Expressivity of Graph Neural Networks Through the Lens of Adversarial Robustness

Francesco Campi [1] **Lukas Gosch** [1] **Tom Wollschläger** [1] **Yan Scholten** [1] **Stephan Günnemann** [1]

Abstract

We perform the first adversarial robustness study into Graph Neural Networks (GNNs) that are provably more powerful than traditional Message Passing Neural Networks (MPNNs). In particular, we use adversarial robustness as a tool to uncover a significant gap between their theoretically possible and empirically achieved expressive power. To do so, we focus on the ability of GNNs to count specific subgraph patterns, which is an established measure of expressivity, and extend the concept of adversarial robustness to this task. Based on this, we develop efficient adversarial attacks for subgraph counting and show that more powerful GNNs fail to generalize even to small perturbations to the graph's structure. Expanding on this, we show that such architectures also fail to count substructures on out-of-distribution graphs.

1. Introduction

In recent years, significant efforts have been made to develop Graph Neural Netwoks (GNNs), for several graph-related tasks, such as molecule property predictions (Gasteiger et al., 2020), social network analysis (Fan et al., 2019), or combinatorial problems (Gasse et al., 2019), to name a few. The most commonly used architectures are based on message passing, which iteratively updates the embedding of each node based on the embeddings of its neighbors (Gilmer et al., 2017). Despite their broad success and wide adoption, different works have pointed out that so called Message Passing Neural Networks (MPNNs) are at most as powerful as the 1-Weisfeiler-Lehman (WL) algorithm (Morris et al., 2019; Xu et al., 2019) and thus, have important limitations in their expressive power (Chen et al., 2020). This encouraged

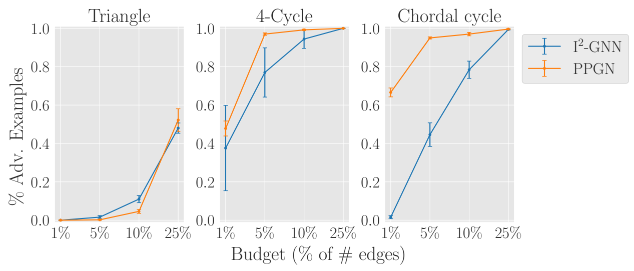

Figure 1. GNNs more powerful than 1-WL are not adversarially robust for subgraph-counting tasks.

the development of provably more powerful architectures.

However, there is no guarantee that the training process also yields models that are as powerful as theoretically guaranteed. Thus, this work investigates if and to what extent the empirically achieved expressivity of such GNNs lacks behind their theoretic possibilities by taking a novel look from the perspective of adversarial robustness. In particular, we focus on the task of counting different subgraphs, which is provably impossible for MPNNs (Chen et al., 2020) (except for very limited cases), but important for many downstream tasks (Huang et al., 2023; Liu et al., 2019; Monti et al., 2018).

Using our new adversarial framework for subgraph counting, we find that the counting ability of theoretically more powerful GNNs fail to generalize even to small perturbations to the graph's structure (see Figure 1). A result even more interesting given that subgraph counting is polynomially solvable for fixed subgraph sizes (Shervashidze et al., 2009).[1] We expand on these results and show that these architectures also fail to count substructures on out-of-distribution (OOD) graphs. Furthermore, retraining the last MLP layers responsible for the prediction based on the graph embedding does not entirely resolve this issue.

Contributions. (i) We perform the first study into the adversarial robustness of GNNs provably more powerful than the 1-WL and use it as an effective tool to uncover a significant gap between the theoretically possible and empirically achieved expressivity for substructure-counting tasks

[1]TUM School of Computation, Information and Technology & Munich Data Science Institute, Technical University of Munich, Germany. Correspondence to: Francesco Campi <francesco.campi98@gmail.com>.

2nd AdvML Frontiers workshop at 40th International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023. Copyright 2023 by the author(s).

[1]In general, this problem is NP-complete (Ribeiro et al., 2021)

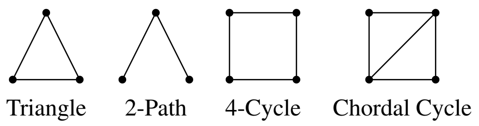

Figure 2. Examples of graph patterns used for subgraph-counting.

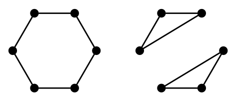

Figure 3. Pair of undistinguishable graphs for MPNNs with different triangle counts.

(see Section 6). (ii) We extend the concept of an adversarial example from classification to (integer) regression tasks and develop multiple perturbations spaces interesting for the task of subgraph counting (see Section 4). (iii) We develop efficient and effective adversarial attacks for subgraph counting, operating in these perturbations spaces and creating *sound* perturbations, i.e., where we know the ground truth (see Section 5). (iv) In Section 6.2 we show that subgraph-counting GNNs also fail to generalize to out-of-distribution graphs, providing additional evidence that these GNNs fail to reach their theoretically possible expressivity. Our code implementation can be found at https://github.com/francesco-campi/Rob-Subgraphs.

2. Background

We consider undirected, unattributes graphs G=(V, E) with nodes V=$\{1, \dots, n\}$ and edges E⊆$\{\{i,j\} \mid i,j \in V, i{\neq}j\}$, represented by adjacency matrix $\mathbf{A} \in \{0,1\}^{n\times n}$. A graph $G_S = (V_S, E_S)$ is a *subgraph* of G if $V_S \subseteq V$ and $E_S \subseteq E$. We say G_S is an *induced* subgraph if E_S contains all edges in E that connect pairs of nodes in V_S. An egonet $\text{ego}_l(i)$ is the induced subgraph containing all nodes with a distance of at most l from root node i. Furthermore, two graphs G, G' are isomorphic ($\simeq$) if there exists a bijection $f : V \to V'$ such that $\{i,j\} \in E$ if and only if $\{f(i), f(j)\} \in E'$. Lastly, the diameter $\text{diam}(G)$ denotes the length of the largest shortest path in graph G.

2.1. Subgraph-Counting

Consider a fixed graph H which we call a pattern (Figure 2). A classic graph-related problem is the *(induced-) subgraph-counting* of the pattern H (Ribeiro et al., 2021), which consists of enumerating the (induced) subgraphs of G isomorphic to H. The subgraph-count of H is denoted by $\mathcal{C}(G, H)$, and by $\mathcal{C}_I(G, H)$ in the induced case. To simplify the notation we will also refer to it as $\mathcal{C}(G)$ if H is given in the context. Several algorithms have been developed to solve the task of subgraph-counting. In this work we specifically consider the (exact) algorithm of Shervashidze et al. (2009) (presented in Appendix B) due to its low computational cost.

2.2. Expressivity of Graph Neural Networks

The expressivity of machine learning models is about which functions they can and cannot approximate. There are different ways of studying the expressive power of GNNs. In this work we specifically consider their ability to count subgraphs (Chen et al., 2020) because it is strictly related to different real-world tasks such as computational chemistry (Jin et al., 2020) and social network analysis (Jiang et al., 2010). We define the ability to count subgraphs as follows:

Definition 2.1. A family of functions $\mathcal{F}$ can perform *subgraph-counting* of a target pattern H on a graph class $\mathcal{G}$ if for any two graphs $G_1, G_2 \in \mathcal{G}$ with $\mathcal{C}(G_1, H) \neq \mathcal{C}(G_2, H)$ there exists a function $f \in \mathcal{F}$ such that $f(G_1) \neq f(G_2)$.

Surprisingly, MPNNs have considerable limitations in subgraph-counting. In fact, Chen et al. (2020) show that MPNNs are not able to count induced patterns with three or more nodes, leaving out only the ability to count edges. For example, Figure 3 shows two graphs that, despite having different triangle counts, will always return identical outputs when fed in the same MPNN.

A different perspective to measure the expressive power is graph isomorphism. In this regard, Xu et al. (2019); Morris et al. (2019) demonstrated that an MPNN is at most as powerful as 1-WL isomorphism test at distinguishing pairs of non-isomorphic graphs. Moreover, since the WL algorithms are designed to extract representation vectors from graphs, they could be used also to perform subgraph-counting. In particular, Chen et al. (2020) showed that k-WL, and equivalently powerful architectures, can perform substructure-counting for patterns with at most k nodes, creating a connection between the two approaches.

2.3. More Expressive Graph Neural Networks

In this work, we analyze two state-of-the-art architectures for the task of subgraph counting: PPGN (Maron et al., 2019a), and I^2-GNN (Huang et al., 2023). PPGN represents the graph structure in a tensor and exploits tensor multiplications to enhance the expressivity. It reaches the same expressive power of 3-WL, which makes it capable of counting patterns of size three. I^2-GNN, following the approach of subgraph GNNs (Frasca et al., 2022), decomposes the whole graph into different subgraphs and processes them independently with an MPNN. It has been explicitly developed to be expressive enough for counting different substructures and most important for this work, can count arbitrary patterns of size four. Both, PPGN and I^2-GNN are effective architectures for downstream tasks such as molecular property predictions.

3. Related Work

Chen et al. (2020) were the first to study the expressivity of GNNs w.r.t. their ability to count substructures. They, and later Tahmasebi et al. (2021) proposed architectures for counting substructures. However, these suffer from high computational complexity. Yu et al. (2023) proposed an architecture purely focusing on subgraph counting. However, subgraph counting alone can be solved by efficient randomized algorithms (Bressan et al., 2021). Thus, in this work, we focus on efficient architectures, which leverage their subgraph counting ability to improve generalization for other downstream tasks. In particular, we focus on PPGN (Maron et al., 2019b) and I^2-GNN (Huang et al., 2023). Both achieve state-of-the-art results for substructure counting while having formal expressivity guarantees.

Different works have studied the adversarial robustness of GNNs for graph-level classification (Dai et al., 2018) and node-level classification (Zügner et al., 2018). Regarding the latter, Gosch et al. (2023) exactly define (semantic-preserving) adversarial examples. Moreover, Geisler et al. (2022) use adversarial attacks with sound perturbation models, i.e., where the ground truth change is known, to investigate the generalization of neural combinatorial solvers. Conversely, adversarial robustness for regression tasks has currently received very little attention (Deng et al., 2020).

4. Robustness in Subgraph-Counting

The field of adversarial robustness is about the problem that machine learning models are vulnerable to small changes to their inputs (Goodfellow et al., 2015). In particular, for the subgraph-counting problem we want to analyze whether the error of the models increases when tested on perturbed input graphs $\tilde{G}$ of a graph from a set of perturbed graphs $\mathcal{P}(G)$. To evaluate the performance of a model f on perturbed graphs $\tilde{G} \in \mathcal{P}(G)$ we use the following adversarial loss:

$$\ell_{adv}(\tilde{G}) := |f(\tilde{G}) - \mathcal{C}(\tilde{G}, H)|.$$

4.1. Subgraph-Counting Adversarial Examples

To empirically evaluate the expressivity of machine learning models for subgraph-counting via adversarial robustness, we have to introduce a notion of adversarial example. In classification tasks adversarial examples are simply perturbations that change the predicted class. In general regression tasks one can define a threshold on ℓ_{adv} for which we call a perturbed graph an adversarial example (Deng et al., 2020). However, this definition is application-dependent and, in our work, we define a specific threshold exploiting the fact that subgraph-counting is an *integer* regression task.

Definition 4.1. Given a model f and clean graph G, we say that $\tilde{G} \in \mathcal{P}(G)$ is an *adversarial example* for f if:

(i) $\lfloor f(G) + 0.5 \rfloor = \mathcal{C}(G)$

(ii) $\lfloor f(\tilde{G}) + 0.5 \rfloor \neq \mathcal{C}(\tilde{G})$

(iii) $\dfrac{\ell_{adv}(\tilde{G}) - \ell_{adv}(G)}{\ell_{adv}(G)} > \delta.$

The conditions (i) and (ii) guarantee that the model prediction, when approximated to the nearest integer, is correct for G and wrong for $\tilde{G}$. Here, having a correct initial prediction is essential to clearly distinguish the performances on the original graph from the perturbed graph. In addition, the condition (iii) ensures that a margin exists between the errors on the original data instance and the perturbed one, and the size of the margin depends on the value of δ. This requisite prevents easily generating adversarial examples from graphs that are almost wrongly predicted, i.e. $\ell_{adv}(G) \approx 0.5$.

4.2. Perturbation Spaces

We define different perturbation spaces for a graph G as constrained sets of structurally perturbed graphs constructed from G. In particular, we consider different combinations of edge deletions and additions, for example $E' = E \cup \{i, j\}$ with $\{i, j\} \notin E$ represents an edge addition. We always consider sound perturbation models, i.e, where we know the ground truth change. These are efficiently implemented as described in Section 5. It is meaningful to limit the number of perturbations in order to control how shifted the distribution of the perturbed subgraph-counts is compared to the distribution of the original ones. Then, we define the *constrained* perturbation space with maximal budget Δ as:

$$\mathcal{P}_\Delta(G) := \{\tilde{G} \mid \frac{1}{2}\|\mathbf{A} - \mathbf{A}'\|_0 \leq \Delta\}, \tag{1}$$

where $\|\cdot\|_0$ represents the number of non-zero elements, i.e. the number of perturbed edges.

Semantic-Preserving Perturbations. Additionally, we conduct a robustness analysis more closely in line with adversarial examples for classification tasks, by incorporating a further constraint to guarantee the preservation of a specific level of semantic meaning. In particular, we define the *count-preserving* perturbation space as:

$$\mathcal{P}^c_\Delta(G) := \{\tilde{G} \mid \tilde{G} \in \mathcal{P}_\Delta(G) \ \wedge \ \mathcal{C}(\tilde{G}) = \mathcal{C}(G)\}. \tag{2}$$

Additionally, when considering induced subgraphs, keeping the count constant does not guarantee that the subgraphs isomorphic to the pattern remain the same. In fact, perturbations can simultaneously delete a subgraph isomorphic to the pattern and generate a new one (see Figure 4). We will denote the *subgraph-preserving* perturbation space by

$$\begin{aligned}\mathcal{P}^s_\Delta(G) := \{&\tilde{G} \mid \tilde{G} \in \mathcal{P}_\Delta(G) \wedge \\ &G_S \subseteq G, G_S \simeq H \iff G_S \subseteq \tilde{G}, G_S \simeq H\}.\end{aligned} \tag{3}$$

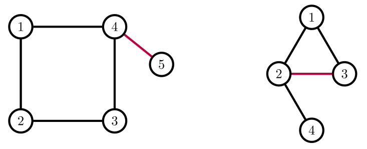

Figure 4. This Figure shows examples demonstrating that not all the count-preserving perturbations are also subgraph-preserving ones. On the left a subgraph- and count-preserving perturbation for 4-cycles where the red edge has been deleted. On the right a perturbation that leaves unchanged the count of 2-paths, but it deletes the induced substructure $\{2, 3, 4\}$ to generate $\{1, 2, 3\}$.

Algorithm 1 Beam search (greedy search for $k = 1$)

```
Input: $G, \Delta, k$
$\mathcal{G}^{(0)} = \{G\}$
for $i = 0$ to $\Delta - 1$ do
  $\mathcal{P}^{(i)} = \{\}$
  for $\tilde{G}$ in $\mathcal{G}^{(i)}$ do
    $\mathcal{P}^{(i)} = \mathcal{P}^{(i)} \cup \mathcal{P}_1(\tilde{G})$        {or $\mathcal{P}_1^c(G), \mathcal{P}_1^s(G)$}
  end for
  $\mathcal{G}^{(i+1)}$ = greatest $k$ in $\{\ell_{adv}(\tilde{G}) \mid \tilde{G} \in \mathcal{P}^{(i)}\}$
end for
Return: $G^* = \mathrm{argmax}_{\tilde{G} \in \mathcal{G}^{(\Delta)}} \{\ell_{adv}(\tilde{G})\}$
```

5. Subgraph-Counting Adversarial Attacks

For a subgraph-counting model f, the goal of an adversarial attack is to find the pertubed graph $G^* \in \mathcal{P}(G)$ that causes the maximal error increase. This problem can be formulated as an optimization problem:

$$G^* = \underset{\tilde{G} \in \mathcal{P}(G)}{\mathrm{argmax}}\ \ell_{adv}(\tilde{G}). \tag{4}$$

Attacking subgraph-counting GNNs for studying their empirical expressivity is particularly challenging. In fact, (i) the subgraph-count can vary significantly even for slight structural changes, and (ii) finding G^* of Equation (4) requires solving a discrete optimization problem.

5.1. Sound Perturbations for Subgraph-Counting

To tackle the sensitivity of the counts to structural changes, we exploit the exact algorithm to update the ground-truth count after every perturbation. In this way, we generate sound perturbations since the exact ground-truth value is know. In order to prevent this step to become computationally prohibitive, we develop an efficient count updating scheme that uses only a small portion of the graph.

Proposition 5.1. *Consider a graph G and a pattern H with* $\mathrm{diam}(H) = d$*. Then, for every edges* $\{i, j\}$ *we have that* $\mathrm{ego}_d(i)$ *and* $\mathrm{ego}_d(j)$ *contain all the subgraphs* $G_S \subset G$ *such that* $G_S \simeq H$ *and* $i, j \in V_S$*.*

Proof in Appendix A.

When an edge $\{i, j\}$ is perturbed, only the subgraphs containing both the end nodes can be affected and potentially change their isomorphism relation with H. Therefore, according to Proposition 5.1, it is sufficient to verify potential count changes only in $\mathrm{ego}_d(i)$ (or equivalently $\mathrm{ego}_d(j)$). Specifically, the theorem assumes that $\{i, j\}$ is contained in the graph, hence we extract the egonet from the graph including $\{i, j\}$ (original for edge deletion and perturbed for addition). Next, from the nodes of $\mathrm{ego}_d(i)$ we generate the induced subgraphs G_S and $\tilde{G}_S$ from the original and perturbed graphs respectively. Since the possible alterations of the subgraph-count are enclosed in G_S and $\tilde{G}_S$, we have the following count update rule.

Proposition 5.2. *Let* $\tilde{G}$ *be a perturbation of a single edge of a graph G, then there holds:*

$$\mathcal{C}(\tilde{G}) = \mathcal{C}(G) + \mathcal{C}(\tilde{G}_S) - \mathcal{C}(G_S).$$

Following Proposition 5.2 we need to run the subgraph-counting algorithm only on the smaller subgraphs G_S and $\tilde{G}_S$, rather than on the whole graph $\tilde{G}$. Additionally, Proposition 5.1 guarantees that potential changes in the subgraphs isomorphic to the patterns are also constrained in the egonet, thus it can be used also identify perturbations belonging to the subgraph-preserving perturbation space $\mathcal{P}^s_\Delta$.

5.2. Construction of Adversarial Examples

To create adversarial examples we need to solve the discrete optimization problem in Equation (4). To do so we develop algorithms that generate more powerful perturbation one change at a time, in this way, we keep track of the exact count with the update rule (Proposition 5.2).

Greedy Search. We develop an efficient and effective greedy search algorithm (Algorithm 1). At each step we select the most effective perturbation of the current perturbed graph $\tilde{G}$ in $\mathcal{P}_1(\tilde{G})$ (or in $\mathcal{P}_1^c(\tilde{G}), \mathcal{P}_1^s(\tilde{G})$) until the budget limit is reached. The new subgraph-counts of perturbations in $\mathcal{P}_1(\tilde{G})$ are computed with Proposition 5.2, whereas the preserving perturbation spaces are generated with Algorithm 2.

Beam search. A more advanced algorithm that does not increase the computational complexity is beam search. Concretely, it follows simultaneously k different paths to explore more extensively the perturbation space (see Algorithm 1).

To improve the computational efficiency the perturbations in $\mathcal{P}_1$ can be randomly selected according to the degrees of the end nodes of the perturbed edge. Concretely, the probability to pick the perturbation where the edge $\{i, j\}$ has been added (or deleted) is proportional to $d(i)^2 + d(j)^2$, since intuitively these are the most relevant edges.

Algorithm 2 $\mathcal{P}_1^c(G)$ generation (analogous for $\mathcal{P}_1^s(G)$)

```
**Input:** $G, \mathcal{C}(G)$
$\mathcal{P}_1^c(G) = \{\}$
**for** $\tilde{G}$ **in** $\mathcal{P}_1(G)$ **do**
    $\mathcal{C}(\tilde{G}) = \mathcal{C}(G) + \mathcal{C}(G'_S) - \mathcal{C}(G_S)$
    **if** $\mathcal{C}(\tilde{G}) = \mathcal{C}(G)$ **then**
        $\mathcal{P}_1^c(G) = \mathcal{P}_1^c(G) \cup \{\tilde{G}\}$
    **end if**
**end for**
**Return:** $\mathcal{P}_1^c(G)$
```

Table 1. AUC of the functions in Figure 5 normalized by the area under the unity function. The label NR stands for Non-Robust and NR (tr.) for Non-Robust (Transfer).

Pert. Space	Arch.	Trangle NR	Trangle NR (tr.)	4-Cycle NR	4-Cycle NR (tr.)	Chord. C. NR	Chord. C. NR (tr.)
$\mathcal{P}_\Delta$	PPGN	0.18	0.10	0.95	0.87	0.95	0.86
	I^2-GNN	0.20	0.17	0.88	0.63	0.72	0.46
$\mathcal{P}_\Delta^c$	PPGN	0.090	0.050	0.92	0.84	0.93	0.73
	I^2-GNN	0.0	0.0	0.76	0.40	0.41	0.097
$\mathcal{P}_\Delta^s$	PPGN	0.09	0.50	0.85	0.58	0.90	0.59
	I^2-GNN	0.0	0.0	0.66	0.23	0.38	0.082

6. Experiments

In Section 6.1, we analyze the empirical expressivity of GNNs using our subgraph-counting adversarial attacks and using generalization as a (proxy) measure. Extending on this, in Section 6.2 we investigate if the same GNNs can count subgraph patterns for out-of-distribution graphs. Here we present the results of the induced subgraph-counting of triangles, 4-cycles and chordal cycles, for other patterns refer to Appendix C.

6.1. Adversarial Robustness

Here, we analyze the empirical expressivity of GNNs using our subgraph-counting adversarial attacks.

Dataset and models. We generate a synthetic dataset of $5{,}000$ Stochastic-Block-Model graphs with 30 nodes divided into 3 different communities. The probabilities of generating edges connecting nodes within the same community are $[0.2, 0.3, 0.4]$, while the probability of generating edges between nodes of different communities is 0.1. We randomly split the dataset into training, validation, and test sets with percentages $30\%, 20\%, 50\%$. We then train PPGN (Maron et al., 2019a) and I^2-GNN (Huang et al., 2023).

Experimental Settings. We train each model 5 times using different initialization seeds to prevent bad weight initialization influencing the final results. Then, for each of the trained models f_i with seed i, we use our adversarial attacks (see Section 5) to generate adversarial examples from 100 correctly predicted test graphs and average the percentage of successful attacks over all seeds. Furthermore, we investigate if the adversarial graphs for a model f_i transfer to the other models f_j trained with a different initialization seed $j \neq i$. We inspect all three different perturbation spaces with budgets Δ of $1\%, 5\%, 10\%$ and 25% with respect to the average number of edges of the graphs in the dataset and use $\delta = 1$ as margin. In detail, we use beam search with beam width $k = 10$ to explore $\mathcal{P}_\Delta^c$ and $\mathcal{P}_\Delta^s$, while we rely on greedy search for $\mathcal{P}_\Delta$.

Results. The plots in Figure 5 show the percentage of perturbations found by the optimization algorithms that represent a successful adversarial example according to Definition 4.1. To condensate the results in a numerical value, we report in Table 1 the area under the curve (AUC) of the functions Non-Robust and Non-Robust (Transfer) in Figure 5. The results are reported as the proportion with respect to the area under the unity function $f(x) = 1$, which represents the worse case where all permutations generate an adversarial example already at $\Delta = 1\%$. Interestingly, the results show that we can find several adversarial examples for both architectures. In particular, PPGN is highly unrobust in the subgraph-counting of patterns with four nodes. However, several adversarial examples can be found also for the triangle count, even though the theoretical expressivity of PPGN claims that it is a family of functions that can count 3-dimensional subgraphs in the sense of Definition 2.1. Similarly, the more expressive model I^2-GNN is fooled on 4-dimensional patterns, in spite of being sufficiently powerful to count them. This indicates that the empirical expressivity achieved does not match the theoretical expressivity since the models are not able to generalize to subgraph-counting tasks that they should in theory be able to solve. Additionally, in Appendix C.2 we investigate some structural properties of the adversarial examples.

6.2. Out-of-Distribution Generalization

Next, we examine the performance of the same architectures when tested on graphs that are OOD with respect to the original training. Here, we retrain the last Multi-Layer Percepton (MLP) prediction layers to investigate more in detail whether the problem lies in the expressivity of the extracted graph representations or in the last layers that leverage the graph representations for the predictions.

Dataset. For these experiments we use Erdos-Renyi graphs with 10 nodes and edge generation probabilities of 0.3 for the first dataset (d$_1$) and 0.8 for the OOD dataset (d$_2$). The size and splits of the datasets are analogous to Section 6.1. Therefore, we consider as OOD graphs that are generated from a distribution with the same number of nodes but different edge generation probabilities.

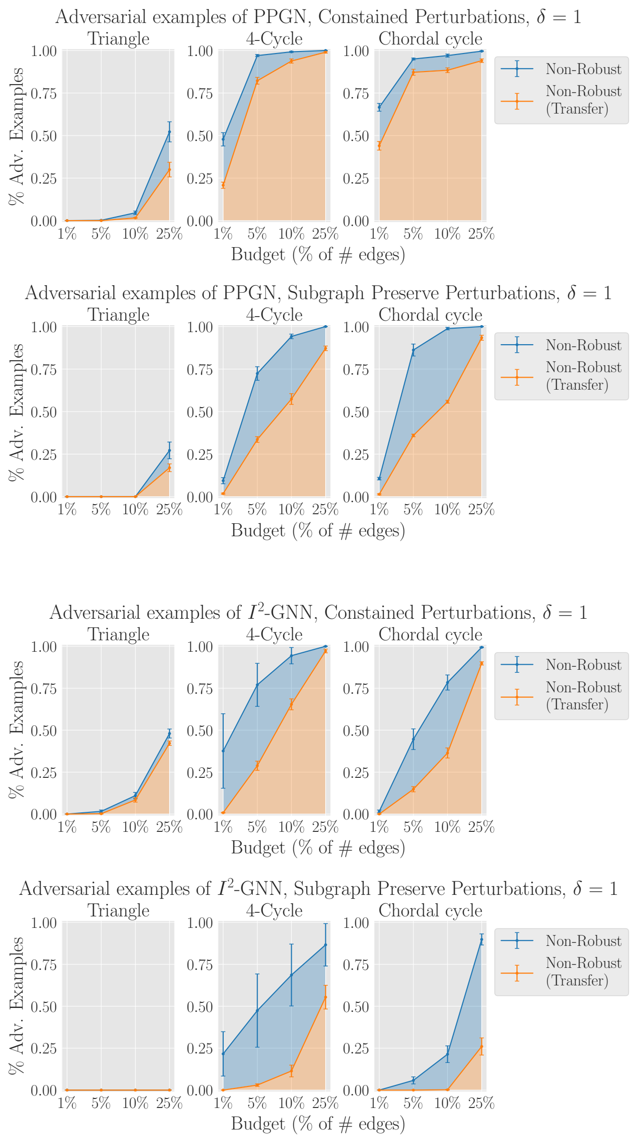

Figure 5. The plots illustrate in blue the success rate of our subgraph-counting adversarial attacks at finding perturbations that represent adversarial examples according to Definition 4.1 constrained and subgraph preserving perturbation spaces. In orange, we represent how effective the adversarial examples are when transferred to the models trained with a different initialization seed. The values are the average of the results obtained with 5 different initialization seeds with the relative standard errors.

Experimental Settings. Firstly, we train the PPGN and I^2-GNN architectures on the dataset d_1 and test them on both d_1 and d_2 to investigate the OOD generalization performances of the architectures. Additionally, we train the models directly on d_2 to have a comparison of the best performances achievable on this dataset. The errors are expressed using the mean absolute error (ℓ_1) and an extension of it, which is obtained by normalizing by the ground-truth count (ℓ_c).

Table 2. Test errors of the OOD experiments that investigate the generalization abilities of the architectures. Specifically, d_i represents models trained and tested on the same dataset d_i, OOD models trained on d_1 and tested in d_2 and in MLP we additionally retrain the final layers on d_2.

Arch.	Exp.	Trangle		4-Cycle		Chord. C.	
	Setting	ℓ_1	ℓ_c	ℓ_1	ℓ_c	ℓ_1	ℓ_c
PPGN	d_1	0.0058	7.8e-4	0.059	0.010	0.10	0.011
	OOD	2.98	0.041	5.40	1.17	20.0	0.25
	d_2	0.0091	1.7e-4	0.040	0.0050	0.12	0.0017
	MLP	0.059	9.8e-4	0.29	0.0.043	1.083	0.014
I^2-GNN	d_1	0.0027	2.8e-4	0.035	0.0062	0.020	0.0023
	OOD	3.25	0.044	2.16	0.45	6.75	0.084
	d_2	0.032	6.2e-4	0.028	0.0031	0.30	0.0042
	MLP	0.20	0.0031	0.19	0.025	1.56	0.020

Results. Table 2 shows the test errors of the aforementioned settings averaged over five different initialization seeds. Here we observe that the models achieve very poor performances on general OOD graphs compared to their ideal performances (OOD and d_2 rows). However, if the model were able to perform subgraph-counting, as theoretically claimed, they should be able to perform this task regardless of the graph distribution. This result matches with Section 6.1 and shows that the models do not learn to detect the patterns and they rather overfit on the training distribution. However, this behavior could be intrinsic to the models' architecture. The models are designed to extract a vector representation from each input graph, which is then mapped to the prediction through an MLP. Then, the fact that different graph distributions might generate different graph representations leads us to investigate whether the problem is a poor generalization of the map between the graph embedding and the count. To test this possibility, we retrain on d_2 only the final MLP of the models previously trained on d_1 (row MLP in Table 2). While this adjustment is helpful, the errors are consistently one order of magnitude higher than the ones obtained training directly on d_2. This indicates that the graph representations do not achieve their theoretic separation power and that the problem does *not* only lie in the last MLP prediction layers.

7. Conclusion

We propose a novel approach to assess the empirical expressivity of subgraph-counting achieved by GNNs via adversarial robustness. We show that despite being theoretically capable of counting certain patterns, the models lack generalization as they struggle to correctly predict adversarially perturbed and OOD graphs. Therefore, the training algorithms are not able to find weights corresponding to a maximally expressive solution. Extending our study to other related GNNs such as KP-GNN (Feng et al., 2022) is an interesting direction for future work.

References

Bressan, M., Leucci, S., and Pancones, A. Faster motif counting via succinct color coding and adaptive sampling. In *TKDD*, 2021.

Chen, Z., Chen, L., Villar, S., and Bruna, J. Can graph neural networks count substructures? *Advances in neural information processing systems*, 33:10383–10395, 2020.

Dai, H., Li, H., Tian, T., Huang, X., Wang, L., Zhu, J., and Song, L. Adversarial attack on graph structured data. In *International conference on machine learning*, pp. 1115–1124. PMLR, 2018.

Deng, Y., Zheng, X., Zhang, T., Chen, C., Lou, G., and Kim, M. An analysis of adversarial attacks and defenses on autonomous driving models. In *2020 IEEE international conference on pervasive computing and communications (PerCom)*, pp. 1–10. IEEE, 2020.

Fan, W., Ma, Y., Li, Q., He, Y., Zhao, Y. E., Tang, J., and Yin, D. Graph neural networks for social recommendation. *The World Wide Web Conference*, pp. 417–426, 2019.

Feng, J., Chen, Y., Li, F., Sarkar, A., and Zhang, M. How powerful are k-hop message passing graph neural networks. In *Advances in Neural Information Processing Systems*, 2022.

Frasca, F., Bevilacqua, B., Bronstein, M., and Maron, H. Understanding and extending subgraph gnns by rethinking their symmetries. *Advances in Neural Information Processing Systems*, 35:31376–31390, 2022.

Gasse, M., Chételat, D., Ferroni, N., Charlin, L., and Lodi, A. Exact combinatorial optimization with graph convolutional neural networks. In *Neural Information Processing Systems*, 2019.

Gasteiger, J., Groß, J., and Günnemann, S. Directional message passing for molecular graphs. In *International Conference on Learning Representations*, 2020.

Geisler, S., Sommer, J., Schuchardt, J., Bojchevski, A., and Günnemann, S. Generalization of neural combinatorial solvers through the lens of adversarial robustness. In *International Conference on Learning Representations*, 2022.

Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. Neural message passing for quantum chemistry. In *International conference on machine learning*, pp. 1263–1272. PMLR, 2017.

Goodfellow, I. J., Shlens, J., and Szegedy, C. Explaining and harnessing adversarial examples. In *International Conference on Learning Representations*, 2015.

Gosch, L., Sturm, D., Geisler, S., and Günnemann, S. Revisiting robustness in graph machine learning. In *ICLR*, 2023.

Huang, Y., Peng, X., Ma, J., and Zhang, M. Boosting the cycle counting power of graph neural networks with i^2-GNNs. In *International Conference on Learning Representations*, 2023.

Jiang, C., Coenen, F., and Zito, M. A. A. Finding frequent subgraphs in longitudinal social network data using a weighted graph mining approach. In *International Conference on Advanced Data Mining and Applications*, 2010.

Jin, W., Barzilay, R., and Jaakkola, T. Multi-objective molecule generation using interpretable substructures. In *International conference on machine learning*, pp. 4849–4859. PMLR, 2020.

Liu, S., Demirel, M. F., and Liang, Y. N-gram graph: Simple unsupervised representation for graphs, with applications to molecules. In *Advances in Neural Information Processing Systems*, 2019.

Maron, H., Ben-Hamu, H., Serviansky, H., and Lipman, Y. Provably powerful graph networks. *Advances in neural information processing systems*, 32, 2019a.

Maron, H., Ben-Hamu, H., Shamir, N., and Lipman, Y. Invariant and equivariant graph networks. In *International Conference on Learning Representations*, 2019b.

Monti, F., Otness, K., and Bronstein, M. M. Motifnet: a motif-based graph convolutional network for directed graphs. In *2018 IEEE Data Science Workshop (DSW)*, pp. 225–228, 2018.

Morris, C., Ritzert, M., Fey, M., Hamilton, W. L., Lenssen, J. E., Rattan, G., and Grohe, M. Weisfeiler and leman go neural: Higher-order graph neural networks. In *AAAI*, 2019.

Ribeiro, P., Paredes, P., Silva, M. E., Aparicio, D., and Silva, F. A survey on subgraph counting: concepts, algorithms, and applications to network motifs and graphlets. *ACM Computing Surveys (CSUR)*, 54(2):1–36, 2021.

Shervashidze, N., Vishwanathan, S., Petri, T., Mehlhorn, K., and Borgwardt, K. Efficient graphlet kernels for large graph comparison. In *Artificial intelligence and statistics*, pp. 488–495. PMLR, 2009.

Tahmasebi, B., Lim, D., and Jegelka, S. Counting substructures with higher-order graph neural networks: Possibility and impossibility results. In *arXiv 2012.03174*, 2021.

Xu, K., Hu, W., Leskovec, J., and Jegelka, S. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2019.

Yu, X., Liu, Z., Fang, Y., and Zhang, X. Learning to count isomorphisms with graph neural networks. In *AAAI*, 2023.

Zügner, D., Akbarnejad, A., and Günnemann, S. Adversarial attacks on neural networks for graph data. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 2847–2856, 2018.

A. Proof of Proposition 5.1

Proposition A.1. *Let consider a graph G and a pattern H with $\mathrm{diam}(H) = d$. Then, for every edges $\{i, j\}$ we have that $\mathrm{ego}_d(i)$ and $\mathrm{ego}_d(j)$ contain all the subgraphs $G_S \subset G$ such that $G_S \simeq H$ and $i, j \in V_S$.*

Proof. Let's consider any subgraph $G_S = (V_S, E_S)$ of the graph $G = (V, E)$ such that $G_S \simeq H$. Firstly we show that $\mathrm{ego}_d(i) = (V_e, E_e)$ contains G_S, that is equivalent to showing that $V_S \subseteq V_e$ and $E_S \subseteq E_e$. By construction, we have that $\mathrm{diam}(H) = \mathrm{diam}(G_S) = d$, which implies that for every node $l \in V_S$ the shortest path connecting i and l has at most length d. Therefore, by definition of egonet, we have that $V_S \subseteq V_e$ Moreover, we know by construction that $E_S \subseteq E$ and can write the following series of set inclusions:

$$E_S \subseteq V_S \times V_S \subseteq V_e \times V_e.$$

All in all, we have that:

$$E_S \subseteq E \cap (V_e \times V_e) = E_e.$$

Showing that $\mathrm{ego}_d(j)$ contains V_S is analogous.

We can also easily show that for any $\mathrm{ego}_l(i)$ (or $\mathrm{ego}_l(j)$) with $l > d$ Proposition A.1 does not hold. In fact, it is sufficient to consider i such that there exists a pair of nodes $\{i, l\}$ where the shortest path connecting the two nodes has length d. Therefore, by construction $l \notin V_e$ and Proposition A.1 does not hold.

B. Counting algorithm

To generate synthetic datasets and to perform adversarial attacks (Section 5) we need an algorithm that is capable of counting subgraphs. However, subgraph counting is a highly complex procedure, and naive approaches could scale to a prohibitive amount of computations. As a matter of fact, only enumerating all the possible k-dimensional subgraph requires $\binom{n}{k}$ iterations, which can become intractable for sufficiently large graphs. Since the computation of groud-truth will be extensively used by adversarial attacks, finding an efficient algorithm to count subgraphs is in our interest to reduce the overall computational complexity of the attacks. (Shervashidze et al., 2009) proposed a method to count induced graphlets of size 3, 4, and 5 with a computational complexity of $\mathcal{O}(Nd^{k-1})$, where d is the maximum node degree and k is the size of the graphlets, which is linear in the number of nodes and is especially efficient sparser graphs. In particular, in this work we will consider only connected subgraphs of 3 and 4 nodes to have a broad overview of the performances on patterns with different structural properties (Figure 6).

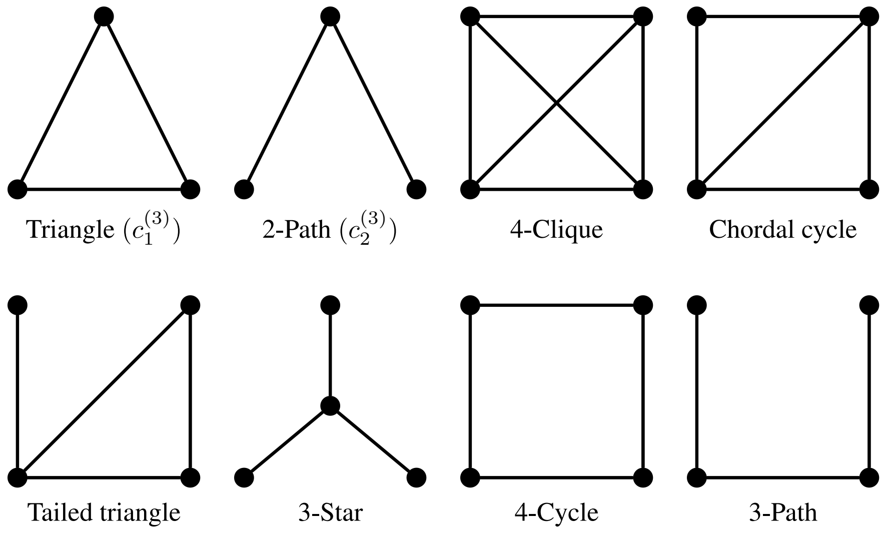

Figure 6. Connected subgraphs with 3 or 4 nodes. In brackets the name with which we represent the count variables in Algorithm 3

The core idea of the algorithm is to extract connected paths of length $k - 1$ and a single node connected to the path, then this set of nodes is assigned to a substructure using handcrafted conditions based on the additional connections between nodes. For example, for a 2-path $\{v_1, v_2, v_3\}$ and an additional node v_4 that is connected only to the head and the tail of the path, one can conclude that the four nodes generate a 4-Cycle.

The algorithm that counts 3-dimensional graphlets (Algorithm 3) starts by considering every edge $\{v_1, v_2\}$ in the graph, which has $\mathcal{O}(Nd)$ complexity. Next, one can count the subgraphs with specific computations on the neighboring sets of v_1 and v_2, in particular, the nodes in both neighborhoods will constitute a triangle, and the ones belonging only to one neighborhood will constitute a 2-path. This last step has $\mathcal{O}(d)$ complexity because every neighborhood contains at most d nodes. Finally, every subgraph is detected more than once as every pattern can be identified starting with any of its edges, then every variable is normalized by dividing by the number of edges of the pattern it represents.

The 4-dimensional version has an analogous working principle adapted to the 4 four nodes case. As before, the algorithm iterates over every edge in the graph, then all the nodes v_3 adjacent to this edge ($\mathcal{O}(d)$) are considered and separated into 3 disjoint groups based on the connections with $\{v_1, v_2\}$ (connected with both nodes, only the first one or only the second one). At this point, similarly to before, the patterns can be identified with specific computations of the neighborhoods of v_1, v_2, v_3. For example, if v_3 is only in the neighborhood of v_1 and v_4 connected to all the previous nodes, then $\{v_1, \dots, v_4\}$ is a tailed triangle. In the last step, the variables need to be normalized to take into account multiple detections of the same subgraph. By construction, once the triplet $\{v_1, v_2, v_3\}$ is set, the algorithm can detect a subgraph containing this triplet only once, hence normalization constant is the number of times the algorithm obtains a triplet included in a subgraph. In particular, triangles can be obtained in three different ways, starting from each edge, and, for the same reason, 2-paths can be obtained in two ways. All in all, the normalization constant for a general pattern is $2p + 3t$, where p is the number of 2-paths in the pattern and t is the number of triangles. Moreover, these algorithms can not only count the induced subgraphs, but also identify them by returning the set of nodes $N' \subset N$ that generates them, which will be essential for subgraph preserving attacks (Section 5).

Algorithm 3 Count 3-dimensional induced subgraphs in a graph $G = (N, E)$

```
$c_i^{(3)} \leftarrow 0,\ i \in \{1, 2\}$                                {Initialize the variables}
for $\{v_1, v_2\} \in E$ do
    $c_1^{(3)} \leftarrow c_1^{(3)} + |\mathcal{N}(v_1) \cap \mathcal{N}(v_2)|$            {Update the substructure counts}
    $c_2^{(3)} \leftarrow c_2^{(3)} + |\mathcal{N}(v_1)/(\mathcal{N}(v_2) \cup \{v_2\})|$
    $c_2^{(3)} \leftarrow c_2^{(3)} + |\mathcal{N}(v_2)/(\mathcal{N}(v_1) \cup \{v_1\})|$
end for
$c_1^{(3)}(G) \leftarrow c_1^{(3)}(G)/3$                                {Normalize the variables }
$c_2^{(3)}(G) \leftarrow c_2^{(3)}(G)/2$
Return: $c_i^{(3)},\ i \in \{1, 2\}$
```

C. Additional Results

In this section, we extend the experiments of Section 6 to all the connected patterns with three and four nodes Figure 6. In particular, in Appendix C.1 we extend the adversarial robustness experiments, in Appendix C.2 we compare some structural properties of the adversarial examples with the clean graphs, and in Appendix C.3 we present the complete results of the OOD experiments.

C.1. Adversarial Robustness

In Figure 7 we present the complete adversarial robustness results of PPGN on all the 3- and 4-dimensional patterns on all the 3 perturbations paces ($\mathcal{P}_\Delta, \mathcal{P}_\Delta^c$ and $\mathcal{P}_\Delta^s$). In Figure 8 we present the analogous results for I^2-GNN. The dataset and experimental settings are identical to Section 6.1. Additionally, in Table 3 we include the training errors of the models with respect to the MAE (ℓ_1) and the MAE normalized by the ground-truth count (ℓ_c). Specifically, we present the average error over five models trained using different initialization seeds and the standard errors. Note ℓ_1 error of 3-Path is considerably higher than 0.5, which makes the search of correctly predicted test graphs almost impossible. For this reason, we excluded the 3-Path pattern from the adversarial robustness experiments. On the new patterns, the results remain aligned to the discussion in Section 4 except for 2-Path in $\mathcal{P}_\Delta^c, \mathcal{P}_\Delta^s$, and 3-Star in $\mathcal{P}_\Delta^s$. However, these specific cases are caused by the fact that the perturbation spaces contain almost no elements (see bold elements in Table 4).

Table 3. Test error of the architectures GIN, PPGN, and I^2-GNN on the Stochastic Block Model dataset with respect to ℓ_1 and ℓ_c metrics for different subgraph-counting tasks. The results are expressed as the average over the 5 different weight initialization $\pm$ the standard error.

Arch.	Loss	Triangle	2-Path	4-Clique	Chord. C.	Tailed Tr.	3-Star	4-Cycle	3-Path
GIN	ℓ_1	2.6 $\pm$ 0.017	7.6 $\pm$ 0.062	0.81 $\pm$ 0.0041	4.4 $\pm$ 0.0087	15.0 $\pm$ 0.14	21.0 $\pm$ 0.076	6.3 $\pm$ 0.012	40.0 $\pm$ 0.15
	ℓ_c	0.13 $\pm$ 6.2e-4	0.03 $\pm$ 2.3e-4	0.34 $\pm$ 0.0015	0.27 $\pm$ 0.0019	0.098 $\pm$ 7.8e-4	0.087 $\pm$ 4.0e-4	0.18 $\pm$ 0.0015	0.053 $\pm$ 1.5e-4
PPGN	ℓ_1	0.014 $\pm$ 5.5e-4	0.15 $\pm$ 0.0098	0.038 $\pm$ 0.001	0.38 $\pm$ 0.0069	1.1 $\pm$ 0.065	0.81 $\pm$ 0.075	0.23 $\pm$ 0.0083	2.7 $\pm$ 0.12
	ℓ_c	6.9e-4 $\pm$ 2.2e-5	6.0e-4 $\pm$ 4.0e-5	0.013 $\pm$ 6.5e-4	0.017 $\pm$ 4.7e-4	0.0061 $\pm$ 4.0e-4	0.0033 $\pm$ 3.0e-4	0.0061 $\pm$ 2.0e-4	0.0035 $\pm$ 1.7e-4
I^2-GNN	ℓ_1	0.0069 $\pm$ 4.4e-4	0.26 $\pm$ 0.025	0.0031 $\pm$ 2.9e-4	0.058 $\pm$ 0.005	0.39 $\pm$ 0.022	0.6 $\pm$ 0.042	0.25 $\pm$ 0.093	1.5 $\pm$ 0.31
	ℓ_c	3.8e-4 $\pm$ 3.0e-5	0.001 $\pm$ 9.9e-5	6.7e-4 $\pm$ 7.0e-5	0.0028 $\pm$ 2.2e-4	0.0022 $\pm$ 1.4e-4	0.0024 $\pm$ 1.7e-4	0.0068 $\pm$ 0.0026	0.0021 $\pm$ 4.0e-4

Table 4. Average size of the preserving perturbations with budget 1. The values represent an estimation of the number of alternatives that each step of the search algorithm has.

	Trangle	2-Path	4-Clique	Chord. C.	Tailed Tr.	3-Star	4-Cycle	3-Path
$\lvert\mathcal{P}_1^c(G)\rvert$	205.32	**1.69**	410.93	244.04	27.94	12.24	88.98	1.64
$\lvert\mathcal{P}_1^s(G)\rvert$	205.32	**0.02**	410.93	241.82	27.62	**1.6**	74.34	0.62

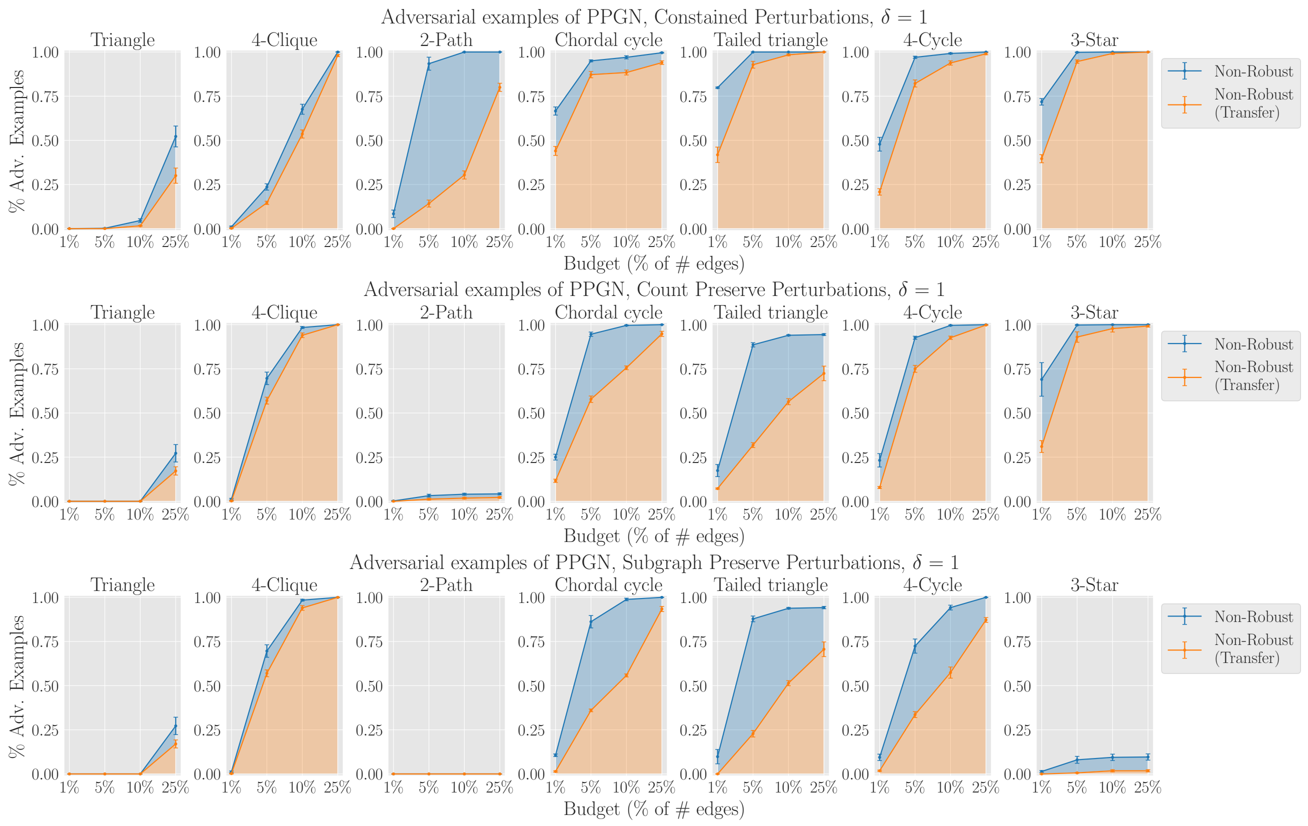

Figure 7. The plots illustrate in blue the success rate of our subgraph-counting adversarial attacks at finding perturbations that represent adversarial examples according to Definition 4.1 in the three perturbation spaces we defined for PPGN. In orange, we represent how effective the adversarial examples are when transferred to the models trained with a different initialization seed. The values are the average of the results obtained with 5 different initialization seeds with the relative standard errors.

C.2. Structural Properties of Adversarial Examples

The previous experiments identify the adversarial examples for the two more expressive architectures, then we can analyze them to get a few insights about the reasons why the architectures are not robust. In Figure 9 we compare the distribution of

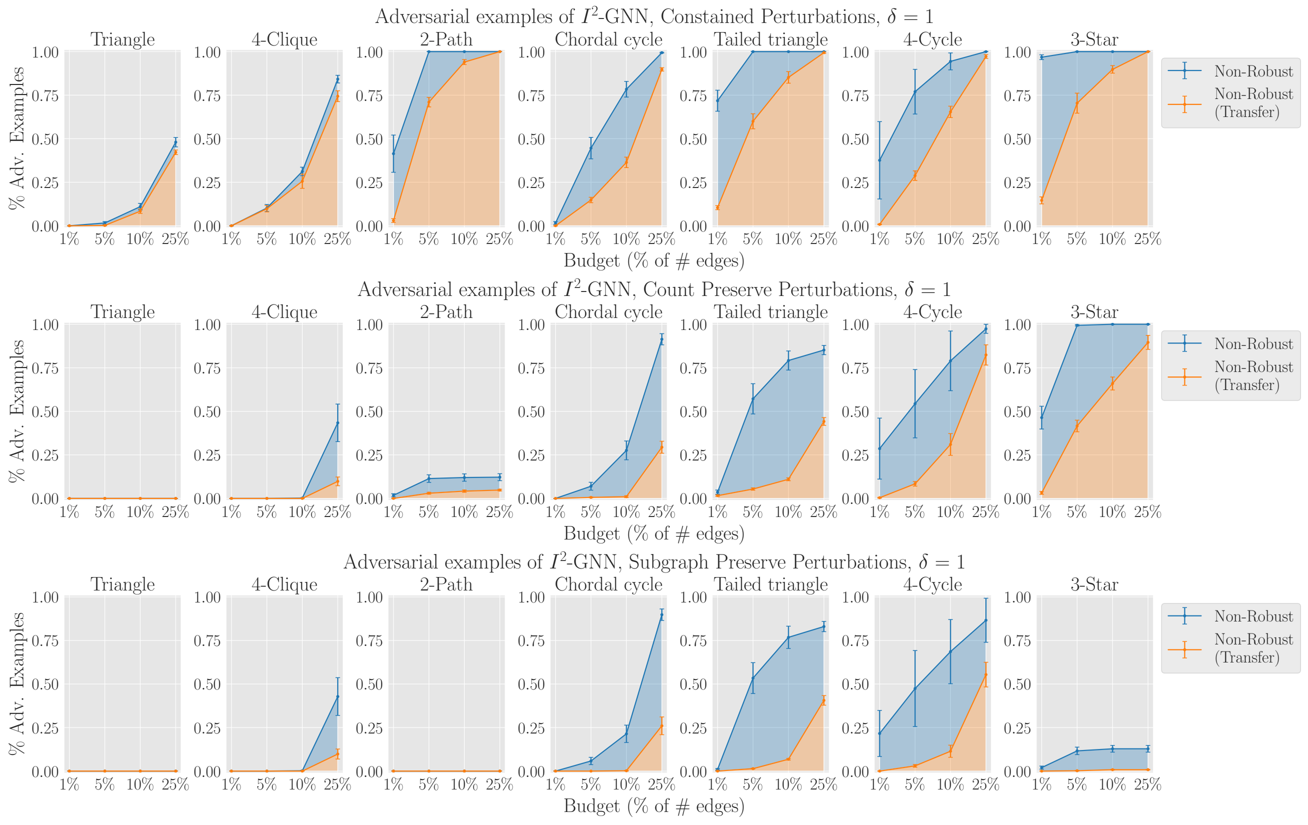

Figure 8. The plots illustrate in blue the success rate of our subgraph-counting adversarial attacks at finding perturbations that represent adversarial examples according to Definition 4.1 in the three perturbation spaces we defined for I^2-GNN. In orange, we represent how effective the adversarial examples are when transferred to the models trained with a different initialization seed. The values are the average of the results obtained with 5 different initialization seeds with the relative standard errors.

the subgraph counts of the test graphs and the adversarial graphs generated from the constrained perturbation space. This experiment is designed to explore whether the adversarial examples generation exploits the fact that this perturbation space allows changing significantly the semantic meaning of the graphs. Concretely, for each pattern, we extract the successful transferring adversarial examples, i.e. perturbations that fool all five trained models, with budget $\Delta = 10\%$, and compare their subgraph-count distribution to the distribution of the corresponding clean graphs. We specifically choose to consider exclusively the adversarial examples that transfer to the other models because they intuitively represent the failure modes that affect the architectures in general. Moreover, we only consider patterns where at least 5% of attacks have produced a successful adversarial example. This makes the representations more reliable since the distributions are estimated over at least 25 samples [2]. To plot the two distributions we use a *violinplot*, which, differently from a box plot, gives also a visual representation of probability density. From the plots, we notice a variation in the distribution for several patterns. In particular, all the adversarial distributions have heavier tails, which means that outliers counts are more frequent, and in some cases also the whole distribution is shifted. This result suggests that altering the ground truth count is a failure mode for both architectures and justifies the introduction of more restrictive perturbation spaces which preserve the subgraph count. However, the adversarial attacks are capable of generating adversarial examples also for the count and subgraph preserving spaces, which encourages us to further investigate additional failure modes. In Figure 10 we continue with this analysis by comparing the distribution of the number of edges of the adversarial graphs and the clean graphs. The plots specifically show the distributions of the transferring adversarial examples generated from the count preserving perturbation space with budget $\Delta = 25\%$ and the corresponding test graphs. Here we find that the adversarial graphs tend to have more edges they seem to belong to a different graph distribution.

[2]We run 500 attacks, 100 for each one of the 5 models

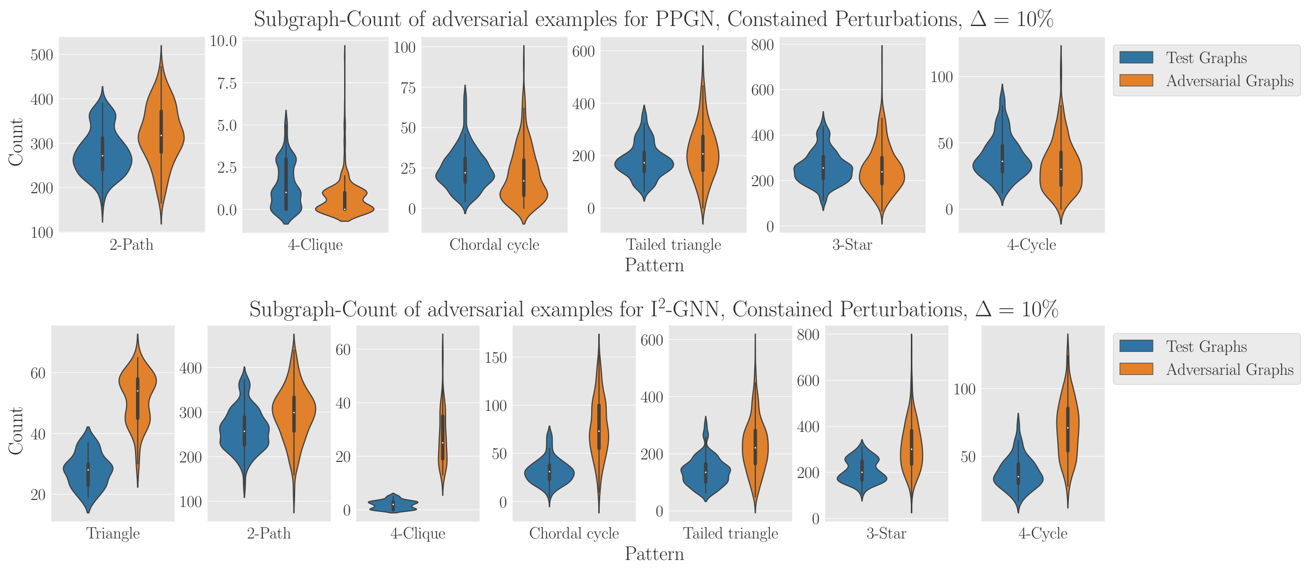

Figure 9. Distribution of the **subgraph counts** of the transferring adversarial examples (in orange) and their relative clean graphs (in blue). In particular, the adversarial examples are generated from the perturbation space $\mathcal{P}_{10\%}$, and we present only the patterns having at least 5% of successful attacks.

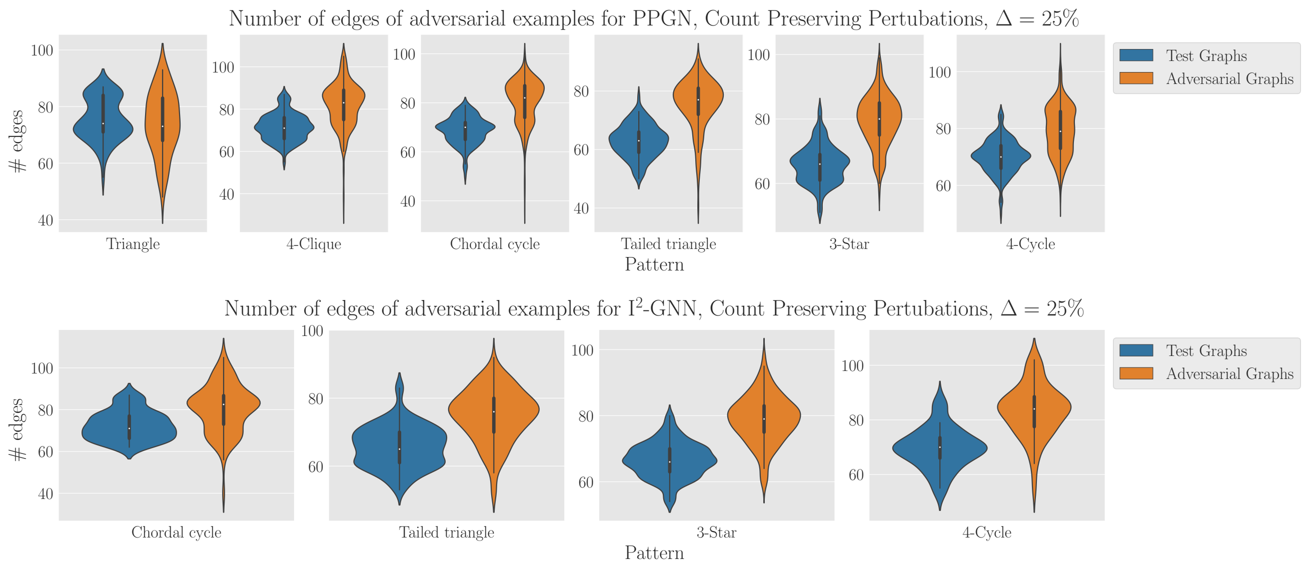

Figure 10. Distribution of the **number of edges** of the transferring adversarial examples (in orange) and their relative clean graphs (in blue). In particular, the adversarial examples are generated from the perturbation space $\mathcal{P}^{c}_{25\%}$, and we present only the patterns having at least 5% of successful attacks.

Table 5. P-values that represent the statistical significance of the shift between the **subgraph count distributions** of the transferring adversarial graph extracted from $\mathcal{P}_\triangle$ and their relative clean graphs. In particular, we report the p-values of a t-test for the null hypothesis that the expected values of the distributions are the same. The p-values of samples with less than 25 elements are omitted.

Arch.	Budget	Triangle	2-Path	4-Clique	Chord. C.	Tailed Tr.	3-Star	4-Cycle
PPGN	1%	-	-	-	0.4	0.0095	0.63	0.87
	5%	-	0.0067	0.11	0.64	7.0e-11	0.25	0.32
	10%	-	8.4e-10	3.0e-11	9.7e-5	2.6e-9	0.19	2.8e-9
	25%	2.2e-6	4.6e-10	3.0e-16	1.6e-28	0.062	1.0e-16	1.3e-116
I^2-GNN	1%	-	-	-	-	0.04	0.15	-
	5%	-	8.2e-6	5.7e-38	9.5e-17	1.9e-22	1.5e-22	1.6e-10
	10%	2.1e-24	1.1e-19	7.9e-59	1.9e-45	2.7e-51	2.1e-57	2.0e-33
	25%	2.2e-103	1.1e-80	1.5e-102	8.5e-112	3.5e-86	1.5e-78	7.9e-105

Table 6. P-values that represent the statistical significance of the shift between the **distributions of the number of edges** of the transferring adversarial graph extracted from $\mathcal{P}^c_\triangle$ and their relative clean graphs. In particular, we report the p-values of a t-test for the null hypothesis that the expected values of the distributions are the same. The p-values of samples with less than 25 elements are omitted.

Arch.	Budget	Triangle	2-Path	4-Clique	Chord. C.	Tailed Tr.	3-Star	4-Cycle
PPGN	1%	-	-	-	5.2e-5	0.49	0.89	0.0073
	5%	-	-	1.1e-4	0.53	0.0017	7.3e-12	1.7e-5
	10%	-	-	0.0069	1.2e-11	1.8e-18	5.0e-37	4.1e-4
	25%	0.51	-	1.3e-67	5.2e-99	2.3e-83	7.4e-147	6.8e-75
I^2-GNN	1%	-	-	-	-	-	0.69	-
	5%	-	-	-	-	0.32	7.9e-6	-
	10%	-	-	-	-	0.029	1.1e-20	-
	25%	-	-	-	2.7e-8	1.6e-26	3.8e-110	1.8e-19

Additionally, we investigate the statistical significance of the aforementioned distributional shifts. To do so, we compute the p-values of the t-tests (with no equal population variance assumption) for the null hypothesis that the distributions of the adversarial and clean graphs have the same expected value. Table 5 shows the p-values for the tests for the subgraph count distributions of adversarial examples extracted from the constrained perturbation space and the clean graphs. Instead, Table 6 shows the analogous result for the distribution of the number of edges and for adversarial graphs searched in the count-preserving perturbation space. Moreover, as before we only consider the subgraphs and budget from which at least 5% of the attacks are successful, for all the other cases we omit the p-value. All in all, the results show that from budget 10% we can affirm for almost all the distribution shift is statistically significant.

C.3. Out-of-Distribution Generalization

Similarly, we present also the OOD experiments results for all the patterns in Table 7. Also in this case the discussion of the results in Section 6.2 complies also with the results on the new patterns.

Table 7. Test errors of the OOD experiments that investigate the generalization abilities of the architectures. Specifically, d_i represents models trained and tested on the same dataset d_i, OOD models trained on d_1 and tested in d_2 and in MLP we additionally retrain the final layers on d_2.

Arch.	Exp.	Trangle		2-Path		4-Clique		Chord. C.		Tailed Tr.		3-Star		4-Cycle		3-Path	
	Setting	ℓ_1	ℓ_c	ℓ_1	ℓ_c	ℓ_1	ℓ_c	ℓ_1	ℓ_c	ℓ_1	ℓ_c	ℓ_1	ℓ_c	ℓ_1	ℓ_c	ℓ_1	ℓ_c
PPGN	d_1	0.0058	7.8e-4	0.016	4.7e-1	0.0098	0.0027	0.11	0.011	0.28	0.011	0.12	0.012	0.058	0.010	0.28	0.0090
	OOD	2.98	0.041	5.42	0.16	7.87	0.12	20.12	0.25	28.57	2.40	5.47	4.16	5.47	1.21	13.91	5.53
	d_2	0.0091	1.7e-4	0.012	2.6e-4	0.049	0.0016	0.12	0.0017	0.12	0.0029	0.017	0.0021	0.040	0.0050	0.063	0.0058
	MLP	0.059	9.8e-4	0.11	0.0027	0.32	0.0067	1.083	0.014	1.04	0.054	0.24	0.081	0.29	0.044	0.72	0.10
I^2-GNN	d_1	0.0027	2.8e-4	0.027	8e-4	0.0054	5.8e-4	0.021	0.0023	0.065	0.0030	0.080	0.0079	0.035	0.0062	0.065	0.0032
	OOD	3.27	0.044	1.61	0.046	21.1	0.30	6.83	0.086	21.51	1.97	3.14	2.34	2. 16	0.45	7.29	1.96
	d_2	0.032	6.2e-4	0.028	5.9e-4	0.048	0.0015	0.30	0.0042	0.18	0.0057	0.070	0.0093	0.028	0.0031	0.12	0.011
	MLP	0.2	0.0031	0.19	0.0048	0.91	0.015	1.59	0.021	1.2	0.032	0.22	0.045	0.19	0.026	0.37	0.066