Attacking Graph Neural Networks at Scale

Simon Geisler, Daniel Zügner, Aleksandar Bojchevski, Stephan Günnemann
Department of Informatics
Technical University of Munich
{geisler, zuegnerd, a.bojchevski, guennemann}@in.tum.de

Abstract

Adversarial robustness of Graph Neural Networks (GNNs) has become exceedingly important due to the popularity and diverse applications of GNNs. Specifically, structure perturbations are very effective, but designing attacks that add or remove edges is difficult because of the discrete optimization domain. Existing adversarial attacks for structure perturbations that rely on first-order optimization require a dense adjacency matrix and, therefore, can only be applied to small graphs (space complexity $\Theta(n^2)$ in the number of nodes n). PubMed is among the largest graphs adversarial attacks and defenses have been evaluated on. However, PubMed is tiny in the context of many real-world applications. In this work-in-progress report, we propose three attacks based on first-order optimization that do not require a dense adjacency matrix and, hence, can be used on graphs more than 100 times larger than previously evaluated. Moreover, one of the proposed attacks considers the very practical case of node injection.

1 Introduction

Attacks based on combinatorial approaches easily become computationally infeasible because of the vast amount of potential adjacency matrices (2^{n^2}). With first-order optimization we can approximate the discrete optimization problem. First-order optimization attacks typically require the gradient towards the entries in the adjacency matrix and, hence, reduce the complexity to $\Theta(n^2)$. To attack a small dataset such as PubMed (19717 nodes), typically more than 20 GB are required if using a dense adjacency matrix. We argue that such memory requirements are still impractical and hinder practitioners to assess adversarial robustness. We identify the necessity to research scalable attacks for GNNs, due to the lack of such attacks for real-world graphs

We propose two strategies how one may apply first-order optimization, without the burden of a dense adjacency matrix. In Section 3 we propose an attack that adds adversarial nodes and was one of the top 5 attacks of the KDD Cup 2020 (Biendata 2020). Thereafter in Section 4, we describe how one might add/remove edges between existing nodes based on Randomized Block Coordinate Descent (R-BCD).

Even though our attacks can be generalized to graph classification, we focus on the important task of node classification.

In this work, we set the foundation for the study of adversarial robustness of GNNs on real-world citation/social networks. Importantly, it is unknown how the adversarial robustness/vulnerability relates to the graph size. In our experiments (Section 5), we point out that a GNN applied to a larger graph is more fragile w.r.t. structure perturbations.

Our contributions are the following:

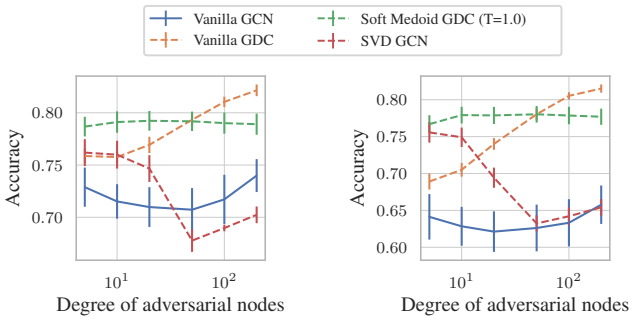
- We propose one scalable adversarial attack that adds adversarial nodes (space complexity of $\Theta(n)$).
- We propose two scalable adversarial attacks that add/remove edges between the existing nodes. One relies on projected gradient descent and the other uses a greedy FGSM-like optimization scheme (space complexity of $\Theta(m)$ in the number of edges m).
- We study the adversarial robustness on graphs substantially larger than PubMed. Empirically, we find that the graph size negatively related to the adversarial robustness.

2 Related Work

Large scale optimization. In a big data setting, the cost to calculate the gradient towards all variables can be prohibitively high. For this reason, coordinate descent has gained importance in machine learning and large scale optimization (Wright 2015). Nesterov (2012) proposed (and analyzed the convergence) of Randomized Block Coordinate Descent (R-BCD). For R-BCD's pseudo code see Algorithm 1. In R-BCD only a subset of variables is optimized at a time and, hence, only the gradients towards those variables are required. In many cases, this allows for a lower memory footprint and in some settings even converges faster than standard methods (Nesterov and Stich 2017). Constrained optimization with first-order methods can be solved with

Algorithm 1 R-BCD (Nesterov 2012)

1: Choose $\boldsymbol{x}_0 \in \mathbb{R}^d$
2: **for** $k \in \{1, 2, \dots, K\}$ **do**
3: Draw random indices $\boldsymbol{i}_k \in \{0, 1, \dots, n\}^b$
4: $\boldsymbol{x}_k \leftarrow \boldsymbol{x}_{k-1} - \alpha_k \nabla_{\boldsymbol{i}_k} \mathcal{L}(\boldsymbol{x}_{k-1})$
5: **end for**



(a) (b)

Figure 1: Influence of the degree of the added nodes on the perturbed accuracy on Cora ML. (a) shows the perturbed accuracy with a budget of changing $\epsilon = 0.1$ edges, and (b) for $\epsilon = 0.25$. The number of nodes is determined as $\Delta_n = \epsilon/\Delta_e$. We report the mean perturbed accuracy and its three-sigma error over five random seeds.

methods such as Projected Gradient Descent (PGD) or Fast Gradient Sign Method (FGSM) (Goodfellow, Shlens, and Szegedy 2015). Similarly, one can extend R-BCD to constrained optimization (Nesterov 2012).

Adversarial attacks. Beginning with (Dai et al. 2018; Zügner, Akbarnejad, and Günnemann 2018), many adversarial attacks on the graph structure have been proposed (Zügner and Günnemann 2019; Xu et al. 2019; Bojchevski and Günnemann 2019; Wu et al. 2019; Wang and Gong 2019; Tang et al. 2020). We limit the scope to an adversary with perfect knowledge about the graph, GNN and test labels. Even this white-box scenario has not been studied for large graphs and our primary goal is to asses adversarial robustness. Dai et al. (2018) scale their reinforcement learning approach to a very sparse, large-scale graph for financial transactions. In contrast to our work, they only use a very small budget Δ (their time complexity scales linearly with Δ). Wang and Gong (2019) also scale their attack to a larger graph than PubMed but they do not attack GNNs. First-order optimization attacks such as Metattack (Zügner and Günnemann 2019) or integrated gradients (Wu et al. 2019) rely on the gradient towards all possible entries in the *dense* adjacency matrix $\boldsymbol{A}$ (quadratic space complexity) to solve the optimization problem for structure perturbations:

$$\underset{\boldsymbol{A}}{\text{maximize}}\, \mathcal{L}(f_\theta(\boldsymbol{A}, \boldsymbol{X})) \tag{1}$$

with the adjacency matrix $\boldsymbol{A}$, node features $\boldsymbol{X}$, loss $\mathcal{L}$, and the trained network f_θ.

3 Adding Adversarial Nodes

We solve the attack optimization problem via adding nodes

$$\underset{\boldsymbol{A}', \boldsymbol{X}'}{\text{maximize}}\, \mathcal{L}(f_\theta(\boldsymbol{A}|\boldsymbol{A}', \boldsymbol{X}|\boldsymbol{X}')) \tag{2}$$

with the space complexity of $\mathcal{O}(m)$ (on top of the complexity of the attacked model; in the following we will neglect this fact). With $\boldsymbol{A}|\boldsymbol{A}'$ we denote the addition of rows & columns and with $\boldsymbol{X}|\boldsymbol{X}'$ the concatenation of the respective attributes ($\boldsymbol{A}$ & $\boldsymbol{X}$ are const.). For imperceptibility, we further limit the number of nodes Δ_n and their degree Δ_e.

Algorithm 2 Greedy Adversarial Node Generation (GANG)

Input: Adj. $\boldsymbol{A}$, feat. $\boldsymbol{X}$, labels $\boldsymbol{y}$, GNN $f_\theta(\boldsymbol{A}, \boldsymbol{X})$, loss $\mathcal{L}$
Parameter: budgets Δ_n & Δ_e, step size s_e, steps features s_x
Initialize empty $\boldsymbol{A}'$ and $\boldsymbol{X}'$
for $k \in \{1, \dots, \Delta_n\}$ **do**
  $\boldsymbol{A}' \leftarrow$ concatenate new node to $\boldsymbol{A}'$ (empty row and column)
  $\boldsymbol{X}' \leftarrow$ concatenate $\boldsymbol{X}'$ vector $\tilde{\boldsymbol{x}}_k \sim \Pi(\mathcal{N}(0, \sigma_n^2))$
  for $j \in \{0, \dots, \Delta_e/s_e\}$ **do**
    $\hat{\boldsymbol{y}} \leftarrow f_\theta(\boldsymbol{A}|\boldsymbol{A}', \boldsymbol{X}|\boldsymbol{X}')$
    $g \leftarrow \nabla_{\boldsymbol{A}'_k} \mathcal{L}(\hat{\boldsymbol{y}}, \boldsymbol{y})$ for all nodes where $\hat{\boldsymbol{y}} = \boldsymbol{y}$
    $\boldsymbol{A}' \leftarrow$ add the top s_e edges to $\boldsymbol{A}'$ w.r.t. g
  end for
  for $j \in \{1, \dots, s_x\}$ **do**
    $\boldsymbol{X}' \leftarrow \Pi(\tilde{\boldsymbol{X}} + \alpha_x \nabla_{\boldsymbol{X}'} \mathcal{L}(f_\theta(\boldsymbol{A}|\boldsymbol{A}', \boldsymbol{X}|\boldsymbol{X}'))$
  end for
end for

The attacks add one node (or a small group of nodes) at a time to the sparse adjacency matrix and connect it to every other node with edge weight zero. Subsequently, we perform a constrained gradient-based optimization to determine the best edges with a given budget. We decide to add nodes in an greedy manner. For each new node, we determine the edges in s_e steps via a greedy FGSM-like procedure (PGD is an alternative). Then, the initial features (randomly sampled) are optimized via PGD (s_x epochs). In Algorithm 2, we give a formal definition of GANG.

In Fig. 1 we analyze the influence of the degree of the added nodes via GANG. Interestingly, low degree nodes seem to be more effective than high degree nodes. This could be due to the normalization by the square root of the inverse node degree for a GCN (Kipf and Welling 2017). Surprisingly, we find that especially GDC (Klicpera, Weißenberger, and Günnemann 2019) (i.e. personalized PageRank) and a low-rank SVD approximation (Entezari et al. 2020) are effective defenses (prepossessing techniques of the adjacency matrix). SVD is a strong defense against low-degree nodes and personalized page rank is particularly strong against high degree nodes. The recent defense Soft Medoid GDC (Geisler, Zügner, and Günnemann 2020), seems to be effective regardless of the node degree.

4 Adding and Removing Edges

In this section we discuss the case where the attack vector is to perturb the existing, binary graph structure:

$$\underset{\boldsymbol{P} \text{ s.t. } \sum \boldsymbol{P} \leq \Delta}{\text{maximize}}\, \mathcal{L}(f_\theta(\boldsymbol{A} \oplus \boldsymbol{P}, \boldsymbol{X}))\,. \tag{3}$$

Here, $\oplus$ stands for an element-wise exclusive or and Δ denotes the edge budget (i.e. the number of altered entries in the perturbed adjacency matrix). In the following, we use set $\mathbb{P}$ and matrix notation $\boldsymbol{P}$ for the sparse perturbations $\mathbb{P}$ interchangeably. Naively, applying R-BCD to optimize towards the dense adjacency matrix would only save some computation on obtaining the respective gradient, but still has a space

complexity of $\mathcal{O}(n^2)$. Note that in R-BCD we interpret each possible entry in the perturbation set $\mathbb{P}$ as one dimension of our optimization problem. To mitigate the quadratic complexity, we make use of the fact that the solution is going to be sparse (L_0 perturbation). As in evolutionary algorithms, in each epoch, we keep that part of the search space which is promising and resample the rest. We can view the underlying problem as a combination of L_0-norm PGD and an adaptive version of Randomized Block Coordinate Descent (R-BCD). R-BCD comes with a space complexity of $\Theta(m)$, as we typically choose Δ to be a fraction of m. We give a formal definition of Projected and Randomized Block Coordinate Descent (PR-BCD) in Algorithm 3.

As proposed by Xu et al. (2019), $\boldsymbol{p}$ is interpreted as the probability mass for each potential entry in the perturbation set $\mathbb{P}$ and is used in the end to sample the final edge additions/removals. In each epoch $e \in \{1, 2, \ldots, E\}$, this probability mass $\boldsymbol{p}$ is used as edge weight. The projection $\Pi_{\mathbb{E}[\text{Bernoulli}(\boldsymbol{p})]=\Delta}(\boldsymbol{p})$ adjusts the probability mass such that $\mathbb{E}[\text{Bernoulli}(\boldsymbol{p})] = \sum_{i \in b} p_i \approx \Delta$ and that $\boldsymbol{p} \in [0, 1]$ (line 8). If using an undirected graph, the potential edges are restricted to the the upper/lower triangular $n \times n$ matrix. In the end we sample $\mathbb{P} \sim \text{Bernoulli}(\boldsymbol{p})$ (line 16).

Note that the projection of the perturbation set $\Pi_{\mathbb{E}[\text{Bernoulli}(\boldsymbol{p})]=\Delta}(\boldsymbol{p})$ contains many zero elements, but is not guaranteed to be sparse. If $\boldsymbol{p}$ has more than 50% non-zero entries, we remove the entries with the lowest probability mass such that 50% of the search space is resampled. Otherwise, we resample all zero entries in $\boldsymbol{p}$. However, one also might apply a more sophisticated heuristic $h(\boldsymbol{p})$ (see line 11).

With growing n it is unrealistic that each possible entry of the adjacency matrix was part of at least one random search space of R-BCD. Apparently, with a constant search space size, the number of mutually exclusive chunks of the perturbation set grows with $\Theta(n^2)$ and this would imply a quadratic runtime. However, as evident in randomized black-box attacks (Waniek et al. 2018), it is not necessary to test every possible edge to obtain an effective attack. In Fig. 2, we analyze the influence of the random block size on the perturbed accuracy. For a sufficient block size b our method performs comparably to its dense equivalent.

We keep the random block fixed and run K_{resample} epochs. Thereafter, we decay the learning rate as in (Xu et al. 2019). We also employ early stopping for both stages ($k \leq K_{\text{resample}}$ and $k > K_{\text{resample}}$ with the epoch k) such that we take the result of the epoch with highest loss $\mathcal{L}$.

We also compare this approach to a Greedy R-BCD (GR-BCD), that greedily flips the entries with largest gradient in the random search space, such that after K iterations the budget requirements are met.

If we performed a targeted instead of a global attack, we could further reduce the space requirements since we only need to consider the node's "receptive field". Such an attack could be very similar to adding one node in the introduced GANG attack with further constraints (see Section 3).

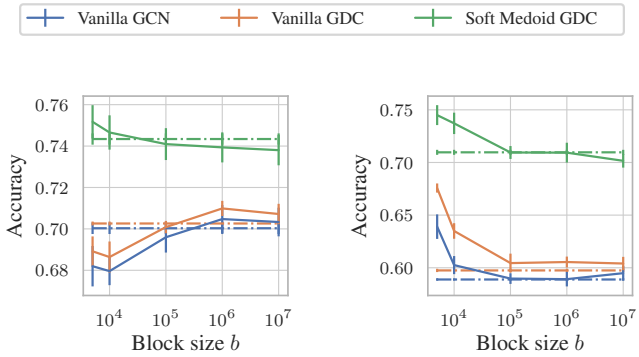


Figure 2: We perform the proposed PR-BCD (solid lines) to obtain a perturbed adjacency matrix with the fraction of perturbed edges $\epsilon = 0.25$ (i.e. $\Delta = 1995$ edges). We run 50 epochs where we resample the search space and subsequently fine-tune for 250 epochs. The dashed lines show the performance of vanilla PGD (Xu et al. 2019). We report the mean and its three-sigma error over five random seeds.

5 Empirical Evaluation

In the following, we present our experiments to show the effectiveness and scalability of our proposed attacks. We first describe our setup and then discuss the results over wide range of graphs of different scale.

Defenses. We also report the results on state of the art defenses of (Entezari et al. 2020; Geisler, Zügner, and Günnemann 2020; Wu et al. 2019; Zhu et al. 2019). For the Soft Medoid GDC, we use the temperature $T = 0.5$ as it is a good compromise between accuracy and robustness. The SVD GCN (Entezari et al. 2020) uses a (dense) low-rank approximation (here rank 50) of the adjacency matrix to filter adversarial perturbations. RGCN (Zhu et al. 2019) models the neighborhood aggregation via Gaussian distribution to filter outliers, and Jaccard GCN (Wu et al. 2019) filters edges based on attribute dissimilarity (here threshold 0.01).

Attacks. We compare our GANG, PR-BCD, and GR-BCD attacks (see Sections 3-4) with the global DICE (Waniek et al. 2018), PGD (Xu et al. 2019), and greedy FGSM attacks Geisler, Zügner, and Günnemann (2020). The greedy FGSM-like attack is the dense equivalent of our GR-BCD attack with the exception of flipping one edge at a time. DICE is a greedy, randomized black-box attack that flips one randomly determined entry in the adjacency matrix at a time. An edge is deleted if both nodes share the same label and an edge is added if the labels of the nodes differ. We ensure that a single node does not become disconnected. Moreover, we use 60% of the budget to add new edges and otherwise remove edges.

Datasets. We use the common Cora ML (Bojchevski and Günnemann 2018), Citeseer (McCallum et al. 2000), and PubMed (Sen et al. 2008) for comparing against the other state of the art attacks. For large scale experiments, we use two graphs of the recent Open Graph Benchmark (Hu et al. 2020). In comparison to Pubmed, we scale by 2.5 orders of magnitude (number of nodes), or by factor 15,000 if count-

Table 1: Perturbed accuracy for the proposed attacks (see Sections 3-4) and baselines on all datasets (see Table 2). ϵ denotes the fraction of edges perturbed (relative to the clean graph). The last column contains the clean accuracy. As this a work-in-progress report, the experiments for the defenses on the large datasets are due and on Products we did not optimize the hyperparameters for GANG. For each architecture we italicize the strongest attack where $\epsilon = 0.05$, underline where $\epsilon = 0.1$, and embolden where $\epsilon = 0.25$. From an attack perspective, a lower perturbed accuracy is better. We rerun the experiments with three different seeds. For OGB we use the provided data splits and otherwise we use random split with 20 nodes per class.

	Attack		**DICE**			**GANG (ours)**			**greedy FGSM**			**GR-BCD (ours)**			**PGD**			**PR-BCD (ours)**			**Accuracy**
	Frac. edges ϵ	0.05	0.1	0.25	0.05	0.1	0.25	0.05	0.1	0.25	0.05	0.1	0.25	0.05	0.1	0.25	0.05	0.1	0.25	
Cora ML	Vanilla GCN	0.813	0.806	0.766	0.766	0.732	0.658	0.758	0.719	0.635	0.763	0.724	0.638	0.755	0.705	**0.590**	*0.753*	0.711	0.598	0.825
	Vanilla GDC	0.821	0.815	0.777	0.788	0.762	0.712	0.759	0.718	0.639	0.765	0.724	0.642	0.755	0.704	**0.595**	*0.754*	0.710	0.607	0.831
	SVD GCN	0.749	0.736	0.682	0.764	0.760	0.722	0.741	0.715	0.640	0.742	0.719	0.647	*0.735*	0.696	**0.603**	0.736	0.714	0.610	0.761
	Jaccard GCN	0.808	0.802	0.770	0.788	0.768	0.737	0.760	0.722	0.643	0.765	0.729	0.644	0.756	0.708	**0.601**	*0.753*	0.716	0.611	0.819
	RGCN	0.791	0.785	0.756	*0.701*	0.674	0.603	0.743	0.708	0.631	0.746	0.711	0.638	0.739	0.699	**0.594**	0.741	0.704	0.603	0.800
	Soft Medoid GDC	0.813	0.808	0.794	*0.769*	0.765	0.755	0.772	0.743	0.679	0.773	0.745	**0.678**	0.774	0.743	0.683	0.770	0.742	0.689	0.817
Citeseer	Vanilla GCN	0.702	0.692	0.670	0.675	0.644	0.593	0.669	0.632	0.540	0.659	0.611	**0.523**	0.660	0.618	0.526	*0.657*	0.616	**0.523**	0.712
	Vanilla GDC	0.694	0.684	0.657	0.686	0.662	0.630	0.659	0.622	0.533	0.650	0.606	0.517	0.655	0.610	0.517	*0.647*	0.605	**0.515**	0.709
	SVD GCN	0.630	0.615	0.573	0.627	0.420	0.539	0.624	0.602	**0.363**	0.623	0.604	0.518	*0.614*	0.593	0.490	0.615	0.581	0.488	0.641
	Jaccard GCN	0.708	0.700	0.677	0.705	0.698	0.691	0.680	0.648	0.573	0.675	0.641	0.568	0.672	0.639	**0.554**	*0.670*	0.641	0.565	0.714
	RGCN	0.634	0.622	0.596	0.624	0.601	0.543	0.608	0.577	0.513	*0.602*	0.574	0.502	0.607	0.584	0.504	0.607	0.578	**0.498**	0.646
	Soft Medoid GDC	0.704	0.701	0.693	0.704	0.700	0.695	*0.685*	0.666	**0.620**	0.686	0.667	0.625	*0.685*	0.671	0.628	*0.685*	0.669	0.638	0.707
PubMed	Vanilla GCN	0.757	0.747	0.708	*0.702*	0.688	0.566	-	-	-	0.725	0.679	0.590	-	-	-	0.714	0.661	**0.534**	0.774
	Vanilla GDC	0.759	0.749	0.709	0.727	0.719	0.662	-	-	-	0.728	0.677	0.588	-	-	-	*0.712*	0.661	**0.563**	0.776
arXiv	Vanilla GCN	0.689	0.669	0.609	0.516	0.482	**0.291**	-	-	-	0.547	0.482	0.381	-	-	-	*0.503*	0.427	0.304	0.705
	Vanilla GDC	0.573	0.538	0.474	0.609	0.607	0.604	-	-	-	0.483	0.436	0.364	-	-	-	*0.446*	0.392	**0.305**	0.618
Products	Vanilla GCN	0.674	0.636	0.545	0.659	0.594	0.390	-	-	-	0.555	0.494	**0.388**	-	-	-	*0.535*	0.480	0.450	0.719
	Vanilla GDC	0.677	0.657	0.618	0.700	0.695	0.685	-	-	-	0.551	0.511	**0.442**	-	-	-	*0.543*	0.510	0.492	0.709

Algorithm 3 Projected and Randomized Block Coordinate Descent (PR-BCD)

Input: Adj. $\boldsymbol{A}$, feat. $\boldsymbol{X}$, labels $\boldsymbol{y}$, GNN $f_\theta(\boldsymbol{A}, \boldsymbol{X})$, loss $\mathcal{L}$
Parameter: budget Δ, block size b, epochs K, heur. $h(\dots)$
Draw random indices $\boldsymbol{i}_0 \in \{0, 1, \dots, N\}^b$
Initialize zeros for $\boldsymbol{p}_0 \in \mathbb{R}^b$
for $k \in \{1, 2, \dots, K\}$ **do**
  $\hat{\boldsymbol{y}} \leftarrow f_\theta(\boldsymbol{A} \oplus \boldsymbol{p}_{k-1}, \boldsymbol{X})$
  $\boldsymbol{p}_k \leftarrow \boldsymbol{p}_{k-1} + \alpha_{k-1} \nabla_{\boldsymbol{i}_{k-1}} \mathcal{L}(\hat{\boldsymbol{y}}, \boldsymbol{y})$
  Projection $\boldsymbol{p}_k \leftarrow \Pi_{\mathbb{E}[\text{Bernoulli}(\boldsymbol{p}_k)]=\Delta}(\boldsymbol{p}_k)$
  $\boldsymbol{i}_k \leftarrow \boldsymbol{i}_{k-1}$
  if $k \leq K_{\text{resample}}$ **then**
    $\text{mask}_{\text{resample}} \leftarrow h(\boldsymbol{p}_k)$
    $\boldsymbol{p}_k[\text{mask}_{\text{resample}}] \leftarrow \boldsymbol{0}$
    Resample $\boldsymbol{i}_k[\text{mask}_{\text{resample}}] \in \{0, 1, \dots, N\}^{|\text{mask}_{\text{resample}}|}$
  end if
end for
$\boldsymbol{P} \sim \text{Bernoulli}(\boldsymbol{p}_k)$ s.t. $\sum \boldsymbol{P} \leq \Delta$
Return $\boldsymbol{A} \oplus \boldsymbol{P}$

ing the possible adjacency matrix entries (see Table 2). For Cora, Citeseer, as well as PubMed we use an 11GB GeForce GTX 1080 Ti and a 32GB Tesla V100 otherwise. We exclusively perform the calculations on GPU, but for products where we coalesce the adjacency matrix on CPU.

Checkpointing. Empirically, almost 30 GB are required to train a three-layer GCN on Products (our largest dataset) using sparse matrices. However, obtaining the gradient, e.g. towards the perturbation set, requires extra memory. We notice that most operations in modern GNNs only depend on the neighborhood size (i.e. a row in the adjacency matrix). As proposed by Chen et al. (2016), the gradient is obtainable with sublinear memory cost via checkpointing. The idea is to discard some intermediate results in the forward phase and recompute them in the backward phase. Specifically, we chunk some operations (e.g. matrix multiplication) within the message passing step to successfully scale to larger graphs. This allows us to attack a three-layer GCN on Products with full GPU acceleration.

Table 2: Statistics of the used datasets. For the dense adjacency matrix we assume that each element is represented by 4 bytes. In the sparse case we use two 8 byte integer pointers and a 4 bytes float value.

Dataset	**#Nodes** n	**#Edges** e	**#Features** d	**Size (dense)**	**Size (sparse)**
Cora ML	2,810	15,962	2,879	31.58 MB	319.24 kB
Citeseer	2,110	7,336	3,703	17.81 MB	146.72 kB
PubMed	19,717	88,648	500	1.56 GB	1.77 MB
arXiv	169,343	1,166,243	128	114.71 GB	23.32 MB
Products	2,449,029	123,718,280	100	23.99 TB	2.47 GB

Hyperparameters. We use the same setup as Geisler, Zügner, and Günnemann (2020) in their evaluation and for models on OGB we follow Hu et al. (2020), For the attacks GR-BCD and PR-BCD, we run the attack for 500 epochs (100 epochs fine-tuning with PR-BCD). We choose the search space size to be at least twice the edge perturbation ratio ϵ (depends on the dataset). Since the edge budget Δ_e in GANG influences the perturbed accuracy (see Figure 1), we decide for a relatively low value of 250. As these are preliminary results, the only exception is Products on which we report the results with the budget of $\Delta_e = 25,000$. Moreover with GANG we binarize the attributes on Cora ML, Citeseer and PubMed and use L_0-norm PGD analogously to PR-BCD.

Results overview. In Table 1 we present the preliminary experimental results for our proposed attacks. We do not observe that sampling the search space harms the attack strength. Similarly to Figure 2, we even outperform the dense PGD on Cora ML. We conclude that our attacks are as effective as the other state of the art attacks.

GNNs' fragility on large graphs. In the following, we analyze the results of PR-BCD, with a budget of $\epsilon = 0.25$, and the GCN. We observe a relative drop in the perturbed

accuracy by 20% on Cora, 25% on PubMed, 33 % on arXiv. On products with the lower budget of $\epsilon = 0.1$, we already see a drop of the perturbed accuracy of 31%. We conclude that there is likely some relationship between the fragility and the graph size. This relationship is similar for GR-BCD but much stronger for GANG. However, for GANG we have to consider that we may choose the attributes as well. arXiv as well as Products have *dense* continuous features, and all the other datasets have *sparse* continuous features. This relationship seems to persist for architectures other than GCN as well. Please note that further experiments are required to confirm this hypothesis. For example, on arXiv and products we use a three-layer GCN (to achieve state of the art accuracy) and for the other datasets we use just two layers. Moreover, the datasets have a different number of classes.

Time and memory cost. On arXiv, we train for 500 epochs and run the PR-BCD attack for 500 epochs. The whole training and attacking procedure requires less than 2 minutes and the peak usage of GPU memory is below 2.5 GB. Note that only loading the adjacency matrix for traditional attacks (no training etc.) would require around 115 GB (see Table 2). Naively attacking the dense adjacency matrix would certainly require more than 1 TB.

6 Conclusion

We propose three new attacks that all have the potential to scale to much larger graphs. We are this first to study adversarial attacks on graphs of practical size and, hence, set the cornerstone for the important evaluation of adversarial robustness at scale. We give some intriguing insights. For example, our experiments suggest that adversarial robustness seems to decrease with the size of the graph.

Moreover, it seems to be very different to defend against adversarially added nodes than edge additions or deletions within the existing graph structure. For most applications, we believe that adding new nodes is more realistic than adding edges between existing nodes and, hence, we argue that this setting should be studied more in future work.

References

Biendata. 2020. KDD Cup 2020: Graph Adversarial Attack and Defense. URL https://www.biendata.xyz/competition/kddcup{_}2020{_}formal/.

Bojchevski, A.; and Günnemann, S. 2018. Deep Gaussian embedding of graphs: Unsupervised inductive learning via ranking. *6th International Conference on Learning Representations, ICLR* 1–13.

Bojchevski, A.; and Günnemann, S. 2019. Adversarial attacks on node embeddings via graph poisoning. *36th International Conference on Machine Learning, ICML* 2019-June: 1112–1123.

Chen, T.; Xu, B.; Zhang, C.; and Guestrin, C. 2016. Training Deep Nets with Sublinear Memory Cost. *arXiv preprint arXiv:1604.06174* URL http://arxiv.org/abs/1604.06174.

Dai, H.; Li, H.; Tian, T.; Xin, H.; Wang, L.; Jun, Z.; and Le, S. 2018. Adversarial attack on graph structured data. *35th International Conference on Machine Learning, ICML* 3: 1799–1808.

Entezari, N.; Al-Sayouri, S. A.; Darvishzadeh, A.; and Papalexakis, E. E. 2020. All you need is Low (rank): Defending against adversarial attacks on graphs. *International Conference on Web Search and Data Mining, WSDM* 169–177. doi:10.1145/3336191.3371789.

Geisler, S.; Zügner, D.; and Günnemann, S. 2020. Reliable Graph Neural Networks via Robust Aggregation. *Neural Information Processing Systems, NeurIPS* (NeurIPS). URL http://arxiv.org/abs/2010.15651.

Goodfellow, I. J.; Shlens, J.; and Szegedy, C. 2015. Explaining and harnessing adversarial examples. *3rd International Conference on Learning Representations, ICLR* 1–11.

Hu, W.; Fey, M.; Zitnik, M.; Dong, Y.; Ren, H.; Liu, B.; Catasta, M.; and Leskovec, J. 2020. Open Graph Benchmark: Datasets for Machine Learning on Graphs. URL http://arxiv.org/abs/2005.00687.

Kipf, T. N.; and Welling, M. 2017. Semi-supervised classification with graph convolutional networks. *5th International Conference on Learning Representations, ICLR* 1–14.

Klicpera, J.; Weißenberger, S.; and Günnemann, S. 2019. Diffusion Improves Graph Learning. *Neural Information Processing Systems, NeurIPS* URL http://arxiv.org/abs/1911.05485.

McCallum, A. K.; Nigam, K.; Rennie, J.; and Seymore, K. 2000. Automating the construction of internet portals with machine learning. *Information Retrieval* ISSN 13864564. doi:10.1023/A:1009953814988.

Nesterov, Y. 2012. Efficiency of coordinate descent methods on huge-scale optimization problems. *SIAM J. Optim.* 22: 341–362.

Nesterov, Y.; and Stich, S. 2017. Efficiency of accelerated coordinate descent method on structured optimization problems. *Siam J. Optim.* 27: 110–123.

Sen, P.; Namata, G. M.; Bilgic, M.; Getoor, L.; Gallagher, B.; and Eliassi-Rad, T. 2008. Collective classification in network data. *AI Magazine* ISSN 07384602. doi:10.1609/aimag.v29i3.2157.

Tang, X.; Li, Y.; Sun, Y.; Yao, H.; Mitra, P.; and Wang, S. 2020. Transferring robustness for graph neural network against poisoning attacks. *Conference on Web Search and Data Mining, WSDM* 600–608. doi:10.1145/3336191.3371851.

Wang, B.; and Gong, N. Z. 2019. Attacking graph-based classification via manipulating the graph structure. *ACM Conference on Computer and Communications Security* 2023–2040. ISSN 15437221. doi:10.1145/3319535.3354206.

Waniek, M.; Michalak, T. P.; Wooldridge, M. J.; and Rahwan, T. 2018. Hiding individuals and communities in a social network. *Nature Human Behaviour* 2(2): 139–147. ISSN 23973374. doi:10.1038/s41562-017-0290-3.

Wright, S. J. 2015. Coordinate Descent Algorithms. *Mathematical Programming* 151: 3–34.

Wu, H.; Wang, C.; Tyshetskiy, Y.; Docherty, A.; Lu, K.; and Zhu, L. 2019. Adversarial examples for graph data: Deep insights into attack and defense. *IJCAI International Joint Conference on Artificial Intelligence* 2019-Augus: 4816–4823. ISSN 10450823. doi:10.24963/ijcai.2019/669.

Xu, K.; Chen, H.; Liu, S.; Chen, P. Y.; Weng, T. W.; Hong, M.; and Lin, X. 2019. Topology attack and defense for graph neural networks: An optimization perspective. *IJCAI International Joint Conference on Artificial Intelligence* 2019-Augus: 3961–3967. ISSN 10450823. doi:10.24963/ijcai.2019/550.

Zhu, D.; Cui, P.; Zhang, Z.; and Zhu, W. 2019. Robust graph convolutional networks against adversarial attacks. *International Conference on Knowledge Discovery and Data Mining, KDD* 1399–1407. doi:10.1145/3292500.3330851.

Zügner, D.; Akbarnejad, A.; and Günnemann, S. 2018. Adversarial attacks on neural networks for graph data. *International Conference on Knowledge Discovery and Data Mining, KDD* 2847–2856. doi:10.1145/3219819.3220078.

Zügner, D.; and Günnemann, S. 2019. Adversarial attacks on graph neural networks via meta learning. *7th International Conference on Learning Representations, ICLR* 1–15.