Graph Unitary Message Passing

Anonymous authors
Paper under double-blind review

Abstract

Unitarity has emerged as a fundamental principle for efficient learning of deep neural networks, from parameter initialization to advanced optimizers, proven effective in various fields, including RNN, CNN, Transformer and Muon optimizer. However, imposing unitarity to the parameters is not enough to improve learning efficiency of graph neural networks (GNNs) due to the instability arising from the graph structure through the message passing mechanism. This data-dependent inefficiency, also known as oversquashing or oversmoothing problems, causes information from distant nodes to decay or node representation to become indistinguishable as the number of layers increases. Motivated by the success of unitarity in stabilizing neural network training, we propose a new graph-learning paradigm called Graph Unitary Message Passing (GUMP) to improve graph learning efficiency by applying unitary adjacency matrices for message passing. GUMP introduces a graph transformation algorithm that equips general graphs with unitary adjacency matrices while preserving original connectivity, and implements Newton-Schulz iteration for efficient unitary projection. Extensive experiments demonstrate that GUMP achieves significant performance improvements over vanilla message passing methods across various graph learning tasks.

1 Introduction

Unitarity has become a principle in modern deep learning for achieving training stability and preventing gradient-related pathologies. This principle manifests consistently across multiple domains through parameter-based unitary constraints: like orthogonal initialization (Saxe et al., 2013; Arjovsky et al., 2016; Orvieto et al., 2023; De et al., 2024) that maintain signal propagation through orthogonal/unitary weight matrices, architectural innovations such as unitary RNNs (Arjovsky et al., 2016; Jing et al., 2017) that impose unitarity on recurrent weight parameters to capture long-range dependencies without vanishing gradients, and cutting-edge optimizers like Muon (Jordan et al., 2024; Liu et al., 2025) that leverage unitary parameter updates for improved convergence. These successes underscore a fundamental principle: preserving the norm and geometric properties of representations through unitary operations on model parameters is essential for preventing training instability and enabling effective learning of complex dependencies.

However, imposing unitarity on parameters alone is insufficient to address the learning inefficiency of graph neural networks (GNNs), which arises from graph structures through message passing mechanisms. GNNs rely on message passing (Gilmer et al., 2017) where nodes aggregate features from neighbors to update representations. Unlike traditional neural networks with mainly parameter-determined gradient flow, GNNs exhibit data-dependent inefficiency because message passing is governed by the graph's adjacency structure. This manifests as oversquashing and oversmoothing problems (Alon & Yahav, 2020; Topping et al., 2022; Chen et al., 2020a), fundamentally limiting GNNs' ability to stack layers for capturing complex dependencies in graphs.

Here, we extend the principle of unitarity from model parameters to the message passing mechanism. We propose Graph Unitary Message Passing (GUMP), a novel approach that improves graph learning efficiency by applying unitary adjacency matrices for message passing, ensuring that information propagation maintains the stability properties (see Figure 1) that have proven crucial in deep learning (Yang et al., 2023).

To implement this vision, we propose a graph transformation algorithm that converts general graphs to have unitary adjacency matrices while preserving their original graph connectivity. This transformation is based on the theoretical result that line graphs of Eulerian graphs possess unitary adjacency matrices. Then, we apply the Newton-Schulz iteration (Kovarik, 1970; Björck & Bowie, 1971) to efficiently compute unitary adjacency matrices. Our experiments show that GUMP outperforms baselines across diverse graph learning tasks.

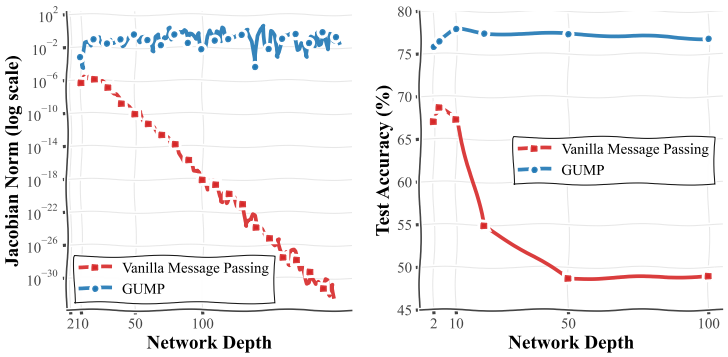


Figure 1: Compared to vanilla message passing (graph convolution), GUMP exhibits a stable Jacobian norm (left), which leads to more stable model performance as the number of layers increases (right).

Notations In this paper, we use bold uppercase letters $\mathbf{X}$ to denote matrices, bold lowercase letters $\mathbf{x}$ to denote vectors, and lowercase letters x to denote scalars. Given a matrix $\mathbf{X}$, the i-th row of matrix $\mathbf{X}$ is denoted as $\mathbf{x}_i$ and the entry of the i-th row and j-th column of matrix $\mathbf{X}$ is denoted as $\mathbf{X}_{ij}$. The transpose and conjugate transpose of matrix $\mathbf{X}$ is denoted as $\mathbf{X}^\top$ and $\mathbf{X}^\dagger$, respectively. A graph with n nodes and e edges is denoted as $G = (V, E, \mathbf{X})$ where $V = \{1, 2, \cdots, n\}$ is the node set, $E \subseteq V \times V$ is the edge set, and $\mathbf{X} \in \mathbb{R}^{n\times d}$ is the d-dimensional node feature matrix. For convenience, the operator $\mathsf{V}[G]$ and $\mathsf{E}[G]$ are used to denote the node set and edge set of graph G respectively, i.e., $\mathsf{V}[G] = V$ and $\mathsf{E}[G] = E$. The adjacency matrix of graph G is denoted as $\tilde{\mathbf{A}}[G] \in \{0, 1\}^{n\times n}$ where $\tilde{\mathbf{A}}_{ij} = 1$ if $(i, j) \in E$ and $\tilde{\mathbf{A}}_{ij} = 0$ otherwise. The normalized adjacency matrix of graph G is denoted as $\hat{\mathbf{A}}[G] \in \mathbb{R}^{n\times n}$ where $\hat{\mathbf{A}}[G] = \mathbf{D}^{-1/2}\tilde{\mathbf{A}}[G]\mathbf{D}^{-1/2}$ and $\mathbf{D}$ is the degree matrix of graph G. We also use matrix $\mathbf{A}[G] \in \mathbb{R}^{n\times n}$ to represent the general adjacency matrix in graph G, i.e., $\mathbf{A}_{ij} \neq 0$ if $(i, j) \in E$ and $\mathbf{A}_{ij} = 0$ otherwise. Therefore, without specifying the type of adjacency matrix, $\mathbf{A}[G]$ can also represent $\tilde{\mathbf{A}}[G]$ and $\hat{\mathbf{A}}[G]$. For convenience, the adjacency matrices above are denoted as $\tilde{\mathbf{A}}$, $\hat{\mathbf{A}}$, and $\mathbf{A}$ respectively. Some preliminaries used in this paper are provided in appendix. Finally, the GNN representation at layer k is denoted as $\mathbf{H}^{(k)} \in \mathbb{R}^{n\times d}$ with d being the dimension of node features, and the vector $\mathbf{h}_i^{(k)} \in \mathbb{R}^d$ denotes the GNN representation of node i at layer k.

2 Related work

2.1 GNN and its Training Instability

Graph neural network (GNN) (Kipf & Welling, 2016; Gilmer et al., 2017) with L layers is a type of neural network that uses graph G and initial node features $\mathbf{H}^{(0)} = \mathbf{X}$ to learn node representations $\mathbf{H}^{(L)}$. The k-th layer of GNN updates node representation via the message-passing formula

$$\mathbf{h}_i^{(k)} = \delta\left(\mathbf{h}_i^{(k-1)}, \phi\left(\left\{\{\psi(\mathbf{h}_j^{(k-1)}), j \in \mathcal{N}(i)\}\right\}\right)\right),$$

where δ, ϕ, and ψ are combination, aggregation, and message functions respectively, $\{\{\cdots\}\}$ is a multiset, and $\mathcal{N}(i) = \{j|(i, j) \in E\}$. Even though GNN achieves success in various graph learning tasks, it suffers from the inefficiency training, which is also known as the oversquashing or oversmoothing problems.

The oversquashing problem is first noted by Alon & Yahav (2020). Inspired by Xu et al. (2018), Topping et al. (2022) proposes to measure oversquashing with the Jacobian between node features at different levels of a GNN. Based on the measure, Topping et al. (2022) propose a rewiring method to increase the curvature of the edges in a graph. Many works combat oversquashing by improving the spectral gap of a graph. Banerjee et al. (2022) measure oversquashing via the spectral gap of a graph, employing a rewiring algorithm based on expander graph construction and effective resistance for edge sampling. Karhadkar et al. (2023) introduce FoSR, a rewiring method that maximizes the first-order change in the spectral gap of the graph. Arnaiz-Rodríguez et al. (2022) introduced a GNN comprising a parameter-free layer for learning commute time and

a rewiring layer to optimize spectral gap according to network characteristics and task requirements. Except from improving spectral gap, Black et al. (2023) focus on minimizing total resistance between node pairs and introduce the Greedy Total Resistance (GTR) rewiring method for oversquashing. Di Giovanni et al. (2023) analyze oversquashing with commute time. Barbero et al. (2023) propose a rewiring method to sequentially increase the number of walks between two nodes and preserve the locality in the original graph. Except for the one-hop message passing neural networks above, Gutteridge et al. (2023) propose a multi-hop message passing neural network (Drew) to alleviate oversquashing with layer-dependent graph rewiring and a delay mechanism for skip connections based on layer and node distance.

Oversmoothing refers to the phenomenon where stacking many GNN layers causes node features to become indistinguishable across the graph (Li et al., 2018; Oono & Suzuki, 2019). In spectral terms, repeated graph convolution (e.g., normalized Laplacian smoothing) drives embeddings to converge to low-frequency eigenvectors. Empirically, Li et al. (2018) showed that the distance between node representations approaches zero as layers increase. Theoretically, Oono & Suzuki (2019) demonstrated that under mild assumptions, a deep GCN outputs only trivial signals—namely, features dependent only on connected component membership and node degrees. Earlier works observed this effect in practice (Kipf & Welling, 2016; Wu et al., 2020), but the rigorous analysis by Oono and Suzuki formalized why deeper graph convolutions fail to capture finer structure. To alleviate oversmoothing, researchers have proposed several strategies. One class of methods modifies the graph structure to limit excessive neighborhood mixing. For example, DropEdge (Rong et al., 2019) randomly removes edges during training to sparsify the graph and reduce neighborhood oversampling. More generally, graph rewiring or sparsification schemes (e.g., GDC (Gasteiger et al., 2019), DropGNN variants) interrupt the smoothing process.

Recently, some works have considered oversmoothing and oversquashing jointly, as both relate to the degradation of node representations with depth. For instance, (Jamadandi et al., 2024) propose a spectral graph pruning framework inspired by the Braess paradox that simultaneously mitigates over-squashing and over-smoothing by removing edges to maximize the spectral gap. (Arroyo et al., 2025) bridge the gap between recurrent and graph learning by identifying “extreme gradient vanishing" as the common underlying cause of both over-smoothing and over-squashing, proposing a state-space formulation (GNN-SSM) that mitigates these issues through direct control of the layer-wise Jacobian spectrum.

Another approach involves normalization layers to maintain embedding diversity. PairNorm (Zhao & Akoglu, 2019) and related techniques rescale node embeddings after each layer to prevent convergence, directly countering oversmoothing. Architectural modifications have also proven effective. Residual or dense skip-connections (Li et al., 2019a; Chen et al., 2020b; Xu et al., 2018) allow nodes to retain features from earlier layers, acting as information highways. Jumping Knowledge networks (Xu et al., 2018) aggregate multi-scale representations from different layers, allowing nodes to preserve local information even in deep architectures. Pooling/unpooling operations (Gao & Ji, 2019) introduce coarsening to maintain feature resolution across scales.

In practice, combinations of these ideas are commonly used. For example, DropEdge with skip connections, or normalization with rewiring, often helps train deeper GNNs. Gasteiger et al. (2018) proposed personalized PageRank to reduce smoothing iterations, while Li et al. (2019a) explored dilated graph convolutions to expand receptive fields. These techniques collectively enable GNNs to scale in depth by mitigating rapid feature homogenization.

2.2 Unitarity in Deep Learning

Unitary matrices $\mathbf{U}$ are square matrices satisfying $\mathbf{U}^\top\mathbf{U} = \mathbf{I}$. Unitary matrices contribute to the stability of neural network training by preserving vector norms and preventing issues like exploding or vanishing gradients. Imposing unitarity on neural networks enhances their ability to seamlessly capture and propagate information across layers. This technique has proven successful in various fields, such as RNNs, CNNs, Transformers and LLM pre-training.

Unitary neural networks (UNNs) are first developed to tackle the problem of vanishing and exploding gradients in deep learning, enabling more efficient learning of information. We take RNN as an example to show how unitary matrices can alleviate the vanishing gradient problem, we denote RNN as $\mathbf{h}_k = \sigma(\mathbf{W}\mathbf{h}_{k-1} + \mathbf{u}_k)$

with $\mathbf{h}_k$ as the hidden state at the k-th layer of RNN, $\mathbf{W}$ as the weight matrix, $\mathbf{u}_k$ as the k-th input of RNN, and σ as the activation function. As shown in Figure 2(a), the long-range dependency can be measured by $\partial \mathbf{h}_r / \partial \mathbf{u}_1$. To simplify the analysis, we assume the activation function σ is identity. The Jacobian of the long-range dependency becomes $\partial \mathbf{h}_r / \partial \mathbf{u}_1 = \mathbf{W}^{r-1}$. When $\mathbf{W}$ is unitary, information in $\partial \mathbf{h}_r / \partial \mathbf{u}_1$ will not vanish or explode, which is the key to alleviate the vanishing/exploding gradient problem in learning long sequences.

Early algorithms on unitary RNN construct a series of parameterized unitary transformations to impose unitarity. EUNN (Jing et al., 2017) achieves this by composing layers of rotations, Fourier transforms, and other unitary transformations to parametrize unitary matrices. uRNN (Arjovsky et al., 2016) and scoRNN (Helfrich et al., 2018), on the other hand, maintain unitarity by performing a Cayley transformation to parametrize the full unitary space. expRNN (Lezcano-Casado & Martınez-Rubio, 2019), in contrast, parametrizes unitary matrices within the Lie algebra of the orthogonal/unitary group. projUNN (Kiani et al., 2022) first optimizes its parameters using gradient-based optimization and then maps the updated parameters to a unitary space. Most recently, linear recurrent unit (LRU) (Orvieto et al., 2023) significantly enhances the long-range learning capability of RNNs by linear RNN and initializing their weight matrix to be nearly unitary. Other works (Peng et al., 2023; Gu & Dao, 2023) on recurrent models also share the similar idea of LRU to effectively capture long-range dependencies in sequences, yielding impressive outcomes in language modeling and other tasks.

Despite RNN, unitarity has also been applied to CNNs and GNNs. Unitary CNNs (Sedghi et al., 2018; Li et al., 2019b; Singla & Feizi, 2021; Trockman & Kolter, 2021) introduce various methods to restrict the convolutional filters to be unitary, e.g., via the Lie algebra of the orthogonal group (Singla & Feizi, 2021) and the Cayley transform (Trockman & Kolter, 2021). Ortho-GConv (Guo et al., 2022) imposes unitarity on the feature transformation matrix in GNNs.

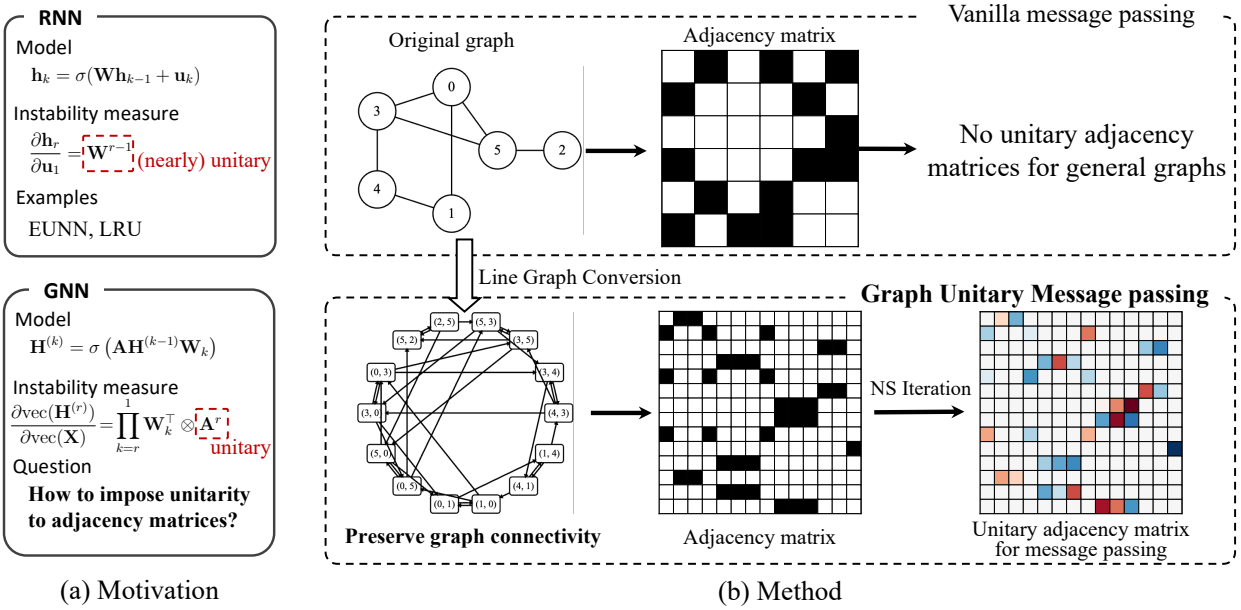
Except the constraints of the network structures to impose unitarity, there are works that initializes the parameters of neural networks to be unitary (Saxe et al., 2013; Balduzzi et al., 2017) and optimization algorithms that impose unitarity during training Jordan et al. (2024).

3 Preliminaries on Unitarity and Training Instability

Unitarity in Deep Learning A matrix $\mathbf{U} \in \mathbb{C}^{n \times n}$ is called *unitary* if $\mathbf{U}^\dagger \mathbf{U} = \mathbf{U}\mathbf{U}^\dagger = \mathbf{I}$. For real matrices, this reduces to the orthogonal condition $\mathbf{U}^\top \mathbf{U} = \mathbf{U}\mathbf{U}^\top = \mathbf{I}$. A fundamental property of unitary matrices is that they preserve vector norms: $\|\mathbf{U}\mathbf{x}\|_2 = \|\mathbf{x}\|_2$ for any vector $\mathbf{x}$. Unitarity has emerged as an important principle in deep learning due to its ability to stabilize training and prevent gradient pathologies. Unitarity manifests across multiple domains:

- **Parameter Initialization:** Orthogonal and unitary initialization schemes (Saxe et al., 2013; Orvieto et al., 2023) have been shown to maintain signal propagation through deep networks by preserving the norms of activations. This prevents the vanishing and exploding gradient problems that plague randomly initialized networks, enabling effective training of deeper models.
- **Unitary Constraints:** The breakthrough work of Arjovsky et al. (2016) demonstrated that constraining the recurrent weight matrix to be unitary allows RNNs to capture long-range dependencies without suffering from vanishing gradients. Subsequent work (Jing et al., 2017; Orvieto et al., 2023; De et al., 2024) has built upon this foundation, showing that unitary RNNs can effectively model sequences with temporal dependencies spanning hundreds or thousands of time steps.
- **Optimizers:** Recent advances in optimization have leveraged unitarity for improved convergence. The Muon optimizer (Jordan et al., 2024; Liu et al., 2025) applies unitary transformations to parameter updates, achieving superior performance in training large language models and other deep architectures.

Training Instability in GNNs To understand the training instability in GNNs, we first introduce the instability in RNNs from Figure 2(a). We simply formulate RNN as $\mathbf{h}_k = \sigma(\mathbf{W}\mathbf{h}_{k-1} + \mathbf{u}_k)$ with $\mathbf{h}_k$ being the hidden state at layer k, $\mathbf{W}$ being the transformation matrix, $\mathbf{u}_k$ being the input at time step k, and σ being the activation function. The Jacobian of the hidden state with respect to the input is used to measure



(a) Motivation (b) Method

Figure 2: Overview of GUMP. (a) RNN versus GNN. The measures of the stability are derived with the identity activation function and $\otimes$ denotes the Kronecker product. (b) GUMP aims to impose unitarity to $\mathbf{A}$ in message passing. GUMP achieves this goal with graph transformation (Section 4.2) and unitary adjacency matrix calculation (Section 4.3).

the training instability of RNN (Arjovsky et al., 2016). If the Jacobian norm $\|\partial \mathbf{h}_k / \partial \mathbf{u}_0\|_2$ grows exponentially with k, the gradient will explode, and if it decays exponentially, the gradient will vanish. When imposing unitarity to $\mathbf{W}$, the Jacobian norm is bounded, thus improves the training stability of RNNs.

For graphs, the training instability is mainly caused by the message passing mechanism. In Figure 2(a), a one-hop MPNN layer for GNN is formulated as $\mathbf{H}^{(k)} = \sigma(\mathbf{A}\mathbf{H}^{(k-1)}\mathbf{W}_k)$ for simplicity. Similar to RNN (Pascanu et al., 2013), the Jacobian $\partial \mathbf{H}^{(r)} / \partial \mathbf{X}$ (Topping et al., 2022) can measure the training instability of GNN. When the Jacobian is large, the gradient will explode, and when it is small, the gradient will vanish. Therefore, we are motivated to impose unitarity to $\mathbf{A}$ in GNN to improve learning efficiency.

4 Graph Unitary Message Passing

4.1 Challenges of Imposing Unitarity

Imposing unitarity to an adjacency matrix is not as straightforward as that in RNNs. The challenge comes from the sparsity of the adjacency matrix, as almost all unitary matrices are dense (Figure 2(b)). Therefore, the amount of graphs with unitary adjacency matrices is limited. Also, the unitary adjacency matrix should be input-dependent and permutation-equivariant because the unitary adjacency matrix depends on input graph, and the order of nodes in a graph should not impact GNN representations.

In this section, the challenge of existence of unitary adjacency matrix is addressed by transforming the original graph to a special graph (Algorithm 1) which is guaranteed to have unitary adjacency matrices and preserves the original graph connectivity at the same time. The challenge of preserving adjacency matrix property is addressed by calculating the unitary adjacency matrix with a unitary projection algorithm (Algorithm 2), which is implemented by utilizing the Newton-Schulz iteration and allows GUMP to be permutation-equivariant (Proposition 2). As a general one-hop message-passing mechanism, any convolution operation can be combined with GUMP by setting the edge weights to be the entries of the unitary adjacency matrix for message passing. The overview of GUMP is shown in Figure 2(b).

Algorithm 1 Graph transformation

Require: A undirected graph $G = (V, E)$;
Initialize a new digraph $G' = (V, E')$;
for $(i, j) \in E$ **do**
Add (i, j) and (j, i) to E';
end for
Convert G' to its line graph $\mathsf{L}(G')$;
Return: A digraph $\mathsf{L}(G')$.

4.2 Graph Transformation: Convert Graph to Have Unitary Adjacency Matrix

Since unitary matrices are generally non-symmetric, the graph transformation algorithm should convert the original graph to a directed graph. We first formally define the unitary adjacency matrix as in Severini (2003). Given an adjacency matrix $\mathbf{A}$, its support matrix $\mathsf{S}[\mathbf{A}] \in \mathbb{R}^{n\times n}$ is a binary matrix with entries equal to one if the corresponding entry of $\mathbf{A}$ is non-zero and equal to zero otherwise, i.e., $\mathsf{S}[\mathbf{A}]_{ij} = 1$, if $\mathbf{A}_{ij} \neq 0$ and $\mathsf{S}[\mathbf{A}]_{ij} = 0$, if $\mathbf{A}_{ij} = 0$. Then, the unitary adjacency matrix $\mathbf{U}_G$ of graph G is a unitary matrix whose support is equal to the support of its adjacency matrix $\mathbf{A}$, i.e., $\mathsf{S}[\mathbf{U}_G] = \mathsf{S}[\mathbf{A}]$.

We propose the transformation in Algorithm 1 for undirected graph G to make it have unitary adjacency matrices and preserve its original graph connectivity. Algorithm 1 first transforms the undirected graph to be Eulerian graph G' (see definition in appendix) by splitting each undirected edge into two directed edges. Then, it converts the Eulerian graph to its line graph $\mathsf{L}(G')$ (see definition in appendix). Finally, $\mathsf{L}(G')$ has unitary adjacency matrices because of its specularity and strong quadrangularity properties (Severini, 2003), which is proved in the following proposition.

Proposition 1. *The line graph $\mathsf{L}(G')$ returned by Algorithm 1 has unitary adjacency matrices.*

Proposition 1 is proved in the appendix. In Algorithm 1, the original graph connectivity is preserved because the splitting of an undirected edge and the conversion to the line graph do not introduce new connectivity between nodes that is absent in the original graph. Finally, Algorithm 1 takes as input graph G with n nodes and e edges, and outputs a line graph $\mathsf{L}(G')$ with $2e$ nodes.

4.3 Unitary Adjacency Matrix Calculation: Compute the Edge Weights for Message Passing

According to Proposition 1, the line graph $\mathsf{L}(G')$ has unitary adjacency matrices. In this section, we propose an algorithm to calculate a unitary adjacency matrix for GUMP, because the unitary adjacency matrix depends on the input graph and should be calculated for each graph.

4.3.1 Permutation-equivariant Projection

Permutation equivariance of message passing is a key property for GNN to apply to graphs with varying node orders. To achieve this, the calculation of unitary adjacency matrix has to be permutation equivariance. GUMP consists of two steps: 1) calculate edge weights to form weighted adjacency matrix; 2) impose unitarity to weighted adjacency matrix.

Firstly, edge weight for $(i, j) \in \mathsf{E}[\mathsf{L}(G')]$ is calculated by

$$\alpha_{ij} = \mathsf{Tanh}\left(\mathbf{w}^\top \cdot \mathsf{LeakyReLU}(\mathbf{W}_s\mathbf{h}_i + \mathbf{W}_t\mathbf{h}_j)\right), \tag{1}$$

where $\mathbf{h}_i$ ($\mathbf{h}_j$) is the representations for node i (j) in $\mathsf{L}(G')$, $\mathbf{W}_s, \mathbf{W}_t \in \mathbb{R}^{d'\times d}$ are transformation matrices for source and target nodes of an edge respectively, and $\mathbf{w} \in \mathbb{R}^{d'}$ is a learnable parameter. Then, the weighted adjacency matrix of $\mathsf{L}(G')$, denoted as $\mathbf{A}_w \in \mathbb{R}^{2e\times 2e}$, is formed from edge weights, i.e., $(\mathbf{A}_w)_{ij} = \alpha_{ij}$.

After calculating $\mathbf{A}_w$, we impose unitarity to $\mathbf{A}_w$ by projection. We use the projection algorithm in Keller (1975), which takes advantage of the fact that the polar transformation yields the closest unitary matrix to a given matrix in terms of the Frobenius norm. The following lemma describes the unitary projection in GUMP:

Lemma 1. *Given a weighted adjacency matrix* $\mathbf{A}_w \in \mathbb{R}^{2e\times 2e}$ *of* $\mathsf{L}(G')$*, unitary projection of* $\mathbf{A}_w$ *is given by* $\mathsf{U}[\mathbf{A}_w]$:=$\arg\min_{\mathbf{U}\ \textit{is unitary}} \|\mathbf{A}_w - \mathbf{U}\|_F^2 = \mathbf{A}_w \left(\mathbf{A}_w^\dagger \mathbf{A}_w\right)^{-\frac{1}{2}}$.

Lemma 1 (proved in the appendix) indicates $\mathsf{U}[\mathbf{A}_w] = \mathbf{A}_w \left(\mathbf{A}_w^\dagger \mathbf{A}_w\right)^{-\frac{1}{2}}$ is the unitary adjacency matrix for GUMP. Also, the unitary projection $\mathsf{U}[\mathbf{A}_w]$ is guaranteed to be permutation-equivariant when $\mathbf{A}_w$ is a full-rank matrix with the following proposition.

Proposition 2 (Strong permutation equivariance)**.** *Given two permutation matrices* $\mathbf{P}_1$ *and* $\mathbf{P}_2$*, if* $\mathbf{A}_w$ *is a full-rank matrix, the unitary projection* $\mathsf{U}[\mathbf{A}_w]$ *is equivariant to both row and column permutations of* $\mathbf{A}_w$*, i.e.,* $\mathbf{P}_1\mathsf{U}[\mathbf{A}_w]\mathbf{P}_2^\top = \mathsf{U}[\mathbf{P}_1\mathbf{A}_w\mathbf{P}_2^\top]$.

By Proposition 2 (proved in appendix), the weighted adjacency matrix $\mathbf{A}_w$ should be full-rank to guarantee permutation equivariance of GUMP. Empirically, inspired by GATv2 (Brody et al., 2021), $\mathbf{A}_w$ induced by equation 1 is full-rank in experiments and thus can guarantee the permutation equivariance of GUMP. However, the unitary projection (Lemma 1) is computationally expensive due to the inverse square root of $\mathbf{A}_w^\dagger \mathbf{A}_w \in \mathbb{R}^{2e\times 2e}$.

4.3.2 Efficient Implementation of Unitarity Projection

Calculating the unitary projection $\mathsf{U}[\mathbf{A}_w] = \mathbf{A}_w(\mathbf{A}_w^\dagger \mathbf{A}_w)^{-\frac{1}{2}}$ directly involves computing the inverse square root of a matrix, which is computationally expensive and numerically unstable for large matrices, typically requiring singular value decomposition (SVD) with cubic complexity $\mathcal{O}((2e)^3)$. To address this, we employ Newton-Schulz iteration (Kovarik, 1970; Björck & Bowie, 1971; Jordan et al., 2024), an iterative method that converges quadratically to the unitary polar factor without expensive decompositions.

The Newton-Schulz iteration (Jordan et al., 2024) for computing the unitary factor $\mathbf{U}$ of a matrix $\mathbf{A}_w$ is given by the recurrence:

$$\mathbf{X}_{t+1} = \frac{15}{8}\mathbf{X}_t - \frac{5}{4}\mathbf{X}_t\left(\mathbf{X}_t^\top \mathbf{X}_t\right) + \frac{3}{8}\mathbf{X}_t\left(\mathbf{X}_t^\top \mathbf{X}_t\right)^2, \quad \mathbf{X}_0 = \frac{\mathbf{A}_w}{\|\mathbf{A}_w\|_F}. \tag{2}$$

Here, $\mathbf{X}_t$ converges to $\mathsf{U}[\mathbf{A}_w]$ as $t \to \infty$. The normalization by the spectral norm $\|\mathbf{A}_w\|_F$ ensures spectral norm is at most 1 for better convergence. In practice, a small number of iterations (e.g., $K = 5$ to 10) is sufficient to obtain a high-quality approximation of the unitary matrix. This iteration involves only matrix multiplications, which are highly parallelizable on GPUs, making it significantly more efficient than SVD-based approaches for the sparse but large adjacency matrices encountered in GNNs.

Algorithm 2 Calculation of unitary adjacency matrix via Newton-Schulz iteration

Require: $\mathsf{L}(G')$ outputted by Algorithm 1, iterations K;
1: Calculate $\mathbf{A}_w$ of $\mathsf{L}(G')$ with equation 1;
2: Normalize $\mathbf{X}_0 = \mathbf{A}_w/\|\mathbf{A}_w\|_F$ (approximation for spectral norm);
3: **for** $t = 0$ to $K-1$ **do**
4: $\quad \mathbf{X}_{t+1} = \frac{15}{8}\mathbf{X}_t - \frac{5}{4}\mathbf{X}_t\left(\mathbf{X}_t^\top \mathbf{X}_t\right) + \frac{3}{8}\mathbf{X}_t\left(\mathbf{X}_t^\top \mathbf{X}_t\right)^2$;
5: **end for**
6: $\mathsf{U}[\mathbf{A}_w] = \mathbf{X}_K$;
7: **Return:** Unitary adjacency matrix $\mathsf{U}[\mathbf{A}_w]$.

4.4 Apply GUMP to GNN

We can apply GUMP to different GNN architectures for graph learning tasks in Algorithm 3. Given a graph G, a base GNN first computes the initial node representations of G, i.e., $\mathbf{X}^{(0)} = \mathsf{GNN}(\mathbf{X}, G)$. Then, Algorithm 1 transforms G to $\mathsf{L}(G')$. The initial node representations $\mathbf{H}^{(0)} \in \mathbb{R}^{2e\times 2d}$ of $\mathsf{L}(G')$ are generated with $\mathbf{h}_{(i,j)} = [\mathbf{x}_i^{(0)}; \mathbf{x}_j^{(0)}]$, $\forall (i,j) \in \mathsf{V}[\mathsf{L}(G')]$ $(i,j \in \mathsf{V}[G])$. Next, the unitary adjacency matrix $\mathsf{U}[\mathbf{A}_w]$ of $\mathsf{L}(G')$ is calculated from Algorithm 2 and applied to propagate messages in graph with

Algorithm 3 GNN with the graph unitary message passing mechanism (GNN-GUMP)

Require: A graph $G = (V, E, \mathbf{X})$;
1: $\mathbf{X}^{(0)} = \mathsf{GNN}(\mathbf{X}, G)$;
2: Transform G to $\mathsf{L}(G')$ with Algorithm 1;
3: Generate initial representation $\mathbf{H}^{(0)}$ for $\mathsf{L}(G')$ with $\mathbf{h}_{(i,j)} = [\mathbf{x}_i^{(0)}; \mathbf{x}_j^{(0)}], \forall (i,j) \in \mathsf{V}[\mathsf{L}(G')]$;
4: Calculate $\mathsf{U}[\mathbf{A}_w]$ with Algorithm 2;
5: **for** $k = 1 \cdots L$ **do**
6: $\quad \mathbf{h}_v^{(k)} = \gamma(\mathbf{h}_v^{(k-1)}, \sum_{u \in N_v} \mathsf{U}[\mathbf{A}_w]_{vu} \phi^{(k)}(\mathbf{h}_v^{(k-1)}, \mathbf{h}_u^{(k-1)}))$, $v \in V$
7: **end for**
8: Scatter $\mathbf{H}^{(L)}$ to nodes of G with $\mathbf{H}_s^{(L)} = \mathsf{Scatter}(\mathbf{H}^{(L)}, G)$;
9: Generate node representations of G with $\mathbf{X}^{(L)} = [\mathbf{X}^{(0)}; \mathbf{H}_s^{(L)}]$.
10: **Return:** Node representations $\mathbf{X}^{(L)}$ of G.

$\mathbf{h}_v^{(k)} = \gamma(\mathbf{h}_v^{(k-1)}, \sum_{u \in N_v} \mathsf{U}[\mathbf{A}_w]_{vu} \phi^{(k)}(\mathbf{h}_v^{(k-1)}, \mathbf{h}_u^{(k-1)}))$, $v \in V$ with any graph convolution operator. After L layers of unitary message passing, we obtain the node representations $\mathbf{H}^{(L)}$, which is later scattered to nodes of G with $\mathbf{H}_s^{(L)} = \mathsf{Scatter}(\mathbf{H}^{(L)}, G) \in \mathbb{R}^{n \times d'}$. Then, $\mathbf{H}_s^{(L)}$ are concatenated with $\mathbf{X}^{(0)}$ to obtain the final node representations $\mathbf{X}^{(L)} = [\mathbf{X}^{(0)}; \mathbf{H}_s^{(L)}] \in \mathbb{R}^{n \times (d+d')}$ of G. Finally, various graph learning tasks, e.g., graph and node classification, link prediction, and graph regression, are performed based on $\mathbf{X}^{(L)}$. In this paper, GUMP is a general one-hop message-passing mechanism for GNN. Therefore, depending on the convolution in line 6 of Algorithm 3, GNN with GUMP is named as [GNN type]-GUMP in Section 5, e.g., GCN-GUMP and GIN-GUMP have graph convolution and isomorphism operators in line 6 of Algorithm 3, respectively.

From the theoretical perspective, GUMP can be proved to improve GNN training stability in terms of the Jacobian stability. Given the ReLU activation function and convolution operation in Figure 2(a), we have the following theorem (proved in appendix) guaranteeing that the Jacobian of GUMP remains a bounded value with respect the number of layers of the model.

Theorem 1. *The expected Jacobian for GUMP, i.e.,* $\mathbf{A}$ *is unitary in GNN, is approximately in the order of* $\mathbb{E}\big[\partial \mathbf{h}_i^{(L)} / \partial \mathbf{x}_s\big] = \mathcal{O}(\mathbf{1})$.

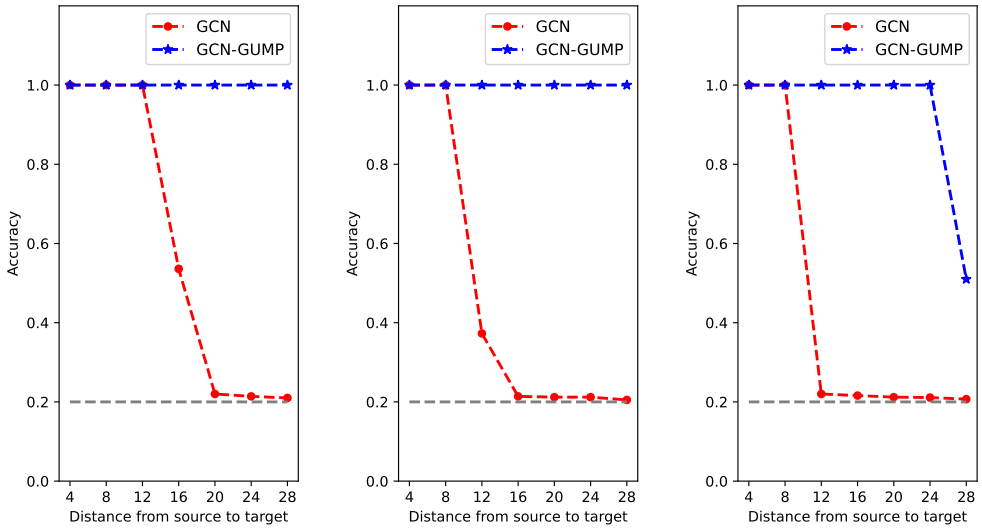
Theorem 1 does not indicate the Jacobian of GUMP is independent of L. In fact, the relation between the Jacobian and L is a trigonometric function in GUMP, which can be bounded by constants. The theorem shows that the Jacobian of GNN with unitary adjacency matrices will not change exponentially with the number of layers, thus improving the GNNs training stability.

5 Experiments

In this section, we perform experiments to evaluate GUMP on graph learning tasks. All experiments are implemented by PyTorch Geometric (Fey & Lenssen, 2019) and conducted on NVIDIA RTX 4090 GPUs and AMD EPYC 7763 CPUs.

5.1 Experiments on Synthetic Dataset

Setup In this section, we conduct experiments on synthetic datasets, i.e., CrossedRing, Ring, and CliquePath, in Di Giovanni et al. (2023) to test GUMP. The performance is evaluated on the distances from source to target in the range of 4 to 28. This experiment is designed to test the ability of GUMP to learn long-range interactions when increasing the model layers. In the experiments, we compare GCN-GUMP and GCN. The layer L of GCN-GUMP and GCN is appropriately set up according to the distance d between source and target in the synthetic datasets (i.e., $L = \lfloor d/2 \rfloor + 1$), such that the long-range interactions can be captured by GNN. We set the hidden dimension to be 32 for both GCN-GUMP and GCN. The hyperparameters of GCN-GUMP for synthetic datasets are in appendix.



(a) CrossedRing (b) Ring (c) CliquePath

Figure 3: The performance of GCN and GCN-GUMP on the CrossedRing, Ring, and CliquePath with different distances from source to target.

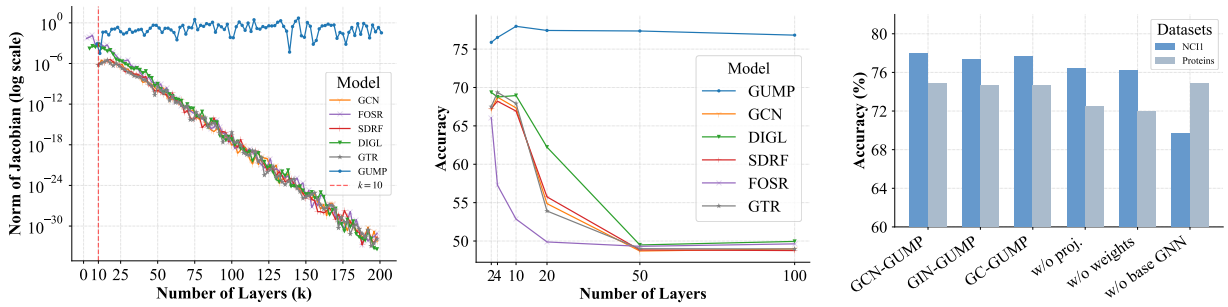
Table 1: Graph classification accuracy on the TUDataset. **First** and second best results are bold and underlined, respectively.

Classes	Methods	Mutag	Proteins	Enzymes	NCI1	NCI109
–	GCN	72.15±2.44	70.98±0.74	27.67±1.16	68.74±0.45	67.90±0.50
	GCN (+layer)	70.05±1.83	69.80±0.99	23.63±1.07	63.94±1.34	55.92±1.26
Rewiring (GCN)	DIGL	79.70±2.15	70.76±0.77	35.72±1.12	69.76±0.42	69.37±0.43
	SDRF	71.05±1.87	70.92±0.79	28.37±1.17	68.21±0.43	66.78±0.44
	FoSR	80.00±1.57	73.42±0.81	25.07±0.99	57.27±0.54	56.82±0.60
	GTR	79.10±1.86	72.59±2.48	27.52±0.99	69.37±0.38	67.97±0.47
	ELDANADD	82.16±0.03	70.53±0.86	26.36±0.01	–	–
Diffusion	ADGN	81.39±1.81	73.81±0.80	28.78±1.25	76.15±0.42	74.31±0.44
	GRAND	77.94±1.73	73.24±0.94	24.13±1.05	68.51±0.48	67.26±0.46
Transformer	GPS	70.12±1.68	69.32±0.86	30.43±1.76	61.32±0.51	61.01±0.22
Ortho-GNN	Ortho-GConv	71.78±2.52	63.80±0.98	18.30±1.13	69.92±0.60	68.91±0.50
K-hop MP	GRIT	80.76±2.18	73.71±0.89	35.22±1.17	72.21±0.46	71.68±0.44
Ours	GCN-GUMP	**84.89±1.63**	**74.88±0.87**	**36.02±1.43**	**77.97±0.42**	**75.85±0.44**

Results We plot the average results from three random seeds of GCN-GUMP and GCN experiments in Figure 3. For two easier datasets, i.e., CrossedRing and Ring, GUMP achieves 100% accuracy when the distance ranges from 4 to 28. For the challenging CliquePath dataset, GCN-GUMP's performance deteriorates to random guessing at a distance of 28. The results show that GUMP can help capture the long-range interactions in graph learning tasks. We compare with more baselines in appendix.

5.2 Experiments on the TUDataset

Datasets We select five datasets, i.e., Mutag, Proteins, Enzymes, NCI1, and NCI109 from the TUDataset (Morris et al., 2020). We chose these datasets because they consist of chemistry or biological graphs



(a) Jacobian norm versus layers on NCI1 (b) Accuracy versus layers on NCI1 (c) Ablation on different components

Figure 4: Model analysis. GCN, GIN, and GUMP in (c) represent the convolution of GUMP. The base GNN of GUMP is GCN. "w/o proj" removes unitary projection in GUMP. "w/o weights" removes weighted adjacency matrix and unitary projection in GUMP. "w/o base GNN" removes base GNN in GUMP.

with complex or long-range interactions, e.g., the atoms far apart may be closer in space. The statistics of these datasets are in appendix.

Baselines Baselines include various rewiring methods, i.e., DIGL (Gasteiger et al., 2019), SDRF (Topping et al., 2022), FoSR (Karhadkar et al., 2023), GTR (Black et al., 2023), and ELDANADD (Jamadandi et al., 2024), diffusion methods, i.e., ADGN (Tönshoff et al., 2023) and GRAND (Chamberlain et al., 2021), transformers method GPS (Rampášek et al., 2022), GNN with orthogonal parameters Ortho-GConv (Guo et al., 2022), and k-hop message passing GRIT (Di Giovanni et al., 2023). The baselines' settings follow Karhadkar et al. (2023).

Experimental Details To evaluate each method, we initially designate a test set comprising 10% of the graphs and a development set encompassing the remaining 90% of the graphs. The accuracies of each configuration are determined through 100 random train/validation splits of the development set, with 80% for training and 10% for validation. During the training phase, a stopping patience of 100 epochs is employed based on validation loss. Subsequently, for the test results, we report 95% confidence intervals for the best validation accuracy observed across the 100 runs. The number of layers for GUMP is manually tuned because the long-range interactions in a graph can only be captured by increasing its layers. The detailed hyperparameters of GUMP for different datasets are presented in appendix.

Results The results of GUMP on TUDataset are shown in Table 1. GUMP achieves superior performance across all five datasets, with GCN-GUMP obtaining the best results, obtaining average performance gains of 1.6% over the second-best methods, demonstrating that the unitary message passing mechanism effectively learns complex interactions in graphs. Moreover, since GUMP usually has more layers than baselines in these datasets, experiments show that GCN with more layers have degraded performance on all datasets, showing that improvement of GUMP does not come from increasing expressivity with more GNN layers.

5.3 Experiments on LRGB

Here, we conduct experiments on the Long Range Graph Benchmark (LRGB) (Dwivedi et al., 2022), which is a set of GNN benchmarks involving long-range interactions. Two datasets are selected from LRGB for comparison, i.e., Peptides-func and Peptides-struct. The statistics of these datasets are shown in appendix.

The experiments of LRGB follow Tönshoff et al. (2023). We set SDRF, FoSR, GTR, LASER (Barbero et al., 2023), GRAND, ADGN, GPS, Exphormer (Shirzad et al., 2023), GRIT, Graph ViT (He et al., 2023), G-MLPMixer (He et al., 2023), Drew-GCN (Gutteridge et al., 2023), ProxyAdd+GCN (Jamadandi et al., 2024), k-GCN-SSM (Arroyo et al., 2025) as baselines. The one-hop message passing of GUMP is set to be GCN. The hyperparameters of GUMP and more results for LRGB are presented in appendix. The results of LRGB are shown in Table 2. The results show that GUMP outperforms all baselines, showing that GUMP can effectively learn long-range interactions in graphs.

5.4 Model Analysis

In this section, we perform more experiments to analyze GUMP from three aspects, i.e., Jacobian, number of layers, and ablation studies.

Table 2: Results of Peptides-func and Peptides-struct. **Bold** are best results.

Methods	Peptides-func Test AP ↑	Peptides-struct Test MAE ↓
GCN	0.6860±0.0050	0.2460±0.0007
SDRF	0.6874±0.0032	0.2453±0.0018
FoSR	0.6878±0.0030	0.2461±0.0016
GTR	0.6740±0.0033	0.2509±0.0013
LASER	0.6440±0.0010	0.3043±0.0019
GRAND	0.5789±0.0062	0.3418±0.0015
ADGN	0.5975±0.0044	0.2874±0.0021
GPS	0.6534±0.0091	0.2509±0.0014
Exphormer	0.6527±0.0043	0.2481±0.0007
GRIT	0.6988±0.0082	0.2460±0.0012
Graph ViT	0.6942±0.0075	0.2449±0.0016
G-MLPMixer	0.6921±0.0054	0.2475±0.0015
Drew-GCN	0.6996±0.0076	0.2781±0.0028
ProxyAdd+GCN	0.6789±0.0002	0.2465±0.0004
k-GCN-SSM	0.6902±0.0138	0.2898±0.0324
GUMP	**0.7088±0.0026**	**0.2438±0.0014**

Jacobian We first visualize the Jacobian of GUMP and other baselines involving message passing based on graph structures (more visualization in appendix). We choose a pair of nodes with a distance of ten from NCI1 and calculate the spectral norm of Jacobian for GUMP and baselines with base GNN as GCN. The visualization is shown in Figure 4(a) with the Jacobian norm in log scale. Firstly, when increasing the number of layers of GNNs, the Jacobian of GUMP does not decay, while the Jacobian of other baselines decays exponentially. The results validate the theoretical analysis in Theorem 1, indicating GUMP is efficient in learning long-range interactions in a graph. Secondly, the norm of the Jacobian varies for different baselines. For example, DIGL has a large norm of Jacobian when the number of layers is smaller than 50. Therefore, DIGL performs better than other rewiring methods in Table 1.

Deep GNN The number of GNN layers indicates the ability of GNNs to capture complex and long-range interactions. We increase the number of layers of GUMP and baselines to see how their performances change. This experiment is conducted in NCI1 with base GNN as GCN and the number of layers in the range of 2 to 100. The results in Figure 4(b) demonstrate that the performance of GUMP increases from 75.88% to 77.97% when increasing the number of layers from 2 to 10. However, the performance of baselines decreases a lot when increasing the number of layers. For example, the performance of FoSR decreases from 66.06% to 52.86% when the number of layers increases from 2 to 10. The performance of baselines changes drastically as the layers increase, while the performance of GUMP is more stable. The results show GUMP can be deeper than previous methods, thus learning more complex interactions in graphs.

Ablation Studies Lastly, we conduct ablation studies to analyze GUMP in Figure 4(c). We first replace the convolution of GUMP with GIN and GraphConv (Morris et al., 2019), showing that the choice of convolution can impact the performance of GUMP. Then, we remove unitary projection (i.e., message passing with $\mathbf{A}_w$) and weighted adjacency matrix (i.e., message passing with $\tilde{\mathbf{A}}[\mathsf{L}(G')]$) in GUMP and the results show that their performances decrease, indicating the importance of GUMP. Finally, we remove the base GNN in GUMP and the results show that the performance of GUMP varies on different datasets, i.e., the performance on NCI1 decreases, while the performance on Proteins does not decrease. This phenomenon is expected because the quality of the unitary adjacency matrix depends on the representations of nodes in the line graph (see equation 1).

In model analysis, GUMP demonstrates the optimal Jacobian stability, and greater stability with increased layers compared to prior methods, highlighting critical role of its design in achieving superior performance.

6 Discussions and Limitations

In this paper, we propose Graph Unitary Message Passing (GUMP), which propagates messages using unitary adjacency matrices to improve graph learning efficiency, which shows superior performance across various graph learning tasks. We discuss below the limitations and future works of GUMP:

Information Loss Here we discuss the potential information loss in applying GUMP. For undirected, unweighted graphs, GUMP's transformation is lossless. For weighted graphs, edge weights are encoded as features following R-GNNs (Battaglia et al., 2018). For directed graphs, directionality is preserved by encoding it as edge features and filtering invalid edges in post-processing (Algorithm 3, step 8). These adaptations ensure GUMP maintains the essential properties of various graphs, enabling its broad application.

Implications for Large Graph Models GUMP represents a promising step toward applying unitary constraints to graph learning. Our work demonstrates that unitary message passing significantly improves training stability and model performance on moderate-scale graphs. The graph transformation we propose could serve as foundational components for scaling up graph models. Future large graph models could leverage our transformation to convert arbitrary graphs into forms amenable to unitary processing, while employing initialization schemes that enable approximate unitarity rather than exact projection. This would enable large graph models that maintain high learning efficiency.

References

Uri Alon and Eran Yahav. On the bottleneck of graph neural networks and its practical implications. *arXiv preprint arXiv:2006.05205*, 2020.

Martin Arjovsky, Amar Shah, and Yoshua Bengio. Unitary evolution recurrent neural networks. In *International conference on machine learning*, pp. 1120–1128. PMLR, 2016.

Adrián Arnaiz-Rodríguez, Ahmed Begga, Francisco Escolano, and Nuria M Oliver. Diffwire: Inductive graph rewiring via the lovász bound. In *The First Learning on Graphs Conference*, 2022. URL `https://openreview.net/forum?id=IXvfIex0mX6f`.

Álvaro Arroyo, Alessio Gravina, Benjamin Gutteridge, Federico Barbero, Claudio Gallicchio, Xiaowen Dong, Michael Bronstein, and Pierre Vandergheynst. On vanishing gradients, over-smoothing, and over-squashing in gnns: Bridging recurrent and graph learning. *arXiv preprint arXiv:2502.10818*, 2025.

David Balduzzi, Marcus Frean, Lennox Leary, JP Lewis, Kurt Wan-Duo Ma, and Brian McWilliams. The shattered gradients problem: If resnets are the answer, then what is the question? In *International conference on machine learning*, pp. 342–350. PMLR, 2017.

Pradeep Kr Banerjee, Kedar Karhadkar, Yu Guang Wang, Uri Alon, and Guido Montúfar. Oversquashing in gnns through the lens of information contraction and graph expansion. In *Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pp. 1–8. IEEE, 2022.

Jørgen Bang-Jensen and Gregory Z Gutin. *Digraphs: theory, algorithms and applications*. Springer Science & Business Media, 2008.

Federico Barbero, Ameya Velingker, Amin Saberi, Michael Bronstein, and Francesco Di Giovanni. Locality-aware graph-rewiring in gnns. *arXiv preprint arXiv:2310.01668*, 2023.

Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

Åke Björck and Clazett Bowie. An iterative algorithm for computing the best estimate of an orthogonal matrix. *SIAM Journal on Numerical Analysis*, 8(2):358–364, 1971.

Mitchell Black, Zhengchao Wan, Amir Nayyeri, and Yusu Wang. Understanding oversquashing in gnns through the lens of effective resistance. In *International Conference on Machine Learning*, pp. 2528–2547. PMLR, 2023.

Shaked Brody, Uri Alon, and Eran Yahav. How attentive are graph attention networks? In *International Conference on Learning Representations*, 2021.

Ben Chamberlain, James Rowbottom, Maria I Gorinova, Michael Bronstein, Stefan Webb, and Emanuele Rossi. Grand: Graph neural diffusion. In *International Conference on Machine Learning*, pp. 1407–1418. PMLR, 2021.

Jianwen Chen, Jun He, Changyou Chen, and Shuiwang Wang. Measuring and relieving the over-smoothing problem for graph neural networks from the topological view. In *AAAI Conference on Artificial Intelligence*, volume 34, pp. 12492–12499, 2020a.

Ming Chen, Zhewei Wei, Zengfeng Huang, Bolin Ding, and Yaliang Li. Simple and deep graph convolutional networks. In *International conference on machine learning*, pp. 1725–1735. PMLR, 2020b.

Soham De, Samuel L Smith, Anushan Fernando, Aleksandar Botev, George Cristian-Muraru, Albert Gu, Ruba Haroun, Leonard Berrada, Yutian Chen, Srivatsan Srinivasan, et al. Griffin: Mixing gated linear recurrences with local attention for efficient language models. *arXiv preprint arXiv:2402.19427*, 2024.

Francesco Di Giovanni, Lorenzo Giusti, Federico Barbero, Giulia Luise, Pietro Lio, and Michael M Bronstein. On over-squashing in message passing neural networks: The impact of width, depth, and topology. In *International Conference on Machine Learning*, pp. 7865–7885. PMLR, 2023.

Vijay Prakash Dwivedi, Ladislav Rampášek, Michael Galkin, Ali Parviz, Guy Wolf, Anh Tuan Luu, and Dominique Beaini. Long range graph benchmark. *Advances in Neural Information Processing Systems*, 35:22326–22340, 2022.

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

Hongyang Gao and Shuiwang Ji. Graph u-nets. In *international conference on machine learning*, pp. 2083–2092. PMLR, 2019.

Johannes Gasteiger, Aleksandar Bojchevski, and Stephan Günnemann. Predict then propagate: Graph neural networks meet personalized pagerank. *arXiv preprint arXiv:1810.05997*, 2018.

Johannes Gasteiger, Stefan Weißenberger, and Stephan Günnemann. Diffusion improves graph learning. *Advances in neural information processing systems*, 32, 2019.

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

Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*, 2023.

Kai Guo, Kaixiong Zhou, Xia Hu, Yu Li, Yi Chang, and Xin Wang. Orthogonal graph neural networks. In *AAAI Conference on Artificial Intelligence*, volume 36, pp. 3996–4004, 2022.

Benjamin Gutteridge, Xiaowen Dong, Michael M Bronstein, and Francesco Di Giovanni. Drew: Dynamically rewired message passing with delay. In *International Conference on Machine Learning*, pp. 12252–12267. PMLR, 2023.

Xiaoxin He, Bryan Hooi, Thomas Laurent, Adam Perold, Yann LeCun, and Xavier Bresson. A generalization of vit/mlp-mixer to graphs. In *International conference on machine learning*, pp. 12724–12745. PMLR, 2023.

Kyle Helfrich, Devin Willmott, and Qiang Ye. Orthogonal recurrent neural networks with scaled cayley transform. In *International Conference on Machine Learning*, pp. 1969–1978. PMLR, 2018.

Adarsh Jamadandi, Celia Rubio-Madrigal, and Rebekka Burkholz. Spectral graph pruning against over-squashing and over-smoothing. *Advances in Neural Information Processing Systems*, 37:10348–10379, 2024.

Li Jing, Yichen Shen, Tena Dubcek, John Peurifoy, Scott Skirlo, Yann LeCun, Max Tegmark, and Marin Soljačić. Tunable efficient unitary neural networks (eunn) and their application to rnns. In *International Conference on Machine Learning*, pp. 1733–1741. PMLR, 2017.

Keller Jordan, Yuchen Jin, Vlado Boza, You Jiacheng, Franz Cesista, Laker Newhouse, and Jeremy Bernstein. Muon: An optimizer for hidden layers in neural networks, 2024.

Kedar Karhadkar, Pradeep Banerjee, and Guido Montufar. Fosr: First-order spectral rewiring for addressing oversquashing in gnns. In *International Conference on Learning Representations*, 2023.

Joseph B Keller. Closest unitary, orthogonal and hermitian operators to a given operator. *Mathematics Magazine*, 48(4):192–197, 1975.

Bobak Kiani, Randall Balestriero, Yann LeCun, and Seth Lloyd. projunn: Efficient method for training deep networks with unitary matrices. *Advances in Neural Information Processing Systems*, 35:14448–14463, 2022.

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

Zdislav Kovarik. Some iterative methods for improving orthonormality. *SIAM Journal on Numerical Analysis*, 7(3):386–389, 1970.

Mario Lezcano-Casado and David Martınez-Rubio. Cheap orthogonal constraints in neural networks: A simple parametrization of the orthogonal and unitary group. In *International Conference on Machine Learning*, pp. 3794–3803. PMLR, 2019.

Qimai Li, Zhichao Han, and Xiao-Ming Wu. Deeper insights into graph convolutional networks for semi-supervised learning. In *AAAI Conference on Artificial Intelligence*, volume 32, pp. 3538–3545, 2018.

Qimai Li, Zhichao Han, and Xiao-Ming Wu. Deepgcns: Can gcns go as deep as cnns? In *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 9267–9276, 2019a.

Qiyang Li, Saminul Haque, Cem Anil, James Lucas, Roger B Grosse, and Jörn-Henrik Jacobsen. Preventing gradient attenuation in lipschitz constrained convolutional networks. *Advances in neural information processing systems*, 32, 2019b.

Jingyuan Liu, Jianlin Su, Xingcheng Yao, Zhejun Jiang, Guokun Lai, Yulun Du, Yidao Qin, Weixin Xu, Enzhe Lu, Junjie Yan, et al. Muon is scalable for llm training. *arXiv preprint arXiv:2502.16982*, 2025.

Christopher Morris, Martin Ritzert, Matthias Fey, William L Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and leman go neural: Higher-order graph neural networks. In *AAAI conference on artificial intelligence*, volume 33, pp. 4602–4609, 2019.

Christopher Morris, Nils M. Kriege, Franka Bause, Kristian Kersting, Petra Mutzel, and Marion Neumann. Tudataset: A collection of benchmark datasets for learning with graphs. In *ICML 2020 Workshop on Graph Representation Learning and Beyond (GRL+ 2020)*, 2020. URL `www.graphlearning.io`.

Kenta Oono and Taiji Suzuki. Graph neural networks exponentially lose expressive power for node classification. *arXiv preprint arXiv:1905.10947*, 2019.

Antonio Orvieto, Samuel L Smith, Albert Gu, Anushan Fernando, Caglar Gulcehre, Razvan Pascanu, and Soham De. Resurrecting recurrent neural networks for long sequences. In *International Conference on Machine Learning*, pp. 26670–26698. PMLR, 2023.

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *International conference on machine learning*, pp. 1310–1318. Pmlr, 2013.

Bo Peng, Eric Alcaide, Quentin Anthony, Alon Albalak, Samuel Arcadinho, Stella Biderman, Huanqi Cao, Xin Cheng, Michael Chung, Matteo Grella, et al. Rwkv: Reinventing rnns for the transformer era. *arXiv preprint arXiv:2305.13048*, 2023.

Ladislav Rampášek, Michael Galkin, Vijay Prakash Dwivedi, Anh Tuan Luu, Guy Wolf, and Dominique Beaini. Recipe for a general, powerful, scalable graph transformer. *Advances in Neural Information Processing Systems*, 35:14501–14515, 2022.

Yu Rong, Wenbing Huang, Tingyang Xu, and Junzhou Huang. Dropedge: Towards deep graph convolutional networks on node classification. *arXiv preprint arXiv:1907.10903*, 2019.

Andrew M Saxe, James L McClelland, and Surya Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *arXiv preprint arXiv:1312.6120*, 2013.

Hanie Sedghi, Vineet Gupta, and Philip M Long. The singular values of convolutional layers. *arXiv preprint arXiv:1805.10408*, 2018.

Simone Severini. On the digraph of a unitary matrix. *SIAM Journal on Matrix Analysis and Applications*, 25(1):295–300, 2003.

Hamed Shirzad, Ameya Velingker, Balaji Venkatachalam, Danica J Sutherland, and Ali Kemal Sinop. Exphormer: Sparse transformers for graphs. In *International Conference on Machine Learning*, pp. 31613–31632. PMLR, 2023.

Sahil Singla and Soheil Feizi. Skew orthogonal convolutions. In *International Conference on Machine Learning*, pp. 9756–9766. PMLR, 2021.

Jan Tönshoff, Martin Ritzert, Eran Rosenbluth, and Martin Grohe. Where did the gap go? reassessing the long-range graph benchmark. *arXiv preprint arXiv:2309.00367*, 2023.

Jake Topping, Francesco Di Giovanni, Benjamin Paul Chamberlain, Xiaowen Dong, and Michael M Bronstein. Understanding over-squashing and bottlenecks on graphs via curvature. In *International Conference on Learning Representations*, 2022.

Asher Trockman and J Zico Kolter. Orthogonalizing convolutional layers with the cayley transform. *arXiv preprint arXiv:2104.07167*, 2021.

Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S Yu. A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems*, 32(1):4–24, 2020.

Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. Representation learning on graphs with jumping knowledge networks. In *International conference on machine learning*, pp. 5453–5462. PMLR, 2018.

Greg Yang, James B Simon, and Jeremy Bernstein. A spectral condition for feature learning. *arXiv preprint arXiv:2310.17813*, 2023.

Lingxiao Zhao and Leman Akoglu. Pairnorm: Tackling oversmoothing in gnns. *arXiv preprint arXiv:1909.12223*, 2019.

A Preliminaries

Definition 1 (Line graph)**.** *Given a graph G, its line graph $\mathsf{L}(G)$ is a graph such that*

- *each vertex of $\mathsf{L}(G)$ represents an edge of G, i.e., $\mathsf{V}[\mathsf{L}(G)] = \mathsf{E}[G]$;*
- *two vertices of $\mathsf{L}(G)$ are adjacent if and only if their corresponding edges share a common endpoint in G, i.e., $\mathsf{E}[\mathsf{L}(G)] = \{((i,j),(j,k)) \in \mathsf{V}[\mathsf{L}(G)] \times \mathsf{V}[\mathsf{L}(G)] \mid (i,j),(j,k) \in \mathsf{E}[G]\}$.*

Definition 2 (Eulerian graph)**.** *An Eulerian graph G is a graph containing an Eulerian cycle, i.e., there is a trail in G that starts and ends on the same vertex and visits every edge exactly once.*

Definition 3 (Permutation matrix)**.** *A permutation matrix $\mathbf{P} \in \mathbb{R}^{n\times n}$ is a square binary matrix that has exactly one entry of 1 in each row and each column with all other entries 0.*

Theorem 2. *Every permutation matrix is orthogonal, i.e., if $\mathbf{P}$ is a permutation matrix, $\mathbf{P}^\top\mathbf{P} = \mathbf{P}\mathbf{P}^\top = \mathbf{I}$.*

B Proof

B.1 Proof of Proposition 1

Proposition 1 is proved based on the following theorem and lemma. Theorem 3 is a direct result from Theorem 3 in Severini (2003). Lemma 2 is a well-known result in graph theory, which can be found in Theorem 1.7.2 of Bang-Jensen & Gutin (2008).

Theorem 3 (Existence of unitary adjacency matrix)**.** *Let G be a single-connected digraph. Its line graph $\mathsf{L}(G)$ (Definition 1) is the digraph of a unitary matrix if and only if G is Eulerian (Definition 2).*

Lemma 2 (A special Eulerian graph)**.** *A digraph graph is Eulerian if and only if it is connected and the in-degree and out-degree are equal at each vertex.*

Proof of Proposition 1. In Algorithm 1, the undirected edges in G are split into two directed edges in G'. Therefore, the in-degree and out-degree of each vertex in G' are equal, indicating G' is Eulerian. Then, Theorem 3 indicates that there exists unitary adjacency matrix $\mathbf{U}$ such that $\mathsf{S}[\mathbf{U}] = \mathbf{A}[\mathsf{L}(G')]$. □

B.2 Proof of Lemma 1

Proof. Given any unitary $\mathbf{U}$, let $\mathbf{U} = \mathbf{M} + \mathsf{U}[\mathbf{A}_w]$ for the properly chosen $\mathbf{M} \in \mathbb{C}^{2e\times 2e}$. Due to the unitarity of $\mathbf{U}$ and $\mathsf{U}[\mathbf{A}_w]$, we have

$$\mathbf{M}\mathsf{U}[\mathbf{A}_w]^\dagger + \mathbf{M}\mathbf{M}^\dagger + \mathsf{U}[\mathbf{A}_w]\mathbf{M}^\dagger = 0. \tag{3}$$

Then, we have

$$\begin{aligned}
\|\mathbf{A}_w - \mathbf{U}\|_F^2 &= \|\mathbf{A}_w - \mathbf{M} - \mathsf{U}[\mathbf{A}_w]\|_F^2 \\
&= \|\mathbf{A}_w - \mathsf{U}[\mathbf{A}_w]\|_F^2 + \mathrm{Tr}[\mathbf{M}\mathsf{U}[\mathbf{A}_w]^\dagger + \mathbf{M}\mathbf{M}^\dagger + \mathsf{U}[\mathbf{A}_w]\mathbf{M}^\dagger] - \mathrm{Tr}[\mathbf{M}^\dagger\mathbf{A}_w + \mathbf{A}_w^\dagger\mathbf{M}] \\
&= \|\mathbf{A}_w - \mathsf{U}[\mathbf{A}_w]\|_F^2 - \mathrm{Tr}[\mathbf{M}^\dagger\mathbf{A}_w + \mathbf{A}_w^\dagger\mathbf{M}] \\
&= \|\mathbf{A}_w - \mathsf{U}[\mathbf{A}_w]\|_F^2 - \mathrm{Tr}[\mathbf{M}^\dagger\mathsf{U}[\mathbf{A}_w](\mathbf{A}_w^\dagger\mathbf{A}_w)^{\frac{1}{2}} + (\mathbf{A}_w^\dagger\mathbf{A}_w)^{\frac{1}{2}}\mathsf{U}[\mathbf{A}_w]^\dagger\mathbf{M}].
\end{aligned}$$

Then, from equation 3, we have $\mathbf{M}\mathsf{U}[\mathbf{A}_w]^\dagger + \mathsf{U}[\mathbf{A}_w]\mathbf{M}^\dagger = -\mathbf{M}\mathbf{M}^\dagger$,

$$\|\mathbf{A}_w - \mathbf{U}\|_F^2 = \|\mathbf{A}_w - \mathsf{U}[\mathbf{A}_w]\|_F^2 + \mathrm{Tr}[(\mathbf{A}_w^\dagger\mathbf{A}_w)^{\frac{1}{2}}\mathbf{M}\mathbf{M}^\dagger]. \tag{4}$$

The second term above is non-negative because $\mathrm{Tr}[(\mathbf{A}_w^\dagger\mathbf{A}_w)^{\frac{1}{2}}\mathbf{M}\mathbf{M}^\dagger] = \mathrm{Tr}[\mathbf{M}^\dagger(\mathbf{A}_w^\dagger\mathbf{A}_w)^{\frac{1}{2}}\mathbf{M}]$ and $(\mathbf{A}_w^\dagger\mathbf{A}_w)^{\frac{1}{2}}\mathbf{M}$ is positive semi-definite. Therefore, for all unitary $\mathbf{U}$,

$$\|\mathbf{A}_w - \mathbf{U}\|_F^2 \geq \|\mathbf{A}_w - \mathsf{U}[\mathbf{A}_w]\|_F^2. \tag{5}$$

The result is proven. □

B.3 Proof of Proposition 2

Lemma 3. *For any unitary matrix* $\mathbf{U}$*, given two permutation matrices* $\mathbf{P}_1$ *and* $\mathbf{P}_2$*,* $\mathbf{P}_1\mathbf{U}\mathbf{P}_2^\top$ *is also unitary.*

Proof. Let $\hat{\mathbf{U}} = \mathbf{P}_1\mathbf{U}\mathbf{P}_2^\top$. Then, we have

$$\hat{\mathbf{U}}\hat{\mathbf{U}}^\dagger = \mathbf{P}_1\mathbf{U}\mathbf{P}_2^\top\mathbf{P}_2\mathbf{U}^\dagger\mathbf{P}_1^\top = \mathbf{P}_1\mathbf{U}\mathbf{U}^\dagger\mathbf{P}_1^\top = \mathbf{P}_1\mathbf{P}_1^\top = \mathbf{I},$$
$$\hat{\mathbf{U}}^\dagger\hat{\mathbf{U}} = \mathbf{P}_2\mathbf{U}^\dagger\mathbf{P}_1^\top\mathbf{P}_1\mathbf{U}\mathbf{P}_2^\top = \mathbf{P}_2\mathbf{U}^\dagger\mathbf{U}\mathbf{P}_2^\top = \mathbf{P}_2\mathbf{P}_2^\top = \mathbf{I}$$

which proves $\mathbf{P}_1\mathbf{U}\mathbf{P}_2^\top$ is unitary. □

Proof of Proposition 2. This proposition is proved by the uniqueness of the unitary matrix $\mathsf{U}[\mathbf{A}_w]$ when $\mathbf{A}_w$ is a full-rank matrix.

Assume there exists another unitary matrix $\mathbf{U} = \mathbf{M} + \mathsf{U}[\mathbf{A}_w]$ such that

$$\mathbf{U} = \arg \min_{\mathbf{U} \text{ is unitary}} \|\mathbf{A}_w - \mathbf{U}\|_F^2 . \tag{6}$$

According to the proof of Lemma 1, we have

$$\|\mathbf{A}_w - \mathbf{U}\|_F^2 = \|\mathbf{A}_w - \mathsf{U}[\mathbf{A}_w]\|_F^2 + \mathrm{Tr}[(\mathbf{A}_w^\dagger\mathbf{A}_w)^{\frac{1}{2}}\mathbf{M}\mathbf{M}^\dagger].$$

Since $\mathbf{A}$ is full-rank matrix, $(\mathbf{A}_w^\dagger\mathbf{A}_w)^{\frac{1}{2}}$ is positive definite. Therefore, we have

$$\|\mathbf{A}_w - \mathbf{U}\|_F^2 > \|\mathbf{A}_w - \mathsf{U}[\mathbf{A}_w]\|_F^2. \tag{7}$$

which contradicts the assumption that $\mathbf{U}$ is the minimizer of $\|\mathbf{A}_w - \mathbf{U}\|_F^2$. Thus, the unitary projection of $\mathbf{A}_w$ is unique when $\mathbf{A}_w$ is full-rank.

Because $\mathsf{U}[\mathbf{A}_w]$ is unique and is the minimizer of $\|\mathbf{A}_w - \mathbf{U}\|_F^2$, from Lemma 3, we have $\mathbf{P}_1\mathsf{U}[\mathbf{A}_w]\mathbf{P}_2^\top$ the minimizer of

$$\arg \min_{\mathbf{U} \text{ is unitary}} \left\|\mathbf{P}_1\mathbf{A}_w\mathbf{P}_2^\top - \mathbf{U}\right\|_F^2$$

for any permutation matrices $\mathbf{P}_1$ and $\mathbf{P}_2$. Because $\mathbf{P}_1\mathbf{A}_w\mathbf{P}_2^\top$ is also full-rank, $\mathbf{P}_1\mathsf{U}[\mathbf{A}_w]\mathbf{P}_2^\top$ is the unitary projection of $\mathbf{P}_1\mathbf{A}_w\mathbf{P}_2^\top$, which proves that $\mathsf{U}[\mathbf{P}_1\mathbf{A}_w\mathbf{P}_2^\top] = \mathbf{P}_1\mathsf{U}[\mathbf{A}_w]\mathbf{P}_2^\top$ for any permutation matrices $\mathbf{P}_1$ and $\mathbf{P}_2$. □

B.4 Proof of Theorem 1

Our proof is based on the GNN model from Figure 2(a) with the activation function being ReLU. GUMP is analyzed with $\mathbf{A}$ being unitary and vanilla message passing is analyzed with $\mathbf{A}$ being the normalized adjacency matrix. Motivated by Xu et al. (2018), we analyze with the expected Jacobian.

Motivated by Xu et al. (2018), we first introduce the expected Jacobian.

Theorem 4. *Given a L-layer GNN with ReLU as activation function, i.e.,* $\mathbf{H}^{(k)} = \mathsf{ReLU}(\mathbf{A}\mathbf{H}^{(k-1)}\mathbf{W}_k), \mathbf{H}^{(0)} = \mathbf{X}, k = 1 \cdots L$*, assume that all paths in the computation graph of the model are activated with the same probability of success* ρ*, the expected Jacobian is*

$$\mathbb{E}\left[\frac{\partial \mathbf{h}_i^{(L)}}{\partial \mathbf{x}_s}\right] = \rho \prod_{l=L}^{1} \mathbf{W}_l^\top \left(\mathbf{A}^L\right)_{is}, \tag{8}$$

Proof. Denote by $\mathbf{f}_i^{(l)}$ the pre-activated feature of $\mathbf{h}_i^{(l)}$, i.e., $\mathbf{f}_i^{(l)} = \sum_{z\in\mathcal{N}(i)} \mathbf{A}_{iz}\mathbf{h}_z^{(l-1)}\mathbf{W}_l$, for any $l = 1 \cdots L$, we have

$$\frac{\partial \mathbf{h}_i^{(l)}}{\partial \mathbf{h}_s^{(0)}} = \mathsf{diag}\left(1_{\mathbf{f}_i^{(l)}>0}\right) \cdot \left(\sum_{z\in\mathcal{N}(i)} \mathbf{A}_{iz}\frac{\partial \mathbf{h}_z^{(l-1)}}{\partial \mathbf{h}_s^{(0)}}\right) \cdot \mathbf{W}_l^\top .$$

By the chain rule, we get

$$\begin{aligned}\frac{\partial \mathbf{h}_i^{(L)}}{\partial \mathbf{h}_s^{(0)}} &= \sum_{p=1}^{\Psi} \left[\frac{\partial \mathbf{h}_i^{(L)}}{\partial \mathbf{h}_s^{(0)}}\right]_p \\ &= \sum_{p=1}^{\Psi} \prod_{l=L}^{1} \mathsf{diag}\left(1_{\mathbf{f}^{(l)}_{v_p^l}>0}\right) \mathbf{A}_{v_p^l v_p^{l-1}} \mathbf{W}_l^\top .\end{aligned}$$

Here, Ψ is the total number of paths $v_p^L v_p^{L-1} \cdots v_p^1 v_p^0$ of length $L+1$ from $v_p^0 = s$ to $v_p^L = i$. For $l = 1 \cdots L-1$, $v_p^{l-1} \in \mathcal{N}(v_p^l)$.

For each path p, the derivative $[\partial \mathbf{h}_i^{(L)} / \partial \mathbf{h}_s^{(0)}]_p$ represents a directed acyclic computation graph. At a layer l, we can express an entry of the derivative as

$$\left[\frac{\partial \mathbf{h}_i^{(L)}}{\partial \mathbf{h}_s^{(0)}}\right]_p^{(m,n)} = \prod_{l=L}^{1} \mathbf{A}_{v_p^l v_p^{l-1}} \sum_{q=1}^{\Phi} Z_q \prod_{l=L}^{1} w_q^{(l)},$$

where Φ is the number of paths q from the input neurons to the output neuron (m,n), in the computation graph of $[\partial \mathbf{h}_i^{(L)} / \partial \mathbf{h}_s^{(0)}]_p$. For each layer l, $w_q^{(l)}$ is the entry of $\mathbf{W}_l^\top$ that is used in the q-th path. Finally, $Z_q \in \{0,1\}$ represents whether the q-th path is active ($Z_q = 1$) or not ($Z_q = 0$) as a result of ReLU activation of the entries of $\mathbf{f}^{(l)}_{v_p^l}$'s on the q-th path.

Under the assumption that Z_q is a Bernoulli random variable with success probability ρ. Because of $\mathbb{P}[Z_q = 1] = \rho, \forall q$, we have

$$\mathbb{E}\left[\left[\frac{\partial \mathbf{h}_i^{(L)}}{\partial \mathbf{h}_s^{(0)}}\right]_p^{(m,n)}\right] = \rho \prod_{l=L}^{1} \mathbf{A}_{v_p^l v_p^{l-1}} \sum_{q=1}^{\Phi} \prod_{l=L}^{1} w_q^{(l)}.$$

Then, the expected Jacobian is

$$\mathbb{E}\left[\frac{\partial \mathbf{h}_i^{(L)}}{\partial \mathbf{x}_s}\right] = \sum_{p=1}^{\Psi} \mathbb{E}\left[\left[\frac{\partial \mathbf{h}_i^{(L)}}{\partial \mathbf{h}_s^{(0)}}\right]_p\right] = \rho \prod_{l=L}^{1} \mathbf{W}_l^\top \left(\mathbf{A}^L\right)_{is}.$$

□

Proof of Theorem 1. We analyze different components in the expected Jacobian equation 8.

Firstly, $\prod_{l=L}^{1} \mathbf{W}_l^\top$ is the product of weight matrices in GNNs, which will not change exponentially with respect to L because the weight matrices are appropriately initialized to avoid exploding or decay.

Then, we focus the analysis on $\mathbf{A}^L$. We first diagonalize $\mathbf{A}$ in complex space, i.e., $\mathbf{A} = \mathbf{P}\mathbf{\Lambda}\mathbf{P}^{-1}$ with $\mathbf{\Lambda}$ being a diagonal matrix with its diagonal elements being eigenvalues of $\mathbf{A}$. Since $\mathbf{A}$ is unitary, the elements in $\mathbf{\Lambda}$ are complex units, i.e., $\mathbf{\Lambda} = \mathsf{diag}(e^{i\theta_1}, e^{i\theta_2}, \cdots, e^{i\theta_{2e}})$. Therefore, $\mathbf{A}^L$ is equal to $\mathbf{P}\mathbf{\Lambda}^L\mathbf{P}^{-1}$, where $\Lambda = \mathsf{diag}(e^{i\theta_1 L}, e^{i\theta_2 L}, \cdots, e^{i\theta_{2e} L})$ which is also a diagonal matrix with its diagonal elements being complex units. Then, we have $(\mathbf{A}^L)_{is} = \mathbf{P}_i \mathsf{diag}(e^{i\theta_1 L}, e^{i\theta_2 L}, \cdots, e^{i\theta_{2e} L})(\mathbf{P}^{-1})_s$, which is a value that does not change exponentially with respect to L and can be bounded by constants. With Euler's formula, i.e., $e^{ix} = \cos x + i \sin x$, the relation between $(\mathbf{A}^L)_{is}$ and L is a trigonometric function.

Since the trigonometric function can be bounded by constants, we have $\prod_{l=L}^{1} \mathbf{W}_l^\top \left(\mathbf{A}^L\right)_{is}$ is bounded by constants, i.e., $\mathbb{E}\left[\partial \mathbf{h}_i^{(L)} / \partial \mathbf{x}_s\right] = \mathcal{O}(1)$. □

B.5 Theory of Vanilla Message Passing

The next theorem analyzes the expected Jacobian of vanilla message passing.

Theorem 5. *The expected Jacobian for vanilla message passing, i.e.,* $\mathbf{A} = \hat{\mathbf{A}}$ *in GNN, is approximately in the order of* $\mathbb{E}\left[\partial \mathbf{h}_i^{(L)} / \partial \mathbf{x}_s\right] = \mathcal{O}(c^L)$*, where* $c \in (0, 1)$*.*

Theorem 5 shows that the Jacobian of vanilla message passing decays exponentially concerning L, thus leading to inefficient learning. Therefore, Theorems 1 and 5 indicate that GUMP can achieve an optimal Jacobian stability, i.e., $\mathcal{O}(1)$, while the vanilla message passing cannot. Theorems 1 and 5 are validated by experiments in Section 4.4.

proof of Theorem 5. We analyze the different components in the expected Jacobian equation 8.

Firstly, $\prod_{l=L}^{1} \mathbf{W}_l^\top$ is the product of weight matrices in GNNs, which will not change exponentially with respect to L because the weight matrices are appropriately initialized to avoid exploding or decay.

Then, we focus the analysis on $\mathbf{A}^L$. We first diagonalize $\mathbf{A}$ in complex space, i.e., $\mathbf{A} = \mathbf{P}\mathbf{\Lambda}\mathbf{P}^{-1}$ with $\mathbf{\Lambda}$ being a diagonal matrix with its diagonal elements being eigenvalues of $\mathbf{A}$. Since $\mathbf{A} = \hat{\mathbf{A}}$, the elements in $\mathbf{\Lambda}$ are in the range of 0 and 1, i.e., $\mathbf{\Lambda} = \mathsf{diag}(\theta_1, \theta_2, \cdots, \theta_{2e}), \theta_i \in [0, 1]$. Therefore, $\mathbf{A}^L$ is equal to $\mathbf{P}\mathbf{\Lambda}^L\mathbf{P}^{-1}$, where $\Lambda = \mathsf{diag}(\theta_1^L, \theta_2^L, \cdots, \theta_{2e}^L)$. Without any assumption on the graph structure, there are many eigenvalues smaller than one. Then, we have $(\mathbf{A}^L)_{is} = \mathbf{P}_i \mathsf{diag}(\theta_1^L, \theta_2^L, \cdots, \theta_{2e}^L)(\mathbf{P}^{-1})_s$, which is a value that change exponentially with respect to L.

Finally, we have $\prod_{l=L}^{1} \mathbf{W}_l^\top \left(\mathbf{A}^L\right)_{is}$ in the order of $\mathcal{O}(c^L)$, i.e., $\mathbb{E}\left[\partial \mathbf{h}_i^{(L)} / \partial \mathbf{x}_s\right] = \mathcal{O}(c^L), c \in (0, 1)$. □

C More Experimental Results

Statistics of datasets The statistics of datasets used in experiments are shown in Table 3.

Table 3: Statistics of datasets.

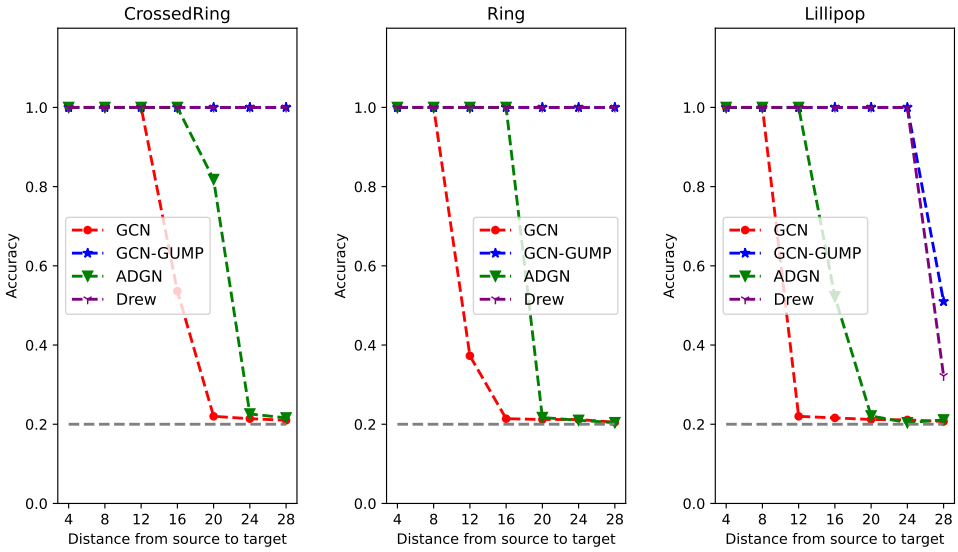
	#graphs	Avg. nodes	Avg. edges	Task type
Mutag	188	17.9	39.6	Graph Classification
Proteins	1,113	39.1	145.6	Graph Classification
Enzymes	600	32.6	124.3	Graph Classification
NC1	4110	29.87	32.30	Graph Classification
NC109	4127	29.68	32.13	Graph Classification
Peptides-func	15,535	150.94	307.30	Graph Classification
Peptides-struct	15,535	150.94	307.30	Graph Regression

Hyperparameters of GUMP The hyperparameters of GUMP for both synthetic and real datasets are shown in Table 4.

Table 4: Hyperparameters of GUMP for datasets in experiments. $\text{layer}_{\text{GUMP}}$, lr_{base}, wd_{base}, lr_{GUMP}, wd_{GUMP}, drop., d', d, batch size, $\text{layer}_{\text{base}}$, opt., sched., and epoch denotes the number of layers of GUMP, the learning rate of base GNN, weight decay of base GNN, the learning rate of GUMP, weight decay of GUMP, dropout rate, dimension of calculating (1), hidden dimension of GNN, batch size, number of layers of base GNN, optimizer, scheduler, and number of epochs, respectively.

	$\text{layer}_{\text{GUMP}}$	lr_{base}	wd_{base}	lr_{GUMP}	wd_{GUMP}	drop.	d'	d	batch size	$\text{layer}_{\text{base}}$	opt.	sched.	epoch
CrossedRing	-	10^{-4}	10^{-6}	10^{-4}	0	0	32	32	20	0	adam	none	200
Ring	-	10^{-4}	10^{-6}	10^{-4}	0	0	32	32	20	0	adam	none	200
CliquePath	-	10^{-4}	10^{-6}	10^{-4}	0	0	32	32	20	0	adam	none	200
Mutag	16	10^{-2}	10^{-4}	10^{-4}	0	0	32	64	16	5	adam	none	100
Proteins	20	10^{-2}	10^{-2}	10^{-4}	10^{-2}	0	32	64	64	3	adam	none	100
Enzymes	10	10^{-2}	10^{-4}	10^{-4}	0	0	32	64	16	1	adam	none	100
NC1	10	10^{-2}	10^{-4}	10^{-4}	0	0	32	64	16	1	adam	none	100
NC109	10	10^{-2}	10^{-4}	10^{-4}	0	0	32	64	16	1	adam	none	100
Peptides-func	12	0.005	0.1	0.1	0.1	0.2	32	256	200	3	adam	cos.	250
Peptides-struct	12	0.005	0.1	0.005	0.1	0.2	32	256	200	3	adam	cos.	250

Comparison GUMP with more methods For the synthetic datasets, we compare GUMP with Drew and ADGN on synthetic datasets in Figure 5. The results show that the performances of GUMP and Drew are close, while ADGN performs worse.



(a) CrossedRing (b) Ring (c) CliquePath

Figure 5: The performance of GCN and GCN-GUMP on the CrossedRing, Ring, and CliquePath with different distances from source to target.

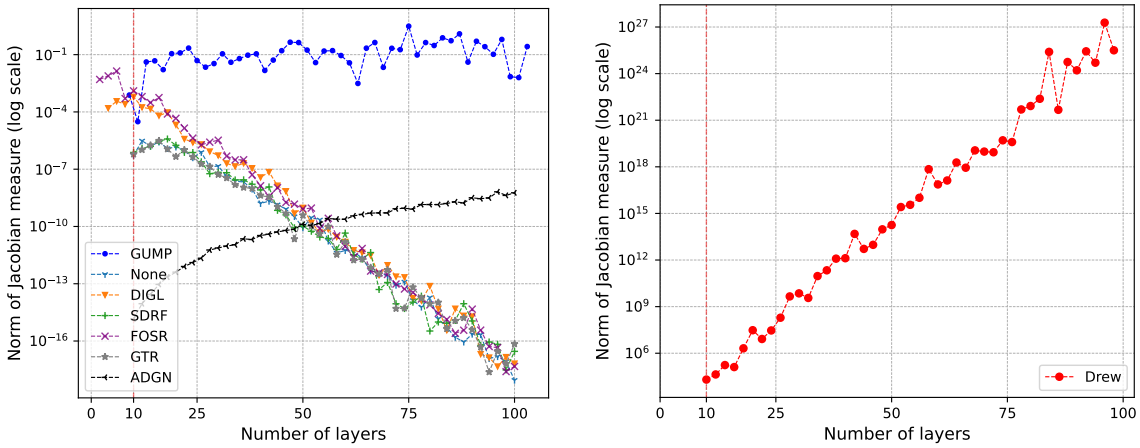
We further compare the performances of GUMP by applying different convolution operator. The results are shown in Table 5. The results show that GUMP can improve the performance of both GCN and GIN, while GIN-GUMP achieves better performance than GCN-GUMP.

Table 5: Graph classification accuracy on the TUDataset. **First**, second, and *third* best results are bold, underlined, and underwaved, respectively.

Base GNN	Methods	Mutag	Proteins	Enzymes	NCI1	NCI109	Rank
GCN	None	72.15±2.44	70.98±0.74	27.67±1.16	68.74±0.45	67.90±0.50	4.2
	None (+layer)	70.05±1.83	69.80±0.99	23.63±1.07	63.94±1.34	55.92±1.26	6.8
	DIGL	*79.70±2.15*	70.76±0.77	35.72±1.12	69.76±0.42	69.37±0.43	3.0
	SDRF	71.05±1.87	70.92±0.79	*28.37±1.17*	68.21±0.43	66.78±0.44	4.8
	FoSR	80.00±1.57	73.42±0.81	25.07±0.99	57.27±0.54	56.82±0.60	4.6
	GTR	79.10±1.86	*72.59±2.48*	27.52±0.99	*69.37±0.38*	*67.97±0.47*	3.6
	GCN-GUMP	**84.89±1.63**	**74.88±0.87**	**36.02±1.43**	**77.97±0.42**	**75.85±0.44**	1.0
GIN	None	77.70±3.60	70.80±0.83	33.80±1.12	*75.65±0.49*	74.93±0.46	4.0
	None (+layer)	69.80±2.75	68.71±0.96	25.92±1.07	73.49±0.46	72.47±0.53	6.6
	DIGL	79.80±2.08	70.71±0.67	*35.74±1.20*	79.37±0.43	76.88±0.39	2.8
	SDRF	*78.40±2.80*	69.81±0.79	35.82±1.09	74.55±0.54	73.89±0.43	4.2
	FoSR	78.00±2.22	75.11±0.82	29.20±1.38	70.15±0.47	69.93±0.45	5.2
	GTR	77.60±2.84	*73.13±0.69*	30.57±1.42	75.45±0.44	*75.28±0.42*	4.2
	GIN-GUMP	**86.72±1.53**	**75.43±0.70**	**48.43±1.24**	**81.25±0.37**	**78.45±0.44**	1.0

We also compare the Jacobian of GUMP with Drew and ADGN in Figure 6. Jacobian of Drew exponentially increases, suggesting its potential numerical instability when training Drew with more layers. The Jacobian

of ADGN is small when the ADGN layer is small and steadily increases to 10^{-8} as the ADGN layer reaches 100. Although the Jacobian of ADGN does not exhibit exponential decay, the correlation between distant nodes is significantly weaker compared to GUMP.



(a) (b)

Figure 6: Jacobian norm versus layers on NCI1

Performance on node classification tasks We also apply GUMP to node classification tasks on Cora and Citeseer datasets. The results are shown in Table 6.

Table 6: Accuracy of node classification datasets: Cora and Citeseer

Layers		2	4	8	16	64
Cora	GCN	81.1	80.4	69.5	60.3	28.7
	GCNII	82.2	82.6	84.2	84.6	85.5
	GCN-GUMP	84.6	86.2	84.8	85.4	87.4
Citeseer	GCN	70.8	67.6	30.2	18.3	20.0
	GCNII	68.2	68.9	70.6	72.9	73.4
	GCN-GUMP	73.0	73.0	72.8	72.4	75.8

Training time of GUMP The time of training GCN and GCN-GUMP 100 epochs on the TUDataset is shown in Table 7.

Table 7: Training seconds of GCN and GCN-GUMP on TUDataset for 100 epochs

	MUTAG	Proteins	Enzymes	NCI1	NCI109
GCN	4.39	20.57	11.26	71.79	74.56
GCN-GUMP	7.21	26.19	20.29	82.02	93.38

D Apply GUMP to Directed Graphs

GUMP can also be applied to directed graphs in Table 4. The transformation of directed graphs is also based on Lemma 2.

Algorithm 4 Graph transformation for directed graphs

Require: A directed graph $G = (V, E)$;
Initialize a new digraph $G' = (V, E')$;
for $(i, j) \in E$ **do**
  Add (i, j) and (j, i) to E';
end for
Remove duplicated edges in E';
Convert G' to its line graph $\mathsf{L}(G')$;
Return: A digraph $\mathsf{L}(G')$.