GSN: A Graph Neural Network Inspired by Spring Networks

Guanyu Cui  
School of Information  
Renmin University of China  
Beijing, China  
cuiguanyu@ruc.edu.cn

Zhewei Wei  
Gaoling School of Artificial Intelligence  
Renmin University of China  
Beijing, China  
zhewei@ruc.edu.cn

Ji-Rong Wen  
Gaoling School of Artificial Intelligence  
Renmin University of China  
Beijing, China  
jrwen@ruc.edu.cn

Abstract

The design of Graph Neural Networks (GNNs) that operate on both homophilous and heterophilous graphs has received research attention in recent years. Existing heterophilous GNNs, particularly those designed in the spatial domain, lack a convincing theoretical or physical motivation. Inspired by an old-fashioned spring network model, we propose the Graph Spring Network (GSN), a universal GNN model that works for homophilous and heterophilous graphs. We show that the GSN framework can interpret many GNN models from the perspective of potential energy minimization of a spring network with respect to various metrics, which entrusts strong physical motivations to these models. We also conduct experiments to demonstrate the performance of our GSN model on real-world datasets.

1 Introduction

With the growth of graph-structured data, Graph Neural Networks (GNNs) has recently emerged as a research hotspot in the machine learning field. In the computer vision field, Convolutional Neural Networks (CNNs) made success on many tasks such as image classification [13, 19, 20, 31, 32]. However, the spatial convolution defined on Euclidean data is difficult to be generalized to graphs. In order to generalize CNNs onto graph-structured data, [7] proposed ChebNet, which approximates the graph spectral convolution with Chebyshev polynomials. [17] simplifies ChebNet with the first-order Chebyshev approximation and obtain the well-known Graph Convolutional Networks (GCN), which achieved state-of-the-art on semi-supervised node classification tasks. These GNN models can be roughly classified into four categories based on their design domain (spectral or spatial) and whether they support homophilous or heterophilous graphs (see Table 1).

GNNs designed for homophilous graphs. Inspired by the message-passing nature of GCN, many GNNs are designed in the spatial domain via different message-passing schemes. For example, GraphSAGE [12] focuses on inductive learning and uses neighborhood sampling and the aggregator estimation methods. GAT [35] introduces attention mechanics in its neighborhood aggregation function. Motivated by the random walk on graphs, PPNP and its approximation version APPNP [18] use the PageRank proximity designing the message-passing scheme. SGC [37] simplifies GCN by removing the learnable matrices and non-linear activation functions in convolution layers. Some researchers notice that GCN loses its performance rapidly when piling up too many layers due to
the over-smoothing phenomenon \[21, 26\]. GCNII \[5\] introduces the initial residue and identity mapping to relieve over-smoothing. Moreover, ElasticGNN \[23\] aims to enhance the local smoothness of GNN via \(\ell_1\) based graph smoothing.

**GNNs designed for heterophilous graphs.** An explanation for the over-smoothing phenomenon is that many GNNs have the homophily assumption (which means that nodes with same labels tend to have connections) when aggregating neighboring nodes \[41\]. Therefore, it is practical to design GNNs for heterophilous graphs. To capture long-range dependencies and to improve the performances on heterophilous graphs, Geom-GCN \[28\] maps nodes to a latent space and aggregates neighboring nodes in that space. \(H_2\)GCN \[41\] separates ego-embedding with neighbor-embedding and considers higher-order neighborhoods in its message-passing scheme. Inspired by two iterative algorithms, TWIRLS \[39\] designs GNN layers to overcome the over-smoothing problem and also generalize GNNs to heterophilous graphs. TDGNN \[36\] uses tree decomposition method to disentangle the neighborhoods with different orders. DMP \[38\] tries to learn weights on each edge during the message-passing process. Another explanation is that the graph convolution operation behaves like a low-pass filter and smoothen the node features \[21, 23\]. Therefore, designing or learning a complex graph filter will alleviate this problem. Motivated by this, GNN-LF/HF \[42\] designs a low-pass filter and a high-pass filter from an optimization framework. GPR-GNN \[6\] introduces the Generalized PageRank (GPR) and allows the model to learn GPR coefficients. Meanwhile, FAGCN \[2\] separates the low- and high-frequency information in its aggregation operation. Moreover, BernNet \[14\] uses the Bernstein basis to approximate any arbitrary graph filter. In Section \[6\] we will analyze some of these models in a more detailed manner. Although many heterophilous GNNs have been proposed, a common drawback of them, particularly those created in the spatial domain (e.g., \(H_2\)GCN and DMP), is that they lack a strong physical motivation. In this paper, we will show that the spring network is a natural physical motivation for modeling homophilous and heterophilous GNNs. To be more specific, the contributions of this paper are:

- Motivated by the lack of theoretical foundation of spatial heterophilous GNNs, and inspired by the nature of spring, we examine the invariant property of the energy function of the spring networks and propose the Graph Spring Network (GSN).

- We analyze the universality of the GSN framework, and prove that some GNNs can be simulated by GSN directly or with little modification.

- We have done experiments on different kinds of graph datasets, and empirical results reveal the effectiveness of the GSN framework.

### 2 Preliminaries

**Mathematical Notations.** We use boldface uppercase letters such as \(X\) for matrices; and we use boldface lowercase characters such as \(x\) for (column) vectors. For matrices, \(X_i\) denotes the \(i\)-th row of \(X\), \(X_j\) denotes the \(j\)-th column of \(X\), \(X_{ij}\) denotes the \(ij\)-th element of \(X\), and \(X^\top\) denotes the transposition of \(X\). Meanwhile, \(\text{diag}(X)\) denotes the column vector consists of all diagonal elements of \(X\), i.e., \(\text{diag}(X)_i = X_{ii} (i = 1, 2, \ldots, n)\). The trace of \(X\) is the sum of all its diagonal elements, i.e., \(\text{tr}(X) := \sum_i X_{ii}\). The Frobenius norm of \(X\), indicated by \(\|X\|_F\), is defined as \(\|X\|_F := \sqrt{\sum_{i,j} |X_{ij}|^2}\). For vectors, \(x_i\) denotes the \(i\)-th element of \(x\), whereas \(x^\top\) denotes the transposition of \(x\). And \(\text{diag}(x)\) denotes the square matrix whose diagonal elements are \(x\), i.e., \(\text{diag}(x)_{ij} = \begin{cases} x_i, & i = j \\ 0, & i \neq j \end{cases}\). The \(p\)-norm \(\|\cdot\|_p\) of vector \(x\) is defined as \(\|x\|_p := (\sum_i |x_i|^p)^{1/p}\). The
Hadamard product is the element-wise product of two matrices or vectors. To be precise, \((X \odot Y)_{ij} = X_{ij}Y_{ij}\), and \((x \odot y)_i = x_iy_i\).

**Concepts from Graph Theory.** A graph \(G\) is a pair \((V, E)\), where \(V\) is the node set and \(E \subseteq V^2\) is the edge set. Given an order for the nodes, we may denote \(V\) as \([1, 2, \ldots, |V|]\), and use the adjacency matrix \(A \in \{0, 1\}^{\left|V\right|\times \left|V\right|}\) to represent the edges \(E\), i.e., \(A_{ij} = 1\) if edge \((i, j) \in E\), otherwise \(A_{ij} = 0\). Note that for an undirected graph, \((i, j) \in E \iff (j, i) \in E\), resulting in a symmetric \(A\). Unless otherwise noted, the graphs below are undirected. The neighborhood of node \(i\) is the set of nodes that share an edge with it, i.e., \(N(i) := \{j : (i, j) \in E\}\), whereas the closed neighborhood of \(i\) is \(\overline{N}(i) := N(i) \cup \{i\}\). The degree of node \(i\) is the number of nodes in its neighborhood, i.e., \(d_i := |N(i)| = |\{j : (i, j) \in E\}|\). And the degree matrix of graph is \(D := \text{diag}(\{d_1, d_2, \ldots, d_{|V|}\})\). The edge homophily \(\mathcal{H}(G, Y) := |\{(u, v) \in E : Y_u = Y_v\}|/|E|\). The Laplacian of graph \(L := D - A\) has two essential and critical properties: it is positive semidefinite \(\lambda_1 = 0\) is the smallest eigenvalue of \(L\) and all-one vector \(1\) is the corresponding eigenvector. We sometimes refer to the eigenvalues of \(L\) as frequencies. We usually use the symmetric normalized Laplacian \(L^{sym} := D^{-1/2}LD^{-1/2} = I - D^{-1/2}AD^{-1/2}\), and the symmetric normalized propagation matrix \(P^{sym} := D^{-1/2}AD^{-1/2}\) (or \(P\) for short).

### 3 Design Motivation

**What Constitutes a “Good” Node Embedding?** We investigate the node classification task by asking: what constitutes a “good” node representation? We can answer the question according to our intuition: a “good” node embedding will draw connected nodes in the same class closer and push connected nodes in different classes further. In other words, it should decrease the intra-class distances while increasing the inter-class distances. Note that the all-one vector \(1\) is the eigenvector of the smallest eigenvalue \(\lambda_1 = 0\) of \(L\), so the perfectly smooth vector has the lowest frequency or energy. On the contrary, if a vector is not smooth, it will contain high-frequency components in the spectral domain (w.r.t. \(L\)). The convolution operation in GCN behaves like a low-pass filter, and the nodes’ features are smoothed each time they pass through a convolution layer \([21, 25]\), consistent with the fact that the weighted mean aggregation operation can draw nodes’ embeddings closer. On the other hand, a high-pass filter will push nodes further. Therefore, we cannot just apply low- or high-pass filters to create a suitable GNN design for graphs with coexisting homophilous and heterophilous edges. In the spectral domain, a well-designed GNN should learn a complex filter; whereas in the spatial domain, it should learn a good push-and-pull relation between nodes.

**Node Embeddings and Spring Networks.** Recall that a “good” node embedding will draw connected homophilous nodes closer and push connected heterophilous nodes further. A natural idea is to use springs to model the edge between homophilous or heterophilous nodes, which also push close objects further and pull distant objects closer. The spring network models have been used in the graph drawing field as a special case of the force-directed methods which can be dated back to 1960s \([4, 8, 10, 16, 33, 34]\), and they have also been used in the biophysics and biochemistry fields to analyze protein’s properties \([1, 11, 22]\). According to our previous discussion, the push and pull relationship that we need to learn should act like “similar objects appeal, dissimilar objects repel”, thus the spring network is an ideal tool to model homophily and heterophily of graphs.

To be concrete, suppose we have a spring network in Figure 1 where edges between nodes are springs. Initially, homophilous edges are extended springs, while heterophilous edges are compressed. Letting the spring network evolve freely, the total potential energy of the system is minimized. Nodes with the same attributes are pulled closer, while nodes with different attributes are pushed further, making the nodes with different labels easier to be separated, regardless of the homophily of the graph.

Let us examine the physics behind spring networks. In Figure 2 suppose we have two nodes whose coordinates are \(r_1, r_2 \in \mathbb{R}^d\). According to Hooke’s Law, the force node 1 exerts on node 2 is \(F_{12} = k(||r_1 - r_2||^2 - l_0)r_1 - r_2/||r_1 - r_2||^2\), and the potential energy of this two-node system is

\[
E_p = \frac{1}{2}k(||r_1 - r_2||^2 - l_0)^2.
\]  

(1)
we have the relation $E(i, j)$.

Then, we introduce soft variables called $M_{ij}$ as a spring whose initial length is $\|Z_i(0) - Z_j(0)\|_2$. We denote the relaxed length of each spring $(i, j)$ as $M_{ij}$, resulting in a relaxed length matrix, or a metric matrix $M \in \mathbb{R}^{n \times n}$. We also assume that each spring $(i, j)$ stores the initial potential energy $E_{ij}$, resulting in an energy matrix $E \in \mathbb{R}^{n \times n}$ to be determined later. According to Equation 1, for any edge $(i, j)$, we have the relation $E_{ij} = \frac{1}{2}k_{ij}(\|Z_i(0) - Z_j(0)\|_2 - M_{ij})^2$, then we have

$$M_{ij} = A_{ij}\left(\|Z_i(0) - Z_j(0)\|_2 + \sqrt{\frac{2E_{ij}}{k_{ij}}}\right) = \begin{cases} 0, & (i, j) \notin E, \\ \|Z_i(0) - Z_j(0)\|_2 + \sqrt{\frac{2E_{ij}}{k_{ij}}}, & (i, j) \text{ compressed}, \\ \|Z_i(0) - Z_j(0)\|_2 - \sqrt{\frac{2E_{ij}}{k_{ij}}}, & (i, j) \text{ stretched}. \end{cases}$$

Then, we introduce soft variables called $\alpha_{ij} \in [-1, 1]$ to indicate whether the spring $(i, j)$ is initially stretched or compressed, and eliminate unnecessary variables $k_{ij}$ by reparameterizing $E_{ij} = 2E_{ij}/k_{ij}$ for all $(i, j) \in E$. We obtain

$$M_{ij} = A_{ij}(\|Z_i(0) - Z_j(0)\|_2 + \alpha_{ij}E_{ij}).$$

Note that $\alpha_{ij}$ approaches 1 when the spring $(i, j)$ is initially stretched and corresponds to an homophilous edge, whereas $\alpha_{ij}$ approaches -1 when the spring $(i, j)$ is initially compressed and corresponds to an heterophilous edge. According to the discussions in Section 3, we need to decrease the intra-class distances and increase the inter-class distances. Therefore, by designing the elements in the energy matrix to be $E_{ij} = \frac{4(\|Z_i(0) - Z_j(0)\|_2)^2}{\alpha_{ij}(\alpha_{ij} + 1)^2}$ for $(i, j) \in E)$, we have $M_{ij} = \frac{1 - \alpha_{ij}}{1 + \alpha_{ij}}\|Z_i(0) - Z_j(0)\|_2$.

Note that there are two different position with the same energy, thus, the spring network is an excellent model for both homophilous and heterophilous graphs.

$$F_{21} \leftrightarrow \begin{array}{c} r_1 \\ r_2 \end{array} \rightarrow F_{12}$$

Figure 2: Two nodes positioned in $r_1$ and $r_2$ with a compressed ($\|r_1 - r_2\|_2 < l_0$) spring connecting them. Each node feels a force pushing them further.

4 Potential Energy of Spring Network

Before presenting our GSN model, we first propose a potential energy function that models both homophilous and heterophilous graphs. We will also discuss the invariant property of it.

**Metric Matrix $M$.** For a graph $G = (V, E)$ with an initial node embedding matrix $Z(0) = f_\theta(X)$, we can treat each edge $(i, j)$ as a spring whose initial length is $\|Z_i(0) - Z_j(0)\|_2$. We denote the relaxed length of each spring $(i, j)$ as $M_{ij}$, resulting in a relaxed length matrix, or a metric matrix $M \in \mathbb{R}^{n \times n}$. We also assume that each spring $(i, j)$ stores the initial potential energy $E_{ij}$, resulting in an energy matrix $E \in \mathbb{R}^{n \times n}$ to be determined later. According to Equation 1, for any edge $(i, j)$, we have the relation $E_{ij} = \frac{1}{2}k_{ij}(\|Z_i(0) - Z_j(0)\|_2 - M_{ij})^2$, then we have

$$M_{ij} = A_{ij}\left(\|Z_i(0) - Z_j(0)\|_2 + \sqrt{\frac{2E_{ij}}{k_{ij}}}\right) = \begin{cases} 0, & (i, j) \notin E, \\ \|Z_i(0) - Z_j(0)\|_2 + \sqrt{\frac{2E_{ij}}{k_{ij}}}, & (i, j) \text{ compressed}, \\ \|Z_i(0) - Z_j(0)\|_2 - \sqrt{\frac{2E_{ij}}{k_{ij}}}, & (i, j) \text{ stretched}. \end{cases}$$

Then, we introduce soft variables called $\alpha_{ij} \in [-1, 1]$ to indicate whether the spring $(i, j)$ is initially stretched or compressed, and eliminate unnecessary variables $k_{ij}$ by reparameterizing $E_{ij} = 2E_{ij}/k_{ij}$ for all $(i, j) \in E$. We obtain

$$M_{ij} = A_{ij}(\|Z_i(0) - Z_j(0)\|_2 + \alpha_{ij}E_{ij}).$$

Note that $\alpha_{ij}$ approaches 1 when the spring $(i, j)$ is initially stretched and corresponds to an homophilous edge, whereas $\alpha_{ij}$ approaches -1 when the spring $(i, j)$ is initially compressed and corresponds to an heterophilous edge. According to the discussions in Section 3, we need to decrease the intra-class distances and increase the inter-class distances. Therefore, by designing the elements in the energy matrix to be $E_{ij} = \frac{4(\|Z_i(0) - Z_j(0)\|_2)^2}{\alpha_{ij}(\alpha_{ij} + 1)^2}$ for $(i, j) \in E)$, we have $M_{ij} = \frac{1 - \alpha_{ij}}{1 + \alpha_{ij}}\|Z_i(0) - Z_j(0)\|_2$.

---

1“Relaxed length” is different from “initial length”.

2The rationality and the physical meaning of this design are included in the supplementary materials.
Designing Stretch / Compress Attention $\alpha_{ij}$. Directly learning $\alpha_{ij}$ is unfeasible, since it introduces $|E|$ parameters, leading to over-fitting. In GSN, we adopt attention mechanisms to obtain $\alpha_{ij}$.

For $(i, j) \in E$, suppose their initial embeddings are $Z^{(0)}_i$ and $Z^{(0)}_j$, we first get their representation in a hidden space (for example, by MLP), then can calculate $\alpha_{ij}$ by following methods,

- “dot”: $\alpha_{ij} = g(h_i^\top h_j)$;
- “concat”: $\alpha_{ij} = g(a^\top [h_i \| h_j])$, where $a \in \mathbb{R}^{2d}$ and $\|$ denotes concatenation;
- “bilinear”: $\alpha_{ij} = g(h_i^\top W h_j)$, where $W \in \mathbb{R}^{d \times d}$;

where $g : \mathbb{R} \rightarrow [-1, 1]$ is a function such as $\tanh(x)$. Different from GAT [35] and TWIRLS [39], the edge attention $\alpha_{ij}$ can be negative, corresponding to a compressed spring and heterophilous edge.

Embedding Potential Energy w.r.t a Metric $E_p(Z; G, M)$. Having the metric matrix $M$, we can use Equation[1] to calculate the potential energy when the node embedding is $Z$,

$$E_p(Z; G, M) = \sum_{(i, j) \in E} \frac{1}{2} k_{ij}(\|Z_i - Z_j\|_2 - M_{ij})^2$$

$$= \text{tr}(Z^\top LZ) - 2 \sum_{i,j} M_{ij}\|Z_i - Z_j\|_2 + \|M\|_F^2$$

$$= \text{tr}(Z^\top LZ) - 2 1^\top \left(M \odot D^2(Z)^{\odot \frac{1}{2}} \right) 1 + \|M\|_F^2,$$

where $D^2 : \mathbb{R}^{n \times d} \rightarrow \mathbb{R}^{n \times n}$, $Y \mapsto \text{diag}(YY^\top)1^\top + 1\text{diag}(YY^\top)^\top - 2YY^\top$ is the square of node embedding distance operation, i.e., $(D^2(Y))_{ij} = \|Y_i - Y_j\|_2^2$, and $Y^{\odot k}$ is the Hadamard (element-wise) power, that is, $(Y^{\odot k})_{ij} = (Y_{ij})^k$. Obviously $E_p(Z; G, M) \geq 0$ always holds, and $E_p = 0$ if and only if $\forall (i, j) \in E, \|Z_i - Z_j\|_2 = M_{ij}$, i.e., the nodes are arranged in a way that the distances between them are consistent with the metric matrix $M$.

$E(d)$ Invariance of $E_p(Z; G, M)$. We will show that the energy function $E_p(Z; G, M)$ is invariant under $d$-dimensional orthogonal transformations (including rotation and reflection) $O(d)$ and translation transformations $T(d)$, or $E(d)$ transformations for the two types of transformations. The critical observation is that $E_p$ is invariant under $\ell_2$-isometric, or distance-preserving transformations. For the convenience of discussion, we define the isometric transformations first.

Definition 4.1. A reversible transformation $T : \mathbb{R}^{n \times d} \rightarrow \mathbb{R}^{n \times d}$, $X \mapsto T(X)$ is called $(C, d)$-isometric w.r.t a constraint set of pairs $C \subseteq \{1, 2, \cdots, n\}^2$ and a metric $d : \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}_{\geq 0}$, if for any $(i, j) \in C, d(T(Z)_i, T(Z)_j) = d(Z_i, Z_j)$ always holds.

Then, we have the following theorem:

Theorem 4.2. For any node embedding matrix $Z$, the energy function of GSN is invariant under any $(E, \ell_2)$-isometric transformation $T$, i.e., $E_p(T(Z); G, M) = E_p(Z; G, M)$.

Proof. We can express the embedding potential energy as

$$E_p(Z) = \sum_{(i, j) \in E} (\|Z_i - Z_j\|_2 - M_{ij})^2 = \sum_{(i, j) \in E} (\|T(Z)_i - T(Z)_j\|_2 - M_{ij})^2 = E_p(T(Z)),$$

and the Theorem follows. $\square$

Note that for any $d$-dimensional orthogonal matrix $Q \in O(d)$ and any translation matrix $S = 1s^\top \in T(d)$ (where $s \in \mathbb{R}^d$ stands for a translation vector), the Euclidean transformation $T(Z) = ZQ + S$ is $(V^2, \ell_2)$-isometric (and is also $(E, \ell_2)$-isometric). Therefore, $E_p$ has $E(d)$ invariant property.

*Discussions of the relation of $(C, \ell_2)$-isometry and graph rigidity are included in supplementary materials.
5 The GSN Model

Below is the update rule of GSN, \( \mathbf{L}_{\mathbf{H}}^{\text{sym}} := \mathbf{D}^{-1/2} \mathbf{L}_{\mathbf{H}} \mathbf{D}^{-1/2} \), where \( \mathbf{L}_{\mathbf{H}} = \text{diag}(\mathbf{H}_1) - \mathbf{H} \) and \( \mathbf{H} = \mathbf{M} \odot \mathcal{D}^2(\mathbf{D}^{-1/2} \mathbf{Z})^{\odot -1/2} \). In this section, we will derive this rule from the energy function of GSN.

**Update Rule of GSN**

\[
\mathbf{Z}^{(k+1)} = \sigma \left( \left( (1 - \alpha) \mathbf{P}^\text{sym} + 2\beta \mathbf{L}_{\mathbf{H}}^{\text{sym}} \right) \mathbf{Z}^{(k)} + \alpha f_{\theta}(\mathbf{X}) \right) \mathbf{W}^{(k)}. 
\]

**Optimization Objection.** Imagine that we put a spring network into a medium with resistance and let the system evolve freely. When the system is at the equilibrium state, the final embedding \( \mathbf{Z}^\ast \) should optimize the energy function \( E_p \). Recall that the energy function reaches its minimum zero when the nodes are arranged with the guidance of the metric matrix. Therefore, GSN aims to find a good metric between nodes and arrange them according to the metric easily. Meanwhile, many GNN models can be viewed as minimizing different energy functions on graphs \([39, 42]\). Specifically, in Section 6, we will show that many existing GNN models are designed with \( \mathbf{M} = \mathbf{O} \). Therefore, we can make reasonable speculation that the performance of GSN comes from this carefully designed energy function. In order to be consistent with the symmetric normalized Laplacian, which is widely used in various GNN models such as GCN \([17]\), APPNP \([18]\), etc., we need to reparameterize \( \mathbf{Z} \) as \( \mathbf{Z} = \mathbf{D}^{-1/2} \mathbf{Z} \). Then we define the “normalized” energy as

\[
\tilde{E}_p(\mathbf{Z}; \mathbf{G}, \mathbf{M}) = \text{tr}(\mathbf{Z}^\top \mathbf{L}_{}^{\text{sym}} \mathbf{Z}) - 2 \sum_{i,j} \mathbf{M}_{ij} \left\| \frac{\mathbf{Z}_{ij}}{\sqrt{d_i}} - \frac{\mathbf{Z}_{ji}}{\sqrt{d_j}} \right\|_2 + \| \mathbf{M} \|_F^2.
\]

From Theorem 4.2 above, we know that if \( \mathbf{Z}^\ast \) minimizes \( E_p(\mathbf{Z}; \mathbf{G}, \mathbf{M}) \), then \( \mathbf{Z}' = \mathbf{Z}^\ast \mathbf{Q} + \mathbf{S} \) also minimizes the energy function. According to the locality concept in physics, we think the “best” solution should be close to the initial embedding \( \mathbf{Z}^{(0)} = f_0(\mathbf{X}) \), resulting in a trade-off regularization term \( \alpha \left\| \mathbf{Z} - \mathbf{Z}^{(0)} \right\|_F^2 \). Now we have the objection \( \mathcal{E} = (1 - \alpha) \tilde{E}_p(\mathbf{Z}; \mathbf{G}, \mathbf{M}) + \alpha \left\| \mathbf{Z} - f_0(\mathbf{X}) \right\|_F^2 \).

**Formula of a GSN Layer.** First, we compute the gradient of \( \mathcal{E} \) w.r.t the node embedding \( \mathbf{Z} \), and we get (denoting \( \mathbf{H} = \mathbf{M} \odot \mathcal{D}^2(\mathbf{D}^{-1/2} \mathbf{Z})^{\odot -1/2} \) for short)

\[
\frac{\partial \mathcal{E}}{\partial \mathbf{Z}} = 2(1 - \alpha) \left( \mathbf{L}^{\text{sym}} - 2\beta \mathbf{L}_{\mathbf{H}}^{\text{sym}} \right) \mathbf{Z} + 2\alpha (\mathbf{Z} - f_0(\mathbf{X})),
\]

where \( \mathbf{L}_{\mathbf{H}} \) stands for \( \text{diag}(\mathbf{H}_1) - \mathbf{H} \), since if we treat \( \mathbf{H} \) as an adjacency matrix of some graph, then \( \text{diag}(\mathbf{H}_1) - \mathbf{H} \) can be thought of the Laplacian of that graph. By setting the gradient to be zero, it leads to \( \mathbf{Z} = (1 - \alpha) \mathbf{P}^{\text{sym}} \mathbf{Z} + 2(1 - \alpha) \mathbf{L}_{\mathbf{H}}^{\text{sym}} \mathbf{Z} + \alpha f_0(\mathbf{X}) \). We can define \( \mathbf{P}^{\text{sym}} \mathbf{Z} \) part as “topological message”, and \( \mathbf{L}_{\mathbf{H}}^{\text{sym}} \mathbf{Z} \) part as “position message”. Finally, by decoupling coefficients of the two types of messages, using the renormalization trick \([17]\) to the topological message part and introducing non-linearity, we obtain the update rule of a GSN layer\(^4\):

\[
\mathbf{Z}^{(k+1)} = \sigma \left( \left( (1 - \alpha) \mathbf{P}^\text{sym} + 2\beta \mathbf{L}_{\mathbf{H}}^{\text{sym}} \right) \mathbf{Z}^{(k)} + \alpha f_{\theta}(\mathbf{X}) \right) \mathbf{W}^{(k)}. 
\]

**Structure of the Framework.** The structure and formula of GSN is summarized below.

1. **(Embedding)** \( \mathbf{Z}^{(0)} = f_{\theta}^{(1)}(\mathbf{X}) \). \( f_{\theta}^{(1)} \) can be a dense layer or an MLP to reduce dimension.

2. **(Attention)** \( \mathbf{H}_k = f_{\theta}^{(2)}(\mathbf{Z}_k) \), and \( \alpha_{ij} = \begin{cases} g(\mathbf{H}_i^\top \mathbf{H}_j) & ((i,j) \in E) \end{cases} \). \( f_{\theta}^{(2)} \) can be a dense layer, or an MLP to map initial embeddings to a hidden space to extract the edge attention.

\[^4\]We use the iterative method to get the propagation rule of GSN and to avoid the high cost of computation. We also include the derivation of the node-level message-passing scheme in the supplementary materials.
3. (Convolution) $Z^{(k+1)} = \sigma \left( \left( (1-\alpha)\tilde{P}^{\text{sym}} + 2\beta L_h^{\text{sym}} \right) Z^{(k)} + \alpha f_\theta^{(3)}(X) \right) W^{(k)}$, for $k = 0, 1, \cdots, K-1$. Here we may decouple $f_\theta^{(1)}$ and $f_\theta^{(3)}$ to allow more flexibility. Usually, we can still keep $f_\theta^{(3)} = f_\theta^{(1)}$ to reduce the computation cost.

4. (Output) $Z^{(\text{out})} = \text{softmax}(Z^{(K)}W^{(K)})$.

6 Reinterpret Related Works Using GSN

In this section, we review some specific GNNs and show that they can be simulated by GSN directly or under minor modification without knowing the output embedding of a model in advance, which provides physical motivation for these models.

Simulate and Interpret GCN and SGC. If we set the metric matrix $M = O$ (or $\beta = 0$) and $\alpha = 0$ in Equation 4, it will lead to $Z^{(k+1)} = \sigma \left( PZ^{(k)} W^{(k)} \right)$, which is the same as the propagation rule of GCN [17]. SGC [37] removes the linear transformation and non-linear activation function in GCN; thus, it can be thought of as a special case of GCN, and our GSN framework can also simulate SGC by doing the same. If we treat GCN and SGC as the special cases of GSN, their metric $M$ is $O$. Therefore, if we stack too many layers of them, the distance between node $i$ and $j$ will be close in the final embedding, causing the over-smoothing problem.

Simulate and Interpret PPNP and APPNP. If we only set the metric matrix $M = O$ (or $\beta = 0$), which means that we both care about the topological message and the initial embedding, and we do not care about the positional message. Moreover, if we remove the non-linear activation function and the linear transformation, it will lead to APPNP’s [18] propagation rule $Z^{(k+1)} = (1-\alpha)\tilde{P}Z^{(k)} + \alpha Z^{(0)}$. Since the APPNP model is the approximate version of PPNP, one can stack infinity APPNP layers to reach PPNP, and the GSN framework can also achieve this by doing the same. Compared to GCN and SGC above, PPNP and APPNP introduce the regularization term $\|Z - Z^{(0)}\|_F^2$, which may alleviate the over-smoothing problem. However, it does not change the metric $M = O$; thus, it can not deal with heterophilous graphs.

Modify GSN to Simulate GCNII. Furthermore, we can modify our GSN to simulate GCNII [5]. We can introduce the identity mapping by substituting $W^{(k)}$ in Equation 4 with $((1-\gamma)I + \gamma W^{(k)})$, leading to the augmented propagation rule of GSN with identity mapping (which we name it GSN+). Note that when $\gamma = 1$, it degenerates to GSN. If we set $\beta = 0$ in GSN+’s propagation equation, we will get $Z^{(k+1)} = \sigma \left( \left( (1-\alpha)\tilde{P}Z^{(k)} + \alpha Z^{(0)} \right) \left( (1-\gamma)I + \gamma W^{(k)} \right) \right)$, which is the same as GCNII’s propagation rule. Therefore, the modified version GSN+ can simulate GCNII. GCNII can be regarded as APPNP with linear transformation and identity mapping, but it also does not change the metric, so it is not good at heterophilous graphs.

Interpret ElasticGNN (with $\ell_{21}$ Regularization Term). Interestingly, suppose we set $M = A \odot 11^T$ and add self-loops to the graph. In that case, the second term of our normalized energy $\hat{E}_p(Z; G, M)$ will be $\sum_{(i,j) \in E} \left\| \frac{Z_i}{\sqrt{d_i+1}} - \frac{Z_j}{\sqrt{d_j+1}} \right\|_2$ (see Equation 3), which is the same as the $\ell_{21}$ regularization term $\|\Delta F\|_{21}$ defined in ElasticGNN [23]. Therefore, by changing the coefficients of regularization terms and setting $M = A \odot 11^T$, our GSN framework can also simulate the ElasticGNN with $\ell_{21}$ norm. However, the limitation of fixed $M$ also leads to its failure on the heterophilous datasets.

Generalize GSN to Simulate Any Spectral GNN. Spectral GNNs aim to learn a complex graph filter $h(\tilde{L})$ (or $h'(\tilde{P}) = h'(I - \tilde{L})$ equivalently), and produce the final embedding by $Z = h(\tilde{L})Z^{(0)}$. Many models approximate the filter $h$ using polynomial basis. For example, GPR-GNN [4] uses $Z^{(k)} = \sum_{k=0}^K \gamma_k \tilde{P}^{(k)}Z^{(0)}$. We can also generalize GSN to approximate any polynomial filter. We remove linear transformations and non-linear activation functions, set $\beta = 0$ in GSN, then allow each layer to learn different coefficients of $\tilde{P}Z^{(k)}$ and $Z^{(0)}$. As a result, we get the propagation rule $Z^{(k)} = a_{k-1}\tilde{P}Z^{(k-1)} + b_{k-1}Z^{(0)}$. Inductively using the propagation rule for
$K$ times, we get $Z^{(K)} = \sum_{k=0}^{K} \left( \prod_{i=K-k}^{K-1} a_i \right) b_{K-1-k} \tilde{P}^k Z^{(0)}$ (define $b_{-1} = 1$). Compare it with a $K$ layer GPR-GNN’s propagation rule, if for any series of $\gamma_k$ always exists corresponding $a_k$ and $b_k$, such that $\forall k \in \{0, 1, \cdots, K\}$, $\left( \prod_{i=K-k}^{K-1} a_i \right) b_{K-1-k} = \gamma_k$ always holds, then GSN can simulate GPR-GNN. Obviously, if we set

$$a_k = \begin{cases} \frac{\gamma_{K-k}}{|\gamma_{K-1-k}|}, & 0 \leq k < K-1 \\ |\gamma_1|, & k = K-1 \end{cases}$$

and

$$b_k = \begin{cases} \text{sign}(\gamma_{K-1-k}), & 0 \leq k < K-1 \\ \gamma_0, & k = K-1 \end{cases},$$

the equation above holds; thus the generalized GSN can also approximate GPR-GNN and any polynomial graph filter.

7 Experiments

All the experiments are done on a server with 40 Intel Xeon Silver 4114 CPUs (2.20GHz) (only one are used), an Nvidia Quadro RTX 8000 GPU (48GB memory) and 1TB RAM (<64GB are used).

Datasets. For homophilous graphs, we use three public citation networks: Cora, Citeseer, and PubMed from [40], and CoraFull dataset from [3]. For heterophilous graphs, we choose the WebKB networks: Cornell, Texas, and Wisconsin from [28], the Wikipedia networks: Chameleon, and Squirrel from [29], and the Actor dataset from [28]. The statistics of the datasets are listed in Table 2.

| Dataset   | # Nodes | # Edges  | # Features | # Classes | $\mathcal{H}(G, Y)$ |
|-----------|---------|----------|------------|-----------|-------------------|
| Cora      | 2,708   | 10,556   | 1,433      | 7         | 0.81              |
| Citeseer  | 3,327   | 9,104    | 3,703      | 6         | 0.74              |
| PubMed    | 19,717  | 66,648   | 500        | 3         | 0.80              |
| CoraFull  | 19,793  | 126,842  | 8,710      | 70        | 0.57              |
| Cornell   | 183     | 298      | 1,703      | 5         | 0.31              |
| Texas     | 183     | 325      | 1,703      | 5         | 0.11              |
| Wisconsin | 251     | 515      | 1,703      | 5         | 0.20              |
| Chameleon | 2,277   | 36,101   | 2,325      | 5         | 0.24              |
| Squirrel  | 5,201   | 217,073  | 2,089      | 5         | 0.22              |
| Actor     | 7,600   | 30,019   | 932        | 5         | 0.22              |

Supervised Node Classification. To compare our GSN framework with previous works, we choose representative models from four categories, including GCN [17], GraphSAGE [12], GAT [35], GCNII [5], GPR-GNN [6], FACGCN [2], H$_2$GCN [41], and DMP-1/2-Sum [38] as baseline models. We implement the GSN model with the PyTorch [27] and PyTorch Geometric [9] libraries. Following the experiment setting in [41] and [38], we randomly generate ten train/validation/test splits with the 48%/32%/20% ratio for all datasets. We train our model for 500 epochs on each of the ten splits for a specific hyper-parameter group. For GSN, we fix the hidden_dim to be 64, and use grid search method to select other hyper-parameters. The selection ranges of hyper-parameters are listed below:

- \(lr\): \{5e-4, 1e-3, 5e-3, 1e-2, 5e-2\};
- \(weight\_decay\): \{1e-6, 1e-5, \cdots, 1e-2\};
- \(dropout\): \{0.25, 0.35, 0.45, 0.55\};
- \(num\_layers\): \{2, 4\};
- \(alpha\ and \beta\): \{0.1, 0.2, \cdots, 0.9\};
- \(attention\_method\): \{‘concat’, ‘bilinear’\}.

Table 3 lists the classification accuracy on homophilous and heterophilous datasets. The results of GCN, GraphSAGE, GAT and H$_2$GCN are taken from [41], and the results of GPR-GNN and DMP are taken from [38]. The results show that our GSN framework performs well on both homophilous and heterophilous datasets.

Table 3 lists the classification accuracy on homophilous and heterophilous datasets. The results of GCN, GraphSAGE, GAT and H$_2$GCN are taken from [41], and the results of GPR-GNN and DMP are taken from [38]. The results show that our GSN framework performs well on both homophilous and heterophilous datasets.

The best hyper-parameters are included in the supplementary materials.
Table 3: Mean accuracy (± standard derivation) of the supervised node classification experiments. Numbers with **boldface** denotes the best result, and numbers with **underline** stands for the second.

| Dataset    | Homophilous | Heterophilous |
|------------|-------------|---------------|
|            | Spectral    | Spatial       | Spectral | Spatial       |
|            | GCN         | SAGE GAT GCNII | GPR. FAGCN | H₂GCN DMP GSN |
| Cora       | 87.28       | 86.90 82.68 87.90 | 88.14 87.58 87.81 85.31 | **88.41 ± 0.94** |
| CiteSeer   | 76.68       | 76.04 75.46 75.14 | 74.07 76.24 **77.07** | 76.27 76.78 ± 1.38 |
| PubMed     | 87.38       | 88.45 84.68 88.47 | 88.27 88.82 89.59 88.15 | **89.88 ± 0.41** |
| CoraFull   | 68.39       | 65.14 59.81 | 69.75 | 69.25 N/A 69.05 N/A | **69.79 ± 0.78** |
| Cornell    | 57.03       | 75.95 58.92 72.97 | **91.14** | 76.49 82.16 83.78 | 85.95 ± 4.65 |
| Texas      | 59.46       | 82.43 58.38 73.78 | **90.49** | 81.62 84.86 86.48 | **87.03 ± 4.80** |
| Wisconsin  | 59.80       | 81.18 55.29 81.00 | 85.33 **86.86** | 86.67 86.27 | **88.60 ± 3.99** |
| Chameleon  | **67.90**   | 58.73 54.69 61.32 | 66.31 **60.00** | 59.39 55.92 | **69.91 ± 1.61** |
| Squirrel   | **54.41**   | 41.61 30.62 40.42 | 50.56 **38.69** | 37.90 43.42 | **58.89 ± 1.13** |
| Actor      | 30.26       | 34.23 26.28 34.78 | 34.17 **36.04** | 35.86 34.93 | **36.94 ± 1.12** |

| Avg. Rank  | 5.9         | 5.9 8.5 5.3 | 4.0 | 4.4 | 3.9 | 5.0 | 1.3 |

1 Results are taken from [24]. It is reported that these models can perform well on Chameleon and Squirrel by hyper-parameter tuning.

**Alleviating the Over-Smoothing Problem.** To show that our GSN model also alleviate the over-smoothing issue, we also perform experiments on GCN and GSN of 2, 4, 8, 16, 32, 64, and 128 layers with 64 hidden channels on relatively large datasets: CoraFull, PubMed, Chameleon and Squirrel. Generally, we set $lr = 1e$-2, $weight\_decay = 1e$-4 and $dropout = 0.25$ across different datasets. And for GSN, we add $alpha = 0.2$, $beta = 0.1$ and $attention\_method = \text{‘bilinear’}$ (and we set $lr = 1e$-3 to avoid divergence on Squirrel).

Table 4 lists the performance of GCN and GSN with various layers, showing that GSN does not suffer from the over-smoothing problem on both homophilous and heterophilous datasets. This is because a GSN model with an infinite number of layers effectively minimizes the designed energy function (1) whose solution contains information about both node features and graph structure.

Table 4: Mean accuracy (%) with various layers. ↓ (%) stands for the accuracy of the best model minus that of the deepest (128-layer) model.

| Dataset    | Method | Layers | 2 | 4 | 8 | 16 | 32 | 64 | 128 | ↓ |
|------------|--------|--------|---|---|---|----|----|----|-----|---|
| PubMed     | GCN    |        | **88.49** | 86.39 | 84.40 | 49.45 | 48.64 | 48.16 | 49.61 | 38.88 |
|            | GSN    |        | 88.90 | **89.24** | 88.90 | 89.05 | 88.97 | 89.02 | 88.79 | **0.45** |
| Chameleon  | GCN    |        | **68.07** | 64.97 | 58.55 | 47.52 | 46.59 | 47.08 | 46.02 | 22.05 |
|            | GSN    |        | 66.31 | 68.88 | **69.49** | 67.47 | 68.15 | 68.42 | 68.77 | **0.72** |
| Squirrel   | GCN    |        | **52.13** | 46.46 | 38.92 | 35.91 | 35.22 | 35.47 | 34.97 | 17.16 |
|            | GSN    |        | 58.01 | 58.28 | 57.46 | 57.83 | 58.31 | **58.56** | 57.32 | **1.24** |

**Ablation Study** The main difference between GCNII and GSN is that GSN introduces the positional message term $\beta L_{\text{sym}}^Z$. In order to show the effectiveness of our design, we conduct experiments with GSN on four relatively small heterophilous datasets. We use a 2-layer GSN with 64 hidden units, setting $lr = 1e$-2, $weight\_decay = 1e$-2, $dropout = 0.25$, $alpha = 0.1$, $attention\_method = \text{‘concat’}$ and vary $\beta$ from 0 to 1.0 to adapt the ratio of the positional message.

Table 5 lists the result of our experiments. The accuracies on three datasets increase when we increase $\beta$, thus, we can conclude that the design of the GSN is effective.
Table 5: Mean accuracy (%) of GSN on four heterophilous datasets with $\alpha = 0.1$ and various $\beta$. ↑(%) stands for the accuracy of the best model minus that of the $\beta = 0$ model.

| Dataset   | 0   | 0.1 | 0.2 | 0.3 | 0.4 | 0.5 | 0.6 | 0.7 | 0.8 | 0.9 | 1.0 | ↑  |
|-----------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|----|
| Cornell   | 66.22| 75.95| 81.35| 76.38| 78.38| 80.54| 81.62| 77.84| 78.92| 75.95| 74.86| 15.80|
| Texas     | 62.43| 76.49| 82.70| 80.27| 80.54| 80.00| 81.62| 82.16| **84.59**| 81.35| 82.70| 22.16|
| Wisconsin | 68.40| 81.20| 84.40| **84.80**| 84.40| 83.00| 81.80| 84.40| 81.40| 82.00| 83.00| 16.40|
| Actor     | 26.09| 26.38| 26.61| 27.74| 31.02| 33.08| 33.76| 34.05| 34.17| 34.23| **34.38**| 8.29|

8 Conclusions

In this paper, inspired by the spring network model, we propose the Graph Spring Network which fills the gap that most GNNs designed in the spatial domain do not have theoretical supports or physical motivations. We discover invariant properties of the energy function and the universality of GSN through mathematical derivations. We have also done experiments with our GSN and the results confirm the good performance of it.

Limitations and Broader Impacts

We consider our work to be a theoretical contribution and has no immediate societal impacts. However, there are lots of aspects, such as more universality of GSN, more relations between GSN and graph rigidity, and the industrial applications of GSN, still waiting for explorations. We leave explorations of those aspects to future work.

References

[1] Romain Amyot, Yuichi Togashi, and Holger Flechsig. Analyzing fluctuation properties in protein elastic networks with sequence-specific and distance-dependent interactions. *Biomolecules*, 9(10), 2019.

[2] Deyu Bo, Xiao Wang, Chuan Shi, and Huawei Shen. Beyond low-frequency information in graph convolutional networks. In *AAAI*. AAAI Press, 2021.

[3] Aleksandar Bojchevski and Stephan Günnemann. Deep gaussian embedding of graphs: Unsupervised inductive learning via ranking. In *International Conference on Learning Representations*, 2018.

[4] Jonathan Bourne. The spring bounces back: introducing the strain elevation tension spring embedding algorithm for network representation. *Applied Network Science*, 5, 11 2020.

[5] Ming Chen, Zhewei Wei, Zengfeng Huang, Bolin Ding, and Yaliang Li. Simple and deep graph convolutional networks. *Proceedings of the 37th International Conference on Machine Learning*, 2020.

[6] Eli Chien, Jianhao Peng, Pan Li, and Olgica Milenkovic. Adaptive universal generalized pagerank graph neural network. In *International Conference on Learning Representations*, 2021.

[7] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems*, 2016.

[8] Peter Eades. A heuristic for graph drawing. *Congressus Numerantium*, 42:149–160, 1984.

[9] Matthias Fey and Jan E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[10] Thomas M. J. Fruchterman and Edward M. Reingold. Graph drawing by force-directed placement. *Software: Practice and Experience*, 21, 1991.
[11] Turkan Haliloglu, Ivet Bahar, and Burak Erman. Gaussian dynamics of folded proteins. *Phys. Rev. Lett.*, 79:3090–3093, Oct 1997.

[12] William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *NIPS*, 2017.

[13] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[14] Mingguo He, Zhewei Wei, Zengfeng Huang, and Hongteng Xu. Bernnet: Learning arbitrary graph spectral filters via bernstein approximation. In *Advances in Neural Information Processing Systems*, 2021.

[15] Bill Jackson. Notes on the rigidity of graphs. 2007.

[16] Tomihisa Kamada and Satoru Kawai. An algorithm for drawing general undirected graphs. *Information Processing Letters*, 31(1):7–15, 1989.

[17] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations (ICLR)*, 2017.

[18] Johannes Klicpera, Aleksandar Bojchevski, and Stephan Günnemann. Predict then propagate: Graph neural networks meet personalized pagerank. In *International Conference on Learning Representations (ICLR)*, 2019.

[19] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012.

[20] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[21] Q. Li, Z. Han, and X.-M. Wu. Deeper Insights into Graph Convolutional Networks for Semi-Supervised Learning. In *The Thirty-Second AAAI Conference on Artificial Intelligence*. AAAI, 2018.

[22] Tu-Liang Lin and Guang Song. Generalized spring tensor models for protein fluctuation dynamics and conformation changes. In *2009 IEEE International Conference on Bioinformatics and Biomedicine Workshop*, pages 136–143, 2009.

[23] Xiaorui Liu, Wei Jin, Yao Ma, Yaxin Li, Hua Liu, Yiqi Wang, Ming Yan, and Jiliang Tang. Elastic graph neural networks. In *Proceedings of the 38th International Conference on Machine Learning*, 2021.

[24] Yao Ma, Xiaorui Liu, Neil Shah, and Jiliang Tang. Is homophily a necessity for graph neural networks? In *International Conference on Learning Representations*, 2022.

[25] Hoang NT, Takanori Maehara, and Tsuyoshi Murata. Revisiting graph neural networks: Graph filtering perspective. In *25th International Conference on Pattern Recognition, ICPR 2020, Virtual Event / Milan, Italy, January 10-15, 2021*, pages 8376–8383. IEEE, 2020.

[26] Kenta Oono and Taiji Suzuki. Graph neural networks exponentially lose expressive power for node classification. In *International Conference on Learning Representations*, 2020.

[27] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32:8026–8037, 2019.

[28] Hongbin Pei, Bingzhe Wei, Kevin Chen-Chuan Chang, Yu Lei, and Bo Yang. Geom-gcn: Geometric graph convolutional networks. In *International Conference on Learning Representations*, 2020.
[29] Benedek Rozemberczki, Carl Allen, and Rik Sarkar. Multi-Scale attributed node embedding. *Journal of Complex Networks*, 9(2), 05 2021.

[30] James B. Saxes. Embeddability of weighted graphs in k-space is strongly np-hard. Technical report, Carnegie Mellon University, USA, 1979.

[31] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, 2015.

[32] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–9, 2015.

[33] Daniel Tunkelang. A practical approach to drawing undirected graphs. Technical report, Carnegie Mellon University, USA, 1994.

[34] William T. Tutte. How to draw a graph. *Proceedings of The London Mathematical Society*, 13:743–767, 1963.

[35] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph Attention Networks. *International Conference on Learning Representations*, 2018.

[36] Yu Wang and Tyler Derr. Tree decomposed graph neural network. In *CIKM ’21: The 30th ACM International Conference on Information and Knowledge Management*, Virtual Event, Queensland, Australia, November 1 - 5, 2021, pages 2040–2049. ACM, 2021.

[37] Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. Simplifying graph convolutional networks. In *Proceedings of the 36th International Conference on Machine Learning*, pages 6861–6871. PMLR, 2019.

[38] Liang Yang, Mengzhe Li, Liyang Liu, Bingxin Niu, Chuan Wang, Xiaochun Cao, and Yuan-fang Guo. Diverse message passing for attribute with heterophily. In *Advances in Neural Information Processing Systems*, 2021.

[39] Yongyi Yang, Tang Liu, Yangkun Wang, Jinjing Zhou, Quan Gan, Zhewei Wei, Zheng Zhang, Zengfeng Huang, and David Wipf. Graph neural networks inspired by classical iterative algorithms. In *Proceedings of the 38th International Conference on Machine Learning*, 2021.

[40] Zhilin Yang, William W. Cohen, and Ruslan Salakhutdinov. Revisiting Semi-Supervised Learning with Graph Embeddings. In *Proceedings of the 33rd International Conference on Machine Learning*, pages 40–48, 2016.

[41] Jiong Zhu, Yujun Yan, Lingxiaozhao, Mark Heimann, Leman Akoglu, and Danai Koutra. Beyond homophily in graph neural networks: Current limitations and effective designs. *Advances in Neural Information Processing Systems*, 33, 2020.

[42] Meiqi Zhu, Xiao Wang, Chuan Shi, Houye Ji, and Peng Cui. Interpreting and unifying graph neural networks with an optimization framework. In *WWW ’21: The Web Conference 2021, Virtual Event / Ljubljana, Slovenia, April 19-23, 2021*, pages 1215–1226, 2021.

**Appendix**

A **Designing the Initial Energy Matrix E**

To apply the spring network framework on GNN, we need to calculate the potential energy of the spring network for any intermediate node embedding $Z$, which requires the derivation of the initial energy matrix $E$. Intuitively, $E$ should be a function of the initial node embedding $Z^{(0)}$ and the Stretch / Compress Attention $\alpha_{ij}$. We will first discuss some physical and intuitive conditions that $E$ should fulfill:
1. (Non-negativity) Since energy in springs and the length of the springs are greater than 0, we have \( E_{ij} \geq 0 \) and \( M_{ij} \geq 0 \) if \( \|Z_{i:}^{(0)} - Z_{j:}^{(0)}\|_2 \geq \alpha_{ij} \sqrt{E_{ij}}, \) leading to \( 0 \leq E_{ij} \leq \frac{\|Z_{i:}^{(0)} - Z_{j:}^{(0)}\|_2^2}{\alpha_{ij}}, \) \( 0 < \alpha_{ij} \leq 1 \), \( -1 \leq \alpha_{ij} \leq 0 \).

2. (Homophilous limit) We should minimize the distances of the nodes in the same class. In other words, when \( \alpha \to 1^- \), the metric \( M_{ij} \to 0 \), i.e. \( E_{ij} \to \|Z_{i:}^{(0)} - Z_{j:}^{(0)}\|_2^2 \).

3. (Heterophilous limit) We should maximize the distances of the nodes in different classes. In other words, when \( \alpha \to -1^+ \), \( M_{ij} \to +\infty \) (or a large number), i.e. \( E_{ij} \to +\infty \).

For simplicity, we set

\[
E_{ij} = \frac{4\|Z_{i:}^{(0)} - Z_{j:}^{(0)}\|_2^2}{(\alpha_{ij} + 1)^2},
\]

which satisfies the above three conditions. It follows that

\[
M_{ij} = \|Z_{i:}^{(0)} - Z_{j:}^{(0)}\|_2 - \frac{2\alpha_{ij}}{\alpha_{ij} + 1}\|Z_{i:}^{(0)} - Z_{j:}^{(0)}\|_2 = \frac{1 - \alpha_{ij}}{1 + \alpha_{ij}}\|Z_{i:}^{(0)} - Z_{j:}^{(0)}\|_2.
\]

### B Discussions of the \((C, d)\)-isometric Transformations and Graph Rigidity

Recall the definition of \((C, d)\)-isometric transformations.

**Definition B.1.** A reversible transformation \( T : \mathbb{R}^{n \times d} \to \mathbb{R}^{n \times d}, X \mapsto T(X) \) is called \((C, d)\)-isometric w.r.t a constraint set of pairs \( C \subseteq \{1, 2, \cdots, n\}^2 \) and a metric \( d : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}_{\geq 0} \), if for any \((i, j) \in C, d(T(Z)_i, T(Z)_j) = d(Z_i, Z_j)\) always holds.

We will discuss some properties of them in this section. We first define the \((C, d)\)-isometric (binary) relation among real matrices.

**Definition B.2.** Two matrices \( Z_1, Z_2 \in \mathbb{R}^{n \times d} \) have the \((C, d)\)-isometric relation, denoted as \( \sim_{(C, d)} \), if there is a \((C, d)\)-isometric transformation \( T \) such that \( Z_2 = T(Z_1) \).

Then we have the following theorem,

**Theorem B.3.** \( \sim_{(C, d)} \) is an equivalence relation.

**Proof.** One can easily verify that \( \sim_{(C, d)} \) satisfy:

1. **Reflexivity**. For any matrix \( Z, Z \sim_{(C, d)} Z \) holds, and the corresponding transformation is the identity mapping \( \text{id}(X) = X \).

2. **Symmetry**. If \( Z_1 \sim_{(C, d)} Z_2 \) w.r.t. the transformation \( T \), then \( Z_2 \sim_{(C, d)} Z_1 \) also holds, and the corresponding transformation is the inverse transformation \( T^{-1} \).

3. **Transitivity**. If \( Z_1 \sim_{(C, d)} Z_2 \) and \( Z_2 \sim_{(C, d)} Z_3 \) w.r.t. the transformation \( T_1 \) and \( T_2 \), then \( Z_1 \sim_{(C, d)} Z_3 \) also holds, and the corresponding transformation is \( T_2 \circ T_1(X) = T_2(T_1(X)) \).

Therefore, the theorem holds.

Since \( \sim_{(C, d)} \) is an equivalence relation, the set of all \( n \times d \) real matrices \( \mathbb{R}^{n \times d} \) can be divided into many equivalence classes. That is, for any matrix \( Z \), the equivalence class of it is \( [Z]_{\sim_{(C, d)}} = \{Z' : Z \sim_{(C, d)} Z'\} \).
B.1 Some Concepts in Graph Rigidity

For a graph $G = (V, E)$, we can treat the nodes as joints and the edges as bars. Intuitively, there are some graphs or structures that are stable or rigid under “motions”, while others are unstable or flexible. For example, in Figure 3 we have a triangle $K_3$ (left) and a square $C_4$ (right). The structure of a triangle is very stable, the shape of it does not change with the motion which keeps the lengths of the edges. On the contrary, the structure of a square is not stable, one can push a node to change it to a parallelogram. The shape of it changes, while the lengths of the edges do not change.

![Figure 3: A stable (rigid) graph, and an unstable (flexible) graph.](image)

In the theory of graph rigidity, the definitions of equivalent, congruent, rigid and globally rigid are relevant to these concepts. We find those definitions from [15], and we list them as equivalent forms:

**Definition B.4 (Equivalent).** Two embedding matrices $Z^{(1)}$ and $Z^{(2)}$ of a graph $G$ are equivalent if \[ \|Z^{(1)}_{i} - Z^{(1)}_{j}\|_2 = \|Z^{(2)}_{i} - Z^{(2)}_{j}\|_2 \] for all $(i, j) \in E$.

**Definition B.5 (Congruent).** Two embedding matrices $Z^{(1)}$ and $Z^{(2)}$ of a graph $G$ are congruent if \[ \|Z^{(1)}_{i} - Z^{(1)}_{j}\|_2 = \|Z^{(2)}_{i} - Z^{(2)}_{j}\|_2 \] for all $i, j \in V$.

**Definition B.6 (Globally Rigid).** An embedding matrix $Z$ of a graph $G$ is globally rigid if all its equivalent embedding matrices $Z'$ are also congruent to $Z$.

**Definition B.7 (Rigid).** An embedding matrix $Z$ of a graph $G$ is rigid if there exists $\varepsilon > 0$ such that every equivalent embedding $Z'$ and satisfies $\|Z_{v'} - Z'_{v}\|_2 < \varepsilon$ for all $v \in V$, is congruent to $Z$. Or informally, all embeddings that can be obtained by continuous motion from $Z$ are congruent to $Z$.

It is reasonable to separate rigid graphs with globally rigid graphs, since not all rigid graphs are globally rigid. For example, in Figure 4 we have a 2-dimensional embedding of a graph (left). It is a rigid embedding, since all continuous motion keeps the “shape” of the embedding. But it is not globally rigid, since there exists a equivalent but not congruent embedding (which can not be reached through continuous motions in $\mathbb{R}^2$) of $G$ (right).

![Figure 4: A rigid but not globally rigid graph.](image)

B.2 Relations Between Graph Rigidity and $(C, d)$-isometric Transformations

Notably, two special cases in our definition of the $(C, d)$-isometric transformations are relevant to the theory of graph rigidity. For two embedding matrices $Z_1$ and $Z_2$, they are equivalent (or congruent) iff. they are $(E, \ell_2)$-isometric (or $(V^2, \ell_2)$-isometric correspondingly). One can easily verify that all congruent transformations are equivalent, and all $(V^2, \ell_2)$-isometric transformation are $(E, \ell_2)$-isometric. We summarize their relations in Figure 5.

It is obvious that only orthogonal transformations (rotation, reflection) and translation transformations will keep the shape of an object, thus they are congruent and $(V^2, \ell_2)$-isometric transformations, regardless of the topological structure of the graph. A natural question arises: are there $(E, \ell_2)$-
Then the third term can be rearranged as

\[
\sum_{j=1}^{V} (L_{H}^{sym})_{ij} Z_{j}^{(k)}
\]

\[
= (L_{H}^{sym})_{ii} Z_{i}^{(k)} + \sum_{j, i \neq j}^{V} (L_{H}^{sym})_{ij} Z_{j}^{(k)}
\]

\[
= \sum_{j \in N(i)} M_{ij} \frac{\|Z_{j}^{(k)} - Z_{i}^{(k)}\|_2}{\sqrt{D_{ii}D_{jj}}} Z_{i}^{(k)} - \sum_{j \in N(i)} M_{ij} \frac{\|Z_{i}^{(k)} - Z_{j}^{(k)}\|_2}{\sqrt{D_{ii}D_{jj}}} Z_{j}^{(k)}
\]

\[
= \sum_{j \in N(i)} M_{ij} (Z_{i}^{(k)} - Z_{j}^{(k)}) \frac{\|Z_{j}^{(k)} - Z_{i}^{(k)}\|_2}{\sqrt{D_{ii}D_{jj}}}.
\]

Therefore, we have the message-passing scheme of GSN:

\[
Z_{i}^{(k+1)} = \alpha Z_{i}^{(0)} + (1 - \alpha) \sum_{j \in N(i)} M_{ij} (Z_{i}^{(k)} - Z_{j}^{(k)}) \frac{\|Z_{j}^{(k)} - Z_{i}^{(k)}\|_2}{\sqrt{D_{ii}D_{jj}}} + 2\beta \sum_{j \in N(i)} \frac{M_{ij}(Z_{i}^{(k)} - Z_{j}^{(k)})}{\sqrt{D_{ii}D_{jj}}} \frac{\|Z_{i}^{(k)} - Z_{j}^{(k)}\|_2}{\sqrt{D_{ii}D_{jj}}}.
\]

### C Derivation of Message-Passing Scheme of GSN

Given the matrix-form update rule of a GSN layer (ignore the linear transformation and non-linear activation function first),

\[
Z^{(k+1)} = (1 - \alpha) \hat{P}_{sym} Z^{(k)} + 2\beta L_{1}^{sym} Z^{(k)} + \alpha Z^{(0)},
\]

we are interested in the message-passing scheme of GSN. For an arbitrary node \(i\), we have

\[
Z_{i}^{(k+1)} = \alpha Z_{i}^{(0)} + (1 - \alpha) \sum_{j \in N(i)} \frac{Z_{j}^{(k)} - Z_{i}^{(k)}}{\sqrt{D_{ii}D_{jj}}} + 2\beta \sum_{j = 1}^{V} (L_{H}^{sym})_{ij} Z_{j}^{(k)}.
\]

Note that \(L_{H}^{sym} = D^{-\frac{1}{2}} L_{H} D^{-\frac{1}{2}}, \ L_{H} = \text{diag}(H) - H\), and \(H = M \circ \mathcal{O}(D^{-\frac{1}{2}} Z)\), thus

\[
(L_{H}^{sym})_{ij} = \begin{cases} \frac{M_{ik}}{\sqrt{D_{ii}D_{kk}}} & \text{if } i = j \\ \frac{M_{ij}}{\sqrt{D_{ii}D_{jj}}} & \text{if } i \neq j, (i, j) \in \mathcal{E} \\ 0, & \text{otherwise.} \end{cases}
\]

Figure 5: The relations of some concepts.

isometry that is not \((V^2, \ell_2)\)-isometry? The answer is yes, and Figure 4 is an example. Therefore, there are gaps between these concepts.

From the definition of globally rigid, we can conclude that the \((E, \ell_2)\)-isometric transformations are equivalent to \((V^2, \ell_2)\)-isometric transformations if and only if the graph is globally rigid. However, Saxes [30] has shown that determining whether a \(d\)-dimensional embedding is globally rigid is NP-hard, which makes it almost impossible to check whether the \((E, \ell_2)\)-isometric transformations are equivalent to \((V^2, \ell_2)\)-isometric transformations for a graph.
D Analysis of Possible Defect of Existing Models

Some previous works, such as [21] and [26], try to explain why GCN-like models have over-smoothing problems from the perspective of Laplacian smoothing or dynamic systems. Here we give another possible explanation from a different perspective.

In Section “Related Work and Universality of GSN”, we have concluded that many GNN models can be regarded as minimizing the potential energy of a spring network with respect to a given metric $M$. For GCN and SGC, their metric matrices are simply $O$. Thus, if we stack too many layers, the distance between node $i$ and $j$ is close in the final embedding, making the nodes hard to appropriately classified.

PPNP and APPNP are equivalent to adding a trade-off regularization term $\|Z - Z^{(0)}\|_F^2$ into energy function, which leads to the initial residue $Z^{(0)}$ in its propagation rule $Z^{(k+1)} = (1 - \alpha)\tilde{P}Z^{(k)} + \alpha Z^{(0)}$. This regularization term may alleviate the node embeddings to be too close, but it never changes the metric $M = O$ between nodes.

Compared with APPNP, GCNII introduces identity mapping into its framework, which alleviates the over-smoothing problem in graph learning. However, it does not change the metric $M$ to a non-zero matrix, which may be why GCNII does not perform well on heterophilous graphs.

ElasticGNN introduces $\ell_{21}$ regularization term in its optimization objection. However, the authors do not realize the physical meaning of $\ell_{21}$ regularization term, it is, in fact, equivalent to a special case (fix $M = 11^T$) of our GSN. This shortage may be the reason why ElasticGNN ($\ell_{21}$) does not perform well on heterophilous graphs.

Inspired by spring networks and metric learning, our GSN framework first change the target metric between nodes to a non-zero matrix, which may benefit the performance of GNNs on heterophilous datasets. Meanwhile, our GSN framework also keeps the initial residue module, which alleviates the over-smoothing problem.

Table 6 lists the components of some GNN models.

| Components | GCN | SGC | (A)PPNP | GCNII | ElasticGNN ($\ell_{21}$) | GSN |
|------------|-----|-----|---------|-------|------------------------|-----|
| Metric     | $O$ | $O$ | $O$     | $O$   | $A \odot 11^T$          | $\mathbb{R}^{V \times V}_{\geq 0}$ |
| Residue    | $O$ | $O$ | $\alpha Z^{(0)}$ | $\alpha Z^{(0)}$ | $\alpha Z^{(0)}$ (as the $\ell_{21}$ reg. term) | $\mathbb{R}^{V \times V}_{\geq 0}$ |
| Transformation | $W^{(k)}$ | $I$ | $I$ | $W^{(k)}$ | $W^{(k)}$ | $W^{(k)}$ |
| Activation | ReLU | id  | id      | ReLU  | ReLU                  | ReLU |

E Hyper-parameter Settings

In Table 7, we list the best hyper-parameters for the GSN model in our node classification experiments.

|         | Cora  | CiteSeer | PubMed | CoraFull | Cornell | Texas   | Wisconsin | Chameleon | Squirrel | Actor |
|---------|-------|----------|--------|----------|---------|---------|-----------|-----------|----------|-------|
| lr      | 5e-4  | 1e-3     | 5e-2   | 1e-3     | 1e-3    | 1e-2    | 5e-3      | 1e-3      | 1e-2     | 5e-3  |
| weight_decay | 1e-4  | 1e-6     | 1e-3   | 1e-6     | 1e-6    | 1e-2    | 1e-5      | 1e-5      | 1e-5     | 1e-5  |
| dropout | 0.35  | 0.35     | 0.35   | 0.25     | 0.25    | 0.35    | 0.25      | 0.25      | 0.25     | 0.55  |
| hidden_dim | 64    | 64       | 64     | 64       | 64      | 64      | 64        | 64        | 64       | 64    |
| alpha   | 0.2   | 0.3      | 0.3    | 0.3      | 0.3     | 0.3     | 0.4       | 0.2       | 0.1      | 0.1   |
| beta    | 0.1   | 0.2      | 0.3    | 0.1      | 0.3     | 0.4     | 0.2       | 0.1       | 0.2      | 0.7   |
| attention | 'bilinear' | 'bilinear' | 'bilinear' | 'concat' | 'concat' | 'concat' | 'concat' | 'concat' | 'concat' | 'concat' |
| initial (f³(3)) | 'XW' | 'XW' | 'XW' | 'XW' | 'XW' | 'XW' | 'XW' | 'AXW' | 'AXW' | 'XW' |