Appendix: Robust Neural Architecture Search for Graph Neural Networks

A. Broader Impact

The key result of this research could aid the GNN family since G-RNA is able to search for robust architectures for learning graphs automatically. The proposed approach can simultaneously enhance the accuracy as well as the resilience to adversarial assaults of GNNs, allowing them to be used in more safety-critical applications including power grids, financial transactions, and transportation.

B. Details of G-RNA

B.1. Detailed denotation for Graph Structure Masks

Here, we explain the detailed denotation introduced in Table 1. For *N*ode *F*eature *S*imilarity (*NFS*), J_{ij} stands for the Jaccard similarity between node i and j. For *N*eighbor *I*mportance *E*stimation (*NIE*), $\hat{\alpha}_{ij}^{(l)}$ is the importance weight as calculated in GNN-Guard [46]. For *V*ariable *P*ower *O*perator (*VPO*), V is a hyper-parameter indicating the power order of VPO.

B.2. Correlation coefficient operations

For correlation coefficient operations, we follow the literature, and the detailed formulas are given in Table 4.

B.3. Examples of popular GNNs based on search space of G-RNA

Our search space could recover some classic and manually designed GNNs and state-of-the-art robust GNNs such as GCN-SVD, GCN-Jaccard, GNN-Guard, and VPN, as shown in Table 5.

Table 4. Commonly used correlation coefficient operations.

$\mathcal{O}_e$	**Formula**
Identity	$e_{ij}^{iden} = 1$
GCN	$e_{ij}^{gcn} = 1/\sqrt{d_i d_j}$
GAT	$e_{ij}^{gat} = leaky\_relu(\mathbf{W}_l\mathbf{h}_i + \mathbf{W}_r\mathbf{h}_j)$
GAT-Sym	$e_{ij}^{sym} = e_{ij}^{gat} + e_{ji}^{gat}$
Cos	$e_{ij}^{cos} =< \mathbf{W}_l\mathbf{h}_i, \mathbf{W}_r\mathbf{h}_j >$
Linear	$e_{ij}^{lin} = tanh(sum(\mathbf{W}_l\mathbf{h}_i))$
Gene-Linear	$e_{ij}^{gene} = \mathbf{W}_a tanh(\mathbf{W}_l\mathbf{h}_i + \mathbf{W}_r\mathbf{h}_j)$

Table 5. Recover existing GNN layers from our search space.

Method	$[\mathcal{O}_{\mathcal{D}}, \mathcal{O}_e, \mathcal{O}_{aggr}, \mathcal{O}_{comb}, \mathcal{O}_{skip}]$
GCN [21]	[*Identity, GCN, Sum, Identity, None*]
JK-Net [41]	[*Identity, GCN, Sum, Identity, Skip*]
GAT [34]	[*Identity, GAT, Sum, Identity, None*]
GIN [40]	[*Identity, Identity, Sum, GIN, None*]
GraphSAGE [15]	[*Identity, Identity, Mean, SAGE, None*]
GNN-Guard [46]	[*NIE, GCN, Sum, Identity, None*]
VPN [19]	[*VPO, GCN, Sum, Identity, None*]

B.4. Evolutionary search algorithm

Based on our proposed robustness metric, we adopt the evolutionary algorithm to search for the optimal robust architectures, as shown in Algorithm 1. Inspired by the biological evolution process, the evolutionary algorithm solves optimization problems by mutation, crossover, and selection. The selection operation is conducted via a fitness function, which is set as $\mathrm{ACC}_{val}(\boldsymbol{\alpha}) + \lambda R(\boldsymbol{\alpha})$ in our algorithm. The inference function in Line 3 calculates the fitness score from the validation set.

C. Additional Experimental Results

C.1. Defense Performance Against Targeted Attacks

Targeted attacks and non-targeted attacks are two vital branches of the adversarial attack field. In this section, we compare our proposed methods to baselines under a targeted attack, Nettack [53], as a complement to defensive results under non-targeted attacks. For each dataset, we choose 40 correctly classified nodes (10 nodes with the highest margin, 10 nodes with

Algorithm 1: Evolutionary Search Algorithm

Input: The maximum iteration number max_iter, supernet weights $\mathbf{W}$, the population size P, the mutation size s, the mutation probability p, the crossover size n, the original graph $G = (\mathbf{A}, \mathbf{X})$, the perturbed adjacency matrices $\{\mathbf{A}'_t, t = 1, .., T\}$, the number of optimal architectures k.

Output: The top-K architectures with the highest robustness metric.

$candidates \leftarrow$ initialize_population(P) ;
for $iter = 1, ..., max\_iter$ **do**
  $Q \leftarrow$ Inference$(candidates, G, \{\mathbf{A}'_i\})$;
  Top-$k \leftarrow$ Select_top$(Q, candidates, k)$;
  $P_{\text{crossover}} \leftarrow$ Crossover(Top-k, n);
  $P_{\text{mutation}} \leftarrow$ Mutation(Top-k, s, p);
  $candidates \leftarrow P_{\text{crossover}} \cup P_{\text{mutation}}$;
Return Top-k

the lowest margin, and 20 nodes randomly) and report the average classification accuracy for these target nodes. Note that this setting leads to higher reported accuracy than the typical accuracy evaluated on the whole test set. We conduct this experiment for five runs and the defensive performance is shown in Table 6. It reads that G-RNA still outperforms the other baselines almost across all datasets under the perturbed setting while maintaining on-par performance on the clean graphs.

Table 6. Defensive performance on targeted nodes (mean in percentages) under targeted attacks Nettack. "-" indicates the result is unavailable due to the high time complexity of the model.

Dataset	Model	GCN	GCN-JK	GAT	GAT-JK	GCN-Jaccard	Pro-GNN	GraphNAS	G-RNA
Cora	No Attack	93.25	94.75	94.75	92.25	98.50	97.50	96.00	94.95
	Attack	16.75	16.25	19.25	23.75	46.25	50.00	25.75	51.25
Citeseer	No Attack	88.00	86.25	91.00	88.25	95.25	92.50	96.25	92.50
	Attack	14.75	12.50	17.25	14.75	22.00	45.00	9.00	60.00
PubMed	No Attack	95.00	95.75	95.50	97.00	99.00	-	95.25	95.25
	Attack	12.50	16.25	13.00	23.25	1.50	-	9.00	59.75
ogbn-arxiv	No Attack	85.00	90.50	92.50	93.50	95.00	-	97.50	92.25
	Attack	5.00	0.50	5.00	6.75	0.50	-	2.50	22.50
Amazon_photo	No Attack	96.25	98.25	89.50	98.50	99.75	-	97.50	94.50
	Attack	12.50	8.00	12.00	28.50	9.00	-	5.00	32.50

C.2. Defense on Heterophily Graph

To better illustrate the effectiveness of our proposed method, we select one heterophily graph, Wisconsin and evaluate the performance of G-RNA on it. Table 7 demonstrates the performance of various methods under both clean and perturbed data, where 5% edges are manipulated by Mettack. We could find that even without the assumption of graph homophily, our method could outperform other baselines under both settings.

Table 7. Experimental result on heterophily graph.

GCN		GAT		GCN-Jaccard	
clean	perturbed	clean	perturbed	clean	perturbed
50.55±1.69	48.95±2.16	52.40±2.22	52.1±2.20	53.60±3.32	52.15±2.06
Pro-GNN		GraphNAS		G-RNA	
51.05±0.58	48.2±1.99	58.10±4.70	57.60±2.80	60.50±2.97	59.25±2.76

C.3. Details of Searched Architectures

The optimal searched architectures by G-RNA for each dataset are listed in Table 8. The optimal model depth searched is 2-layer for Cora, 3-layer for CiteSeer, and 1-layer for PubMed. All three architectures use *NFS* as the pre-processing adjacency mask. *NIE* is selected in the second layer for CiteSeer. Besides, *LSTM* is chosen to be the layer aggregator for Cora and PubMed, while *Concat* is selected for CiteSeer.

Table 8. Searched architecture results.

Dataset	Layer	Intra-layer Operations	Layer aggregator
Cora	layer1: *Skip* layer2: *Skip*	[*NFS, GAT-Cos, Sum, Identity*] [*Identity, Identity, Sum, Identity*]	*LSTM*
CiteSeer	layer1: *Skip* layer2: *None* layer3: *Skip*	[*NFS, Gene-Linear, Mean, Identity*] [*NIE, GAT, Max, GIN*] [*VPO, Identity, Max, Identity*]	*Concat*
PubMed	layer1: *Skip*	[*NFS, Identity, Max, SAGE*]	*LSTM*
ogbn-arxiv	layer1: *Skip* layer2: *Skip*	[*NFS, gat, Sum, GIN*] [*Identity, GAT-Sym, Sum, GIN*]	*Concat*
Amazon_photo	layer1: *Skip* layer2: *Skip* layer3: *Skip*	[*NFS, GAT-Linear, Sum, GIN*] [*Identity, Identity, Sum, GIN*] [*Identity, GAT-Sym, Sum, Identity*]	*Concat*

C.4. Sensitivity Analysis

To in-depth understand the effect of the hyper-parameter λ in the search process, we conduct an ablation study on its sensitivity. Fig. 5 shows the performance for $\lambda = 0.01, 0.05, 0.5$ on Cora dataset. The experiments on the other datasets show similar patterns. When λ is very small, e.g., 0.01, the performance on clean data is gratifying, but the performance drops fast under adversarial perturbations. On the contrary, when λ is relatively large, e.g., 0.5, the prediction accuracy on the clean graph is poor, but the performance is stable even when the graph is heavily attacked. As a result, we need to properly balance the accuracy and robustness tradeoff by choosing a suitable λ. In practice, we find that tuning λ in the range of $[0.03, 0.3]$ usually leads to satisfactory results.

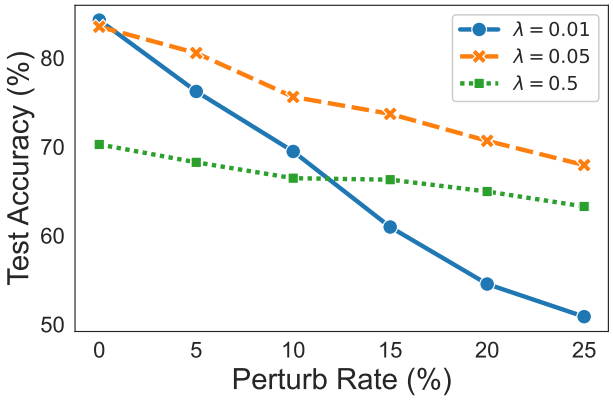

Figure 5. Parameter sensitivity analysis for λ.

Table 9. Running time ($hours$) comparison on Cora dataset.

Pro-GNN	GraphNAS	G-RNA
0.22	4.12	2.23

C.5. Computational Efficiency Analysis

We also empirically demonstrate the computational efficiency of G-RNA in comparison with two comprehensive baselines, GraphNAS and Pro-GNN, on the Cora dataset in Table 9. Due to the usage of graph structure mask operations and the evolutionary algorithm, our methods are not as efficient as darts-based NAS methods or plain robust GNN solutions. However, we can find that the computation cost of our G-RNA is comparable to GraphNAS. This lack of efficiency is consistent with one of the limitations that we claimed in Section 6, which is a shared issue for many NAS methods and we choose to leave its improvement as a future work.

D. Experimental Setup

In this section, we start with the dataset statistics and then describe the implementing details for G-RNA and other baselines.

D.1. Dataset

We mainly consider the semi-supervised node classification task on three citation graph datasets [32][1], including Cora, CiteSeer, and PubMed. For each graph, we randomly select 10% nodes for training, 10% nodes for validation, and the rest 80% nodes for testing to be consistent with existing literature. Besides, we also evaluate the effectiveness of G-RNA on one heterophily graph, one citation network from Open Graph Benchmark (OGB) [17], and a co-purchase network that intends to predict the product category on Amazon [28]. The specific statistics for each dataset are shown in Table 10.

Table 10. Dataset statistics.

Dataset	# Nodes	# Edges	# Features	# Classes
Cora	2,708	5,429	1,433	7
Citeseer	3,327	4,732	3,703	6
PubMed	19,717	44,338	500	3
ogbn-arxiv	50,802	108,554	128	40
Amazon_photo	7,650	119,081	745	8
Winsconsin	251	499	1703	5

D.2. Baselines

In order to validate the effectiveness and robustness of our G-RNA, we compare it with state-of-the-art GNNs, manually-designed robust GNNs, and Graph NAS methods:

- **GCN** [21]: Graph Convolution Network (GCN) is the pioneer of GNN and represents as a classic victim model against adversarial attacks.
- **GCN-JK** [41]: GCN-JK combines GCN with jumping knowledge networks (JK-Net). JK-Net adds skip connections among different hidden layers and adaptively learns node-wise neighborhoods when aggregating node representations.
- **GAT** [34]: Graph Attention Network (GAT) utilizes the self-attention mechanism to learn different weights for edges.
- **GAT-JK** [41]: GAT-JK is the combination of GAT and JK-Net. Again, We use GNNs and their JK-Net variants to study the effect of the JK-Net backbone.
- **RGCN** [51]: Robust Graph Convolutional Network (RGCN) is an attention-based defense model by treating node representations as Gaussian distribution and assigning less attention to nodes with high variance.
- **GCN-Jaccard** [38]: GCN-Jaccard conducts edge pruning according to the jaccard similarity among node representations. It is a simple yet effective pre-processing defensive baseline.
- **Pro-GNN** [20]: Based on the graph properties of sparsity, low rank, and feature smoothness, Property GNN (Pro-GNN) learns parameters and purifies the adjacency matrix at the same time.

[1] https://github.com/kimiyoung/planetoid/tree/master/data

- **DropEdge** [31]: DropEdge randomly drops edges for each graph convolution layer.
- **PTDNet**[2] [27]: PTDNet improves the robustness of GNNs and prunes task-irrelevant edges with parameterized networks.
- **GraphNAS**[3] [10]: GraphNAS serves as a baseline that validates the robustness of plain graph NAS under adversarial attacks.
- **GASSO**[4] [30]: GASSO conducts graph structure learning with architecture search to search under potential noisy graph data.

Except for GraphNAS, we use the public implementation for all baselines via PyTorch Geometric (PyG) [9] and DeepRobust [23].

D.3. Parameter Settings

To search for the optimal robust GNNs, we first construct a supernet and train it afterward. Then, we search with our robustness metric using the weights of the trained supernet. At last, we retrain the top-5 selected architectures from scratch. For the construction of the supernet, we limit the maximum layer number to 3.

Since the random attack is one simple yet effective attack method, we use random attack as our attack proxy and generate 5 perturbed graphs ($T = 5$) with a 5% perturbation rate. Meanwhile, it is easy to implement, so that we use it to evaluate the robustness of architectures without the knowledge of the specific attacker. Some other attack algorithms could also be utilized to generate adversarial examples for the evaluation of the robustness metric if some prior knowledge from the attacker is given.

For the proposed G-RNA, the supernet is trained for 1,000 epochs with a learning rate of 0.005 and a weight decay of 3e-4. The linear dropout rate is fixed at 0.5, and the attention dropout rate is fixed at 0.6. λ is determined through grid-search and selected as 0.05 for Cora and PubMed, and 0.1 for CiteSeer. For robust operations, the reconstruction rank r for LRA is set to 20, and the threshold for NFS operation $\gamma = 0.01$. We maintain the power order V of VPO to be 2 for all datasets. In the generic search algorithm, we set population size $P = 50$, mutation size $s = 25$, mutation probability $p = 0.1$, crossover size $n = 25$, and optimal architecture number $k = 10$. After we finish the search, we continue to tune the hyper-parameters using hyperopt[5] with the following options to gain the best results:

- hidden size: {16, 32, 64, 128, 256}
- learning rate: {0.005, 0.01}
- weight decay: {5e-3, 1e-3, 5e-4, 1e-4}
- optimizer: {adam, adagrad}
- linear dropout: {0, 0.3, 0.5, 0.7}
- attention dropout: {0.5, 0.6, 0.7}

For all models, we train them on ogbn-arxiv for 500 epochs and on all the other datasets for 200 epochs. For vanilla GNNs, we set the learning rate as 0.005 using the Adam optimizer. Other hyper-parameters are kept the same as the original papers. For GNN-Jaccard, we tune the threshold for Jaccard similarly from {0.01,0.02,0.03,0.04,0.05}. For RGCN, the hidden size is tuned from {16, 32, 64, 128}. For Pro-GNN, we follow the original settings to fix the hidden size as 64 for Cora and CiteSeer. The results of Pro-GNN for the PubMed dataset are not available due to its high time complexity. In addition, its performances differ from the original paper due to the data split and whether to select the largest connected component. For GraphNAS, we keep the same setting as reported in the original paper and rerun it because their code leaks the test set of data. We ran all experiments on a single machine with a 16GB GeForce GTX TITAN X GPU.

[2]https://github.com/flyingdoog/PTDNet
[3]https://github.com/GraphNAS/GraphNAS
[4]https://github.com/THUMNLab/AutoGL
[5]https://github.com/hyperopt/hyperopt

E. Extension to feature attack

To extend our method for feature-based attacks, we redefine the robustness metric as

$$-\mathbb{E}_{\mathbf{X}'}\left[\frac{1}{N}\sum_{i=1}^{N} D_{KL}\left(f(\mathbf{X})_i||f(\mathbf{X}')_i\right)\right], \mathbf{X}' = \mathcal{T}_{\Delta}(\mathbf{X}). \tag{11}$$

We leave the validation of the experimental effectiveness for G-RNA's feature attack extension as our future work.