Appendix

A Extended Related Work

Network utility. Network utility refers to the system's quality to provide a specific service, for example, transmitting electricity in power networks and transmitting packages in routing networks. A popular metric for network utility is the network efficiency Latora & Marchiori (2003). In many previous work, despite that network resilience could be improved, the utility may dramatically drop at the same time Carchiolo et al. (2019); Schneider et al. (2011); Chan & Akoglu (2016); Buesser et al. (2011). This contradicts the idea behind improving network resilience and will be infeasible in real-world applications. Our goal is to enhance network resilience with moderate loss of network utility via edge rewiring.

Graph structure learning. Unlike the graph generation task which focuses on the quality of the generated graph, graph structure learning (GSL) aims to jointly learn an optimized graph structure and corresponding graph representations only for better performance on downstream tasks. Although conceptually related, GSL differs from graph generation since GSL mostly cares for the downstream task while graph generation focuses on the generated graphs Jin et al. (2020); Zhu et al. (2021). Currently, GSL relies on the existence of rich features to construct a graph, while there are generally no rich features in graph generation. Moreover, GSL cannot control the graph's node degree during graph optimization. We refer the interested readers to a survey of GSL Zhu et al. (2021) since our work is a constrained graph generation task unrelated to GSL.

Multi-views graph augmentation for GNNs. Multi-views graph augmentation is one efficient way to improve the expressive power of GNNs or combine domain knowledge, which is adapted based on the task's prior Hu et al. (2020). For example, GCC generates multiple subgraphs from the same ego network Qiu et al. (2020). GCA adaptively incorporates various priors for topological and semantic aspects of the graph You et al. (2020). Hassani & Khasahmadi (2020) contrasts representations from first-order neighbors and a graph diffusion. DeGNN Jin et al. (2020) was proposed as an automatic graph decomposition algorithm to improve the performance of deeper GNNs. These techniques rely on the existence of rich graph feature and the resultant GNNs cannot work well on graphs without rich features. In the resilience task, only the graph topological structure is available. Motivated by the calculation process of persistent homology Edelsbrunner & Harer (2008), we apply the filtration process to enhance the expressive power of GNNs for handling graphs without rich features.

Deep graph generation. Deep graph generation models learn the distribution of given graphs and generate more novel graphs. Some work use the encoder-decoder framework by learning latent representation of the input graph through the encoder and then generating the target graph through the decoder. For example, GCPN You et al. (2018) incorporates chemistry domain rules on molecular graph generation. GT-GAN Guo et al. (2018) proposes a GAN-based model on malware cyber-network synthesis. GraphOpt Trivedi et al. (2020) learns an implicit model to discover an underlying optimization mechanism of the graph generation using inverse reinforcement learning. GFlowNet learns a stochastic policy for generating molecules with the probability proportional to a given reward based on flow networks and local flow-matching conditions Bengio et al. (2021). Boosting network resilience in a degree-preserving way can be viewed as a constrained graph generation task. However, none of existing graph generation methods can generate desired graphs with the exact node degree preserving constraint, which is required by the resilience task.

B Implementation Details

This section provides the implementation details, including dataset and baseline setup.

B.1 Dataset

We first present the data generation strategies. Table 3 summarizes the statistics of each dataset. Synthetic datasets are generated using the Barabasi-Albert (BA) model (known as scale-free graphs) (Albert &

Table 3: Statistics of graphs used for resilience maximization. Both transductive and inductive settings ($\star$) are included. Consistent with our implementation, we report the number of edges by transforming undirected graphs to directed graphs. The edge rewiring has a fixed execution order. For the inductive setting, we report the maximum number of edges. The action space size of the edge rewiring is measured by $2|E|^2$.

Dataset	Node	Edge	Action Space Size	Train/Test	Setting
BA-15	15	54	5832	✗	Transductive
BA-50	50	192	73728	✗	Transductive
BA-100	100	392	307328	✗	Transductive
BA-500	500	996	1984032	✗	Transductive
BA-1000	1000	999	1996002	✗	Transductive
EU	217	640	819200	✗	Transductive
p2p-Gnutella05	400	814	1325192	✗	Transductive
p2p-Gnutella09	300	740	1095200	✗	Transductive
BA-10-30 ($\star$)	10-30	112	25088	1000/500	Inductive
BA-20-200 ($\star$)	20-200	792	1254528	4500/360	Inductive

Barabási, 2002), with the graph size varying from $|N|$=10 to $|N|$=1000. During the data generation process, each node is connected to two existing nodes for graphs with no more than 500 nodes, and each node is connected to one existing node for graphs with near 1000 nodes. BA graphs are chosen since they are vulnerable to malicious attacks and are commonly used to test network resilience optimization algorithms (Bollobás & Riordan, 2004). We test the performance of ResiNet on both transductive and inductive settings.

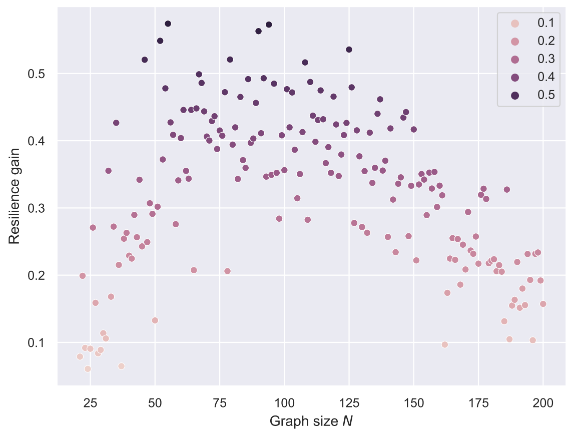
- **Transductive setting.** The algorithm is trained and tested on the same network.
  - Randomly generated synthetic BA networks, denoted by BA-m, are adopted to test the performance of ResiNet on networks of various sizes, where $m \in \{15, 50, 100, 500, 1000\}$ is the graph size.
  - The Gnutella peer-to-peer network file sharing network from August 2002 (Leskovec et al., 2007; Ripeanu et al., 2002) and the real EU power network (Zhou & Bialek, 2005) are used to validate the performance of ResiNet on real networks. The random walk sampling strategy is used to derive a representative sample subgraph with hundreds of nodes from the Gnutella peer-to-peer network (Leskovec & Faloutsos, 2006).
- **Inductive setting.** Two groups of synthetic BA networks denoted by BA-m-n are randomly generated to test ResiNet's inductivity, where m is the minimal graph size, and n indicates the maximal graph size. We first randomly generate the fixed number of BA networks as the training data to train ResiNet and then evaluate ResiNet's performance directly on the test dataset without any additional optimization.

B.2 Baseline Setup

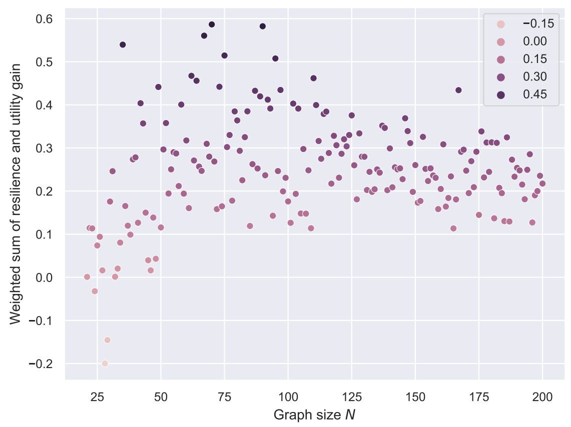
All baselines share the same action space with ResiNet and use the same action masking strategy to block invalid actions as ResiNet does. The maximal objective evaluation is consistent for all algorithms. Other settings of baselines are consistent with the default values in their paper. The early-stopping strategy is used for baselines, which means that the search process terminates if no improvement is obtained in successive 1000 objective function calling trials. All algorithms are repeated for 3 random seeds using default hyper-parameters.

C Extended Experimental Results

In this section, we present additional experimental results to show that ResiNet generalizes to unseen graphs, different utility and resilience metrics.



(a) Inductivity on resilience



(b) Inductivity on resilience and utility

Figure 9: The inductive ability of ResiNet on the test dataset (BA-20-200) when optimizing (a) network resilience and (b) the combination of resilience and utility.

C.1 The Effect of Filtration Order K on FireGNN

In this section, we report the ratio of the remaining edges in subgraphs versus different filtration order K in Table 5 and visualize it in Figure 8. The maximum filtration order K of FireGNN of each dataset is summarized in Table 6. Table 6 demonstrates that the maximum filtration order K is nearly around the half size of the graph since we gradually remove the node with the largest degree in the filtration process.

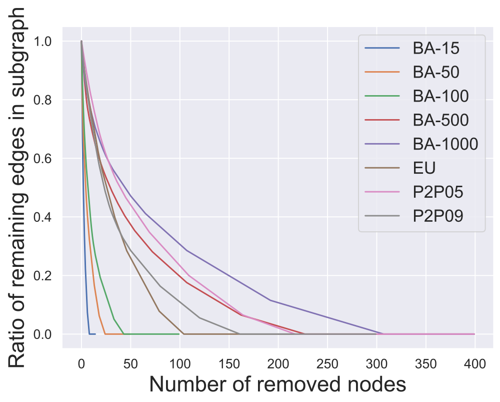


Figure 8: Ratio of the remaining edges in subgraphs versus different filtration order K.

C.2 Inductivity on Larger Datasets

To demonstrate that ResiNet can learn from networks to accommodate different utility and resilience metrics, we conduct experiments based on BA-15 using multiple resilience and utility metrics. The Pareto points shown in Figure 7 denote the optimum under different objectives on BA-15, implying that ResiNet can obtain the approximate Pareto frontier. Surprisingly, the initial gain of resilience (from around 0.21 to around 0.24) is obtained without loss of the utility, which incentivizes almost every network to conduct such optimization to some extent when feasible. More results are included in Appendix C.3 and the optimized network structures are visualized in Figure 10 and Figure 11.

Even with limited computational resources, armed with the autoregressive action space and the power of FireGNN, ResiNet can be trained fully end-to-end on graphs with thousands of nodes using RL. We demonstrate the inductivity of ResiNet on graphs of different sizes by training ResiNet on the BA-20-200 dataset, which consists of graphs with the size ranging from 20 to 200, and then report its performance on directly guiding the edges selections on unseen test graphs. The filtration order K is set to 1 for the computational limitation. As shown in Figure 9, we can see that ResiNet has the best performance for $N \in [70, 100]$. The degrading performance with the graph size may be explained by the fact that larger graphs require a larger filtration order for ResiNet to work well. A more stable performance improvement of ResiNet is observed with the increment of graph size when trained to optimize network resilience and utility simultaneously, and ResiNet possibly finds a strategy to balance these two metrics.

Table 4: The effect of the filtration order K on ResiNet in improving network resilience (percentage).

K	BA-15	BA-50	BA-100	EU
0	11.8 ± 2.7	0 ± 0	0 ± 0	6.1 ± 4.2
1	17.6 ± 0	49.6 ± 2.3	74.9 ± 0.8	54.5 ± 0.4
2	17.6 ± 0	51.0 ± 0.1	76.3 ± 1.2	57.4 ± 1.6
3	17.6 ± 0	55.7±2.3	73.1 ± 0.9	54.9 ± 0.9

Table 5: The ratio of remaining edges in subgraphs versus the filtration order K.

K	BA-15	BA-50	BA-100	BA-500	BA-1000	EU	P2P-Gnutella05	P2P-Gnutella09
1	0.6667	0.7708	0.8571	0.9538	0.9479	0.9625	0.9779	0.9608
2	0.4815	0.6667	0.7602	0.9096	0.9139	0.9313	0.957	0.927
3	0.3333	0.5938	0.6837	0.8665	0.8809	0.9031	0.9361	0.8973
4	0.2222	0.5208	0.6224	0.8313	0.8559	0.8781	0.9152	0.8703
5	0.1481	0.4583	0.5663	0.8002	0.8348	0.8562	0.8956	0.8446
6	0.0370	0.4062	0.5255	0.7741	0.8158	0.8344	0.8771	0.8189
7	0	0.3646	0.4847	0.754	0.7998	0.8125	0.8587	0.7959
8	0	0.3229	0.4439	0.7359	0.7848	0.7906	0.8403	0.773
9	0	0.2917	0.4031	0.7199	0.7698	0.7719	0.8231	0.75

Table 6: Maximum filtration order K of each dataset.

Dataset	BA-15	BA-50	BA-100	BA-500	BA-1000	EU	P2P-Gnutella05	P2P-Gnutella09
Size (node)	15	50	100	500	1000	217	400	300
Maximum of K	7	23	42	226	306	103	216	160

C.3 Learning to Balance Different Utility and Resilience Metrics

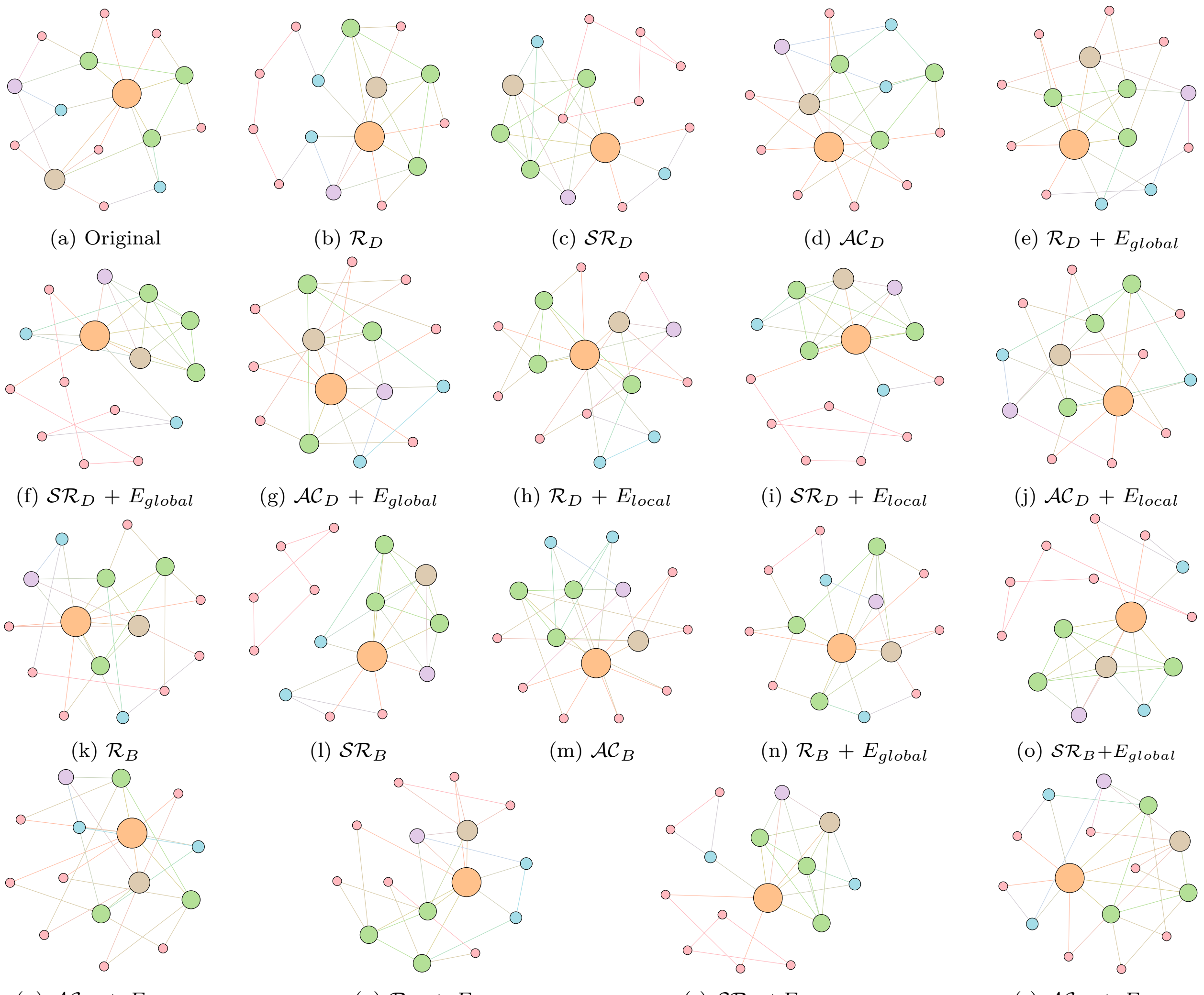
As shown in Figure 10, we conduct extensive experiments on the BA-15 network to demonstrate that ResiNet can learn to optimize graphs with different resilience and utility metrics and to defend against other types of attacks besides the node degree-based attack, such as the node betweenness-based attack.

Table 7 records the improvements in percentage of ResiNet for varying objectives on the BA-15 dataset. As visualized in Figure 10, ResiNet is not limited to defend against the node degree-based attack (Figure 10 (b)-(j)) and also learns to defend against the betweenness-based attack (Figure 10 (k)-(s)). Total three resilience metrics are used, with $\mathcal{R}$ denoting the graph connectivity-based resilience metric, $\mathcal{SR}$ being the spectral radius and $\mathcal{SR}$ representing the algebraic connectivity. Total two utility metrics are adopted, including the global efficiency E_{global} and the local efficiency E_{local}. Not surprisingly, the optimized network with an improvement of about 3.6% for defending the betweenness-based attack also has a higher resilience (around 7.8%) against the node-degree attack. This may be explained as the similarity between node degree and betweenness for a small network.

C.4 Performance Comparisons Under a Large Rewiring Budget

In this section, we present the resilience improvement and the required number of edge rewiring of each algorithm under a large rewiring budget of 200. The running speed is presented to compare the running time efficiency of each algorithm.

As shown in Table 8, traditional methods improve the network resilience significantly compared to ResiNet under a large rewiring budget of 200. However, traditional methods are still undesired in such a case since a solution with a large rewiring budget is not applicable in practice due to the vast cost of adding many new edges into a real system. For example, the actual number of rewiring budget for EA is hard to calculate since it is a population-based algorithm, so it is omitted in Table 8. All baselines adopt the early-stopping strategy that they will terminate if there is no positive resilience gain in a successive 1000 steps.



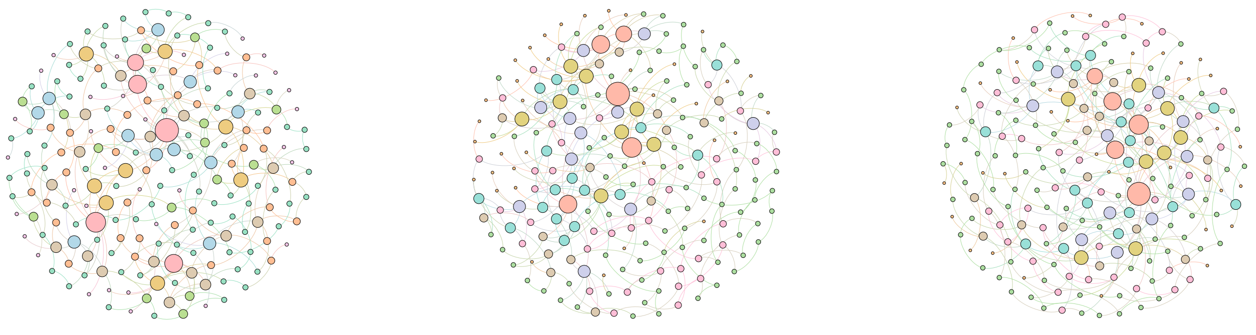
(a) Original (b) $\mathcal{R}_D$ (c) $\mathcal{SR}_D$ (d) $\mathcal{AC}_D$ (e) $\mathcal{R}_D$ + E_{global}

(f) $\mathcal{SR}_D$ + E_{global} (g) $\mathcal{AC}_D$ + E_{global} (h) $\mathcal{R}_D$ + E_{local} (i) $\mathcal{SR}_D$ + E_{local} (j) $\mathcal{AC}_D$ + E_{local}

(k) $\mathcal{R}_B$ (l) $\mathcal{SR}_B$ (m) $\mathcal{AC}_B$ (n) $\mathcal{R}_B$ + E_{global} (o) $\mathcal{SR}_B$+E_{global}

(p) $\mathcal{AC}_B$ + E_{global} (q) $\mathcal{R}_B$ + E_{local} (r) $\mathcal{SR}_B$+E_{local} (s) $\mathcal{AC}_B$ + E_{local}

Figure 10: The resilience maximization on the BA-15 dataset with 15 nodes and 27 edges with (a) original network, (b)-(j) results of defending the node degree-based attack with different combinations of resilience and utility, and (k)-(s) results of defending against the node betweenness-based attack with varying combinations of resilience and utility. For three resilience metrics, $\mathcal{R}$ denotes the graph connectivity-based resilience metric; $\mathcal{SR}$ is the spectral radius; $\mathcal{SR}$ represents the algebraic connectivity. For two utility metrics, E_{global} denotes the global efficiency, and E_{local} is the local efficiency.



(a) Original EU network (b) $\mathcal{R}$ (c) E_{global}

Figure 11: Visualizations of the original EU network and optimized networks using ResiNet with different objectives: $\mathcal{R}$ means the connectivity-based resilience measurement and E_{global} is the global efficiency.

Table 7: Performance gain (in percentage) of ResiNet in optimizing varying objectives on the BA-15 network. All objectives are optimized with the same hyper-parameters, which means that we did not tune hyper-parameters for objectives except for R_D.

Objective	Gain (%)	Objective	Gain(%)
$\mathcal{R}_D$	35.3	$\mathcal{R}_B$	14.6
$\mathcal{SR}_D$	15.3	$\mathcal{SR}_B$	15.3
$\mathcal{AC}_D$	48.2	$\mathcal{AC}_B$	43.2

Table 8: Resilience optimization algorithm under the fixed maximal rewiring number budget of 200. Entries are in the format of $X(Y)$, where 1) X: weighted sum of the graph connectivity-based resilience and the network efficiency improvement (in percentage); 2) Y: required rewiring number. Results are averaged over 3 runs and best performance is in bold.

Method	α	BA-15	BA-50	BA-100	BA-500	BA-1000	EU	P2P-Gnutella05	P2P-Gnutella09
HC	0	26.8 (10.0)	52.1 (47.0)	76.9 (97.3)	45.8 (200)	302.5 (200)	71.9 (152.7)	37.5 (193.3)	40.2 (137.7)
	0.5	18.6 (11.3)	43.1 (62.7)	56.9 (121)	30.0 (200)	66.3 (200)	63.2 (200)	27.7 (200)	34.7 (196.3)
SA	0	26.8 (20)	49.7 (59.0)	84.5 (119.7)	43.2 (200)	271.8 (200)	73.5(160.3)	37.1 (200)	37.2 (134)
	0.5	17.8 (21)	41.1 (79.7)	57.7 (127.7)	31.4 (200)	64.9 (200)	62.8 (200)	37.1 (200)	35.2 (200)
Greedy	0	23.5 (6)	48.6 (13)	64.3 (20)	✗	✗	0.5 (3)	✗	✗
	0.5	5.3 (15)	34.7 (13)	42.7 (20)	✗	✗	0.3 (3)	✗	✗
EA	0	35.3 (✗)	50.2 (✗)	61.9 (✗)	9.9 (200)	174.1 (200)	66.2 (✗)	2.3 (200)	0 (200)
	0.5	27.1 (✗)	38.3 (✗)	46.6 (✗)	6.8 (200)	18.7 (200)	58.4 (✗)	3.2 (200)	0 (200)
DE-GNN-RL	0	13.7 (2)	0 (1)	0 (1)	1.6 (20)	41.7 (20)	9.0 (20)	2.2 (20)	0 (1)
	0.5	10.9 (2)	0 (1)	0 (1)	2.7 (20)	20.1 (14)	2.1 (20)	0 (1)	1.0 (20)
k-GNN-RL	0	13.7 (2)	0 (1)	0 (1)	0 (1)	8.8 (20)	4.5 (20)	-0.2 (20)	0 (1)
	0.5	6.3 (2)	0 (1)	0 (1)	0 (20)	-24.9 (20)	4.8 (20)	-0.1 (20)	0 (1)
ResiNet	0	**35.3** (6)	**61.5** (20)	**70.0** (20)	**10.2** (20)	172.8 (20)	**54.2** (20)	**14.0** (20)	**18.6** (20)
	0.5	**26.9** (20)	**53.9** (20)	**53.1** (20)	**15.7** (20)	**43.7** (20)	**51.8** (20)	**12.4** (20)	**15.1** (20)

Table 9 indicates that the time it takes for the benchmark algorithm to solve the problem usually increases as the test data set size increases. In contrast, our proposed ResiNet is suitable for testing on a large dataset once trained.

Table 9: Running speed (in second) of the resilience optimization algorithm under the fixed maximal rewiring number budget. Entries are in the format of $X(Y)$, where 1) X: speed under the budget of 20; 2) Y: speed under the budget of 200 . ✗ means that the result is not available at a reasonable time. Results are averaged over 3 runs and best performance is in bold.

Method	α	BA-15	BA-50	BA-100	BA-500	BA-1000	EU	P2P-Gnutella05	P2P-Gnutella09
HC	0	1.0 (1.0)	1.1 (6.4)	1.3 (22.2)	21.9 (354.1)	80.3 (1288.3)	3.1 (94.2)	15.3 (358.1)	4.5 (89.1)
	0.5	1.5 (11.5)	1.1 (12.8)	2.0 (49.0)	40.9 (589.5)	148.7 (2603.7)	5.3 (193.7)	24.7 (462.8)	7.0 (190.6)
SA	0	0.5 (0.5)	0.3 (6.6)	0.6 (22.6)	12.2 (313.0)	45.7 (1051.8)	2.4 (91.2)	10.8 (286.4)	2.6 (89.4)
	0.5	0.7 (1.7)	0.7 (13.2)	1.7 (47.5)	33.9 (568.9)	99.8 (2166.3)	5.0 (193.5)	23.9 (454.5)	6.3 (188.5)
Greedy	0	0.2 (6.0)	34.1 (34.5)	766.3 (✗)	✗	✗	3061.7 (✗)	✗	✗
	0.5	0.7 (0.7)	64.1 (65.4)	1478.9 (✗)	✗	✗	6192.6 (✗)	✗	✗
EA	0	0.01 (✗)	0.1 (✗)	1.6 (✗)	2.5 (✗)	10.3 (✗)	0.2 (✗)	1.6 (✗)	0.4 (✗)
	0.5	0.01 (✗)	0.1 (✗)	0.8 (✗)	4.7 (✗)	15.0 (✗)	0.4 (✗)	3.0 (✗)	0.8 (✗)
DE-GNN-RL	0	0.1 (✗)	0.1 (✗)	0.1 (✗)	14.9 (✗)	70.3 (✗)	3.6 (✗)	8.7 (✗)	0.5 (✗)
	0.5	0.1 (✗)	0.1 (✗)	0.2 (✗)	13.7 (✗)	60.9 (✗)	4.5 (✗)	1.0 (✗)	6.7 (✗)
k-GNN-RL	0	0.02 (✗)	0.03 (✗)	0.07 (✗)	1.3 (✗)	56.5 (✗)	2.6 (✗)	8.2 (✗)	0.5 (✗)
	0.5	0.02 (✗)	0.04 (✗)	0.08 (✗)	18.3 (✗)	76.1 (✗)	3.6 (✗)	11.5 (✗)	0.6 (✗)
ResiNet	0	0.5 (✗)	1.8 (✗)	2.2 (✗)	17.5 (✗)	66.8 (✗)	4.5 (✗)	14.7 (✗)	9.3 (✗)
	0.5	0.5 (✗)	1.9 (✗)	2.4 (✗)	18.0 (✗)	67.5 (✗)	5.2 (✗)	15.0 (✗)	10.3 (✗)

C.5 Inspection of Optimized Networks

Moreover, to provide a deeper inspection into the optimized network structure, we take the EU power network as an example to visualize its network structure and the optimized networks given by ResiNet with different objectives. Compared to the original EU network, Figure 11 (b) is the network structure obtained by only optimizing the graph connectivity-based resilience. We can observe a more crowded region on the left, consistent with the "onion-like" structure concluded in previous studies. If we consider the combination gain of both resilience and utility, we observe a more compact clustering "crescent moon"-like structure as shown in Figure 11 (c).

D Deep analysis of why regular GNNs fail in the resilience task

It is well-known that GNNs generally work well for graphs with rich features. Unluckily, the graph network in the resilience task has no node/edge/graph feature, with only the topological structure available. No rich feature means that the output of the GNNs is not distinguishable, and then it is difficult for the RL agent to distinguish different vertices/edges, causing large randomness in the output of the policy. This may cause the rewiring process to alternate between two graphs, forming an *infinite loop*. And we suspect that this infinite loop failure may explain the poor empirical performance of optimizing network resilience by selecting edges using existing GNNs and reinforcement learning (RL). The infinite loop failure is presented as follows.

Consider the graph G_t with N nodes and containing two edges AB and CD. The agent selects AB and CD for rewiring, leading to G_{t+1} with news edges AC and BD. A frequent empirical failure of regular GNNs for the robustness task is the *infinite action backtracking* phenomenon. The agent would select AC and BD at step $t+1$, returning back to G_t and forming a cycled loop between G_t and G_{t+1}. Formally, the infinite loop is formulated as

$$\begin{aligned}((A,B),(C,D)) &= \underset{i,j,m,n\in 1:N}{\operatorname{argmax}} \operatorname{SIM}\left(((h_t^i,h_t^j),(h_t^m,h_t^n)),h_{G_t}\right)\\ ((A,C),(B,D)) &= \underset{i,j,m,n\in 1:N}{\operatorname{argmax}} \operatorname{SIM}\left(((h_{t+1}^i,h_{t+1}^j),(h_{t+1}^m,h_{t+1}^n)),h_{G_{t+1}}\right),\end{aligned}$$

where SIM is a similarity metric, h_t^i and h_{G_t} are embeddings of node i and graph G_t at step t, and (A,B) is one edge.

Table 10 compares and summarizes different graph related tasks' characteristics. We can see that *the resilience task is more challenging from many aspects*. No prior rule like action masking or negative penalty can be used to avoid selecting visited nodes as in TSP. For the resilience task, all previously visited edges are also possibly valid to be selected again, resulting in insufficient training signals.

The desired GNN model should not depend on rules like action masking to distinguish edge and graph representations for graphs with little node features. Our proposed FireGNN fulfills these requirements to obtain proper training signals. FireGNN has a distinct expressive power and learns to create more meaningful and distinguishable features for each edge. FireGNN is not a simple aggregation of higher-order information of a static graph. It was inspired by homology filtration and the multi-view graph augmentation. Persistence homology motivates us to aggregate more distinct node features by observing how the graph evolves towards being empty, leading to more distinct and meaningful features for each node/edge, thus avoiding the infinite loop. Extensive experimental results in Table 1 validate the necessity and effectiveness of FireGNN. Existing GNNs perform worse while FireGNN performs well.

Table 10: Characteristics of different graph related tasks.

Approach	**Task**	**RL component**			**Problem**	**Training & Inference**			
		State	**Action**	**Reward**	**Complexity**	**Size Extrapolate**	**Encoder**	**Action Masking**	**Scalability**
S2V-DQN Khalil et al. (2017)	MVC	node level	add node to subset	-1	$\mathcal{O}(N)$	✗	S2V	✓	500
	Max-Cut	node level	add node to subset	change in cut weight	$\mathcal{O}(N)$	✗	S2V	✓	300
	TSP	node level	add node to tour	change in tour cost	$\mathcal{O}(N)$	✗	S2V	✓	300
Local search Hudson et al. (2022)	TSP	edge level	relocate node in tour	global regret	$\mathcal{O}(N^2)$	✓	GNN	✓	100
RNN-RL Bello et al. (2016)	TSP	node level	add node to tour	change in tour cost	$\mathcal{O}(N)$	✗	RNN	✓	100
GNN-RL Joshi et al. (2022)	TSP	node level	add node to tour	change in tour cost	$\mathcal{O}(N)$	✓	GNN	✓	50
Attention-RL Kool et al. (2018)	TSP	node level	add node to tour	change in tour cost	$\mathcal{O}(N)$	✗	Attention	✓	100
	VRP	node level	add node to tour	change in tour cost	$\mathcal{O}(N)$	✗	Attention	✓	100
Local search Böther et al. (2022)	MIS	node level	add node to subset	change in IS size	$\mathcal{O}(N)$	✓	GNN	✓	800
ResiNet	Resilience	***graph level***	***edge rewiring***	change in resilience and utility	$\mathcal{O}(N^4)$	✓	***FireGNN***	✗	***1000***