CS886 Project Report: Network Dismantling using Graph Neural Networks

Lucas Fenaux [1]

Abstract

Graph network dismantling is the challenge of pinpointing a minimal set of nodes (or edges) whose removal causes a large network to break down into many small, isolated clusters. A body of literature studies dismantling strategies, with existing methods mainly based on heuristics and focusing on node removal. With the advent of graph neural network-based (GNN) methods for graph network dismantling, two problems arise: Can we improve upon them, and can we defend against them? In this work, we re-implement and evaluate the state-of-the-art for GNN-based network dismantling: GDM (Graph Dismantling Machine). We then study training methods to improve its performance and enable it to train on larger graphs. Finally, we study whether machine learning and GNNs can be used to improve a network's robustness to dismantling attacks. The code for this work is available at https://github.com/LucasFenaux/cs886-2.

1. Introduction

Graph network dismantling captures the idea of finding the system's "weak links" that hold the network together, and it is known to be NP-hard, meaning that exact solutions become computationally infeasible as the network size grows. Instead, researchers rely on approximations and heuristics, drawing on tools from percolation theory, spectral methods, and machine learning to efficiently identify dismantling strategies. This problem's applications are critical in fields such as epidemic control (Morone & Makse, 2015) and infrastructure protection (Cohen et al., 2001), where strategically disrupting a network can halt harmful processes or cause catastrophic failures. The current state-of-the-art for this machine learning approach is GDM (Grassia et al., 2021) and its follow-up work CoreGDM (Grassia & Mangioni, 2023). GDM trains a GNN model to predict the importance of the nodes in a graph. They then progressively remove nodes in order of importance until the size of the largest connected component (LCC) in the graph is reduced to below a certain threshold (usually 10% of the original size of the largest connected component). CoreGDM improves on GDM by merging this model with the CoreHD (Zdeborová et al., 2016) algorithm to dismantle graphs more efficiently. However, GDM is limited by the generalization capabilities of GNNs. This limitation is due to their training method (that we explain in detail in Section 3). They can only train their model on small (25-node) synthetic networks. We first tried to improve this method but with poor success. Then, we use reinforcement learning, rather than supervised learning, to train their model. This change allows us to train directly on large networks rather than rely purely on the generalization capabilities of GNNs. We explore a defense method against dismantling attacks in Section 5.

2. Previous Work

The work by Artime et al. (Artime et al., 2024) provides a compilation of a large number of existing dismantling algorithms: brute force, Collective Influence (CI) (Morone et al., 2016), Graph Dismantling Machine (GDM) (Grassia et al., 2021), CoreGDM (Grassia & Mangioni, 2023), CoreHD (Zdeborová et al., 2016), Generalized Network Dismantling (GND) (Ren et al., 2019), Ensemble GND (EGND) (Ren & Antulov-Fantulin, 2020), Explosive Immunization (EI) (Clusella et al., 2016), FINDER (Fan et al., 2020), Min-Sum (Braunstein et al., 2016), NetworkEntanglement (Braunstein et al., 2016), and VertexEntanglement (Huang et al., 2024). It implements these works in a combination of Python and C++. However, they are difficult to run, with some having high runtimes. Runtimes can reach hours to tens of hours on a 32-CPU-core machine for a single graph, depending on the size of the graph. We use the datasets they gathered and evaluated, both real and synthetic, as the basis for our experimentation.

[1]University of Waterloo. Correspondence to: Lucas Fenaux <lucas.fenaux@uwaterloo.ca>.

Proceedings of the 42^{nd} International Conference on Machine Learning, Vancouver, Canada. PMLR 267, 2025. Copyright 2025 by the author(s).

3. GDM

3.1. Architecture

In this section, we explore in detail the methodology of GDM and the GNN architecture they use. They use a Graph Attention Convolution model (GAT). Their model can be divided into the graph convolution component and the fully connected component. The graph convolution component contains a series of GATConv layers (from pytorch_geometric) and Linear layers. GAT uses multi-head attention for each layer and dropout for improved generalization. Dropout randomly drops input features with a low probability to prevent the model from relying exclusively on a small subset of features to make a decision. If we let H_k be the k-th GATConv layer and L_k be the k-th linear layer, then the k-th layer F_k in their graph convolution component, given the nodes features at layer k X_k and the edges E computes the following node features:

$$F_k(X_k, E) = \sigma(H_k(X_k, E) + L_k(X_k))$$

Where σ is the ELU (Exponential Linear Unit) activation function, defined as follows:

$$\sigma(x) = \begin{cases} x, & \text{if } x \geq 0, \\ \alpha\left(e^x - 1\right), & \text{if } x < 0, \end{cases}$$

Where α is a hyperparameter that controls the value an ELU saturates for negative inputs.

The output of the convolution component is fed into the fully connected component, where each layer is a linear layer with an ELU activation $F_k(X_k) = \sigma(L_k(X_k))$. Finally, to normalize the importance of each node, they apply a sigmoid activation ($\frac{1}{1+e^{-x}}$) to the output prediction of each node. In their implementation, they provide versions of this architecture with varying numbers of layers for each component and a number of heads for the attention module. We try all the suggested versions to select the best-performing one, as shown in Table 1. The code to re-run the grid can be found in the run_gdm_grid.py in our GitHub repository. The clear winner is the last configuration proposed. (40, 30) means that the convolution component has two layers, one of which takes in the number of features in the input graph (by default, 4) and outputs 40 features per node. The second takes 40 features per node as input and outputs 30. The # Heads represents the number of attention heads used for each GATConv layer. (100, 100) means that the second component is composed of two 100-feature linear layers, with a final third regressor linear layer that outputs a single feature per node (for node regression).

With node reinsertion (as we explain in Section 3.3), all three architectures manage to dismantle the Brazilian corruption graph network by removing 29 nodes (rather than 71.6 nodes on average for the best performer without reinsertion). The Brazilian corruption graph network has 309 nodes and 6562 directed edges.

# Conv	# Fully connected	# Heads	Nodes removed
(30, 20)	(100, 100)	(5, 5)	131.6 ($\pm$52.7)
(40, 30, 20, 10)	(100, 100)	(1, 1, 1, 1)	90.6 ($\pm$25.1)
(40, 30)	(100, 100)	(10, 10)	**71.6** ($\pm$12.7)

Table 1. Comparison of the number of layers for each component and the number of heads in the attention module for the GDM GAT model. Averaged number of nodes removed (less is better) over five runs (with standard deviation in parentheses) on the Brazilian corruption graph network. The LCC threshold is 10%.

3.2. Training

They train this GAT model with supervised learning. To do so, they need to compute labels for each node in the graph. To compute labels for the small 25-node synthetic graphs that they train on, they optimally dismantle these small networks using a brute-force approach. This approach has an exponential runtime with respect to the graph network size. However, this is feasible in practice due to the small size of the graphs. They then gather all the optimal combinations of nodes to dismantle the graph. The label of a node is the ratio of optimal combinations it appears in. For example, if a node appears in every optimal dismantling combination, its label is 1. If it appears in half of them, it is $\frac{1}{2}$; if it never appears, it is 0. The model is trained with the mean-squared error loss to teach it to match these ratios, with the hope that it will generalize to larger networks. The label computation step of their training process scales exponentially with the number of nodes in the graph, preventing them from training their model on larger graphs.

3.3. Removal of nodes

To remove nodes using a GDM-trained GAT model, one can feed the graph and its node features to the model. Then, sort the output prediction logit for each node and remove nodes from the highest logit to the lowest until the largest connected component threshold (10% of the original size of the largest connected component) is achieved. The number of nodes removed is the metric for performance, with a lower amount being better. Any node-removal graph dismantling algorithm can be enhanced with a reinsertion algorithm. A reinsertion algorithm takes in an ordered list of removed nodes and reinserts as many nodes as possible into the graph while remaining below the LCC threshold. We implement a greedy reinsertion algorithm (Braunstein et al., 2016) that starts from the last removed node and reinserts it if the LCC threshold is not reached. This process is repeated until the first removed node is reached.

4. Improvements

As explained in Section 3.3, the GAT model makes a single prediction given the entire network, and then, using that prediction, we remove the nodes. While this approach leads to a computationally efficient training approach on small networks and an inference approach on large networks, it is sub-optimal. We believe that instead, predicting the next node to remove, given the current state of the network, is a better approach to GNN-based network dismantling. This belief is based on the success of LLMs, which are next-token prediction models rather than predicting an entire sequence in a single shot. Using a next-node-to-remove approach also opens up the possibility of various strategies beyond a purely greedy strategy of removing the node of the highest predicted importance. For example, we could use beam search (Lowerre & Reddy, 1976) and other decoding algorithms to search the potential subsets of nodes to remove using the model's predictions.

4.1. Next-node prediction with supervised learning

We considered multiple approaches to train a next-node prediction network with supervised learning. These approaches can be decomposed by their loss function, how the labels are computed, and how often (number of removal steps) the labels are recomputed. We explored using the Mean-Square Error Loss (MSE) and the Cross-Entropy Loss (CE). For labels, we tried four approaches: computing node appearance ratios in optimal solutions using GDM's label computation and their normalized equivalent (all labels for a graph sum to one), computing "keystone" nodes and labeling them as one and their normalized equivalent (the label is 1/number of keystones).

Keystones are the nodes we deem key to be removed in a network at a given step. We first compute the ratio labels for each node, and then we take the nodes with maximal ratios and set their label to one and the others to zero. It would help the model focus only on the most important node in the network to remove at each step. We recompute the keystones at every node-removal step or whenever we run out. When we use occurrence ratios as labels, we recompute them after each node removal.

One advantage of GDM is that it only needs to compute the optimal labels once. Therefore, it is not too consequential if this process is time-consuming (as long as it is feasible). However, we do not have that luxury since we recompute the labels frequently. As such, we perform an approximation of this occurrence count when necessary. When there are too many combinations to evaluate of a given size, we randomly sample a subset. Our label-generating function is described in Algorithm 1.

To help rule out ineffective combinations of approaches, we run an exhaustive grid of each combination and compare their performance averaged over three runs in Table 2 against the Brazilian corruption network. We call our next-node GDM-based removal method NN-GDM. If we also use reinsertion, we will call it NN-GDM-r. With reinsertion, we find that the best-performing combination closely matches GDM, with 30 nodes removed to dismantle the network compared to 29 for GDM. All the results for the CE loss are poor, so we did not include them.

Algorithm 1 Find Optimal Paths

Input: graph G, lcc_threshold l, num_paths k
Initialize $candidates = [\{0, 1, \ldots, G.num\_nodes]$
for $i = G.num\_nodes - 1$ **to** 0 **do**
  Initialize $new\_candidates = []$
  for $subset \in candidates$ **do**
    for $element \in subset$ **do**
      $new\_combination$ = $subset \setminus \{element\}$
      if $new\_combination \notin new\_candidates$ **then**
        $new\_combination = new\_combination \cup [new\_candidate]$
      end if
    end for
  end for
  $num\_candidates = len(new\_candidates)$
  if $k >= num\_candidates$ **then**
    to_check = all combinations of size $i - 1$ that can be generated from each combination in $new\_candidates$
  else
    Sample k random subsets of size $i - 1$ generated from combinations in $new\_candidates$.
  end if
end for

4.2. Reinforcement Learning

A major disadvantage of our and GDM's supervised approach is the computational infeasibility of training on larger graph networks. To bypass the label computation phase, we study whether training their GAT model with reinforcement learning (specifically Q-learning) yields better results.

4.2.1. Methodology

Q-learning is a model-free reinforcement learning algorithm that estimates the expected cumulative reward of taking a given action a in a given state s, commonly denoted as $Q(s, a)$, without needing a model of the environment. The core of Q-learning is its update rule:

$$Q(s,a) \leftarrow Q(s,a) + \alpha\Big(r + \gamma \max_{a'} Q(s',a') - Q(s,a)\Big) \tag{1}$$

Loss	Label	Recompute	Sigmoid	NN-GDM avg	MoE	NN-GDM-r avg	MoE
MSE	Probabilities	Step	Yes	217.67 (±86.74)	215.47	93.33 (±67.15)	166.82
MSE	Probabilities	Step	No	215.67 (±)76.38	189.73	113.00 (±60.49)	150.26
MSE	Normalized probabilities	Step	Yes	263.00 (±8.64)	21.47	157.00 (±32.89)	81.71
MSE	Normalized probabilities	Step	No	160.67(±83.96)	208.57	**30.00** (±0.82)	2.03
MSE	Keystone	Step	Yes	152.67(±42.16)	104.73	91.00 (±49.44)	122.82
MSE	Keystone	Step	No	241.67 (±28.12)	69.86	179.33(±32.06)	79.63
MSE	Keystone	No keystone	Yes	255.33 (±20.50)	50.92	161.00 (±20.02)	49.72
MSE	Keystone	No keystone	No	193.33 (±66.46)	165.11	107.33 (±54.84)	136.22
MSE	Normalized keystone	Step	Yes	215.00 (±87.00)	216.12	97.67 (±78.15)	194.14
MSE	Normalized keystone	Step	No	174.67 (±76.93)	191.09	101.00 (±50.94)	126.54
MSE	Normalized keystone	No keystone	Yes	279.00 (±0.00)	0.00	112.33 (±81.30)	201.96
MSE	Normalized keystone	No keystone	No	182.33 (±73.52)	182.64	123.00 (±58.04)	144.18
GDM				62	/	**29**	/
RL				**33**	/	**29**	/

Table 2. MOE stands for Margin of Error at 95% confidence. We used 1000 paths for the approximate label generation. Our performance numbers are the number of nodes removed to reach an LCC threshold of 10% averaged over three runs.

Where α is the learning rate, r is the reward received after performing action a in state s, γ is the discount factor that balances immediate and future rewards, and $s\prime$ is the state following the action. By iteratively applying this update, the algorithm converges to an optimal policy that maximizes the expected cumulative reward. In cases where the number of possible states and actions is low, it is possible to store the Q-value table explicitly. However, in our case, the state-action space is too large to store explicitly. Hence, we employ deep-Q-learning, where a deep-learning model approximates the Q-values. Instead of a table that can be indexed by (s, a), we have a deep-learning model that outputs the Q-value for that state action pair when given (s, a). We use the same model architecture as the GDM model in Section 3.1, except that we remove the last sigmoid (as Q-values are not bounded between 0 and 1) and append an additional linear layer at the end of the model.

During training, the estimator is trained by minimizing the difference between the predicted and target Q values with gradient descent. The target Q-value is computed using the Bellman equation:

$$Q_t(s, a) \leftarrow r + \gamma \max_{a'} Q(s', a') \tag{2}$$

where s' is the next state.

We use the mean-squared error (MSE) Loss to minimize the difference between the predicted and target Q-values. In our setting, the state is the graph, and the action is the node to remove from the graph. Let lcc be the function that, given a graph G, returns the size of the largest connected component $lcc(G)$. Then, we set our reward function as:

$$r = lcc(G) - lcc(G \backslash \{a\} - 1)$$

This represents the measurable improvement in the model's dismantling capabilities, with the -1 added to add a cost to removing a node. The additional -1 to the reward punishes the model for removing nodes that have no effect on the largest connected component.

4.2.2. Results

Models and data: We evaluate our method against GDM on a 10-graph subset of the test set used in the original GDM paper (Grassia et al., 2021). We use the same model architecture for our and GDM's method, described in the third row of Table 1. For our method, we train a model for each graph. During training, we save the models with the best removal count with and without reinsertion. For GDM, we train three models: one that is validated against only one of the graphs of the 10-graph subset (the corruption network), one that is validated against the first five graphs in alphabetical order, and finally, a model that is validated against all ten models. This matters because we save the models with the best validation removal count with and without reinsertion throughout training (we validate after

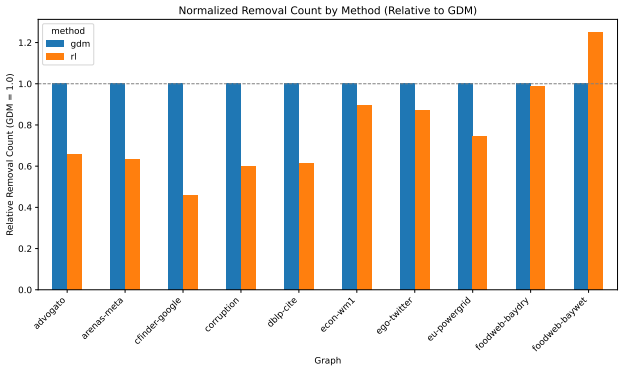

Figure 1. Normalized removal count by method (relative to GDM). We name our method RL. Lower is better.

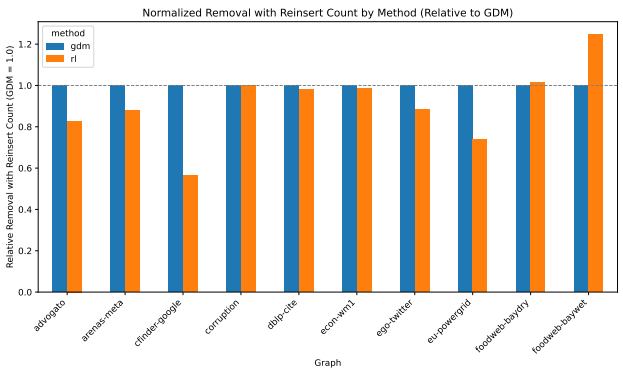

Figure 2. Normalized removal count *with reinsertion* by method (relative to GDM). We name our method RL. Lower is better.

every training epoch).

Metrics: We measure the *removal count*: the number of nodes removed to reach the largest connected component (LCC) size threshold. We also measure the *removal count with reinsertion*, which is the removal count after reinserting the nodes that do not affect the LCC size as specified in Section 3.3. We use the same LCC size threshold as previous work: 10% of the original LCC size.

Results: In Figure 1 and Figure 2, we compare our method (named RL) against GDM in normalized removal count and normalized removal with reinsertion count, respectively. For most graphs, we find that our method improves over GDM for both metrics, requiring significantly fewer nodes than GDM to dismantle the graph. Our method is worse than GDM for one graph (foodweb-baywet) in removal count and two graphs (foodweb-baydry and foodweb-baywet) in removal count with reinsertion. We find that our method tends to outperform GDM on larger graphs. This aligns with the fact that the GDM model is trained on small synthetic networks, compared to our method, which is trained against the target graph networks directly. To measure this finding, we present in Figure 3 and Figure 4 the removal count improvement provided over GDM by our method with respect to the number of nodes in the graph.

To conclude, we improve on other machine learning-based network dismantling attacks using reinforcement learning.

5. Defense

In Section 3 and Section 4, we showed that these network attacks work, and we improved on previous work by presenting a more node-efficient attack. The literature has a dire lack of mitigation methods against these attacks. As such, we study whether one can defend a network against dismantling attacks. It is impossible to completely prevent

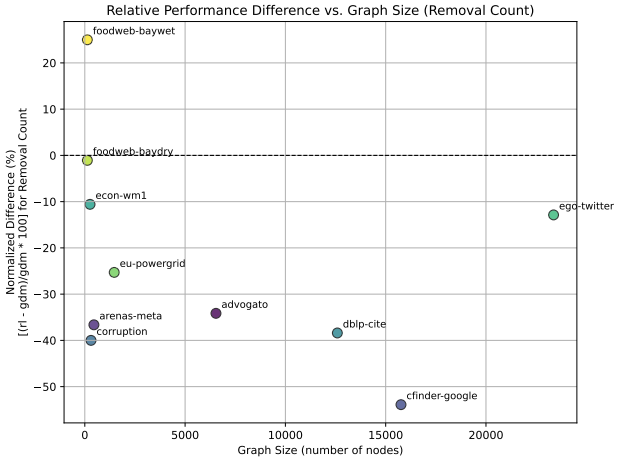

Figure 3. Relative removal count difference (in %) depending on the graph size (number of nodes).

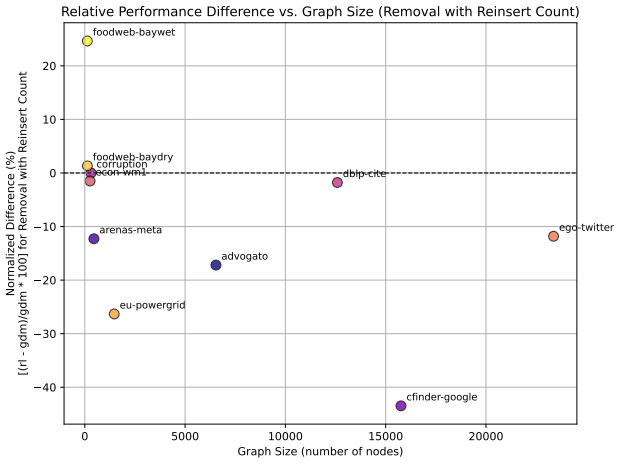

Figure 4. Relative removal count with reinsertion difference (in %) depending on the graph size (number of nodes).

an attacker from dismantling a network, as an attacker could always remove all the nodes in the network. Therefore, we study how to increase the number of node removals required for a given attacker to dismantle the network. In particular, we create a model that learns to efficiently add edges to a graph to disrupt the removal process of a given attacker. We train this next-link-prediction model against a given attacker (or attack model in the case of GDM) using deep-Q-learning. We could also study methods to add nodes instead, but a trivial solution of adding a node with edges to every other node appears to be the best solution in such a case.

5.1. Defense Model Architecture

We re-use the architecture from Section 3.1 with a few modifications. We use the GAT layers of the GDM model as an encoder for the nodes to generate node embeddings. We then provide every pair of node embeddings to the second part of the model that estimates the Q-value of adding an edge between the two nodes (action) given the original input graph (state). This second part of the model is a 2-layer linear model with a relu activation function.

5.2. Defense Model Training

We train our defense model using deep-Q-learning. We train our defense model as a Q-value estimator similarly to Section 4.2, with Equation 2 as our target Q-value and the MSE loss. However, the action space is different. Instead of an action representing removing a node, it represents adding an edge to the graph. To reward the model, we compare the removal count by the attacker before and after the edges were added to the graph. The final reward function is the weighted sum of the removal count difference with and without reinsertion, with parameters α and β tuning the weight of each component, respectively.

5.3. Experiments

We train our model to add $0.001 * m$ directed edges to the graph (0.1% of the original number of directed edges). Due to time constraints, our experimentation was relatively limited, and we could only test a few sets of hyper-parameters against the corruption network. However, our results look promising. We train our defense model against the GDM model from Section 4.2.2 validated against ten graphs. This model must remove 71 nodes (out of 309) from the Brazilian corruption network to dismantle it and 29 nodes with reinsertion. In the best case, our defense model increases that by 155 nodes and 61 nodes with reinsertion for a total of 226 and 90 nodes without and with reinsertion, respectively. This additional number of nodes requiring removal is reached by only adding six directed edges to the network ($\lfloor 0.001 * 6562 \rfloor$ edges) In Figure 5, we provide the defense success throughout the training process, where we measure

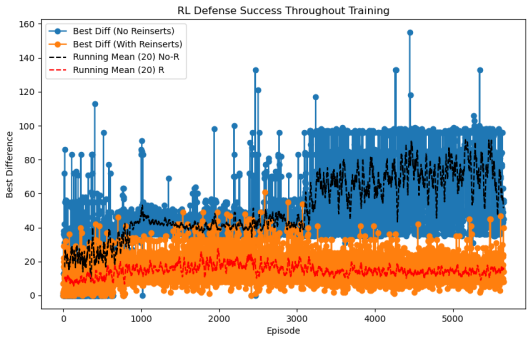

Figure 5. RL defense method success throughout training on the Brazilian corruption network. Success is measured as the number of additional nodes the attacker must remove to dismantle the network with the additional edges. $(\alpha, \beta) = (1, 1)$.

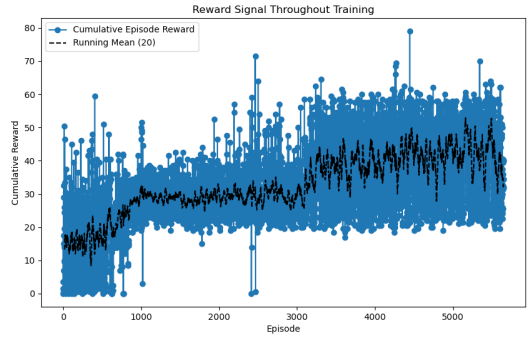

Figure 6. Reward signal throughout training on the Brazilian corruption network. $(\alpha, \beta) = (1, 1)$.

success by the additional number of nodes required to dismantle the network for the attacker. This training was done with $\alpha = 1$ and $\beta = 1$ and shows that the model can effectively learn to defend the network against attacks without reinsertion. In Figure 6, we provide the reward signal received throughout training. To show that the defense model can also learn to defend against attacks with reinsertion, we present a shorter run in Figure 7.

References

Artime, O., Grassia, M., De Domenico, M., Gleeson, J. P., Makse, H. A., Mangioni, G., Perc, M., and Radicchi, F. Robustness and resilience of complex networks. *Nature Reviews Physics*, 2024. doi: 10.1038/s42254-023-00676-y. URL https://doi.org/10.1038/s42254-023-00676-y.

Braunstein, A., Dall'Asta, L., Semerjian, G., and

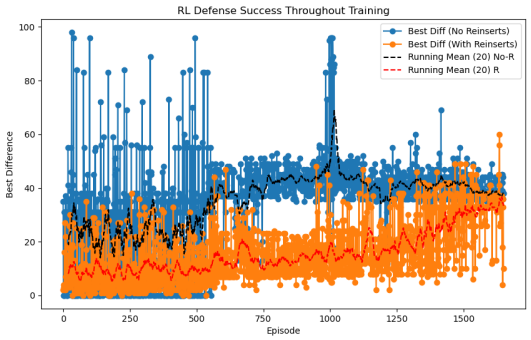

Figure 7. RL defense method success throughout training on the Brazilian corruption network, shorter run with a different set of hyper-parameters. Success is measured as the number of additional nodes the attacker must remove to dismantle the network with the additional edges. $(\alpha, \beta) = (1, 1)$.

Zdeborová, L. Network dismantling. *Proceedings of the National Academy of Sciences*, 113(44):12368–12373, 2016. doi: 10.1073/pnas.1605083113. URL https://www.pnas.org/doi/abs/10.1073/pnas.1605083113.

Clusella, P., Grassberger, P., Pérez-Reche, F. J., and Politi, A. Immunization and targeted destruction of networks using explosive percolation. *Phys. Rev. Lett.*, 117: 208301, Nov 2016. doi: 10.1103/PhysRevLett.117.208301. URL https://link.aps.org/doi/10.1103/PhysRevLett.117.208301.

Cohen, R., Erez, K., Ben-Avraham, D., and Havlin, S. Breakdown of the internet under intentional attack. *Physical review letters*, 86(16):3682, 2001.

Fan, C., Zeng, L., Sun, Y., and Liu, Y.-Y. Finding key players in complex networks through deep reinforcement learning. *Nature Machine Intelligence*, 2(6): 317–324, Jun 2020. ISSN 2522-5839. doi: 10.1038/s42256-020-0177-2. URL https://doi.org/10.1038/s42256-020-0177-2.

Grassia, M. and Mangioni, G. Coregdm: geometric deep learning network decycling and dismantling. In *International Workshop on Complex Networks*, pp. 86–94. Springer, 2023.

Grassia, M., De Domenico, M., and Mangioni, G. Machine learning dismantling and early-warning signals of disintegration in complex systems. *Nature communications*, 12(1):5190, 2021.

Huang, Y., Wang, H., Ren, X.-L., and Lü, L. Identifying key players in complex networks via network entanglement. *Communications Physics*, 7(1):19, 2024. doi: 10.1038/s42005-023-01483-8. URL https://doi.org/10.1038/s42005-023-01483-8.

Lowerre, B. P. and Reddy, B. R. Harpy, a connected speech recognition system. *The Journal of the Acoustical Society of America*, 59(S1):S97–S97, 1976.

Morone, F. and Makse, H. A. Influence maximization in complex networks through optimal percolation. *Nature*, 524(7563):65–68, 2015.

Morone, F., Min, B., Bo, L., Mari, R., and Makse, H. A. Collective influence algorithm to find influencers via optimal percolation in massively large social media. *Scientific Reports*, 6(1):30062, Jul 2016. ISSN 2045-2322. doi: 10.1038/srep30062. URL https://doi.org/10.1038/srep30062.

Ren, X.-L. and Antulov-Fantulin, N. Ensemble approach for generalized network dismantling. In Cherifi, H., Gaito, S., Mendes, J. F., Moro, E., and Rocha, L. M. (eds.), *Complex Networks and Their Applications VIII*, pp. 783–793, Cham, 2020. Springer International Publishing. ISBN 978-3-030-36687-2.

Ren, X.-L., Gleinig, N., Helbing, D., and Antulov-Fantulin, N. Generalized network dismantling. *Proceedings of the National Academy of Sciences*, 116(14):6554–6559, 2019. doi: 10.1073/pnas.1806108116. URL https://www.pnas.org/doi/abs/10.1073/pnas.1806108116.

Zdeborová, L., Zhang, P., and Zhou, H.-J. Fast and simple decycling and dismantling of networks. *Scientific reports*, 6(1):37954, 2016.