# Supplementary Material: From Local Structures to Size Generalization in Graph Neural Networks

## A. Size generalization in Single-layer GNNs

We start our discussion on size generalization with a theoretical analysis of a simple setup. We consider a single-layer GNN and an easy task and show that: (1) The training objective has many different solutions, but only a small subset of these solutions generalizes to larger graphs (2) Simple regularization techniques cannot mitigate the problem.

Assume we train on a distribution of graphs. Our task is to predict the number of edges in the graph using a first-order GNN with a single linear layer and additive readout function, for simplicity also consider the squared loss. We first note that the task of edge count can be solved with this architecture for graphs of any size. The intuition is to count the number of neighbors for each node (can be done with a 1-layer GNN), and summing over all nodes using the readout function would give us $2|E|$, where $E$ is the set of edges.

The objective boils down to the following function for any graph $G$ in the training set:

$$L(w_1, w_2, b; G) = \left( \sum_{u \in V(G)} \left( w_1 \cdot x_u + \sum_{v \in \mathcal{N}(u)} w_2 \cdot x_v + b \right) - y \right)^2 .$$

Here, $G$ is an input graph, $V(G)$ are the nodes of $G$, $\mathcal{N}(v)$ are all the neighbors of node $v$, $w_1, w_2$ and $b$ are the trainable parameters, $y$ is the target and $x_v$ is the node feature for node $v$. Further, assume that we have no additional information on the nodes, so we can just embed each node as a one-dimensional feature vector with a fixed value of 1. In this simple case, the trainable parameters are also one-dimensional.

For a graph with $n$ nodes and $m$ edges the training objective can also be written in the following form:

$$L(w_1, w_2, b; G) = (nw_1 + 2mw_2 + nb - m)^2 ,$$

One can easily find the solution space, which is an affine subspace defined by $w_2 = \frac{m - n(w_1+b)}{2m} = \frac{1}{2} - \frac{n}{2m} \cdot \frac{w_1+b}{2m}$. In particular, the solutions with $w_1 + b = 0,\ w_2 = 1/2$ are the only ones which do not depend on the specific training set graph size $n$, and generalize to graphs of any size and with any number of edges.

It can be readily seen that when training the model on graphs where the ratio $n/m$ between the number of nodes and number of edges is fixed, gradient descent with a small enough learning rate will favor the global solution closest to the initialized point. Hence, by using a standard initialization scheme (e.g. (Glorot & Bengio, 2010)), with probability 1, the solution that gradient descent converges to is not a generalizing solution. Note that we could train on many graphs with different sizes and still end up in a non-generalizing solution, as long as $n/m$ is fixed. On the other hand, training on graphs with different node/edge ratios necessarily leads to some generalizing solution. This is because the problem is convex, and the generalizing solutions are the only solutions that minimize the loss for graphs with different ratios.

We also note that the generalizing solution ($w_1 + b = 0,\ w_2 = 1/2$) is not the least norm solution in general (with respect to both $L_1$ and $L_2$ norms) so simple regularization will not help here (it is the least $L_1$ norm solution if $2m > n$). As we show in Sec. 6, the problem gets worse when considering GNNs with multiple non-linear layers, and this simple solution will not help in this case: we can train deeper GNNs on a wide variety of sizes and the solution will not generalize to other sizes.

## B. Proofs from Sec. 4

*Proof of Thm. 4.2.* We show that from the definition of $d$-patterns, and the 1-WL algorithm (see (Weisfeiler & Lehman, 1968)), the color given by the WL algorithm to two nodes is equal iff their $d$-pattern is equal. For the case of $d = 0$, it is clear.

Suppose it is true for $d-1$, the WL algorithm at iteration $d$ give node $v$ a new color based on the colors given in iteration $d-1$ to the neighbors of $v$. This means, that the color of $v$ at iteration $d$ depends on the multiset of colors at iteration $d-1$ of its neighbors, which by induction is the $(d-1)$-pattern of the neighbors of $v$. To conclude, we use Theorem 1 from (Morris et al., 2019) which shows that GNNs are constant on the colors of WL, hence also constant on the $d$-patterns. □

To prove Thm. 4.3 we will first need to following claim from (Yun et al., 2019) about the memorization power of ReLU networks:

**Theorem B.1.** *Let $\{\mathbf{x}_i, y_i\}_{i=1}^N \in \mathbb{R}^d \times \mathbb{R}$ such that all the $\mathbf{x}_i$ are distinct and $y_i \in [-1,1]$ for all $i$. Then there exists a 3-layer fully connected ReLU neural network $f : \mathbb{R}^d \to \mathbb{R}$ with width $2\sqrt{N}$ such that $f(\mathbf{x}_i) = y_i$ for every $i$.*

We will also need the following lemma which will be used in the construction of each layer of the GNN:

**Lemma B.2.** *Let $N \in \mathbb{N}$ and $f : \mathbb{N} \to \mathbb{R}$ be a function defined by $f(n) = \mathbf{w}_2^\top \sigma(\mathbf{w}_1 n - \mathbf{b})$ where $\mathbf{w}_1, \mathbf{w}_2, \mathbf{b} \in \mathbb{R}^N$, and $\sigma$ is the ReLU function. Then for every $y_1, \ldots, y_N \in \mathbb{R}$ there exists $\mathbf{w}_1, \mathbf{w}_2, \mathbf{b}$ such that $f(n) = y_n$ for $n \leq N$ and $f(n) = (n-N+1)y_N - (n-N)y_{N-1}$ for $n > N$.*

*Proof.* Define $\mathbf{w}_1 = \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix}$, $\mathbf{b} = \begin{pmatrix} 0 \\ 1 \\ \vdots \\ N-1 \end{pmatrix}$. Let $a_i$ be the $i$-th coordinate of $\mathbf{w}_2$, we will define $a_i$ recursively in the following way: Let $a_1 = y_1$, suppose we defined $a_1, \ldots, a_{i-1}$, then define $a_i = y_i - 2a_{i-1} - \cdots - ia_1$. Now we have for every $n \leq N$:

$$f(n) = \mathbf{w}_2^\top \sigma(\mathbf{w}_1 n - \mathbf{b}) = na_1 + (n-1)a_2 + \cdots + a_n = y_n \,.$$

For $n > N$ we can write $n = N + k$ for $k \geq 1$, then we have:

$$\begin{aligned} f(n) &= \mathbf{w}_2^\top \sigma(\mathbf{w}_1(N+k) - \mathbf{b}) = (N+k)a_1 + (N+k-1)a_2 + \cdots + (k+1)a_N \\ &= y_N + k(a_1 + a_2 + \cdots + a_N) = y_N + k(y_N - a_{N-1} - 2a_{N-2} - \cdots - (N-1)a_1) \\ &= (k+1)y_N - ky_{N-1} \end{aligned}$$

□

Now we are ready to prove the main theorem:

*Proof of Thm. 4.3.* We assume w.l.o.g that at the first iteration each node $i$ is represented as a one-hot vector $\mathbf{h}_i^{(0)}$ of dimension $|C|$, with its corresponding node feature. Otherwise, since there are $|C|$ node features we can use one GNN layer that ignores all neighbors and only represent each node as a one-hot vector. We construct the first $d$ layers of the 1-GNN $F$ by induction on $d$. Denote by $a_i = |C| \cdot \left(N^i + N^{i-1}\right)$ the dimension of the $i$-th layer of the GNN for $1 \leq i \leq d$, and $a_0 = |C|$.

The mapping has two parts, one takes the neighbors information and maps it into a feature representing the multiset of $d$-patterns, the other part is simply the identity-saving information regarding the d-pattern of the node itself.

The $d$ layer structure is

$$\mathbf{h}_v^{(d)} = U^{(d+1)} \sigma \left( W_2^{(d)} \mathbf{h}_v^{(d-1)} + \sum_{u \in \mathcal{N}(v)} W_1^{(d)} \mathbf{h}_u^{(d-1)} - \mathbf{b}^{(d)} \right)$$

We set $W_2^{(d)} = [0, I]^T$, $W_1^{(d)} = [\tilde{W}_1^{(d)T}, 0]^T$ and $U^{(d+1)} = [\tilde{U}^{(d+1)T}, 0]^T$ with $\tilde{W}_1^{(d)} \in \mathbb{R}^{Na_{d-1} \times a_{d-1}}$ and $\tilde{U}^{(d+1)} \in \mathbb{R}^{Na_{d-1} \times Na_{d-1}}$. For $\tilde{W}_1^{(d)}$ we set $\mathbf{w}_i^{(1)}$, its $i$-th row, to be equal to $\mathbf{e}_n$ where $n = \left\lceil \frac{i}{N} \right\rceil$. Let $b_i^{(1)}$ be the $i$-th coordinate of $\mathbf{b}^{(d)}$, be equal to $i - 1 \pmod N$ for $i \leq N \cdot a_{d-1}$ and zero otherwise. What this does for the first $a_{d-1}$ elements of $W_2^{(d)} \mathbf{h}_v^{(d)} + \sum_{u \in \mathcal{N}(v)} W_1^{(d)} \mathbf{h}_u^{(d)}$ is that each dimension $i$ hold the number of neighbors with a specific

(d-1)-pattern. We then replicate this vector N times, and for each replica we subtract a different bias integer ranging from 0 to $N-1$. To that output we concatenate the original $\mathbf{h}_v^{(d-1)}$

Next we construct $\tilde{U}^{(d+1)} \in \mathbb{R}^{Na_{d-1} \times Na_{d-1}}$ in the following way: Let $\mathbf{u}_i^{(d+1)}$ be its $i$-th row, and $u_{i,j}^{(d+1)}$ its $j$-th coordinate. We set $u_{i,j}^{(d+1)} = 0$ for every $j$ with $j \neq \lceil \frac{i}{N} \rceil$ and the rest $N$ coordinates to be equal to the vector $\mathbf{w}_2$ from Lemma B.2 with labels $y_\ell = 0$ for $\ell \in \{1, \ldots, N\} \setminus \{(i \bmod N) + 1\}$ and $y_\ell = 1$ for $\ell = (i \bmod N) + 1$.

Using the above construction we encoded the output on node $v$ of the first layer of $F$ as a vector:

This encoding is such that the $i$-th coordinate of $\mathbf{h}_v^{(d+1)}$ for $1 \leq i \leq N \cdot a_{d-1}$ is equal to 1 iff node $v$ have $(i \bmod N) + 1$ neighbors with node feature $\lceil \frac{i}{N} \rceil \in \{1, \ldots, |C|\}$. The last $a_{d-1}$ rows are a copy of $\mathbf{h}_v^{(d-1)}$.

**Construction of the suffix.** Next, we construct the last two layers. First we note that for a node $v$ with $d$-pattern $p$ there is a unique vector $\mathbf{z}_p$ such that the output of the GNN on node $v$, $\mathbf{h}_v^{(d)}$, is equal to $\mathbf{z}_p$. From our previous construction one can reconstruct exactly the (d-1)-pattern of each node, and the exact number of neighbors with each (d-1)-pattern and therefore can recover the d-pattern correctly from the $\mathbf{h}_v^{(d)}$ embedding.

Let $y_{\max} := \max_i |y_i|$, and define $\tilde{y}_i = y_i / y_{\max}$. Finally, we use Thm. B.1 to construct the 3-layer fully connected neural network with width at most $2\sqrt{|P|}$ such that for every pattern $p_i \in P$ with corresponding unique representation $\mathbf{z}_{p_i}$ and label $\tilde{y}_i$, the output of the network on $\mathbf{z}_{p_i}$ is equal to $\tilde{y}_i$. We construct the last two layers of the GNN such that $W_1^{(d+1)}, W_1^{(d)} = 0$, and $W_2^{(d+1)}, \mathbf{b}^{(d+1)}, W_2^{(d+2)}, \mathbf{b}^{(d+2)}, W^{(d+3)}$ are the matrices produced from Thm. B.1. Note that $W^{(d+3)}$ is the linear output layer constructed from the theorem, thus the final output of the GNN for node $v$ is $W^{(d+3)} \cdot h_v^{(d+2)}$, where $h_v^{(d+2)}$ is the output after $d+2$ layers. We multiply the final linear output layer by $y_{\max}$, such that the output of the entire GNN on pattern $p_i$ is exactly $y_i$. □

## C. Proofs from Sec. 5

*Proof of Thm. 5.1.* By the assumption, there is a depth $d$ GNN that solves the task for graphs of any size, denote this GNN by $F$. We can write $F$ operating on a graph $G$ as $F(G) = M\left(\sum_{v \in V(G)} H(G)_v\right)$, where $V(G)$ are the nodes of $G$, $H$ is a $d$-layer GNN with $H(G)_v$ is the output of $H$ on node $v$, and $M$ is an MLP. Note that we used the summation readout function which sums the output of the GNN over all the nodes of the input graph.

We will construct a new GNN $F'$ in the following way: Let $P$ be the set of $d$-patterns which appear in $P_1^d$ and $\tilde{P}$ all the $d$-patterns which appear in $P_2^d$ but not in $P_1^d$. Note that by the assumption that the distributions are with finite support, both $P$ and $\tilde{P}$ are finite. Suppose that $H(G)_v \in \mathbb{R}^k$, that is the dimension of the output feature vector for each $v \in V(G)$ is $k$-dimensional. We define a $d+2$-layer GNN $H'$ with output dimension $k+1$, such that:

1. For each $p \in P$ and node $v$ with $d$-pattern $p$, the output of $H'$ on $v$ is equal to $\begin{pmatrix} H(G)_v \\ 0 \end{pmatrix}$.

2. For each $\tilde{p} \in \tilde{P}$ and node $\tilde{v}$ with $d$-pattern $\tilde{p}$, the output of $H'$ on $\tilde{v}$ is equal to $\begin{pmatrix} 0 \\ 1 \end{pmatrix}$.

This construction is possible using Thm. 4.3 since both $P$ and $\tilde{P}$ are finite. We define an MLP $M'$ which have one more layer than $M$ in the following way: All the weights in all the layers except the last one are block matrices of the form $\begin{pmatrix} W & \mathbf{0} \\ \mathbf{0}^\top & 1 \end{pmatrix}$, where $W$ is the original weight matrix from $M$, and $\mathbf{0}$ is a vector of zeroes of the corresponding size. Let $y \in \mathbb{R}$ be the maximum output of the task (in absolute value) over all the graphs from both $P_1$ and $P_2$. We define the last layer of $M'$ as: $\begin{pmatrix} 1 & 0 \\ 0 & 2y \end{pmatrix}$. Finally, define $F'(G) = M'\left(\sum_{v \in V(G)} H'(G)_v\right)$.

We will now show the correctness of the construction. For any $d$-pattern $p \in P$, and $v$ with $d$-pattern $p$ it is clear that

$H'(G)_v = \begin{pmatrix} H(G)_v \\ 0 \end{pmatrix}$. Hence, for every graph $G$ coming from the distribution $P_1$ we have that

$$F'(G) = M'\left(\begin{pmatrix} \sum_{v \in V(G)} H(G)_v \\ 0 \end{pmatrix}\right) = F(G)$$

On the other side, let $G$ be some graph with from the distribution $P_2$. Then, there is some $\tilde{v} \in V(g)$ with $\tilde{v} \in \tilde{P}$, hence we have that $\sum_{v \in V(G)} H'(G)_v = \begin{pmatrix} \boldsymbol{x} \\ z \end{pmatrix}$, where $\boldsymbol{x}$ is some vector and $z \geq 1$. Hence, we have that $F'(G) = M'\left(\begin{pmatrix} \boldsymbol{x} \\ z \end{pmatrix}\right) > y$. This means that the output of $F'$ on any graph drawn from $P_2$ is not the correct output for the task. □

*Proof of Thm. 5.2.* By the assumption, the output of the task is determined by the $d$-patterns of the nodes. For each node with pattern $p_i$ let $y_i$ be the output of the node prediction task. Define

$$A = \arg \max_{A' : P_1^d(A') < \epsilon} P_2^d(A') \tag{1}$$

By Thm. 4.3 there exists a first order GNN such that for any $d$-pattern $p_i \in A$ gives a wrong label and for every pattern $p_j \in (P_1 \cup P_2) \setminus A$ gives the correct label. Note that we can use Thm. 4.3 since both $A$ and $(P_1 \cup P_2) \setminus A$ are finite, because we assumed that distributions on the graphs have finite supports. The 0-1 loss for small and large graphs is exactly $P_1^d(A)$ and $P_2^d(A)$ respectively.

□

## D. Additional experiments from Sec. 6

### D.1. Experiments on Larger Graphs

We conducted the experiment from Fig. 3 (a) with much larger graphs. We used a three-layer GNN and tested on graphs with $n \in [50, 500]$ nodes. See Fig. 8 The problem of size generalization persists, where increasing the graph size also significantly increases the loss on the test set. We stress that in all the experiments, the loss on the validation set is effectively zero.

### D.2. Max-clique Size

We consider the max-clique problem. The goal of this problem is given a graph, to output the size of the maximal clique. This is in general an NP-hard problem, hence a constant depth GNN will not be able to solve it for graphs of all sizes. For this task we sampled both the train and test graphs from a geometrical distribution, which resembles how graphs are created from point clouds, defined as follows: given a number of nodes $n$ and radius $\rho$ we draw $n$ points uniformly in $[0, 1]^2$, each point correspond to a node in the graph, and two nodes are connected if their corresponding points have a distance less than $\rho$. We further analyzed the effects of how the network depth and architecture affect size generalization.

Table 2 presents the test loss on the max-clique problem. Deeper networks are substantially more stricken due to the domain shift. If the test domain has a similar pattern distribution, increasing the neural network depth from one layer to three layers results in a small decrement of at most $25\%$ to the loss. However, if the pattern distribution is different than the train pattern distribution, such change may increase the loss by more than $5.5\times$. We also show that the problem is consistent in both first-order GNN and GIN architectures.

### D.3. Different Architectures and Node/Graph Level Tasks

We tested on the following tasks: (1) student-teacher task, on both graph and node levels, (2) per node degree prediction task, and (3) predicting the number of edges in the graph. The goal of these experiments is to show that the size-generalization problem persists on different tasks, different architectures, and its intensity is increased for deeper GNN. In all the experiments we draw the graphs from $G(n, p)$ distribution, wherein the test set $n$ is drawn uniformly between 40 and 50, and $p = 0.3$, and in the test set $n = 100$ and $p$ is either 0.3 or 0.15. We note that when $p = 0.15$, the average degree of the test graph is

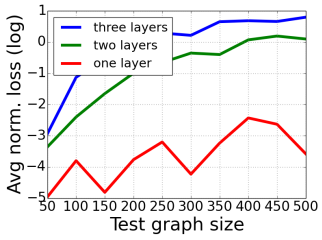


*Figure 7.* Size generalization in PA graphs. The networks were trained

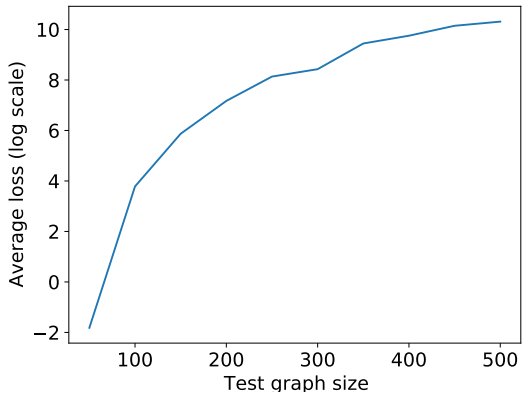


*Figure 8.* Extending the experiment in Fig. 3 (a) to larger graphs. We train on graphs with bounded size $n \in [40, 50]$ and test on graphs with size up to $n = 500$ with constant $p = 0.3$.

equal to (approximately) the average degree of the train graph, while when $p = 0.3$ it is twice as large. We would expect that the model will generalize better when the average degree in the train and test set is similar because then their $d$-patterns will also be more similar.

Table 3 compares the performance of these tasks when changing the graph size of the test data. We tested the performance with the squared loss. We note that the GNNs (both first-order GNN and GIN) successfully learned all the tasks presented here on the train distribution, here we present their generalization capabilities to larger sizes.

### D.4. Different data distribution - Preferential Attachment

We performed an experiment on the preferential attachment graph model. In this model, the graph distribution is determined by two parameters $n$ and $m$. Each graph is created sequentially, where at each iteration a new node is added to the graph, up to $n$ nodes. Each new node is connected to exactly $m$ existing nodes, where the probability to connect to node $v$ is proportional to its degree. This means that higher degree nodes have a higher degree to have more nodes connected to them.

We trained on the task of edge count (i.e. predicting the number of edges in a graph) for graphs sampled from a preferential attachment model with $n$ uniformly sampled from $[10, 50]$ and $m = 4$. We tested on graphs also sampled from a preferential attachment model with $n$ nodes for $n$ varying from 50 to 500, and $m = 4$. Note that this task can be solved efficiently for any graph distribution. Fig. 7 depicts the results. It is clear that for depth 2 and 3 GNNs, as the graph size gets larger, the GNN fails to predict the number of edges. This shows that although for this problem there is a solution that works on all graph sizes, and we trained on graphs of varying sizes, GNNs tend to converge to solutions that do not generalize to larger sizes.

### D.5. Generalization From Large to Small Graphs

We additionally tested size generalization in the opposite direction, i.e. generalizing from large to small graphs. We used the same teacher-student setting as in Fig. 3, where the graphs are drawn from a $G(n, p)$ distribution with a constant $p = 0.3$. We consider three separate training sets with graphs of sizes $[90, 100]$, $[140, 150]$, $[190, 200]$ and tested on graphs of sizes $50,\ 75$. The rows of Table 4 clearly show that when the size difference between the graphs in the train and test sets decreases, the loss also decreases, which is consistent with our theory.

| | first order GNN | | | GIN | | |
|---|---|---|---|---|---|---|
| $\rho_{train}/\rho_{test}$ | 1-layer | 2-layers | 3-layers | 1-layer | 2-layers | 3-layers |
| 1 | $402 \pm 59$ | $926 \pm 245$ | $2325 \pm 3613$ | $367 \pm 56$ | $620 \pm 130$ | $634 \pm 1042$ |
| $\sqrt{2}$ | $96 \pm 5$ | $111 \pm 4$ | $119 \pm 5$ | $101 \pm 3$ | $114 \pm 6$ | $123 \pm 14$ |

*Table 2.* The difference is the predicted max clique size under size generalization domain shift. The train domain graphs were constructed by drawing $n \in [40, 50]$ points uniformly in the unit square, and connecting two points if their distance is less than $\rho_{train} = 0.3$. The test set domain graphs contain $n = 100$ nodes, effectively increasing their density by 2. We tested with two different values of $\rho_{train}/\rho_{test}$, the ratio between the train and test connectivity radius. A proper scaling that keeps the expected degree of each node is $\rho = \sqrt{2}$. Here, although proper scaling does not help solve the problem completely, it does improve performance.

| | | Node regression | | Graph regression | |
|---|---|---|---|---|---|
| | $p$ | Student - Teacher | Degree | Student - Teacher | Edge count |
| first order GNN (Morris et al., 2019) | 0.3 | $3500 \pm 9240$ | $348 \pm 553$ | $(1.8 \pm 3) \cdot 10^4$ | $(1.8 \pm 2.4) \cdot 10^6$ |
| | 0.15 | $\mathbf{0.02 \pm 0.05}$ | $\mathbf{(1.2 \pm 0.8) \cdot 10^{-3}}$ | $\mathbf{0.04 \pm 0.04}$ | $\mathbf{5.1 \pm 2.5}$ |
| GIN | 0.3 | $1384 \pm 3529$ | $73 \pm 86$ | $5487 \pm 9417$ | $(4.2 \pm 4.3) \cdot 10^5$ |
| | 0.15 | $\mathbf{0.96 \pm 0.98}$ | $\mathbf{0.33 \pm 0.04}$ | $\mathbf{0.04 \pm 0.07}$ | $\mathbf{6.7 \pm 5.4}$ |

*Table 3.* Comparing performance on different local distributions (a) A student-teacher graph regression task; (b) A graph regression task, where the graph label is the number of edges; (c) A student-teacher node regression task; (d) A node regression task, where the node label is its degree. In the edge count/degree tasks the loss is the mean difference from the ground-truths, divided by the average degree/number of edges. In the student-teacher tasks the loss is the mean $L_2$ loss between the teacher's value and the student's prediction, divided by the averaged student's prediction. Both the student and teacher share the same 3-layer architecture

## E. SSL task on $d$-pattern tree

First, we will need the following definition which constructs a tree out of the $d$-pattern introduced in Sec. 4. This tree enables us to extract certain properties for each node which can, later on, be learned using a GNN. This definition is similar to the definition of "unrolled tree" from (Morris & Mutzel, 2019).

**Definition E.1** ($d$-pattern tree)**.** *Let $G = (V, E)$ a graph, $C$ a finite set of node features, where each $v \in V$ have a corresponding feature $c_v$, and $d \geq 0$. For a node $v \in V$, its d-**pattern tree** $T_v^{(d)} = (V_v^{(d)}, E_v^{(d)})$ is directed tree where each node corresponds to some node in G. It is defined recursively in the following way: For $d = 0$, $V_v^{(0)} = u_{(0,v)}$, and $E_v^{(0)} = \varnothing$. Suppose we defined $T_v^{(d-1)}$, and let $\tilde{V}_v^{(d-1)}$ be all the leaf nodes in $V_v^{(d-1)}$ (i.e. nodes without incoming edges). We define:*

$$V_v^{(d)} = V_v^{(d-1)} \cup \left\{ u_{(d,v')} : v' \in \mathcal{N}(v''),\ u'_{(d-1,v'')} \in \tilde{V}_v^{(d-1)} \right\}$$
$$E_v^{(d)} = E_v^{(d-1)} \cup \left\{ (u_{(d,v')}, u'_{(d-1,v'')}) : v' \in \mathcal{N}(v''),\ u'_{(d-1,v'')} \in \tilde{V}_v^{(d-1)} \right\}$$

*and for every node $u_{(d,v')} \in V_v^{(d)}$, its node feature is $c_{v'}$ - the node feature of $v'$*

The main advantage of pattern trees is that they encode all the information that a GNN can produce for a given node by running the same GNN on the pattern tree.

This tree corresponds to the local patterns in the following way: the $d$-pattern tree of a node can be seen as a multiset of the children of the root, and each child is a multiset of its children, etc. This completely describes the $d$-pattern of a node. In other words, there is a one-to-one correspondence between $d$-patterns and pattern trees of depth $d$. Thus, a GNN that successfully represents the pattern trees of the target distribution will also successfully represent the $d$-patterns of the target distribution.

Using the $d$-pattern tree we construct a simple regression SSL task where its goal is for each node to count the number of nodes in each layer and of each feature of its $d$-pattern tree. This is a simple descriptor of the tree, which although loses some information about connectivity, does hold information about the structure of the layers.

For example, in Fig. 4 the descriptor for the tree would be that the root (zero) layer has a single black node, the first layer has two yellow nodes, the second layer has two yellow, two gray, and two black nodes, and the third layer has ten yellow, two black and two gray nodes.

| | | Train graph size | | |
|---|---|---|---|---|
| | | $[\mathbf{90, 100}]$ | $[\mathbf{140, 150}]$ | $[\mathbf{190, 200}]$ |
| **Test graph size** | **50** | 1.87 | 3.1 | 4.36 |
| | **75** | 1.93 | 4.19 | 4.29 |

*Table 4.* Training on large graphs and testing on smaller graphs. Teacher-student task with graphs drawn from a $G(n, p)$ distribution with $p = 0.3$ and $n$ varies, as described in the table. Results are on a logarithmic scale. As expected, even generalizing from large to small graphs is difficult, but the results improve as the sizes of the graphs in the train set are close to the sizes of the graphs in the test set.

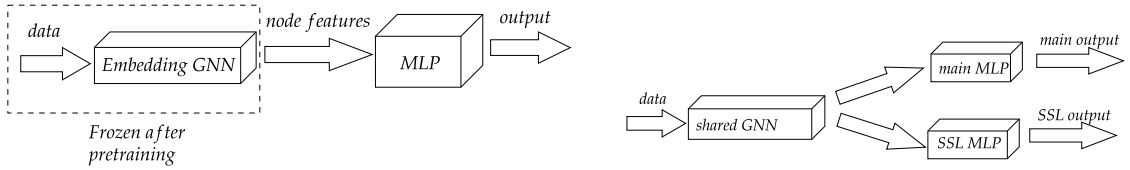


*Figure 9.* Two training procedures for learning with SSL tasks. **Left:** Learning with pretraining: Here, a GNN is trained on the SSL task with a specific SSL head. After training, the weights of the GNN are fixed, and only the main head is trained on the main task. **Right:** Multitask learning: Here, there is a shared GNN and two separate heads, one for the SSL task and one for the main task. The GNN and both heads are trained simultaneously.

## F. Training Procedure

In this section, we explain in detail the training procedure used in the experiments of Sec. 7. Let $X_{Main}$ and $X_{SSL}$ be two labeled datasets, the first contains the labeled examples for the main task from the source distribution, and the second contains examples labeled by the SSL task from both the source and target distributions. Let $\ell : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ be a loss function, in all our experiments we use the cross-entropy loss for classification tasks and squared loss for regression tasks. We construct the following models:

(1) $f_{GNN}$ is a GNN feature extractor. Its input is a graph and its output is a feature vector for each node in the graph. (2) $h_{Main}$ is a head (a small neural network) for the main task. Its inputs are the node feature and it outputs a prediction (for graph prediction tasks this head contains a global pooling layer). (3) $h_{SSL}$ is the head for the SSL task. Its inputs are node features, and it outputs a prediction for each node of the graph, depending on the specific SSL task used.

**Pretraining.** Here, there are two phases for the learning procedure. In the first phase, at each iteration we sample a batch $(\mathbf{x}_1, \mathbf{y}_1)$ from $X_{SSL}$, and train by minimizing the objective: $\ell(h_{SSL} \circ f_{GNN}(\mathbf{x}_1), \mathbf{y}_1)$. In this phase both $h_{SSL}$ and $f_{GNN}$ are trained. In the second phase, at each iteration we sample $(\mathbf{x}_2, \mathbf{y}_2)$ from $X_{main}$ and train on the loss $\ell(h_{Main} \circ f_{GNN}(\mathbf{x}_2), \mathbf{y}_2)$, where we only train the head $h_{Main}$, while the weights of $f_{GNN}$ are fixed.

**Multitask training.** Here we train all the functions at the same time. At each iteration we sample a batch $(\mathbf{x}_1, \mathbf{y}_1)$ from $X_{SSL}$ and a batch $(\mathbf{x}_2, \mathbf{y}_2)$ from $X_{Main}$ and train by minimizing the objective:

$$\alpha \cdot \ell(h_{SSL} \circ f_{GNN}(\mathbf{x}_1), \mathbf{y}_1) + (1 - \alpha) \cdot \ell(h_{Main} \circ f_{GNN}(\mathbf{x}_2), \mathbf{y}_2).$$

Here $\alpha \in [0, 1]$ is the weight for the SSL task, in all our experiments we used $\alpha = 1/2$.

For an illustration of the training procedures see Fig. 9. These procedures are common practices for training with SSL tasks (see e.g. (You et al., 2020b)).

We additionally use a semi-supervised setup in which we are given a dataset $X_{FS}$ of few-shot examples from the target distribution with their correct label. In both training procedures, at each iteration we sample a batch $(\mathbf{x}_3, \mathbf{y}_3)$ from $X_{FS}$ and add a loss term $\beta\ell(h_{Main} \circ f_{GNN}(\mathbf{x}_3), \mathbf{y})$ where $\beta \in [0, 1]$ is the weight of the few-shot loss. In pretraining, this term is only added to the second phase, with weight $1/2$ and adjust the weight of the main task to $1/2$ as well (equal weight to the main task). In the multitask setup, we add this term with weight $1/3$ and adjust the weights of the two other losses to $1/3$ as well, so all the losses have the same weight.

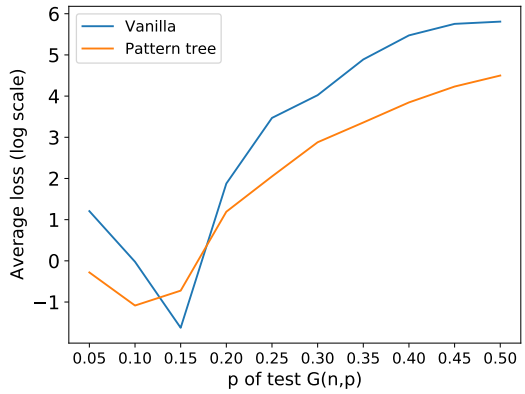


*Figure 10.* Teacher-student setup with a 3-layer GNN. Training is on graphs drawn i.i.d from $G(n,p)$ with $n \in \{40,...,50\}$ uniformly and $p = 0.3$. Testing is done on graphs with $n = 100$ and $p$ vary (x-axis). The "Pattern tree" plot represent training with our pattern tree SSL task, using the pretraining setup.

| TASKS | # TARGET SAMPLES | EDGE COUNT | DEGREE | TEACHER STUDENT | TEACHER STUDENT PER NODE |
|---|---|---|---|---|---|
| VANILLA | **0** | $(1.9 \pm 2.1) \cdot 10^6$ | $363 \pm 476$ | $(3 \pm 5) \cdot 10^4$ | $3311 \pm 8813$ |
| | **1** | $679 \pm 1014$ | $0.27 \pm 0.27$ | $53 \pm 98$ | $1.1 \pm 2.8$ |
| | **5** | $95 \pm 105$ | $(3.8 \pm 6.1) \cdot 10^{-2}$ | $3.1 \pm 5.4$ | $0.4 \pm 1.3$ |
| | **10** | $43 \pm 39$ | $(1.8 \pm 3.5) \cdot 10^{-2}$ | $25 \pm 75$ | $(0.8 \pm 1.3) \cdot 10^{-1}$ |
| $d$-PATTERN PT | **0** | $(1.9 \pm 1.4) \cdot 10^6$ | $1580 \pm 1912$ | $809 \pm 1360$ | $4.4 \pm 5.5$ |
| | **1** | $2528 \pm 1559$ | $2 \pm 1.2$ | $0.5 \pm 1.5$ | $(4 \pm 6) \cdot 10^{-3}$ |
| | **5** | $134 \pm 279$ | $0.55 \pm 0.14$ | $0.11 \pm 0.19$ | $(2 \pm 3) \cdot 10^{-3}$ |
| | **10** | $79 \pm 79$ | $0.55 \pm 0.1$ | $1.4 \pm 4.4$ | $(2 \pm 3) \cdot 10^{-3}$ |

*Table 5.* Results on synthetic datasets (Vanilla vs. $d$-pattern PT).

## G. More experiments from Sec. 7

### G.1. Synthetic datasets

We used the setting of Section 6. Source graphs were generated with $G(n,p)$ with $n$ sampled uniformly in $[40,50]$ and $p = 0.3$. Target graphs were sampled from $G(n,p)$ with $n = 100$ and $p = 0.3$.

Table 5 depicts the results of using the $d$-patterns SSL tasks, in addition to using the semi-supervised setting. It can be seen that adding the $d$-patterns SSL task significantly improves the performance on the teacher-student task, although it does not completely solve it. We also observe that adding labeled examples from the target domain significantly improves the performance of all tasks. Note that adding even a single example significantly improves performance. In all the experiments, the network was successful at learning the task on the source domain with less than $0.15$ averaged error, and in most cases much less.

Fig. 10 depicts a side-by-side plot of the 3-layer case of Fig. 3 (d), where training is done on graphs sampled from $G(n,p)$ with $40$ to $50$ nodes and $p = 0.3$, and testing on graphs with $100$ nodes and $p$ varies. We compare Vanilla training, and our pattern tree SSL task with pretraining. It is clear that for all values of $p$ our SSL task improves over vanilla training, except for $p = 0.15$.

### G.2. Max clique

We tested our SSL task on the task of finding the max-clique size of a given graph, similar to the experiment presented in Table 2. For the train distribution, we constructed graphs by drawing $n \in [10,50]$ points uniformly in the unit square and connecting two points if their distance is less than $\rho = 0.3$. The test set graphs are drawn similarly, but with $n = 100$, effectively increasing the density by 2. We trained on $20,000$ graphs and tested on $2000$ graphs, where the task is to predict

| Task No. | 0 | 47 | 93 | 94 | 110 | 111 | 114 | 115 | average |
|---|---|---|---|---|---|---|---|---|---|
| Pattern PT | $88.5 \pm 0$ | $78.4 \pm 25$ | $55.9 \pm 9.2$ | $68.4 \pm 0$ | $73.1 \pm 0$ | $44.8 \pm 0.2$ | $84.9 \pm 7.3$ | $73.7 \pm 14.9$ | $70.9\%$ |
| Vanilla | $88.4 \pm 0.4$ | $82.1 \pm 23.7$ | $59.1 \pm 5.6$ | $68.4 \pm 0.1$ | $70.3 \pm 6.1$ | $44.9 \pm 0$ | $85.78 \pm 4.8$ | $76.5 \pm 5.7$ | $71.9\%$ |

*Table 6.* Results on the ogbg-molpcba datasets from the OGB collection. The dataset contains 128 different tasks, we used the tasks with the most balanced labels (there is no class with more than $90\%$ of the samples). In both settings, we emitted the node and edge features as our method does not support edge features and continuous node features (only categorical node features). The results are inconclusive, with a slight edge toward vanilla training.

*Table 7.* Dataset statistics.

| | NCI1 | | | NCI109 | | | DD | | |
|---|---|---|---|---|---|---|---|---|---|
| | **all** | **Smallest 50%** | **Largest 10%** | **all** | **Smallest 50%** | **Largest 10%** | **all** | **Smallest 50%** | **Largest 10%** |
| **Class A** | 49.95% | 62.30% | 19.17% | 49.62% | 62.04% | 21.37% | 58.65% | 35.47% | 79.66% |
| **Class B** | 50.04% | 37.69% | 80.82% | 50.37% | 37.95% | 78.62% | 41.34% | 64.52% | 20.33% |
| **Num of graphs** | 4110 | 2157 | 412 | 4127 | 2079 | 421 | 1178 | 592 | 118 |
| **Avg graph size** | 29 | 20 | 61 | 29 | 20 | 61 | 284 | 144 | 746 |

| | Twitch_egos | | | Deezer_egos | | | IMDB-binary | | |
|---|---|---|---|---|---|---|---|---|---|
| | **all** | **Smallest 50%** | **Largest 10%** | **all** | **Smallest 50%** | **Largest 10%** | **all** | **Smallest 50%** | **Largest 10%** |
| **Class A** | 46.23% | 39.05% | 58.07% | 56.80% | 44.78% | 64.97% | 50.00% | 48.98% | 55.55% |
| **Class B** | 53.76% | 60.94% | 41.92% | 43.19% | 55.21% | 35.02% | 50.00% | 51.01% | 44.44% |
| **Num of graphs** | 127094 | 65016 | 14746 | 9629 | 4894 | 968 | 1000 | 543 | 108 |
| **Avg graph size** | 29 | 20 | 48 | 23 | 13 | 68 | 19 | 13 | 41 |

| | PROTEINS | | |
|---|---|---|---|
| | **all** | **Smallest 50%** | **Largest 10%** |
| **Class A** | 59.56% | 41.97% | 90.17% |
| **Class B** | 40.43% | 58.02% | 9.82% |
| **Num of graphs** | 1113 | 567 | 112 |
| **Avg graph size** | 39 | 15 | 138 |

the size of the maximal clique in the graph. We used squared loss. We ran the experiment 10 times and averaged the loss over all the experiments.

Without our SSL, the squared loss on average is 2325, hence not generalizing well to larger sizes. With our SSL task and using pretraining (PT) the loss on average is 1327, improving over vanilla training, but still not solving the problem. Using our SSL task with multitask training (MTL) the average loss provided worse results than vanilla training. We note that this is in general an NP-hard problem, hence solving it might require a specific solution, while our SSL is a general framework for improving size generalization.

### G.3. OGB dataset

We additionally tested our framework on 8 tasks from the "ogbg-molpcba" dataset from the OGB dataset collection (Hu et al., 2020). These are binary classification tasks, where we choose tasks for which there is no one class with more than $90\%$ of the samples (i.e. the tasks with the most balanced labels). We compare our SSL task with pretraining vs. vanilla training. We note that since our task assumes only categorical features and bidirectional edges with no features, we did not use the node and edge features. We split the datasets by size, in the same manner, we did in the experiments from Table 1, where we trained on the $50\%$ smallest graphs and tested on the $10\%$ largest graphs. We report the accuracy. For the results see Table 6. The results are inconclusive due to the lack of features which is valuable information not used by our model. We also tested using the given split from the OGB repository but found the same inconclusive results. In the future, it will be interesting to generalize our method to handle continuous node features and edge features. We hope that by using these features our method could also improve on the size generalization task for datasets from the OGB collection.

### G.4. Datasets statistics

Table G.4 shows the statistics of the datasets that were used in the paper. In particular, the table presents the split that was used in the experiments, we trained on graphs with sizes smaller or equal to the 50-th percentile and tested on graphs with sizes larger or equal to the 90-th percentile. Note that all the prediction tasks for these datasets are binary classification. In all the datasets there is a significant difference between the graph sizes in the train and test sets, and in some datasets, there is also a difference between the distribution of the output class in the small and large graphs.

### G.5. Correlation Between Size Discrepancy and $d$-pattern Discrepancy

In this section, we show that in many real datasets, there is a high correlation between the sizes of the graphs and their $d$-patterns. This motivates the effect that $d$-patterns have on the ability of GNNs to generalize to larger sizes than they were trained on.

We focused on the datasets that were tested in Sec. 7: **Biological datasets:** NCI1, NCI109, D & D, Proteins. **Social networks:** IMDB-Binary, Deezer ego nets, Twitch ego nets. To this end, we split each dataset to the $50\%$ smallest graphs and $10\%$ largest graphs. We calculated the distribution of 2-patterns for the 20 most common two patterns in each split (smallest and largest graphs) and compared these two distributions. The results are depicted in Fig. 11 (The plot for the Twitch dataset similar to the one of Deezer, where the distributions of 2-patterns are disjoint).

It is clear that for the NCI1, NCI109, and D & D datasets there is a very high overlap of 2-patterns between small and large graphs. Indeed, in the result from Table 1 it can be seen that adding an SSL task (and specifically our tree pattern task) does not improve significantly over vanilla training (except for NCI109). On the other hand, for the other datasets, there is a very clear discrepancy between the 2-patterns of small and large graphs. Indeed for these datasets, our SSL task improved over vanilla training. For the social network datasets, it is even more severe, where there is almost no overlap between the 2-patterns, meaning that the small and large graphs have very different local structures. We also calculated the total variation distance between the distributions for every dataset, this appears in the first row of Table 1.

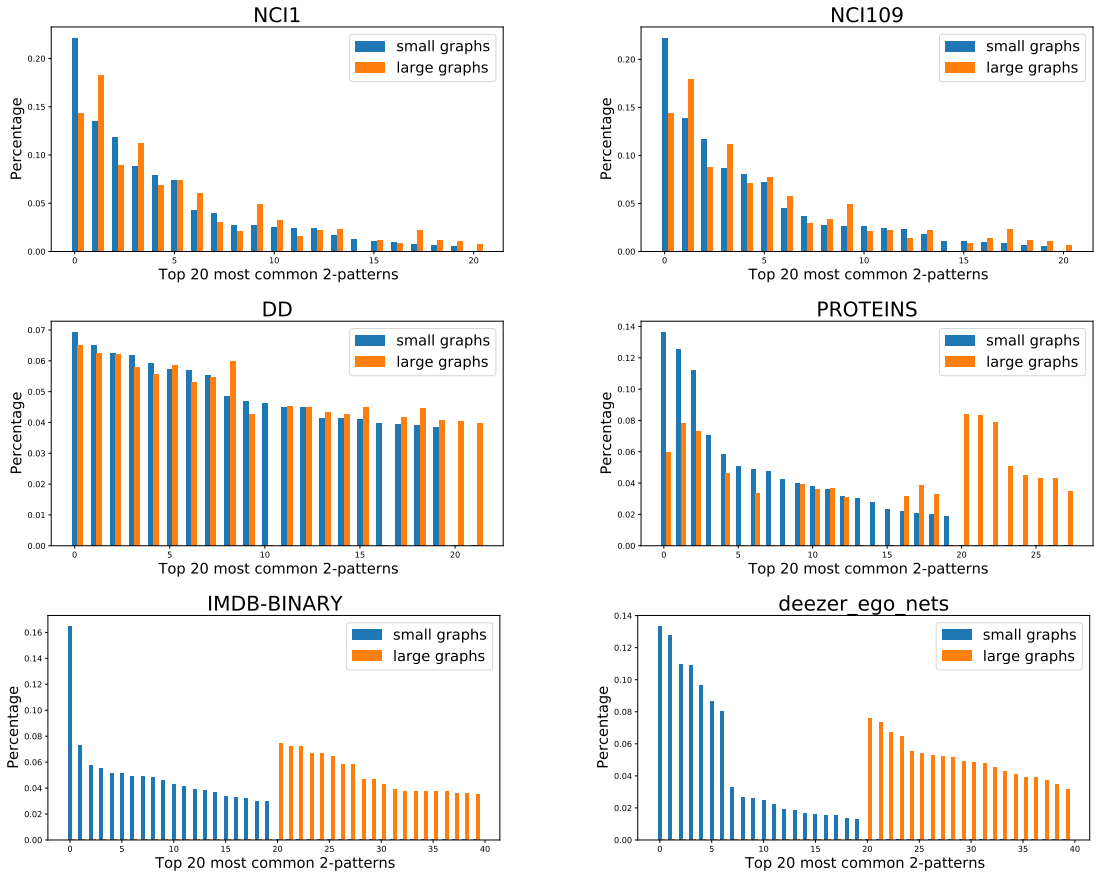


*Figure 11.* Histograms in percentage of 2-patterns of graphs. We used the 50% smallest graphs and 10% largest graphs in each dataset.(same as the split in the experiment from Sec. 7). The X-axis represent the 20 most common 2-patterns from each split, and the y-axis their percentage of appearance. The x-axis contain from 20 to 40 bars - given by how much overlap of 2-patterns there is between small and large graphs.

| DATASETS | TOTAL-VARIATION DISTANCE |
|---|---|
| D & D | 0.15 |
| NCI1 | 0.16 |
| NCI109 | 0.16 |
| PROTEINS | 0.48 |
| IMDB-BINARY | 0.99 |
| DEEZER EGO NETS | 1 |
| TWITCH EGO NETS | 1 |

*Table 8.* Total variation distance, between the 2-patterns of the 50% smallest graphs and 10% largest graphs for all the real datasets that we tested on.