Enhancing Network by Reinforcement Learning and Neural Confined Local Search

Qifu Hu , **Ruyang Li*** , **Qi Deng** , **Yaqian Zhao** and **Rengang Li**
Inspur Electronic Information Industry Co., Ltd
{huqifu, liruyang, dengqi01, zhaoyaqian}@inspur.com, lirengang.hsslab@gmail.com

Abstract

It has been found that many real networks, such as power grids and the Internet, are non-robust, i.e., attacking a small set of nodes would cause the paralysis of the entire network. Thus, the Network Enhancement Problem (NEP), i.e., improving the robustness of a given network by modifying its structure, has attracted increasing attention. Heuristics have been proposed to address NEP. However, a hand-engineered heuristic often has significant performance limitations. A recently proposed model solving NEP by reinforcement learning has shown superior performance than heuristics on in-distribution datasets. However, their model shows considerably inferior out-of-distribution generalization ability when enhancing networks against the degree-based targeted attack. In this paper, we propose a more effective model with stronger generalization ability by incorporating domain knowledge including measurements of local network structures and decision criteria of heuristics. We further design a hierarchical attention model to utilize the network structure directly, where the query range changes from local to global. Finally, we propose neural confined local search (NCLS) to realize the effective search of a large neighborhood, which exploits a learned model to confine the neighborhood to avoid exhaustive enumeration. We conduct extensive experiments on synthetic and real networks to verify the ability of our models.

1 Introduction

Modern society has become increasingly dependent on the availability of networked systems, such as power grids, communication networks, traffic networks, and so on. Network robustness, which measures the ability of networks to maintain reliable operation in the presence of attacks [Ellens and Kooij, 2013], has attracted more and more attention because for many real-world networks, failures of a small number of nodes may cause the malfunction of the entire network [Albert *et al.*, 2000; Holme *et al.*, 2002; Wang and Rong, 2009]. The underlying reason is that robustness is not sufficiently considered when designing these networks. It has been found that the robustness of a network is determined by its structure [Albert *et al.*, 2000]. Thus a natural question is: Can we improve the robustness of a given network as much as possible by slightly modifying its structure? We call this problem the *Network Enhancement Problem (NEP)*.

This paper considers a specific set of NEP, as in [Darvariu *et al.*, 2021], while our methods apply in more generalized settings. First, we specify the network robustness of a given network G against an attack algorithm $\mathcal{A}$ as $R(G) = E_{\xi\sim\mathcal{A}(G)}\, p(G,\xi)$[1], where $p(G,\xi)$ is the minimum fraction of nodes that must be removed from G in the order ξ to disconnect it. Note $\mathcal{A}$ is required in the definition because even the same network may show different levels of robustness under different attack algorithms. For instance, a scale-free network [Barabási and Albert, 1999; Faloutsos *et al.*, 1999] is robust to random attack and vulnerable to targeted attack [Albert *et al.*, 2000]. Second, we consider adding a certain number of links in a network to improve its robustness.

Existing methods can be divided into two categories: heuristic methods [Beygelzimer *et al.*, 2005; Sánchez Martínez, 2009; Wang and Van Mieghem, 2010; Wang *et al.*, 2014; Li *et al.*, 2018] and the Reinforcement Learning (RL) based method [Darvariu *et al.*, 2021]. In the next section, we will review heuristic methods in detail. As we will see, developing a heuristic algorithm often requires extensive expert domain knowledge. In addition, the performance of a hand-engineered heuristic is often limited by the designer's intuition and experience. In contrast, the RL based method, named NEP-DQN [Darvariu *et al.*, 2021], models the problem in the Markov Decision Process (MDP) framework and learns end-to-end solutions with DQN [Mnih *et al.*, 2013]. With NEP-DQN, complex decision rules that are hard to specify by hand are automatically discovered through trial and error.

Although NEP-DQN performs better on In-Distribution (ID) datasets than hand-engineered heuris-

*Corresponding author.

[1] A comprehensive discussion of various robustness measures can be referred to [Liu *et al.*, 2017].

tics , it lacks the ability to generalize well for Out-Of-Distribution (OOD) scenarios when enhancing networks against the degree-based targeted attack, i.e., removing nodes from a network in decreasing order of their degree. Towards a more effective decision model with stronger generalization ability, we incorporate domain knowledge to improve both the state and action representations, which are crucial in the decision-making process. In particular, we introduce measurements from the field of complex networks, such as various centrality measures [Barrat *et al.*, 2004] to characterize the local neighborhood structure of a node. We also use decision criteria of hand-engineered heuristics, such as degree product, distance, and algebraic distance [Fiedler, 1973] to characterize the node pairs. To capture the global position of nodes in the network, we assign a two-dimensional coordinate to each node using the eigenvectors of the graph Laplacian such that adjacent nodes are mapped to neighboring points in the coordinate space. We solve NEP by adapting the Attention Model (AM) [Kool *et al.*, 2018], i.e., adding a feature extraction module to AM. We call the adapted model NEP-AM. NEP-AM uses the network structure information indirectly through the introduced features. To directly utilize the network structure information, we add a structure-aware attention module [Shi *et al.*, 2020] that only allows nodes to query local neighbors to NEP-AM. Thus the resulting model is a hierarchical attention model, i.e., the query range changes from local to global. We name it NEP-HAM.

Let l be the edge budget. A solution can be represented as a node sequence of length $2l$. The k-opt local search [Aarts *et al.*, 2003] is generally used to improve a solution further. The solution is iteratively improved until a local optimum is achieved. In each iteration, we search the *k-exchange neighborhood* for a better candidate to replace the current solution. The k-exchange neighborhood contains feasible solutions that are obtained by exchanging k nodes of the current solution. The size of the k-exchange neighborhood is $O(\binom{2l}{k}n^k)$, where n is the number of nodes of the network. In practice, k is usually set as a small value so that the algorithm can finish in a reasonable amount of time [Ma *et al.*, 2019]. However, removing this constraint can lead to better solutions by exploring a larger neighborhood. To this end, we propose Neural Confined Local Search (NCLS). In particular, we define the *restricted k-exchange neighborhood* by restricting the selection of k nodes to be consecutive. The size of the restricted k-exchange neighborhood is $O(2ln^k)$. However, examining all candidates in the restricted k-exchange neighborhood is also infeasible, especially when k is large. Hence, we use a learned policy π_θ to generate a *neural confined neighborhood* that includes high-quality candidates decoded from π_θ. The neural confined neighborhood is a subset of the restricted k-exchange neighborhood and of size $O(2l\eta)$, where η is a hyperparameter that balances effectiveness and efficiency. NCLS effectively searches the restricted k-exchange neighborhood by only examining candidates in the neural confined neighborhood, which will be illustrated by experiments.

In the experiment, we compare our models with heuristics and NEP-DQN on synthetic and real networks using two attack algorithms. We evaluate the ID and OOD generalization ability on synthetic networks, and the active search [Bello *et al.*, 2016] on real networks. For NCLS, we set large values for k to test whether it can realize the effective search of large neighborhoods. Additionally, we visualize the validation results during the entire training process, where we study the impact of node coordinates. Our results show that: **1)** NEP-AM outperforms baselines in almost all cases. In particular, it has a much stronger OOD generalization ability than NEP-DQN; **2)** NEP-HAM performs better than or comparable to NEP-AM in most cases; **3)** NCLS can improve the quality of solutions by effectively searching large neighborhoods; **4)** Node coordinates are crucial for NEP-AM and provide additional information for NEP-HAM.

To summarize, our contributions are as follows.

1) Towards a more effective decision model with stronger generalization ability, we propose NEP-AM, where domain knowledge is incorporated to improve both the state and action representations. To directly utilize the network structure, we propose NEP-HAM by adding a structure-aware attention module to NEP-AM.
2) To further improve the solving of NEP, we propose NCLS, which exploits a learned policy to effectively search a large neighborhood. Specially, it realizes the effective search of the restricted k-exchange neighborhood of size $O(2ln^k)$ by examining its subset of size $O(2l\eta)$, i.e., the neural confined neighborhood.
3) We demonstrate the superiority of NEP-AM/HAM through extensive experiments, including the evaluation of the ID, OOD generalization ability, and active search. We verify NCLS's ability to search larger neighborhoods. In addition, we compare validation curves and assert the importance of providing node coordinates.

2 Related Work

Network robustness is a wide-ranging area of research that began around 20 years ago. NEP is one of the most important research topics in network robustness and has attracted increasing attention due to its applicability in real-world applications. There are many works addressing NEP by designing heuristic algorithms [Beygelzimer *et al.*, 2005; Sánchez Martínez, 2009; Wang and Van Mieghem, 2010; Wang *et al.*, 2014; Li *et al.*, 2018]. These methods insert an edge between the current best unconnected pair of nodes, where the quality of an unconnected pair of nodes is measured by some criterion, and repeat this process until the end. Let L be the graph Laplacian and $\lambda_2(L)$ be the second smallest eigenvalue of L. Criteria in [Beygelzimer *et al.*, 2005; Sánchez Martínez, 2009; Wang and Van Mieghem, 2010; Li *et al.*, 2018] are designed to maximize $\lambda_2(L)$, which is called *algebraic connectivity* in the network jargon. The rationale lies in that the algebraic connectivity is a lower bound of *node connectivity* [Fiedler, 1973], reflecting the robustness of a network under the strongest attack algorithm, which involves identifying the minimum number of nodes that must be removed to disconnect the network. Since a network G's smallest degree provides an upper bound of its algebraic connectivity [Fiedler, 1973], the smallest degree has been used

as a surrogate objective to maximize $\lambda_2(L)$. For instance, in each step, a link is added to unconnected nodes with lowest degrees [Beygelzimer *et al.*, 2005] or between the minimum degree node and a random chosen node [Sánchez Martínez, 2009]. To optimize $\lambda_2(L)$ directly, a useful property is that the improvement of $\lambda_2(L)$ by inserting (u, v) can be approximated by $|e_u - e_v|^2$, where e is the eigenvector of L corresponding to $\lambda_2(L)$ [Ghosh and Boyd, 2006]. To exploit this property, in each step, [Wang and Van Mieghem, 2010] adds a link between unconnected u, v with the maximal algebraic distance $d^A(u, v) = |e_u - e_v|$. [Li *et al.*, 2018] finds that in a special class of graphs, two nodes maximizing the algebraic distance are the ones with minimum local centrality and maximum distance. They thus propose a criterion based on their finding. Other than algebraic connectivity, the criterion in [Wang *et al.*, 2014] is built based on effective resistance $\Omega(u, v) = (\hat{L}^{-1})_{v,v} + (\hat{L}^{-1})_{u,u} - 2(\hat{L}^{-1})_{v,u}$, where $\hat{L}^{-1}$ is the pseudo inverse of L [Ellens *et al.*, 2011]. Intuitively, for a node pair with few alternative and longer paths, its effective resistance will be larger than that with a high number of alternative and short paths. Thus, the unconnected nodes with maximum effective resistance, i.e., $\arg\max_{u,v} \Omega(u, v)$, are selected in [Wang *et al.*, 2014]. From the above introduction, we can see that the design of a heuristic algorithm is a complex task requiring extensive domain knowledge. In addition, the performance of a heuristic is often limited by the designer's intuition and experience.

In recent years, machine learning techniques are increasingly being used to solve combinatorial optimization problems, such as Travelling Salesman Problem [Vinyals *et al.*, 2015; Bello *et al.*, 2016; Deudon *et al.*, 2018; Kool *et al.*, 2018], KnapSack [Bello *et al.*, 2016], Vehicle Routing Problem [Nazari *et al.*, 2018; Kool *et al.*, 2018], Maximum Cut [Khalil *et al.*, 2017], and so on. These methods can be divided into two categories: supervised learning based and RL based. However, supervised learning does not apply to NEP since the optimal labels are unavailable. [Darvariu *et al.*, 2021] proposes an RL based method NEP-DQN, which uses a neural network $Q_\theta(s, v)$ to estimate the optimal action-value function. In particular, a graph neural network [Dai *et al.*, 2016] is employed to extract the representations of state and action, where the raw feature of a node is a binary variable indicating whether the node is an edge stub. These representations are used to calculate the action-value function. θ is trained based on a variant of DQN [Mnih *et al.*, 2013]. After training, the node chosen under state s is given by $\pi(s) = \arg\max_v Q_\theta(s, v)$. However, the domain knowledge is completely ignored in NEP-DQN, which causes its poor OOD generalization ability. In contrast, our proposed methods exploit domain knowledge to improve the representations of state and action, which are essential for the decision process. Previous work [Hu *et al.*, 2019] has also considered the confine of the neighborhood of 1-opt local search. However, their method can not be directly generalized to larger k. Some studies, including [Wu *et al.*, 2021; Ma *et al.*, 2021], focus on using RL to learn improvement heuristics directly, rather than using pre-trained models to guide classical improvement heuristics as in our work.

3 Methodology

In this section, we first describe the decision process and then present our proposed NEP-AM and NEP-HAM. Finally, we introduce the NCLS.

3.1 The Decision Process and Policy Learning

Let G be the network to enhance, $V = \{1, 2 \cdots, n\}$ be the node set consisting of n nodes. Given the edge budget l, there are $2l$ steps in the decision procedure. At each step, the agent observes a state, decides which node to choose, and then acts to change the state. Specifically, for $0 \le t \le l-1$, the state s_{2t} is a tuple $(G_{2t}, \emptyset)$, where $G_{2t} = G \oplus \{(v_{2i}, v_{2i+1})\}_{i=0}^{t-1}$ is obtained by adding edges $\{(v_{2i}, v_{2i+1})\}_{i=0}^{t-1}$ into G, $v_0, v_1, \cdots, v_{2t-1}$ are nodes chosen in previous steps, $\emptyset$ means there is no edge stub. After observing s_{2t}, the agent chooses an edge stub v_{2t} and enters a new state $s_{2t+1} = (G_{2t+1}, \{v_{2t}\})$, where $G_{2t+1} = G_{2t}$. Then the agent chooses v_{2t+1} and insert (v_{2t}, v_{2t+1}) into G_{2t+1} so that $G_{2t+2} = G_{2t+1} \oplus \{(v_{2t}, v_{2t+1})\}$. The agent repeats this process to the final step and receives a reward as the robustness improvement of G_{2l} against G.

Let π_θ be a parameterized policy, where θ refers to the parameters of a model. We learn π_θ by maximizing the expected reward:

$$\mathcal{L}(\theta) = \mathsf{E}_{\tau \sim \pi_\theta}[R(\tau)] \tag{1}$$

where $\tau = (s_0, v_0, \cdots, s_{2l-1}, v_{2l-1})$ is a trajectory. $R(\tau)$ is the robustness improvement of the final network G_{2l} against G. We use the REINFORCE [Williams, 1992] to estimate the gradient and introduce a baseline $b(s)$ to reduce the variance:

$$\nabla \mathcal{L}(\theta) \sim (R(\tau) - b(s)) \sum_{i=0}^{2l-1} \nabla_\theta \log \pi_\theta(v_i | s_i) \tag{2}$$

where τ is sampled from π_θ and '$\sim$' means the right-hand side is an unbiased estimator of the gradient.

3.2 NEP-AM

NEP-AM consists of three parts: a feature extraction module that extracts node features, an encoder that generates node and state representations from the extracted features, and a decoder that makes decisions based on the node and state representations. We discuss these three parts in detail below.

Feature Extraction Module

High-quality state and action representations help improve the policy's performance. We introduce the following two kinds of features[2]. The first type, aiming to characterize the local structure near a node, includes centrality measures (the degree centrality and the average neighbor degree) and the clustering coefficient. Centrality measures, with a strong ability to distinguish nodes, have been widely used to identify critical nodes in complex networks [Liu *et al.*, 2016]. The clustering coefficient complements centrality measures by characterizing the presence of loops in the local neighborhood [Costa *et al.*, 2007]. The goal of the second type is to describe the relative relationship between the edge stub and the

[2]We provide a detailed introduction to the feature selection in the appendix.

other nodes. Inspired by heuristics [Wang and Van Mieghem, 2010; Li *et al.*, 2018], we compute the distance, product of degree, and algebraic distance between the considered node pairs. And we adopt the Jaccard coefficient [Lü and Zhou, 2011] to characterize the similarity between the considered node pairs. All statistics are normalized between 0 and 1. In addition, to capture the global position of nodes in the network, we project them into a two-dimensional coordinate space by using the eigenvectors associated with the two smallest nonzero eigenvalues of the graph Laplacian, which has the property that adjacent nodes have smaller Euclidean distances in the coordinate space. We will illustrate the indispensable role of node coordinates through experiments.

Encoder

The encoder generates low-dimensional representations for states and actions. The critical step involves computing node representations from the raw features. For node $1 \leq i \leq n$, we put all extracted features into a vector x_i, perform a linear transformation to map x_i to the initial representation h_i, and process h_i by multiple transformer layers [Vaswani *et al.*, 2017]. Let o be the number of transformer layers. For $1 \leq j \leq o$, the computation steps in the j-th layer are listed in (3).

$$\begin{aligned} \hat{h}_i^j &\leftarrow \mathsf{MHA}(h_i^{j-1}, \{h_1^{j-1}, \cdots, h_n^{j-1}\}) \\ \tilde{h}_i^j &\leftarrow \mathsf{BN}_1(\hat{h}_i^j + h_i^{j-1}) \\ h_i^j &\leftarrow \mathsf{BN}_2(\mathsf{FF}(\tilde{h}_i^j) + \tilde{h}_i^j) \end{aligned} \tag{3}$$

where h_i^{j-1} is the representation of i-th node produced by the $j-1$-th layer and $h_i^0 = h_i$. MHA refers to the multi-head attention mechanism [Vaswani *et al.*, 2017]. In each head, h_i^{j-1} is mapped into a query, $\{h_1^{j-1}, \cdots, h_n^{j-1}\}$ are mapped into keys and values, and comparabilities between the query and keys are computed to combine the values into an embedding for node i. Embeddings of node i on all heads are linearly projected to the single representation $\hat{h}_i^j$. BN_1 and BN_2 are batch normalization modules [Ioffe and Szegedy, 2015] with learnable affine parameters. FF is the point-wise feed-forward network.

During the $2t$-th step of the decision process, the action representation of selecting node i is the representation of node i, i.e., h_i^o, computed from G_{2t} by the procedure described above. The state s_{2t} is represented as $h_s = [h_G, h_e]$, where the graph embedding h_G is obtained via the sum or mean of all node embeddings, h_e is the default vector when there is no edge stub. The representation for s_{2t+1} is computed similarly, the only difference is that h_e is replaced by the node representation of the edge stub.

Decoder

The decoder makes decisions based on the state representation h_s and node representations $\{h_i^o\}_{i=1}^n$. In the first step, the decoder computes a *glimpse* as below.

$$g = \mathsf{MHA}(h_s, \{h_1^o, \cdots, h_n^o\}) \tag{4}$$

where MHA is almost the same as described above, except that the compatibility of a node is masked as $-\infty$ if the node is infeasible. At step $2t$, nodes with degree equal to $n-1$ are infeasible; at step $2t+1$, nodes that are connected to the edge stub are infeasible.

In the second step, the glimpse g is mapped into query q, the node embedding h_i^o is mapped into key k_i. The compatibility u_i is computed as below:

$$u_i = \begin{cases} C \cdot \tanh(\dfrac{q^T k_i}{\sqrt{d}}), \text{if } i \text{ is feasible} \\ -\infty, \text{otherwise} \end{cases} \tag{5}$$

where d is the dimension of the query, C is a hyperparameter to balance exploration and exploitation [Bello *et al.*, 2016]. Finally, the compatibilities are transformed by a Softmax operator into probabilities.

3.3 NEP-HAM

In the encoder of NEP-AM, initial node embeddings h_i, which are indirectly computed from the network structure through extracted features, are used as the input of multiple transformer layers. One way to improve the performance is to make the input directly depend on the network structure. To this end, we add a structure-aware multi-head attention module [Shi *et al.*, 2020] before the multiple transformer layers, where node i queries its first order neighbors and combines their values in the added module. The resulting NEP-HAM model is a hierarchical attention model, i.e., the query range changes from local to global. Compared to NEP-AM, NEP-HAM enables the direct utilization of the network structure.

3.4 Neural Confined Local Search

Given an initial solution, k-opt local search is a commonly used technique to improve the solution. As stated in the introduction, k is restricted to a small value to maintain a reasonable running time. To remove this restriction, we propose NCLS to realize the effective search of a restricted k-exchange neighborhood of size $O(2ln^k)$ by only examining candidates in the neural confined neighborhood of size $O(2l\eta)$, where η is a hyperparameter to balance effectiveness and efficiency. We can take a large k for the number of candidates to check is independent of k.

Now we introduce the details of NCLS. Let $w_p = \{p, (p+1)\%2l, \cdots, (p+k-1)\%2l\}$ be the index set of exchangeable nodes, where p is the starting index. Let $\mathcal{F}$ be the set of all feasible solutions, $v = \{v_1, v_2, \cdots, v_{2l}\}$ be a feasible solution. We define the restricted k-exchange neighborhood as $\mathcal{N}(v) = \cup_{p=1}^{2l} \mathcal{N}_p(v)$, where $\mathcal{N}_p(v)$ is defined as below.

$$\mathcal{N}_p(v) = \{v' \in \mathcal{F} : v'_i = v_i, \forall 1 \leq i \leq 2l, i \notin w_p\} \tag{6}$$

It is clear that the size of $\mathcal{N}(v)$ is $O(2ln^k)$ and an exhaustive enumeration over $\mathcal{N}(v)$ is unfeasible for large k. To confine $\mathcal{N}(v)$, we first apply the partial solution $v' = \{v_i : \forall 1 \leq i \leq 2l, i \notin w_p\}$ to G and obtain a state s'. Then we generate η groups of nodes $\{(v_1^i, \cdots, v_k^i)\}_{i=1}^{\eta}$ by a learned policy π_θ from s'. We can either sample these groups from π_θ or perform a beam search. All solutions that can be obtained by replacing exchangeable nodes with a node group form the neural confined neighborhood. Thus, the neural confined neighborhood is of size $O(2l\eta)$. The rationale behind NCLS is that we can decode high-quality groups from a learned policy.

Test	G	$\tau(\%)$	LDP	FV	ERes	Greedy	NEP-DQN	NEP-AM (ours)	NEP-HAM (ours)
ID	BA	1	0.025 (0.036)	0.017 (0.043)	0.019 (0.048)	0.045 (0.033)	0.047* (0.057*)	0.047* (0.057*)	**0.049** (**0.060**)
		2.5	0.087 (0.092)	0.036 (0.112)	0.090 (0.104)	0.091 (0.074)	0.117 (0.130)	0.126* (0.134*)	**0.133** (**0.135**)
		5	0.301 (0.164)	0.175 (0.181)	0.312 (0.187)	0.167 (0.139)	0.289 (0.222)	0.341* (0.225*)	**0.352** (**0.227**)
	ER	1	0.090 (0.087)	0.079 (0.094)	0.088 (0.102)	**0.152** (0.069)	0.128 (0.104*)	0.129 (**0.106**)	0.140* (0.104*)
		2.5	0.189 (0.163)	0.155 (0.169)	0.182 (0.174)	0.262 (0.135)	0.279 (0.173)	0.305* (0.177*)	**0.309** (**0.179**)
		5	0.307 (0.232)	0.247 (0.247)	0.284 (0.254*)	0.418 (0.212)	0.482 (0.249)	0.499* (0.254*)	**0.540** (**0.256**)
OOD (100)	BA	1	**0.163** (0.151)	0.003 (0.139)	0.053 (0.159)	—	0.016 (0.178)	0.160* (**0.206**)	0.158 (0.197*)
		2.5	0.407 (0.308)	0.191 (0.304)	0.313 (0.312)	—	0.115 (0.399)	0.414* (0.415*)	**0.420** (**0.420**)
		5	0.573 (0.463)	0.442 (0.463)	0.484 (0.465)	—	0.478 (0.566*)	0.616* (**0.605**)	**0.630** (**0.605**)
	ER	1	0.135 (0.022)	0.107 (0.022)	0.133 (0.023*)	—	0.082 (0.023*)	0.249* (**0.024**)	**0.297** (**0.024**)
		2.5	0.166 (0.038)	0.138 (0.039*)	0.167 (**0.040**)	—	0.133 (0.038)	0.288* (**0.040**)	**0.303** (**0.040**)
		5	0.184 (0.057)	0.176 (0.059*)	0.200 (0.059*)	—	0.157 (0.056)	0.289* (**0.060**)	**0.305** (**0.060**)
OOD (500)	BA	0.05	0.002 (0.018)	0.000 (0.017)	0.000 (**0.026**)	—	0.000 (0.012)	0.020* (0.019*)	**0.021** (0.014)
		0.1	0.021 (**0.068**)	0.000 (0.036*)	0.000 (**0.068**)	—	0.000 (0.023)	0.028* (0.036*)	**0.029** (0.034)
		0.2	**0.084** (0.113*)	0.000 (0.080)	0.010 (**0.121**)	—	0.000 (0.074)	0.071* (0.079)	0.044 (0.096)
	ER	0.05	0.156 (**0.180**)	0.111 (0.167)	0.145 (0.178*)	—	0.093 (0.174)	**0.248** (0.170)	0.173* (0.165)
		0.1	0.196 (**0.221**)	0.139 (0.212)	0.180 (**0.221**)	—	0.121 (0.217*)	**0.316** (0.215)	0.224* (0.215)
		0.2	0.262 (0.271*)	0.187 (0.265)	0.236 (**0.272**)	—	0.183 (0.268)	**0.397** (0.269)	0.287* (0.268)

Table 1: Comparison of the average RI evaluated on the ID test set and OOD test set. Values outside (inside) the parentheses indicate the average RIs against the targeted (random) attack. The best results are boldfaced and the second-best results are marked with asterisks. Dashs indicate unavailable solutions due to the unbearable computational time.

4 Experiments

The main goal of this section is to provide answers to the following research questions. **RQ1**: Does NEP-AM outperform previous baselines? **RQ2**: How does the performance of NEP-HAM compare with NEP-AM? **RQ3**: Can NCLS effectively search the restricted k-exchange neighborhood? **RQ4**: How important are node coordinates to the performance of NEP-AM/HAM?

4.1 Datasets

We use synthetic networks generated from graph models and real-world networks. The first model is Barabási-Albert (BA) model [Barabási and Albert, 1999]. An instance is generated according to the growth and preferential attachment mechanisms, where newly added nodes preferentially attach to M existing nodes with probabilities proportional to the degrees of those nodes. The second model is Erdős-Rényi (ER) model [Erdos *et al.*, 1960]. An instance is generated by uniformly sampling from the set consisting of all networks with n nodes and m edges. To evaluate RL pretraining, which is introduced below, we generate networks of 3 different sizes: 20, 100, and 500. We set $M = 2$ for BA and $m = p \times \binom{n}{2}$ for ER, where p is set as 0.2 for $20, 100$, and 0.015 for 500.

We take 10 real-world networks: the road networks in Kazakhstan, Poland, Romania, Finland, and Turkey, and the power grids in the Switzerland, Czech Republic, United Kingdom, Hungary, and Sweden[3]. For convenience, we abbreviate these country names by country codes in the following paper, i.e., KZ, PL, RO, FI, TR, CH, CZ, GB, HU, and SE.

4.2 Comparing Methods and Parameter Setup

As in [Darvariu *et al.*, 2021], we set up two attack algorithms: the first is the degree-based targeted attack, as we mentioned before; the second is the random attack, known as random failures in the network jargon, where nodes are removed in random order. We compare with 5 baselines, including 4 heuristics and NEP-DQN. Heuristics, involving Greedy, LDP [Wang *et al.*, 2014], FV [Wang and Van Mieghem, 2010], and ERes [Wang *et al.*, 2014] have been introduced in the related work section. Their criteria are formally listed below.

$$\begin{aligned} &\mathsf{Greedy}(u,v) = R(G \oplus \{(u,v)\}) - R(G) \\ &\mathsf{LDP}(u,v) = -1 \times \mathsf{deg}(u) \times \mathsf{deg}(v) \\ &\mathsf{FV}(u,v) = d^A(u,v) \\ &\mathsf{ERes}(u,v) = \Omega(u,v) \end{aligned} \tag{7}$$

where $\mathsf{deg}(u)$ is the degree of u, $d^A(u,v)$ is the algebraic distance between u, v, $\Omega(u,v)$ is the effective resistance between them. For each method, we evaluate the Robustness Improvement (RI):

$$RI(G) = R(\bar{G}) - R(G) \tag{8}$$

where $\bar{G}$ is the network enhanced by the method.

For NEP-AM and NEP-HAM, we set the initial embedding dimension $d_{h_i} = 32$, the number of transformer layers $o = 3$. In MHA, the number of heads is set as $H = 8$, the key and value dimensions are set to $d_k = d_v = d_{h_i}/H$. The number of hidden neurons in FF is set as 512. The C in (5) is set to 10. Models are trained using Adam optimizer; the batch size is fixed at 128, and the learning rate is fixed at $1e-4$. Other hyperparameters, such as max norm of the gradients (1.0 or ∞), set pooling function (sum or mean) to compute the graph embedding h_G, baseline function $b(s)$ in (2) (the exponential or greedy rollout baseline [Kool *et al.*, 2018]), are adjusted to optimize the performance of our models. The experiments are conducted on a server (56-core Intel Xeon Gold 6348 CPU 2.60GHz, 1T RAM, 8 NVIDIA A100 GPUs) under the Python 3.9.12 and PyTorch 1.12.1 environments.

4.3 Evaluation of RL Pretraining (RQ1 & RQ2)

Our goal in RL pretraining is to test the ability of a pre-trained model to generalize to both ID and OOD test sets. To this end,

[3]All real-world networks are available at this URL. All source code is available at this github repository.

G	LDP	FV	Eres	Greedy	NEP-DQN	NEP-AM (ours)	NEP-HAM (ours)
KZ	0.080 (0.163)	0.160 (0.197)	0.160 (0.186)	0.240 (0.221)	0.400* (0.203)	**0.480** (0.257*)	**0.480** (**0.261**)
PL	0.130 (0.140)	0.087 (0.167)	0.087 (0.186)	0.043 (0.189)	0.304* (0.230)	**0.435** (0.234*)	**0.435** (**0.239**)
RO	0.185 (0.183)	0.074 (0.189)	0.111 (0.216)	0.148 (0.211)	0.370 (0.235)	**0.519** (**0.264**)	0.444* (0.261*)
TR	0.057 (0.180)	0.086 (0.204)	0.086 (0.193)	0.114 (0.220)	0.514 (0.247)	**0.743** (**0.258**)	0.657* (0.256*)
FI	0.143 (0.145)	0.143 (0.154)	0.095 (0.187)	0.190 (0.192)	**0.524** (0.189)	0.476* (**0.235**)	0.476* (0.231*)
CH	0.091 (0.100)	0.091 (0.140)	0.091 (0.175)	0.091 (0.186)	0.136* (0.226)	**0.182** (0.249*)	**0.182** (**0.259**)
CZ	0.116 (0.272)	0.116 (0.268)	0.116 (0.302)	0.093 (0.253)	0.186 (**0.375**)	0.395* (0.357)	**0.419** (0.365*)
GB	0.171 (0.273)	0.122 (0.271)	0.073 (0.290)	0.220 (0.319)	0.463 (**0.379**)	0.488* (0.359)	**0.537** (0.367*)
HU	**0.190** (0.127)	0.000 (0.168)	0.143* (0.162)	0.095 (0.151)	**0.190** (0.185)	**0.190** (**0.225**)	**0.190** (0.215*)
SE	0.156 (0.169)	0.094 (0.202)	0.188 (0.241)	0.094 (0.255)	0.313* (0.276)	0.312 (**0.293**)	**0.344** (0.290*)

Table 2: Comparison of RI evaluated on real networks. Values outside (inside) the parentheses indicate the RIs against the targeted (random) attack. The best results are boldfaced and the second-best results are marked with asterisks.

we conduct experiments on synthetic networks. We generate a training set, a validation set, an ID test set, and two OOD test sets for each network model. The training set consists of 2^{14} networks, while both the validation set and the test set consist of 2^7 networks. The network size in the training, validation, and ID test sets is 20, while the network size in the first/second OOD test set is 100/500. For each network, the edge budget is set as a percentage τ of the maximal possible number of edges, i.e., the number of edges to insert is $l = \binom{n}{2}\tau$. We set the values of τ as 1%, 2.5%, and 5% for networks with 20 or 100 nodes, and decrease these values for networks with 500 nodes to 0.05%, 0.1%, and 0.2% to mitigate the expensive computation cost. We train a model for each attack method, network model, and edge budget combination. The training steps are set as 400k, 300k, and 200k for $\tau = 1\%, 2.5\%, 5\%$, respectively. We then choose the pre-trained model with the best validation performance for each combination and report the best performance achieved by the pre-trained models for each OOD test set.

Table 1 displays the average RIs against both targeted and random attacks. The results for targeted (random) attack are outside (inside) the parentheses. Dashes are used to indicate unavailable solutions due to the unbearable computational time. We use names like "BA-100" ("OOD-100") to represent the BA (OOD) test set consisting of networks with 100 nodes.

Targeted attack. We make the following observations regarding the targeted attack from Table 1. **1)** In the ID test set, NEP-DQN performs better or comparable to all heuristics. NEP-AM improves the results of NEP-DQN for nearly all cases, and the maximum relative increase of average RI (17.99%) is reached at BA-20, τ=5%. **2)** In the OOD test sets, NEP-DQN is beaten by FV and ERes in most cases and performs significantly worse than LDP in all cases. In particular, its performance degrades severely at BA-500, where no robustness improvements are obtained. In comparison, NEP-AM works better than the best-performing heuristic (LDP) in almost all cases. For instance, NEP-AM outperforms LDP by 33.33% for BA-500 at τ=0.1% and by $57.07\% \sim 84.44\%$ ($51.53\% \sim 61.22\%$) for ER-100 (ER-500). The only case where LDP achieves a non-marginal improvement (18.31%) over NEP-AM is BA-500 at τ=0.2%. In addition, NEP-AM can generate meaningful solutions even for BA-500 at τ=0.05%, where all baselines fail to yield improvements. **3)** For ID and OOD-100, NEP-HAM performs better than NEP-AM in almost all cases, especially for the ER model. For instance, compared with NEP-AM, average RIs brought by NEP-HAM have a relative increase of 8.53% and 8.22% for ER-20 at $\tau = 1\%, 5\%$, and of 19.28% for ER-100 at $\tau = 1\%$. One possible reason is that, compared to BA networks, the features are more evenly distributed across nodes in ER networks, making the network structure information more important. And in NEP-HAM, the network structure directly affects node representations by determining the local query range. However, this direct dependence on network structure harms the generalization ability of NEP-HAM when applied to larger networks. This is evidenced by the worse performance of NEP-HAM compared to NEP-AM on ER-500. Despite this, NEP-HAM still outperforms LDP on OOD-500 except at BA-500, τ=0.2%.

Random attack. We make the following observations regarding the random attack from Table 1. **1)** The observations for the ID test set are similar to those for the targeted attack. However, the relative improvements of NEP-AM compared to NEP-DQN against the random attack ($1.35\% \sim 3.08\%$) are less evident than those ($0.78\% \sim 17.99\%$) against the targeted attack. The reason, as pointed out by [Darvariu *et al.*, 2021], is that improving network robustness against the random attack is a simpler task than against the targeted attack such that: a) even heuristics are enough to generate competitive solutions and b) a learned policy can only obtain a limited performance improvement. **2)** In OOD-100, NEP-DQN provides better or at least comparable performance than the best baseline (ERes), which differs significantly from the situation observed under the targeted attack. An explanation is that generalization to OOD-100 for the challenging task (improving against the targeted attack) requires a more effective decision model than for the easy task (improving against the random attack). NEP-AM outperforms NEP-DQN by $4.01\% \sim 15.73\%$ on OOD-100. In OOD-500, NEP-AM outperforms over NEP-DQN in all cases ($0.37\% \sim 58.33\%$) except at ER-500, $\tau = 0.05\%$ and ER-500, $\tau = 0.1\%$, where the differences are small (less than 2.3%). However, competitive heuristics (LDP and ERes) perform best for OOD-500. That is because the severely changed network structures degrade the performance of NEP-AM such that the competitive

heuristics beat it. **3)** In the ID test set, NEP-HAM outperforms NEP-AM in all cases ($0.75\% \sim 5.25\%$) except at ER-20, $\tau = 1\%$, where the average RI of NEP-HAM is 1.89% lower than that of NEP-AM. For OOD-100, NEP-HAM performs on par with NEP-AM. For instance, they have the same result values on ER-100. For OOD-500, the average RIs of NEP-HAM are $0.37\% \sim 26.32\%$ lower than that of NEP-AM in all cases, except for BA-500, $\tau = 0.2\%$, where NEP-HAM outperforms NEP-AM by 21.52%.

Conclusions. **1)** In the ID test set, NEP-AM outperforms all baselines for both attack methods, and NEP-HAM further surpasses NEP-AM. **2)** When it comes to OOD test sets, NEP-AM shows better generalization ability than NEP-DQN, especially for the targeted attack where NEP-AM outperforms the best-performing heuristic in almost all cases. NEP-HAM performs better than or comparable to NEP-AM in OOD-100. Although directly utilizing network structure information contributes to the performance gain of NEP-HAM, it hinders its generalization to more severely changed network structures, i.e., OOD-500. **3)** NEP-AM/HAM also suffers from performance degradation on OOD test sets. One potential solution is to include a competitive heuristic, e.g., LDP, as a sub-policy of the decision model such that the heuristic can be utilized when a significant distribution shift is detected. We will explore this solution in future work.

4.4 Evaluation of Active Search (RQ1 & RQ2)

In active search [Bello *et al.*, 2016], we focus on testing a model's ability to solve a single instance, which involves continuously optimizing the model's parameters while keeping track of the best solution. We test our models on real networks, with a fixed τ value as 2.5%. The edge budget is computed using the same method in RL Pretraining. All models (including NEP-DQN) are trained for 50k steps[4]. We try NEP-AM/HAM with or without the entropy regularization [Haarnoja *et al.*, 2018] and report the best results. For entropy regularization, we set the target entropy to $-n$, initialize the dual variable α to 10, and fix the learning rate for α to $1e-4$. We present the results in Table 2, where the results regarding targeted (random) attack are outside (inside) the parentheses. For NEP-DQN performs significantly better than heuristics on real networks, we mainly compare our models with NEP-DQN in the discussion below.

Targeted attack. We analyze the results regarding the targeted attack in Table 2 as below. **1)** In 7 (8) out of 10 real networks, NEP-AM (NEP-HAM) outperforms NEP-DQN significantly with a relative increase of RIs ranging from 5.40% (9.90%) to 112.37% (125.27%). On SE, NEP-AM performs slightly worse than NEP-DQN with -0.32% lower RI, while NEP-HAM outperforms NEP-DQN by 9.90%. On HU, all models have the same RI. On FI, the RI of NEP-AM (NEP-HAM) is 9.16% lower than NEP-DQN. This indicates the entropy regularization does not effectively prevent the premature convergence of our models. In future work, we will explore incorporating intrinsic rewards [Burda *et al.*, 2018] to improve our models' exploration ability to address this issue. **2)** NEP-HAM performs better than or equal to NEP-AM in 8 out of 10 real networks. Specifically, it outperforms NEP-AM by 6.08%, 10.04%, and 10.26% on CZ, GB, and SE, respectively. However, on RO and TR, the RIs of NEP-HAM are 14.45% and 11.57% lower than NEP-AM.

Random attack. We analyze the results regarding the random attack in Table 2 as below. **1)** NEP-AM outperforms NEP-DQN in 8 out of 10 real networks. Specifically, NEP-AM shows significant improvements on FI, KZ, and HU with relative increases of RIs at 24.34%, 26.60%, and 21.62%, respectively. However, on CZ and GB networks, NEP-AM performs slightly worse than NEP-DQN with about 5% lower RIs. NEP-HAM narrows the performance gap to NEP-DQN on these two networks to about 3%. **2)** In 5 out of 10 real networks, NEP-HAM has a slight advantage over NEP-AM with relative increases of RIs ranging from 1.56% to 4.02%. However, NEP-AM performs slightly better than NEP-HAM by 1.73% and 4.65% on FI and HU networks, respectively. They have similar performance in the remaining three networks, with about or less than 1% performance differences.

Conclusions. **1)** Our models show significant progress in solving NEP for real networks against both attack methods. However, we should improve their exploration ability to further improve their performance, especially for the targeted attack. **2)** NEP-HAM performs better than or similar to NEP-AM in most real networks.

4.5 Evaluation of the NCLS (RQ 3)

We aim to check whether NCLS can realize the effective search of the restricted k-exchange neighborhood with large k values. Due to the extremely long computation time of k-opt local search with large k values, we compare NCLS with 1-opt local search. We evaluate them on the ID test sets of synthetic networks at $\tau = 5\%$ and real networks at $\tau = 2.5\%$. The NEP-AM/HAM model trained in previous experiments is used to generate initial solutions for NCLS and 1-opt local search. In particular, we sample 8 initial solutions for each synthetic network and attack method and 32 (128) initial solutions for each real network for the targeted (random) attack. To generate a neural confined neighborhood, we perform a beam search with width 2 to find the $\eta = \min\{2^k, 100\}$ groups of nodes with the highest probability under a trained model to replace the exchangeable nodes. We set k to $5, 7$. We update the current solution with the best solution in the neighborhood and iteratively improve the solution until obtaining a local optimal solution. We process all initial solutions in parallel, and choose the best-improved solution as the final solution. We present the results in Table 3 and also list the results of NEP-AM and NEP-HAM in the table for easier comparison. We use names like "NCLS-AM" to represent NCLS with both initial solutions and neural confined neighborhoods generated by a trained NEP-AM model and "LS-AM" to represent the 1-opt local search with initial solutions generated by a trained NEP-AM model. When comparing NCLS-AM (NCLS-HAM) with NEP-AM (NEP-HAM), we report the best results among NCLS-AM (NCLS-HAM) with $k = 5, 7$. We observe that for NCLS-AM (NCLS-HAM),

[4]We discuss the results under different training steps in the appendix.

G	NEP-AM	LS-AM	NCLS-AM		NEP-HAM	LS-HAM	NCLS-HAM	
			k=5	k=7			k=5	k=7
BA	0.341 (0.225)	0.550 (0.320)	0.627 (0.349)	**0.642** (**0.363**)	0.352 (0.227)	0.547 (0.325)	0.618 (0.347)	**0.648** (**0.362**)
ER	0.499 (0.254)	0.821 (0.322)	0.832 (0.330)	**0.837** (**0.339**)	0.540 (0.256)	0.816 (0.324)	**0.837** (**0.337**)	**0.837** (**0.337**)
KZ	0.480 (0.257)	**0.520** (0.264)	0.480 (**0.273**)	**0.520** (0.268)	0.480 (0.261)	**0.520** (0.257)	0.480 (**0.269**)	**0.520** (**0.269**)
PL	0.348 (0.234)	**0.435** (0.235)	**0.435** (**0.242**)	**0.435** (0.239)	0.435 (0.239)	0.435 (0.237)	0.435 (0.237)	**0.478** (**0.241**)
RO	0.519 (0.264)	0.444 (0.262)	**0.630** (0.261)	**0.630** (**0.266**)	0.444 (0.261)	0.444 (0.259)	0.556 (0.261)	**0.593** (**0.273**)
TR	0.743 (0.258)	0.743 (0.253)	**0.771** (0.265)	**0.771** (**0.268**)	0.657 (0.256)	0.657 (0.253)	**0.771** (**0.269**)	0.743 (0.268)
FI	**0.476** (0.235)	**0.476** (0.231)	**0.476** (0.235)	**0.476** (**0.240**)	**0.476** (0.231)	**0.476** (0.223)	**0.476** (**0.237**)	**0.476** (0.236)
CH	**0.182** (0.249)	**0.182** (0.237)	**0.182** (0.248)	**0.182** (**0.257**)	**0.182** (0.259)	**0.182** (0.265)	**0.182** (**0.289**)	**0.182** (0.281)
CZ	0.395 (**0.357**)	**0.419** (0.347)	**0.419** (0.347)	**0.419** (0.352)	0.419 (**0.365**)	0.395 (0.358)	0.419 (0.354)	**0.442** (0.358)
GB	0.488 (0.359)	0.585 (0.362)	0.561 (**0.371**)	**0.610** (0.370)	0.537 (0.367)	0.561 (0.365)	**0.585** (**0.370**)	**0.585** (**0.370**)
HU	**0.190** (**0.225**)	**0.190** (0.206)	**0.190** (0.221)	**0.190** (**0.225**)	**0.190** (**0.215**)	**0.190** (0.207)	**0.190** (0.212)	**0.190** (0.213)
SE	0.312 (0.293)	0.344 (0.287)	0.344 (0.291)	**0.375** (**0.298**)	0.344 (0.290)	**0.375** (0.288)	**0.375** (0.291)	**0.375** (**0.292**)

Table 3: Evaluation of NCLS under different k. Values outside (inside) the parentheses indicate the RIs against the targeted (random) attack. The best results are boldfaced.

these best results are consistently achieved with $k = 7$ in most cases.

Targeted attack. We make the following observations regarding the targeted attack in Table 3 as below. **1)** When dealing with synthetic networks (BA and ER), NCLS-AM/HAM performs better than LS-AM/HAM. Specifically, with $k = 5$, NCLS-AM/HAM outperforms LS-AM/HAM by 14%/12.98% on BA networks and by 1.34%/2.57% on ER networks. With $k = 7$, the advantages of NCLS-AM/HAM expand further to 16.73%/18.46% on BA networks and 1.95%/2.57% on ER networks. **2)** For real networks, with $k = 5$, NCLS-AM outperforms LS-AM in two networks (RO and TR) by 41.89% and 3.77%, respectively. It performs equally well as LS-AM in six networks and worse in two networks (KZ and GB) with a 7.69% and 4.10% difference. With $k = 7$, NCLS-AM outperforms LS-AM in four networks (RO, TR, GB, and SE) by 41.89%, 3.77%, 4.27% and 9.01%, and performs equally well as LS-AM in all other networks. It is worth noting that LS-AM, NCLS-AM with $k = 5$, and NCLS-AM with $k = 7$ tend to have the same results in most networks. However, this does not imply our methods cannot search a larger neighborhood effectively. The reason for these results is that the initial solutions sampled from NEP-AM are good enough such that even LS-AM can find high-quality local optima. We find that when we generate initial solutions randomly, 1-opt local search is much worse than NCLS. For instance, in KZ, NCLS-AM with $k = 5$ outperforms 1-opt local search by 50%. The results on other real networks are similar, thus omitted here. We can conduct similar analyses to compare LS-HAM and NCLS-HAM with $k = 5, 7$. NCLS-HAM with $k = 5$ outperforms LS-HAM in four networks (RO, TR, CZ, and GB) by 25.23%, 17.35%, 6.08%, and 4.28%. It performs equally to LS-HAM in five real networks, but performs worse than LS-HAM in one network (KZ) by 7.69%. Meanwhile, NCLS-HAM with $k = 7$ outperforms LS-HAM in five real networks with relative increases of RIs ranging from 4.28% to 33.56%, and performs equally to LS-HAM in all remaining real networks. **3)** NCLS-AM (NCLS-HAM) shows significant improvements over NEP-AM (NEP-HAM) on synthetic networks, with an increase of 88.27% (84.09%) on BA networks and 67.74% (55%) on ER networks. For real networks, NCLS-AM (NCLS-HAM) improves the results of NEP-AM (NEP-HAM) on 7 out of 10 real networks with relative increases of RIs ranging from 3.77% to 25% (5.49% to 33.56%).

Random attack. We make the following observations regarding the random attack in Table 3 as below. **1)** For synthetic networks (BA and ER), with $k = 5$, NCLS-AM (NCLS-HAM) produces better results than LS-AM (LS-HAM). Specifically, NCLS-AM (NCLS-HAM) outperforms LS-AM (LS-HAM) by 9.06% (6.77%) on BA networks and by 2.48% (4.01%) on ER networks. When the value of k is increased to 7, these advantages are further increased to 13.44% (11.38%) on BA networks and 5.28% (4.01%) on ER networks. **2)** For real networks, with $k = 5$, NCLS-AM (NCLS-HAM) outperforms LS-AM (LS-HAM) on eight real networks with relative increases of RIs ranging from 1.39% to 7.28% (0.77% to 9.06%). In the remaining two networks, NCLS-AM (NCLS-HAM) performs equally to LS-AM (LS-HAM) on one network and worse than LS-AM (LS-HAM) on another by 0.38% (1.12%). With $k = 7$, NCLS-AM (NCLS-HAM) outperforms LS-AM (LS-HAM) on all (nine) real networks with relative increases in RIs ranging from 1.44% to 9.22% (1.37% to 6.04%). **3)** For synthetic networks, NCLS-AM (NCLS-HAM) improves the results of NEP-AM (NEP-HAM) by 61.33% (59.47%) on BA networks and by 33.46% (31.64%) on ER networks. For real networks, NCLS-AM (NCLS-HAM) improves the results of NEP-AM (NEP-HAM) on 8 out of 10 real networks with relative increases in RIs ranging from 0.76% to 6.23% (0.69% to 11.58%).

Conclusions. **1)** NCLS can realize the effective search of the restricted k-exchange neighborhood by only examining a few candidates generated by a trained model. **2)** NCLS-AM (NCLS-HAM) as a post-processing tool can significantly improve the results of NEP-AM (NEP-HAM) on both attack methods.

4.6 Comparison of Validation Curves (RQ 4)

In the previous subsection, we compare pre-trained models on synthetic networks. In this subsection, we evaluate the

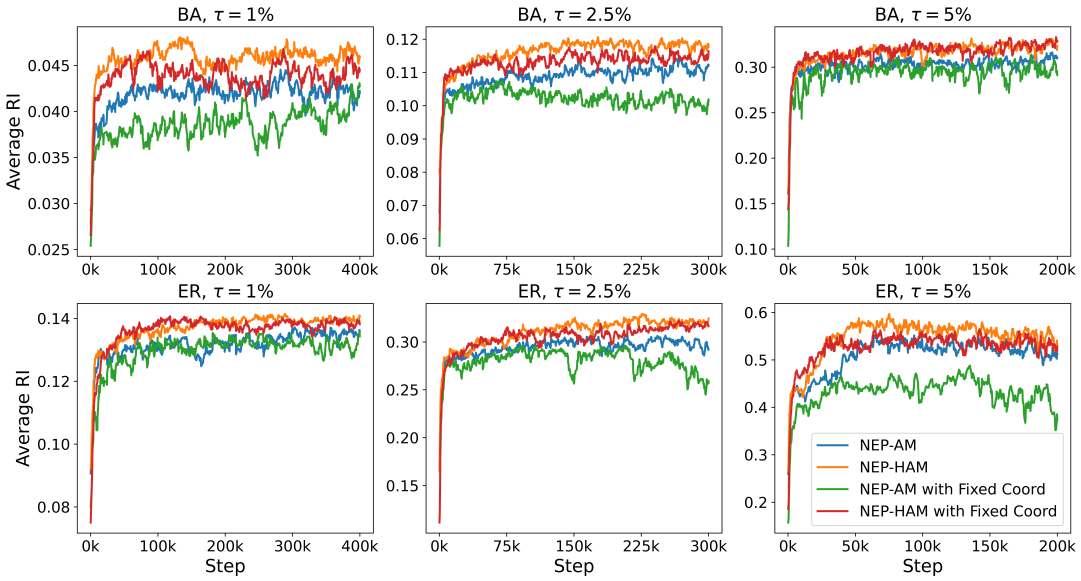

Figure 1: Validation Curves.

performance of NEP-AM and NEP-HAM throughout the entire training process by measuring the average RI on the validation set. We also investigate the necessity of recomputing node coordinates as the network structure changes. Although methods [Demmel, 1997] have been proposed to approximate the eigenvectors of graph Laplacian efficiently, there is still computational overhead. A natural idea is to fix the node coordinates during the decision process. Thus, we consider these models with fixed node coordinates. We display the results of the targeted attack in Fig. 1, which shows the average RI against the training step. The results of the random attack are similar, thus omitted here. From the figure, we can see that: **1)** In all cases, NEP-HAM (orange lines) has an obvious advantage over NEP-AM (blue lines) on all training steps. **2)** With fixed node coordinates, the performance of NEP-AM degrades severely. Thus, updated node coordinates provide essential information for NEP-AM to capture the changing of the network structure. **3)** With fixed node coordinates, the performance of NEP-HAM also degrades somewhat, but not so much as NEP-AM. That is because NEP-HAM exploits the network structure more directly than NEP-AM. Thus, recomputing node coordinates is indispensable for NEP-AM and provides some extra information for NEP-HAM.

5 Discussions

Scalability. Our models face two scalability issues[5]. Firstly, the changing network structure in the decision process requires recomputing node coordinates, which is time-consuming. To address this, we suggest computing exact node coordinates only in the first decision step and adjusting them dynamically in subsequent steps using matrix perturbation theory [Chan *et al.*, 2014]. Secondly, the computation of the multi-head attention mechanism is quadratic with graph size. We propose replacing it with the ProbSparse self-attention [Zhou *et al.*, 2021], which calculates the inner product $O(n \log n)$ times and does not compromise performance.

Generalization to different network patterns. In this work, we only focus on how well our models generalize to different network sizes within the same network model. We run a pilot experiment and find that when we train our models on BA networks and test them on ER networks, their performance is slightly worse than NEP-DQN trained on ER networks. We are working on a solution to address this challenge. We build an encoder $q_\phi(z|G)$ to map the network G to a distribution over the latent vector z, where the distribution distance between two similar structures is small, and learn a policy $\pi_\theta(v|s,z)$ conditioned on z. For decision, we select node v with probability $p(v) = \int q_\phi(z|G)\pi_\theta(v|s,z)dz$.

Defending against an unknown attack method. Our work aims to improve network robustness against two common attack methods. However, it is often difficult to anticipate the attack method beforehand. To address this, we suggest maximizing the robustness against the strongest attack strategy $\mathcal{A}^*$ mentioned in the related work section. Let G^* be the optimal solution under $\mathcal{A}^*$. For arbitrary strategy $\mathcal{A}$, we have $R^{\mathcal{A}^*}(G^*) \leq R^{\mathcal{A}}(G^*) \leq R^{\mathcal{A}}(\bar{G})$, where $\bar{G}$ is the optimal solution under $\mathcal{A}$. However, the gap between $R^{\mathcal{A}^*}(G^*)$ and $R^{\mathcal{A}}(\bar{G})$ may be large. Our methods can be applied to seek G^* by modifying the reward function.

Conclusions. We develop NEP-AM and NEP-HAM for solving NEP. In NEP-AM, we incorporate domain knowledge to get high-quality state and action representations. In NEP-HAM, we add a structure-aware attention module to NEP-AM to directly utilize the network structure information. To further improve the results of NEP-AM/HAM, we introduce NCLS to realize the effective search of a large neighborhood. We demonstrate our models' superiority and NCLS's effectiveness through extensive experiments.

[5]We provide a running time comparison in the appendix.

Acknowledgements

This work is supported by the National Key R&D Program of China (No.2021ZD0113000).

References

[Aarts *et al.*, 2003] Emile Aarts, Emile HL Aarts, and Jan Karel Lenstra. *Local search in combinatorial optimization*. Princeton University Press, 2003.

[Albert *et al.*, 2000] Réka Albert, Hawoong Jeong, and Albert-László Barabási. Error and attack tolerance of complex networks. *nature*, 406(6794):378–382, 2000.

[Barabási and Albert, 1999] Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *science*, 286(5439):509–512, 1999.

[Barrat *et al.*, 2004] Alain Barrat, Marc Barthelemy, Romualdo Pastor-Satorras, and Alessandro Vespignani. The architecture of complex weighted networks. *Proceedings of the national academy of sciences*, 101(11):3747–3752, 2004.

[Bello *et al.*, 2016] Irwan Bello, Hieu Pham, Quoc V Le, Mohammad Norouzi, and Samy Bengio. Neural combinatorial optimization with reinforcement learning. *arXiv preprint arXiv:1611.09940*, 2016.

[Beygelzimer *et al.*, 2005] Alina Beygelzimer, Geoffrey Grinstein, Ralph Linsker, and Irina Rish. Improving network robustness by edge modification. *Physica A: Statistical Mechanics and its Applications*, 357(3-4):593–612, 2005.

[Burda *et al.*, 2018] Yuri Burda, Harrison Edwards, Amos Storkey, and Oleg Klimov. Exploration by random network distillation. *arXiv preprint arXiv:1810.12894*, 2018.

[Chan *et al.*, 2014] Hau Chan, Leman Akoglu, and Hanghang Tong. Make it or break it: Manipulating robustness in large networks. In *Proceedings of the 2014 SIAM International Conference on Data Mining*, pages 325–333. SIAM, 2014.

[Costa *et al.*, 2007] L da F Costa, Francisco A Rodrigues, Gonzalo Travieso, and Paulino Ribeiro Villas Boas. Characterization of complex networks: A survey of measurements. *Advances in physics*, 56(1):167–242, 2007.

[Dai *et al.*, 2016] Hanjun Dai, Bo Dai, and Le Song. Discriminative embeddings of latent variable models for structured data. In *International conference on machine learning*, pages 2702–2711. PMLR, 2016.

[Darvariu *et al.*, 2021] Victor-Alexandru Darvariu, Stephen Hailes, and Mirco Musolesi. Goal-directed graph construction using reinforcement learning. *Proceedings of the Royal Society A*, 477(2254):20210168, 2021.

[Demmel, 1997] James W Demmel. *Applied numerical linear algebra*. SIAM, 1997.

[Deudon *et al.*, 2018] Michel Deudon, Pierre Cournut, Alexandre Lacoste, Yossiri Adulyasak, and Louis-Martin Rousseau. Learning heuristics for the tsp by policy gradient. In *International conference on the integration of constraint programming, artificial intelligence, and operations research*, pages 170–181. Springer, 2018.

[Ellens and Kooij, 2013] Wendy Ellens and Robert E Kooij. Graph measures and network robustness. *arXiv preprint arXiv:1311.5064*, 2013.

[Ellens *et al.*, 2011] Wendy Ellens, Floske M Spieksma, Piet Van Mieghem, Almerima Jamakovic, and Robert E Kooij. Effective graph resistance. *Linear algebra and its applications*, 435(10):2491–2506, 2011.

[Erdos *et al.*, 1960] Paul Erdos, Alfréd Rényi, et al. On the evolution of random graphs. *Publ. Math. Inst. Hung. Acad. Sci*, 5(1):17–60, 1960.

[Faloutsos *et al.*, 1999] Michalis Faloutsos, Petros Faloutsos, and Christos Faloutsos. On power-law relationships of the internet topology. *ACM SIGCOMM computer communication review*, 29(4):251–262, 1999.

[Fiedler, 1973] Miroslav Fiedler. Algebraic connectivity of graphs. *Czechoslovak mathematical journal*, 23(2):298–305, 1973.

[Ghosh and Boyd, 2006] Arpita Ghosh and Stephen Boyd. Growing well-connected graphs. In *Proceedings of the 45th IEEE Conference on Decision and Control*, pages 6605–6611. IEEE, 2006.

[Haarnoja *et al.*, 2018] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. PMLR, 2018.

[Holme *et al.*, 2002] Petter Holme, Beom Jun Kim, Chang No Yoon, and Seung Kee Han. Attack vulnerability of complex networks. *Physical review E*, 65(5):056109, 2002.

[Hu *et al.*, 2019] Qifu Hu, Angsheng Li, Jiamou Liu, and Jun Liu. Time-efficient network monitoring through confined search and adaptive evaluation. In *Pacific Rim International Conference on Artificial Intelligence*, pages 633–646. Springer, 2019.

[Ioffe and Szegedy, 2015] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015.

[Khalil *et al.*, 2017] Elias Khalil, Hanjun Dai, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning combinatorial optimization algorithms over graphs. *Advances in neural information processing systems*, 30, 2017.

[Kool *et al.*, 2018] Wouter Kool, Herke van Hoof, and Max Welling. Attention, learn to solve routing problems! In *International Conference on Learning Representations*, 2018.

[Li *et al.*, 2018] Gang Li, Zhi Feng Hao, Han Huang, and Hang Wei. Maximizing algebraic connectivity via minimum degree and maximum distance. *IEEE Access*, 6:41249–41255, 2018.

[Liu *et al.*, 2016] Jun Liu, Qingyu Xiong, Weiren Shi, Xin Shi, and Kai Wang. Evaluating the importance of nodes in complex networks. *Physica A: Statistical Mechanics and its Applications*, 452:209–219, 2016.

[Liu *et al.*, 2017] Jing Liu, Mingxing Zhou, Shuai Wang, and Penghui Liu. A comparative study of network robustness measures. *Frontiers of Computer Science*, 11(4):568–584, 2017.

[Lü and Zhou, 2011] Linyuan Lü and Tao Zhou. Link prediction in complex networks: A survey. *Physica A: statistical mechanics and its applications*, 390(6):1150–1170, 2011.

[Ma *et al.*, 2019] Qiang Ma, Suwen Ge, Danyang He, Darshan Thaker, and Iddo Drori. Combinatorial optimization by graph pointer networks and hierarchical reinforcement learning. *arXiv preprint arXiv:1911.04936*, 2019.

[Ma *et al.*, 2021] Yining Ma, Jingwen Li, Zhiguang Cao, Wen Song, Le Zhang, Zhenghua Chen, and Jing Tang. Learning to iteratively solve routing problems with dual-aspect collaborative transformer. *Advances in Neural Information Processing Systems*, 34:11096–11107, 2021.

[Mnih *et al.*, 2013] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[Nazari *et al.*, 2018] Mohammadreza Nazari, Afshin Oroojlooy, Lawrence Snyder, and Martin Takác. Reinforcement learning for solving the vehicle routing problem. *Advances in neural information processing systems*, 31, 2018.

[Sánchez Martínez, 2009] Mari Carmen Sánchez Martínez. Robustness optimization via link additions. Master's thesis, Universitat Politècnica de Catalunya, 2009.

[Shi *et al.*, 2020] Yunsheng Shi, Zhengjie Huang, Shikun Feng, Hui Zhong, Wenjin Wang, and Yu Sun. Masked label prediction: Unified message passing model for semi-supervised classification. *arXiv preprint arXiv:2009.03509*, 2020.

[Vaswani *et al.*, 2017] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[Vinyals *et al.*, 2015] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. *Advances in neural information processing systems*, 28, 2015.

[Wang and Rong, 2009] Jian-Wei Wang and Li-Li Rong. Cascade-based attack vulnerability on the us power grid. *Safety science*, 47(10):1332–1336, 2009.

[Wang and Van Mieghem, 2010] Huijuan Wang and Piet Van Mieghem. Algebraic connectivity optimization via link addition. In *3d International ICST Conference on Bio-Inspired Models of Network, Information, and Computing Systems*, 2010.

[Wang *et al.*, 2014] Xiangrong Wang, Evangelos Pournaras, Robert E Kooij, and Piet Van Mieghem. Improving robustness of complex networks via the effective graph resistance. *The European Physical Journal B*, 87(9):1–12, 2014.

[Williams, 1992] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3):229–256, 1992.

[Wu *et al.*, 2021] Yaoxin Wu, Wen Song, Zhiguang Cao, Jie Zhang, and Andrew Lim. Learning improvement heuristics for solving routing problems. *IEEE transactions on neural networks and learning systems*, 33(9):5057–5069, 2021.

[Zhou *et al.*, 2021] Haoyi Zhou, Shanghang Zhang, Jieqi Peng, Shuai Zhang, Jianxin Li, Hui Xiong, and Wancai Zhang. Informer: Beyond efficient transformer for long sequence time-series forecasting. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pages 11106–11115, 2021.