of the most probable edges to be retained and the percentage of the least probable edges to be dropped, ranging from 0 to 0.9 with an interval size of 0.1. For GAug-O [40], we used random search to uniformly draw α, β, and $temp$ parameters within the range of [0,1], [0,4], and [0,2] respectively. As for GDC (heat) [13], we varied the parameter t from 1 to 10 with and interval of 0.5. In the case of GDC (ppr) [13], we fine-tuned the alpha parameter within the range of 0 to 0.95 with an interval of 0.05. For both GDC models, we utilized the *top-k* method to sparsify the influence matrix, selecting from 32, 64, and 128, either along the dimension 0 (row) or dimension 1 (column). GPRGNN [6] and GOAL [41] do not require the use of a backbone model for prediction. We used their available hyperparameters from their respective GitHub sources.

A.4 Hyperparams of Different Backbones

For GAT, we employed a hidden layer with a size of 32, utilizing input attention heads of size 8 and output attention heads of size 1. In the case of CHEBNET, we applied a two-step propagation with a hidden layer of size 32. For JKNET, we implemented a GCN model with a hidden layer size of 32. Regarding the layer aggregation component of JKNET, we incorporated a LSTM with 3 layers, each with a size of 16. For all of these models, the learning rate was set to 0.01, using the Adam optimizer, a weight decay of 1e-6, a dropout rate of 0.5, and a training duration of 5000 epochs with a patience counter set to 100. This configuration was used for original node content features and for learned generalized geodesic distance distance features.

A.5 Efficiency

Training Time. The overall training time depends on that of the backbone in Pipeline2. Regarding backpropagation through the ODE solver (Pipeline1), the training time efficiency (training loss vs time) is known to be a few times lower than an MLP (as shown in the Figure 20 of work done by Dupont *et al.* [11]). However, this can be effectively mitigated by concatenating every feature vector with zeros [11].

Complexity. The runtime complexity of the ODE solver is dominated by $O(|E|k)(F_b + F_f)$. Here, $|E|$ represents the number of edges, k represents the number of classes, and F_b and F_f represent the numbers of backward and forward function evaluations, respectively. The complexity of the backbone GNN depends on the specific backbone.

Inference. Once the MLP in Pipeline1 is trained, it can quickly produce learned generalized geodesic distance features, taking only a fraction of a second. The primary benefit for the dynamic inclusion part is the ability to make fast predictions without needing to retrain the backbone GNN model. For instance, the prediction times for the dynamically added new labels case (green dots in Figure 5, bottom row) is around 0.1 sec for all three citation graphs, whereas retraining a simple backbone model like a GCN for 1k epochs takes around 7 sec for Pubmed dataset on RTX 3090.

Scalability. The overall scalability depends on the scalability of the backbone GNN. Concerning the ODE solver's ability to manage large-scale graphs, it is worth noting that it has been successfully utilized in the literature [3, 4] for handling OGB graphs in node classification task.