A ADDITIONAL PROPERTIES OF HELP

A.1 EXPRESSIVE POWER

An additional advantage of HELP is that, in contrast to GNNs following the message-passing framework (Xu et al., 2019), it can distinguish between graphs that cannot be distinguished by the Weisfeiler-Lehman graph isomorphism test (WL-test). Figure 3 gives an example of two graphs that this test fails to tell apart. However, it is easy to see that the number of connected components in these graphs differs. As our pooling step includes a connected component search, our algorithm should, in theory, be able to determine which of those two graphs was given as the input. In particular, the GNN layers will produce the same embeddings for all nodes due to the mentioned limitation. The pooling will then map the graph on the left to two nodes and the graph on the right to one node—all of them with the exact same embedding which was calculated by mean pooling nodes with the same embedding. For nonzero embeddings, the final global sum pooling will therefore yield different results (the graph on the left will have the result of the graph on the right times two), allowing the final classification layer to make different predictions.

To empirically demonstrate this, we design a simple synthetic dataset based on the example graphs shown in Figure 3. In particular, we generate graphs with even numbers of nodes between 6 and 40 which consist of either one or two circles. The goal is to predict which of those two is the case. As shown in Section 5.1, we indeed achieve perfect accuracy whereas GCN only reaches the accuracy of always guessing the more likely class.

Note that HELP trivially also can distinguish everything the GNN layers it uses could differentiate on their own. This makes it strictly more expressive than standard GNNs that follow the message-passing scheme. Interestingly, even though it does not solve it perfectly, the fact that ASAP achieves higher accuracy than just predicting the most likely class on our Synthetic Expressivity dataset indicates that it is more than 1-WL expressive as well. To the best of our knowledge, this is not mentioned in the original paper.

A.2 RECEPTIVE FIELD

In a standard GNN following the message-passing scheme, the *receptive field* of a node is its L-hop neighborhood, where L is the number of GNN layers. A big receptive field can be necessary to recognize larger structures in the graph. However, increasing the number of GNN layers not only leads to higher computational cost but also has a harmful effect termed *oversmoothing*. This refers to the phenomenon that with a growing number of GNN layers, the embeddings of all nodes will become similar which leads to a loss of expressive power exponential in the number of layers (Oono & Suzuki, 2020).

Our algorithm alleviates this issue by increasing the receptive field beyond the number of GNN layers. Take, for instance, a graph consisting of multiple substructures where the interaction of those substructures is important but they are separated by long chains of intermediate nodes. An example would be inserting more intermediate nodes in our Synthetic Hierarchical dataset. A standard GNN would require enough layers to cover those long chains. In contrast, our method could already pool after only one or two layers. Most nodes on the chains would then be mapped to the same cluster—

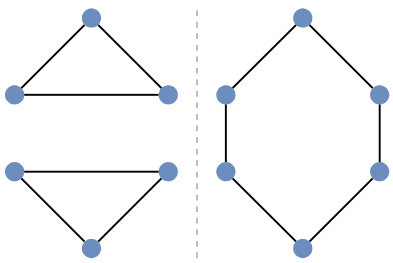


Figure 3: The graph consisting of two triangles (left) and the graph consisting of a hexagon (right) can not be distinguished by standard GNN architectures even though they are not isomorphic.

Table 3: Shared hyperparameters for all datasets

Parameter	Value
Optimizer	Adam (Kingma & Ba, 2015)
Learning Rate	0.001
Weight Decay	0.0005
Batch Size	32 for REDDIT-BINARY, 64 otherwise
GNN Layers	GCN (Kipf & Welling, 2017)
Global Pooling Operation	$\sum$
Activation	LeakyReLU (0.01)
Hidden Dimensions	32 32 [Pool] 32 32 [Pool] 32 4
Train/Test/Validation Split	80% / 10% / 10%
m (no. samples for hyperplane approx.)	10

independent of the length of the chains. Consequently, the nodes on each chain would be mapped to a single node during pooling. After this, a significantly smaller number of layers would be required to cover the whole graph in the receptive field.

B Experimental Setup

B.1 Baselines

We compare our proposed algorithm to a standard GNN (in our case GCN (Kipf & Welling, 2017)) with the exact same architecture but without the pooling layers. Additionally, we compare it to two popular pooling methods: DiffPool and and ASAP (see Section 6). Whereas none of these approaches was designed with the goal of interpretability, we are still able to calculate concept completeness for DiffPool. Recall that DiffPool has a fixed number of nodes after each pooling step and learns a soft assignment from each input node to these output nodes. We therefore define the concept of an input node as the id of the output node to which the assignment is the strongest. This gives us everything that is required for our concept metrics, such as a multiset of present concepts for completeness. ASAP, on the other hand, only selects some percentage of the most important nodes. There is no straight-forward mapping from these nodes to a global concept. Additionally, note that ASAP and DiffPool both employ more complex GNN layers to achieve state-of-the-art performance and ASAP additionally makes use of a layer-wise *readout*[4]. To enable fair comparison, we instead use the GCN layer for all experimental setups and remove the layer-wise readout.

C Implementation Details

C.1 Reproducability

Whereas the most important hyperparameters are given in Tables 3 and 4, the exact commands to reproduce all ablations are given in the `README.md` file of our repository. Additionally, the `analysis.ipynb` notebook contains detailed instructions to easily reproduce the concept/subgraph visualizations used in the qualitative analysis.

C.2 Clustering

C.2.1 Resolving Ambiguities when Merging Clusters

Note that it is possible for connected chains of points that should be merged to occur. For example, there could be three points A, B and C where the distance between A and B and between B and C is below the threshold but the distance between A and C is not. We resolve these ambiguities in a permutation-invariant manner by merging the whole connected chain.

[4]Essentially, this introduces skip connections from the output of each GNN layer to the final prediction layer.

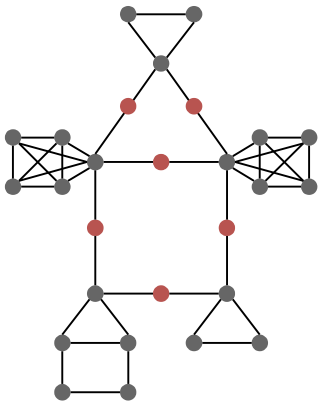


Figure 4: **An example graph from the synthetic hierarchical dataset.** The high-level motif here is a house. It contains two triangles, a house and two fully connected pentagons as lowlevel motifs. The class label is therefore given by $(\text{house}, \{\text{triangle}, \text{house}, \text{fully connected pentagon}\})$. Intermediate nodes are colored red.

Table 4: Hyperparameters varying between datasets

	Epochs	n_{clust} Lvl. 1	n_{clust} Lvl. 2
Synthetic Hierarchical	10000	10	15
Mutagenicity	1000	20	20
REDDIT-BINARY	1500	30	30
BBBP	5000	15	15
Synthetic Expressivity	Ours: 100, others: 1000	10	15

C.3 Datasets

C.3.1 Hierarchical Dataset

Note that we insert the intermediate nodes described in Section 4.1 for two reasons. First, they ensure that the combination of low-level and high-level motif can theoretically be deduced. Whereas we view the graph in a certain way based on how it was generated, it is otherwise possible that graphs from different classes are in fact isomorphic. Additionally, the main goal of this dataset is that we already have a good understanding of what concepts to expect when going into the analysis. In particular, we would like to see decoupled concepts for the different low-level motifs which makes the dynamics easier to understand. As our method pools connected components mapped to the same concept, without the intermediate nodes, neighboring high-level nodes would be pooled together if they are mapped to the same concept. Therefore, our algorithm would be forced to learn more complex concepts that depend on the combination of neighboring low-level motifs in order to solve the task. These would be harder to interpret in the analysis.

C.3.2 Mutagenicity and BBBP

Like previous methods (Magister et al., 2021; 2022), we ignore the edge features in these datasets as they are not supported by the simple GNN layer GCN (Kipf & Welling, 2017). Whereas our method is independent of the message-passing GNN layer and therefore in principle also supports edge features with an appropriate layer, we opt for GCN as a widely used baseline to make our analysis comparable to previous work (Magister et al., 2022; 2021), avoid unexpected side-effects by a complex GNN layer and minimize computational cost. Whereas Mutagenicity has only the atom class as its node labels, BBBP contains various additional properties like *valence* by default. We remove these properties to allow easier visualization and interpretation of the input graphs by someone who is not a domain expert.

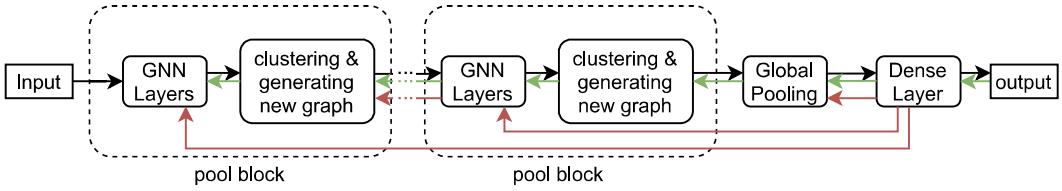


Figure 5: **Gradient flow of our method.** Black arrows denote the forward pass, green arrows show the naïve gradient flow only through the node embeddings (Section D.2) and red arrows denote the gradient flow in our hyperlane-based approximation (Section D.3). In particular, note that the red gradients flow back up to the discontinuous component where the flow is interrupted. Instead, the gradient with respect to the input of these components is approximated from the multiple samples that were propagated from here up to the final dense layer. These approximated input gradients then flow back up to the next discontinuous component, where the process repeats.

C.4 Visualization

Whilst not required in clean datasets like our Synthetic Hierarchical one, for BBBP and Mutagenicity (plots 13–12), we make visualizations slightly more readable by merging small bar segments into bigger ones. In particular, visualizing examples of these segments reveals that they are generally highly related to bigger segments of the same concept. For example, the same functional group but including an additional carbon atom. While this could still be interpreted by a human, it would require looking at a bigger list of examples and thereby be harder to depict in a single plot. As a remedy, we take all pooled components with at most 500 assigned nodes (starting from the one with the least assigned nodes) and merge them with the next bigger one (if any) that (1) is a proper subgraph (including node features) of the current component and (2) consists of at least 2 nodes (to ensure that relevant structure is preserved). Note that this method could not sensibly be applied to GCExplainer as the different k-hop neighborhoods would likely not be subgraphs of each other.

D Details About the Method

D.1 Differentiability

As mentioned earlier, clustering and merging connected components are inherently discontinuous operations. In this section, we will discuss different remedies that allow us to still learn embeddings that give the desired clusters.

D.2 Using the Existing Gradient Flow

Before we dive further into different approaches to estimate gradients, it is important to note that they are not strictly necessary to train a model. As the node embeddings of the pooled graphs are always averages over a subset of node embeddings of the previous graph, there exists an uninterrupted gradient flow from the inputs to the final predictions (see Figure 5). However, these gradients are calculated as if the cluster assignments were fixed. In other words, whereas the loss landscape is not zero almost everywhere (as it would be the case for many classical algorithms like sorting, path finding etc. (Pogancic et al., 2020)), it has discontinuities wherever a change in a node embedding would lead to it being assigned to a different cluster. This means that the gradient information does not give us any way of learning which cluster assignment would be better. As learning a good way to coarsen graphs is a central goal of this work, we will discuss different sampling based remedies in the following sections. Nevertheless, since Magister et al. (2021) find that even in standard GNNs, the trained node embeddings form clusters representing meaningful concepts, we still expect this approach to perform reasonably well. Even though we do not explicitly optimize for a good clustering, it will implicitly evolve as the GNN learns to optimally predict the target.

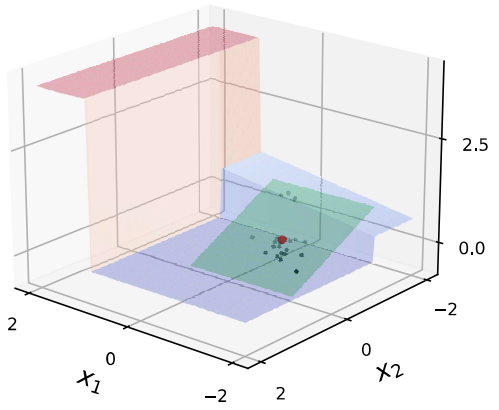


Figure 6: **Visualization of the approximated hyperplane**. For a given point (red) on the function f (blue/red), we take m samples around it (black) and find an approximate hyperplane (green). We then use the gradient of that hyperplane as the approximate gradient at the red point. Note that this example is *overdetermined* as $m > 2$ and $\boldsymbol{x} \in \mathbb{R}^2$. We therefore use the least squares solution to Equation 3 rather then the minimum norm solution. In practice, we have $\boldsymbol{x} \in \mathbb{R}^b$ for some $b \gg m$ and our hyperplane will therefore always go through all sampled points.

D.3 Locally Approximating Gradients as a Hyperplane

An alternative approach is motivated by the definition of the gradient as the locally tangent hyperplane as well as Taylor's theorem which tells us that a continuously differentiable function could be locally approximated as a hyperplane. We therefore propose to locally approximate the gradient of some function $f : \mathbb{R}^b \to \mathbb{R}^d$ at point $\boldsymbol{x} \in \mathbb{R}^b$ by evaluating the function on a number $m - 1$ of slightly perturbed points and finding a hyperplane that goes through all of them. A visualization is shown in Figure 6.

Formally, for noise vectors $\boldsymbol{\varepsilon}_1, \dots, \boldsymbol{\varepsilon}_{m-1} \sim \mathcal{N}(\mathbf{0}, \sigma^2 \boldsymbol{I}_b)$ we determine the minimum norm solution A to

$$\begin{bmatrix} — & \boldsymbol{x}^\top & — & 1 \\ — & (\boldsymbol{x}+\boldsymbol{\varepsilon}_1)^\top & — & 1 \\ & \vdots & & \vdots \\ — & (\boldsymbol{x}+\boldsymbol{\varepsilon}_{m-1})^\top & — & 1 \end{bmatrix} \underbrace{\begin{bmatrix} a_{1,1} & \dots & a_{1,d} \\ \vdots & \ddots & \vdots \\ a_{b+1,1} & \dots & a_{b+1,d} \end{bmatrix}}_{=:A} = \begin{bmatrix} — & (f(\boldsymbol{x}))^\top & — \\ — & (f(\boldsymbol{x}+\boldsymbol{\varepsilon}_1))^\top & — \\ & \vdots & \\ — & (f(\boldsymbol{x}+\boldsymbol{\varepsilon}_{m-1}))^\top & — \end{bmatrix} \tag{3}$$

where we append ones to the input to allow for a constant offset in the hyperplane. We can therefore locally approximate f around $\boldsymbol{x}$ as:

$$\hat{f}(\boldsymbol{z}) := A^\top \begin{pmatrix} \boldsymbol{z}_1 \\ \vdots \\ \boldsymbol{z}_b \\ 1 \end{pmatrix} \tag{4}$$

where we can trivially read off the gradients from A.

Whereas in traditional finite difference methods, the distance between the points should be as small as possible to obtain the best possible gradient approximation, this is not the goal in our case. Instead, we need to strike a balance with the chosen variance. On the one hand, the points should be spread out far enough that when close to a discontinuity, some points will likely be on the other side of this "jump". This is what makes our gradient estimations smooth. On the other hand, increasing the variance of the point distribution also increases the variance of our gradient estimates

which means that we need more samples to attain stable gradients. Moreover, the estimated gradient should not be overly smooth in order to still be able to find the correct minimum.

Whereas the previously proposed black box differentiation methods (Berthet et al., 2020; Pogancic et al., 2020; Blondel et al., 2020; Dalle et al., 2022) focus on linear programs, our approach has an intuitive motivation for arbitrary functions. This is particularly useful as in our setting we need to map a mixture of continuous (node embeddings) and discrete (input graph) features to discrete output features (pooled graph) which continuous output features rely on (new node embeddings). In contrast, these previous approaches focus on mapping purely continuous input features to discrete output features.

Additionally, whereas we opt to sample the perturbed points from the normal distribution, the motivation behind our method does not rely on randomly sampled points and they could easily be chosen in some deterministic way instead. This makes our approach easier to analyze. For instance, if f already corresponds to a hyperplane, for $m > d$ we will always get $f = \hat{f}$ by definition. This holds independent of the choice of perturbed points, as long as they are linearly independent. In methods like the one by Berthet et al. (2020), this only holds in expectation.

D.3.1 Application to Graph Pooling

The gradient approximation methods for black box combinatorial solvers that we discussed so far (including our method described in the previous section) are not directly applicable to our hierarchical pooling setting because they assume that the output dimension d of the black box function will be fixed. However, our black box component outputs arbitrarily sized graphs along with their node embeddings. For each pooling layer, we therefore define the black box as its actual discontinuous component followed by all subsequent pool blocks along with the final, global pooling (see Figure 5). Whereas this results in a constant output dimension, it also leads to black boxes containing learnable parameters—a setting which the discussed approximation methods are not applicable for. We therefore always propagate gradients back until we reach a discontinuous component. However, instead of propagating them through this component as described in Section D.2, we compute the gradient with respect to the inputs of the discontinuous component using the black box differentiation method described earlier in this section.

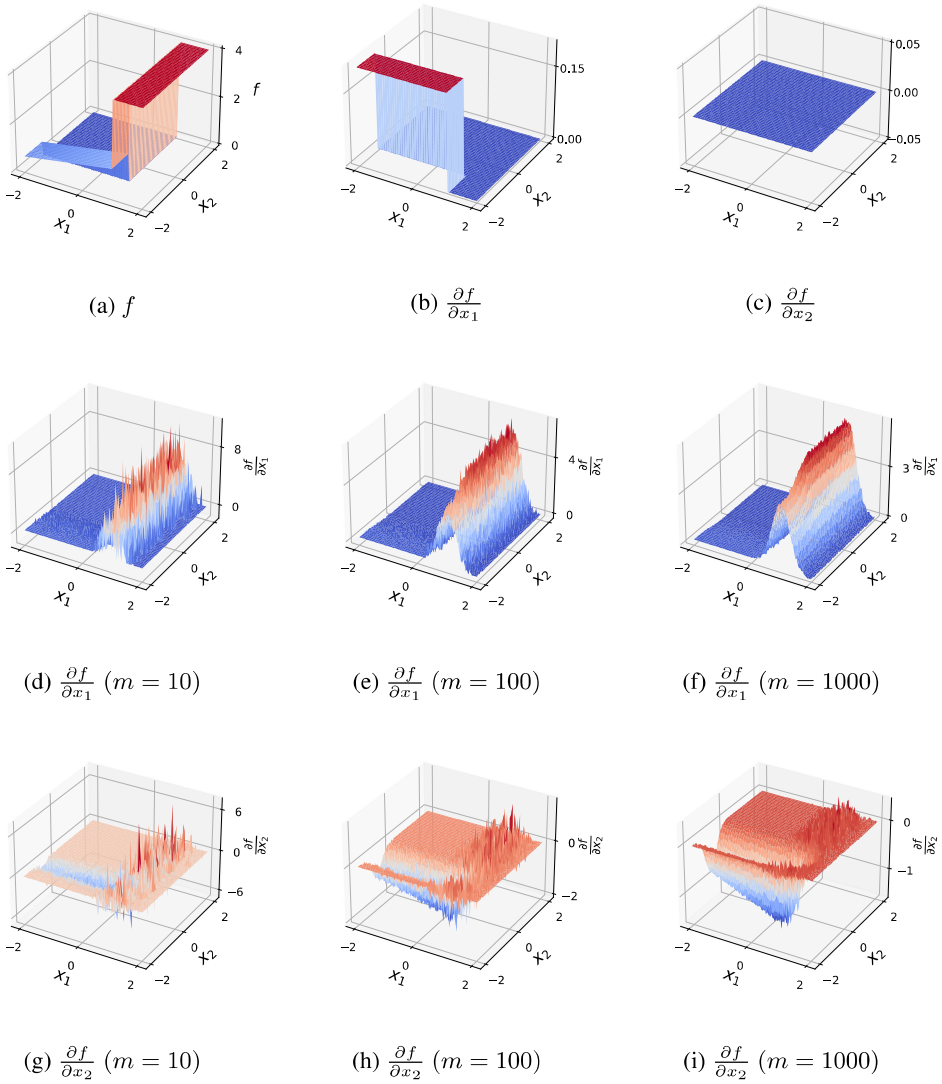
Note that with this method, the number of required forward passes grows in $\Theta(mn_{\text{blocks}})$. Along with the "main" forward pass (for which we calculate gradients), we need $m-1$ additional forward passes from each component up to the final layer. As we do not need to calculate the gradients for these additional forward passes, the number of forward passes does not grow exponentially. Regardless, this still limits us to a relatively low number of samples in practice. In Appendix E we demonstrate that despite yielding relatively stochastic gradients, combined with an optimizer like Adam (Kingma & Ba, 2015), this approach can still lead to convergence. To further stabilize training in our more complex setting, we use a weighted average of the exact gradients with respect to the values (as described in Section D.2) and the gradients obtained by our method, which are stochastic but take different clustering into account.

E Evaluation of Hyperplane Gradient Aproximation

To showcase the smooth gradient estimation generated by our method described in Appendix D.3, we arbitrarily chose the piece-wise linear function f as:

$$f(x_1, x_2) := \begin{cases} 0.2x_1 + 1 & x_1 < 1 \wedge x_2 < -1 \\ 0 & x_1 < 1 \wedge x_2 \geq -1 \\ 4 & x_1 \geq 1 \end{cases} \tag{5}$$

Figure 7 shows that for a higher number of samples we get an increasingly smooth and less stochastic gradient approximation. However, In practice we are limited to a relatively low number of samples in more complex architectures like the one proposed in this thesis. To verify that even noisy gradient estimations allow us to learn, we initialize a point on the red plateau at $(2, -2.5)$ (see Figure 7a) and try to minimize the function using the Adam optimizer (Kingma & Ba, 2015) with learning rate 0.1 and weight decay $5 \cdot 10^{-4}$. For the gradient estimation, we evaluate at $m = 20$ additional points perturbed by the Normal distribution $\mathcal{N}(0, 0.3)$. Since f does not have a unique minimum,



(a) f (b) $\frac{\partial f}{\partial x_1}$ (c) $\frac{\partial f}{\partial x_2}$

(d) $\frac{\partial f}{\partial x_1}$ $(m = 10)$ (e) $\frac{\partial f}{\partial x_1}$ $(m = 100)$ (f) $\frac{\partial f}{\partial x_1}$ $(m = 1000)$

(g) $\frac{\partial f}{\partial x_2}$ $(m = 10)$ (h) $\frac{\partial f}{\partial x_2}$ $(m = 100)$ (i) $\frac{\partial f}{\partial x_2}$ $(m = 1000)$

Figure 7: Plot (a) shows the function from Equation 5 with its exact gradients in the directions x_1 and x_2 in Figures (b) and (c) respectively. Plots (d)–(i) show the corresponding smooth gradient estimations for $m \in \{10, 100, 1000\}$ samples. Perturbations are sampled from $\mathcal{N}(\mathbf{0}, 0.3\boldsymbol{I}_2)$.

the goal should be arriving at any point with $x_1 < 1, x_2 < -1$ and moving down the slope in the direction of smaller x_1 from there. After 1000 steps we end up at $(-76.21, -2.42)$ with $f(x_1, x_2)$ monotonically decreasing over all steps, strongly indicating that this is indeed what happens. In particular, despite the theoretical gradient of 0 we are able to leave the plateau and even though the dark blue area $f(x_1, x_2) = 0$ for $x_1 < 1, x_2 \geq -1$ has 0 gradient and a lower value for $x_1 > -5$ we do not end up in this local minimum.

F Additional Results

F.1 L-hop Neighborhood Conformity

In addition to our definition of conformity, Table 5 also shows the conformity when interpreting examples of a concept as the L-hop neighborhood (like proposed in GCExplainer) rather than the connected component of nodes mapped to the same cluster. Crucially, this is not just a different met-

Table 5: **Comparison of concept conformity scores.** Due to the expense of graph-isomorphism tests we only perform one inference pass per trained model (as opposed to 3 for concept completeness). For comparison, we also show the L-hop scores (Section F.1). Note that scores close to 100% are expected for DiffPool after the first layer as the pooled graphs look identical for all samples (fully connected with the predefined number of nodes). However, this also yields low completeness scores, meaning that the concepts may be easy to understand (everything is considered the same concept) but not meaningful.

	Ours Level 1	**L-hop Level 1**	**Ours Level 2**	**L-hop Level 2**
Synthetic Hierarchical				
Ours	99.8%±0.4	97.3%±0.2	95.0%±2.7	89.7%±0.9
Ours (Global Clustering)	**100.0%±0.0**	**97.5%±0.0**	97.1%±2.2	88.8%±3.9
Ours (Hyperplane)	**100.0%±0.0**	97.1%±0.2	97.0%±2.3	90.3%±4.0
DiffPool	0.0%±0.0	27.2%±0.0	**100.0%±0.0**	**100.0%±0.0**
Mutagenicity				
Ours	83.6%±0.3	47.5%±2.4	63.5%±4.2	23.3%±4.4
Ours (Global Clustering)	**84.2%±1.7**	**47.8%±5.3**	54.0%±6.3	18.2%±5.4
Ours (Hyperplane)	83.0%±1.2	46.7%±3.7	60.0%±6.0	28.2%±3.9
DiffPool	42.9%±0.0	12.4%±0.0	**100.0%±0.0**	**100.0%±0.0**
BBBP				
Ours	**84.8%±1.4**	**33.2%±6.0**	57.5%±2.0	7.5%±2.9
Ours (Global Clustering)	84.4%±2.4	30.1%±0.7	53.4%±4.0	10.5%±4.4
Ours (Hyperplane)	84.0%±2.0	31.3%±2.0	59.1%±5.0	9.2%±4.7
DiffPool	0.0%±0.0	0.0%±0.0	**100.0%±0.0**	**100.0%±0.0**
REDDIT-BINARY				
Ours	**96.2%±0.4**	infeasible	76.0%±11.0	infeasible
DiffPool	93.0%±2.6	infeasible	**100.0%±0.0**	infeasible

ric definition but rather a question of how we define and therefore interpret concepts. As expected, all conformity scores are significantly lower this way as the subgraphs consist of a fixed neighborhood and will therefore inevitably contain nodes that are not relevant to the concepts, leading to many non-isomorphic subgraphs with the same meaning.

The main advantage of taking the L-hop neighborhood is that it guarantees, all isomorphic subgraphs mapped to a concept will have exactly the same meaning as they are defined as the *receptive field* of the node. In contrast, in our approach, two concepts could each pool the subgraph "pair of nodes" where in one of them, this stands for the bottom of a house and in the other it stands for the bottom of a triangle. In Section 5.2, we demonstrate that plotting a few example neighborhoods per subgraph is generally sufficient to get a good understanding of meaningful concepts. Note that in practice, GCExplainer ignores node features and tunes the neighborhood size to some $\ell \leq L$ that gives the best concepts which both imply that this property no longer necessarily holds for their reported concepts. Additionally, by ensuring this, the number of subgraphs to analyze is significantly higher which makes understanding each of them infeasible. When it leads to ignoring other concepts, the guaranteed same meaning is no longer beneficial.

F.2 Completeness of Hierarchical Concepts

Table 6: **Test completeness scores after each of the two pooling steps in comparison to other methods.**

	HELP (Ours)		**DiffPool**	
	Level 1	Level 2	Level 1	Level 2
Synthetic	**100.0%±0.0**	**99.5%±1.4**	27.0%±0.0	27.0%±0.0
Mutagenicity	**73.7%±2.7**	**70.8%±2.4**	53.6%±0.0	53.6%±0.0
BBBP	**80.8%±1.4**	**78.3%±2.5**	77.1%±0.4	77.1%±0.4
Synthetic Expressivity	52.3%±0.0	**100.0%±0.0**	**53.5%±0.0**	53.5%±0.0

We would generally expect the completeness to be higher in later pooling layers where the concepts incorporate more structural information. However, the trend we observe indicates the opposite. To this end, it is important to note that in our hierarchical setup, these concepts can, in fact, be more meaningful in two ways that are not captured by the completeness measure. Firstly, they aggregate local information that might not be meaningful on its own but could facilitate subsequent layers. For instance, the presence of a pair of carbon atoms might not carry significantly more information towards the final prediction than a single atom and thus not increase the completeness. Yet, the following GNNs would require less layers to detect an aromatic ring from pairs of carbon atoms than they would for the original graph. Secondly, pooling always implies a reduction in the number of nodes. This generally means that the set of concepts is smaller and each of them will therefore need to carry more information in order for the completeness to stay constant.

F.3 Additional visualizations

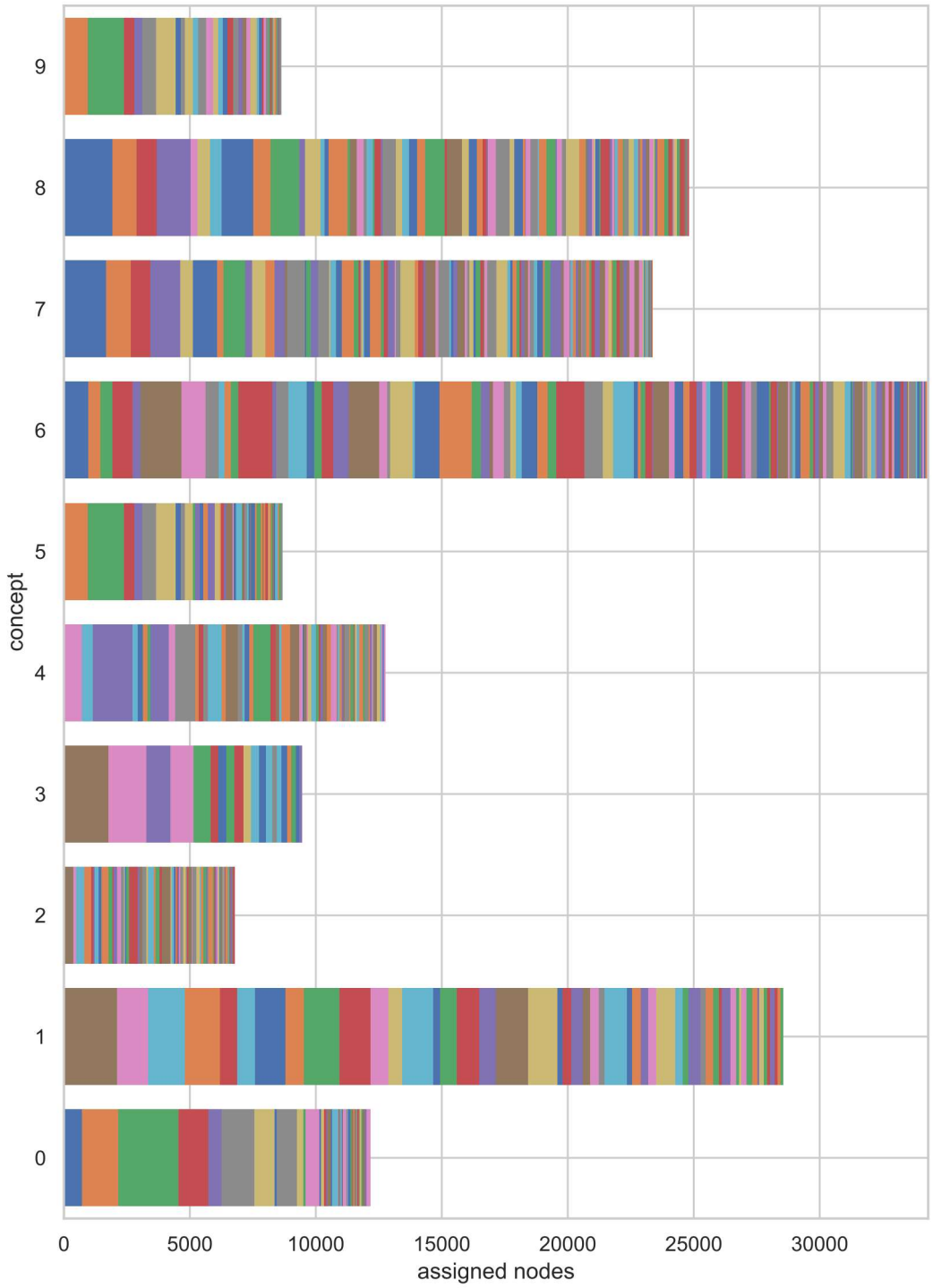


Figure 8: **Subgraphs matched to each concept by GCExplainer in our hierarchical dataset and how often they occur**

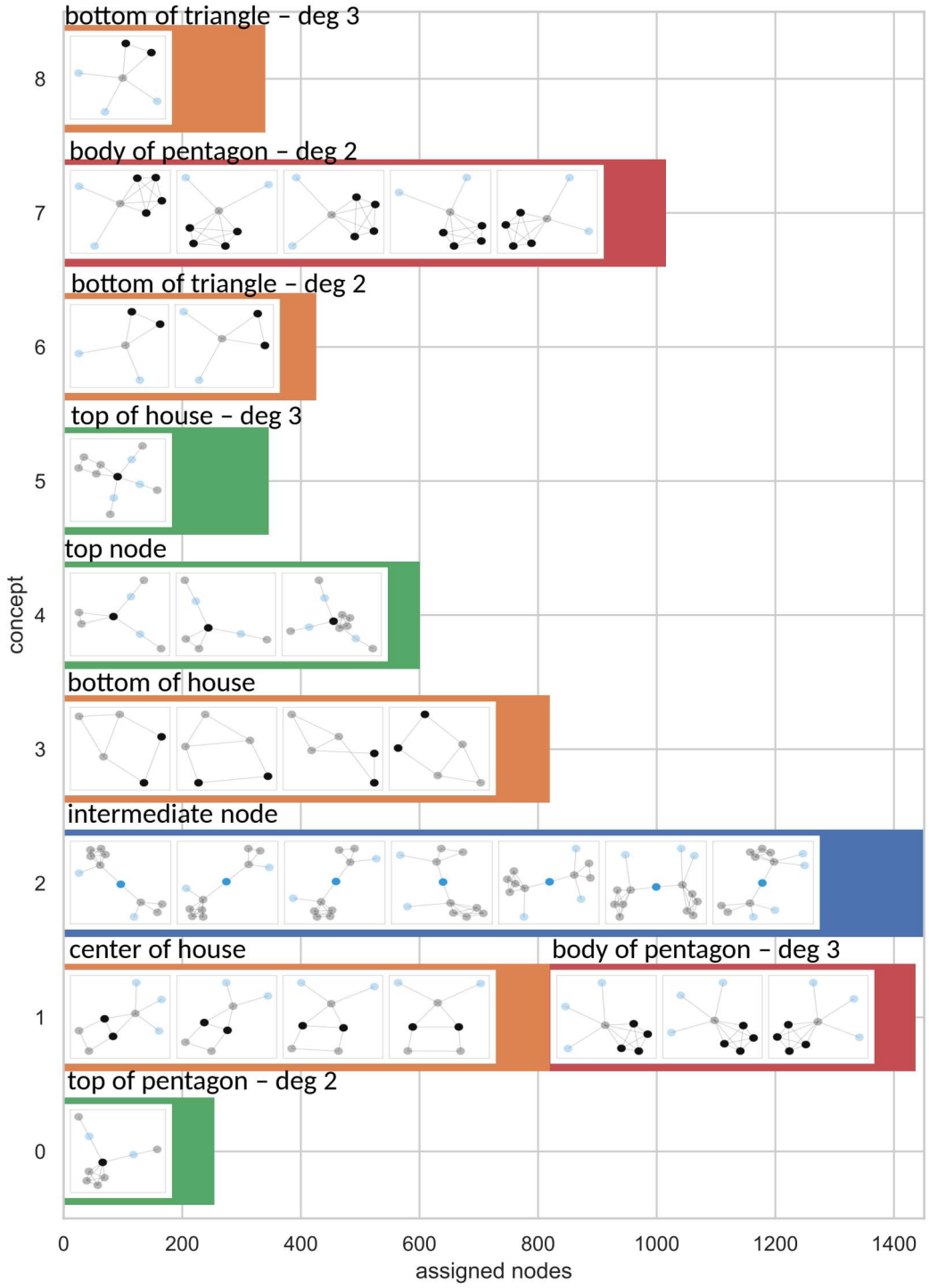


Figure 9: **Subgraphs matched to each concept in the first pooling layer of our hierarchical dataset and how often they occur**

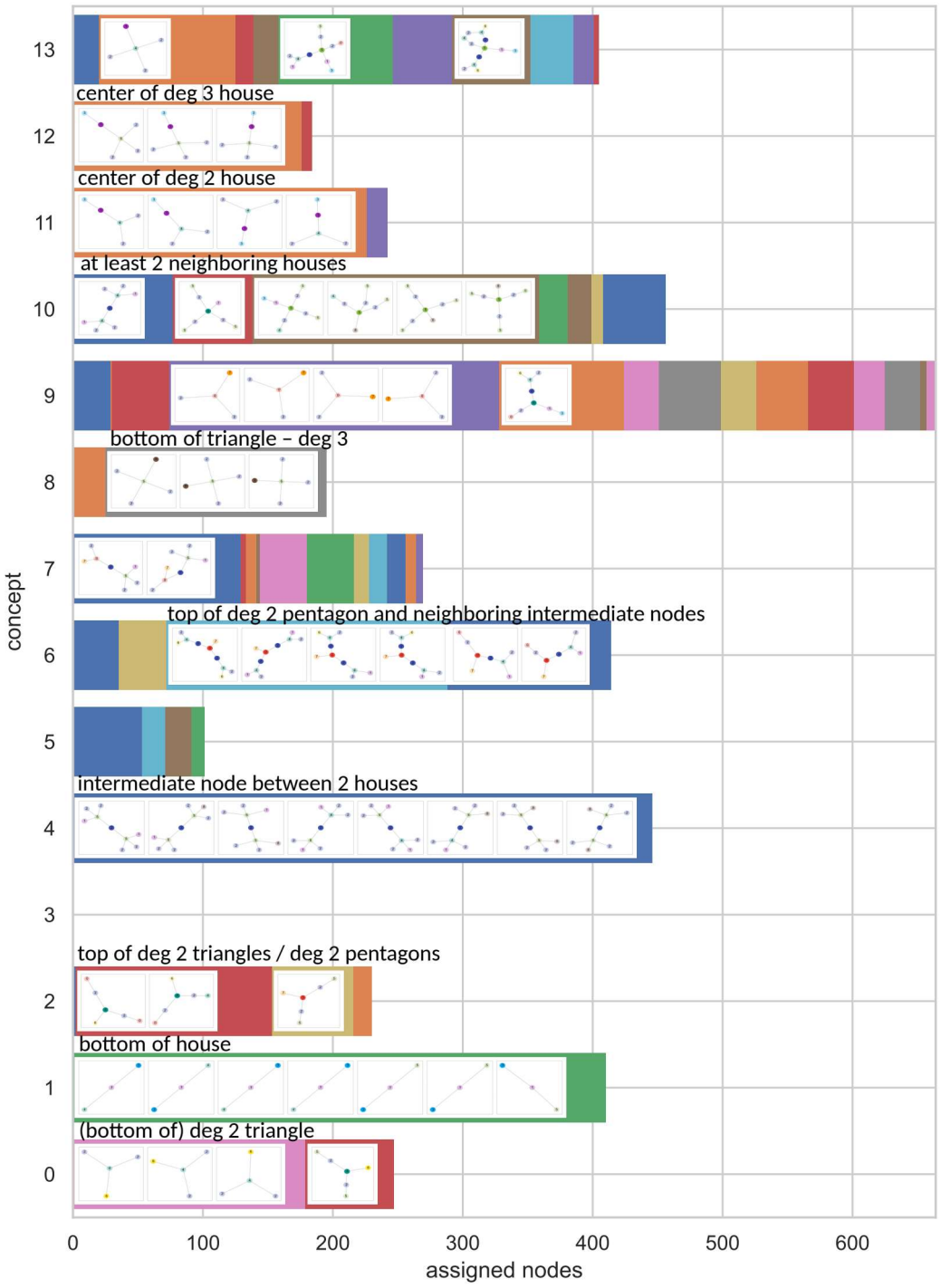


Figure 10: **Subgraphs matched to each concept in the second pooling layer of our hierarchical dataset and how often they occur**

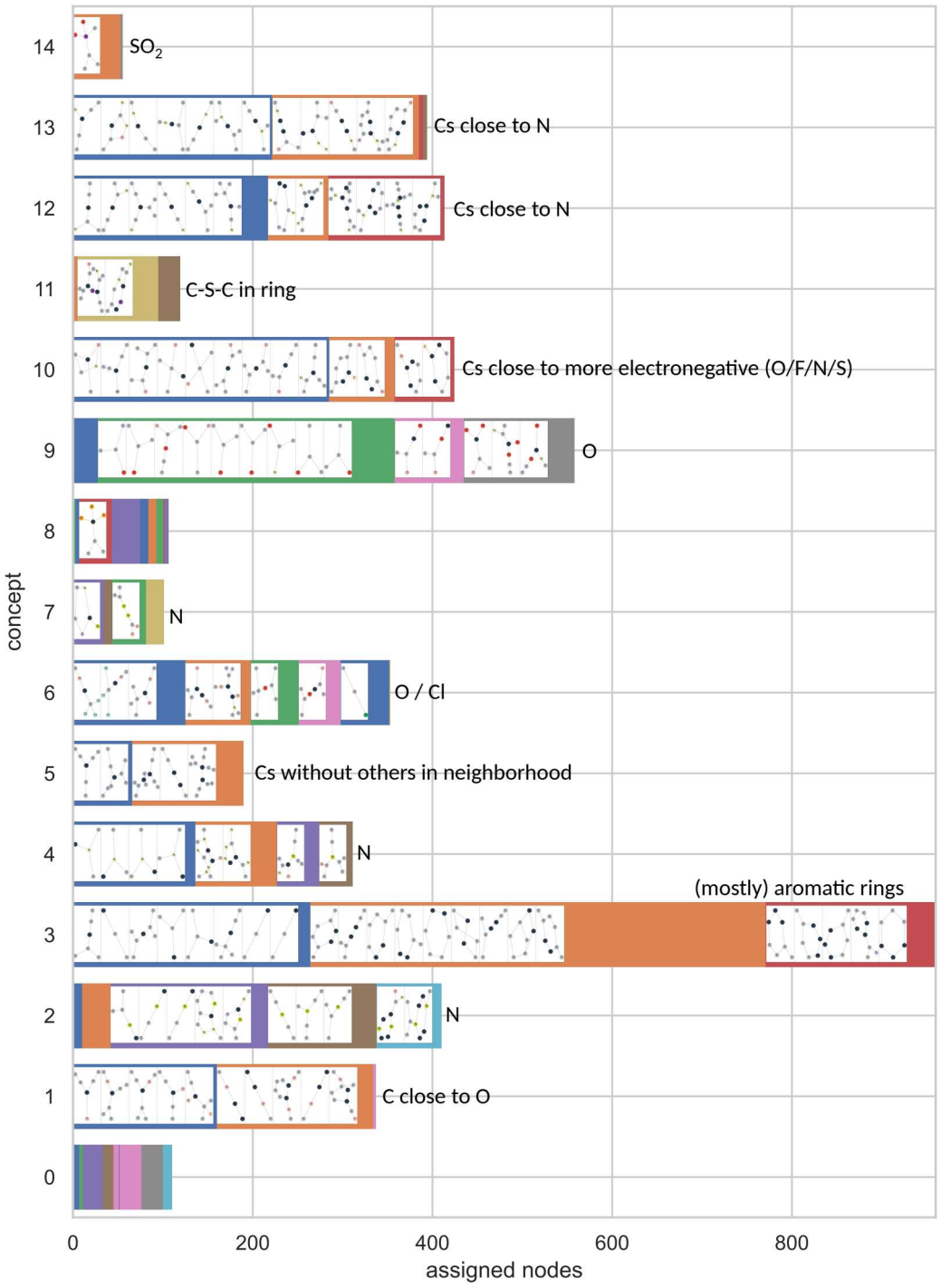


Figure 11: **Subgraphs matched to each concept in the first pooling layer of BBBP and how often they occur**

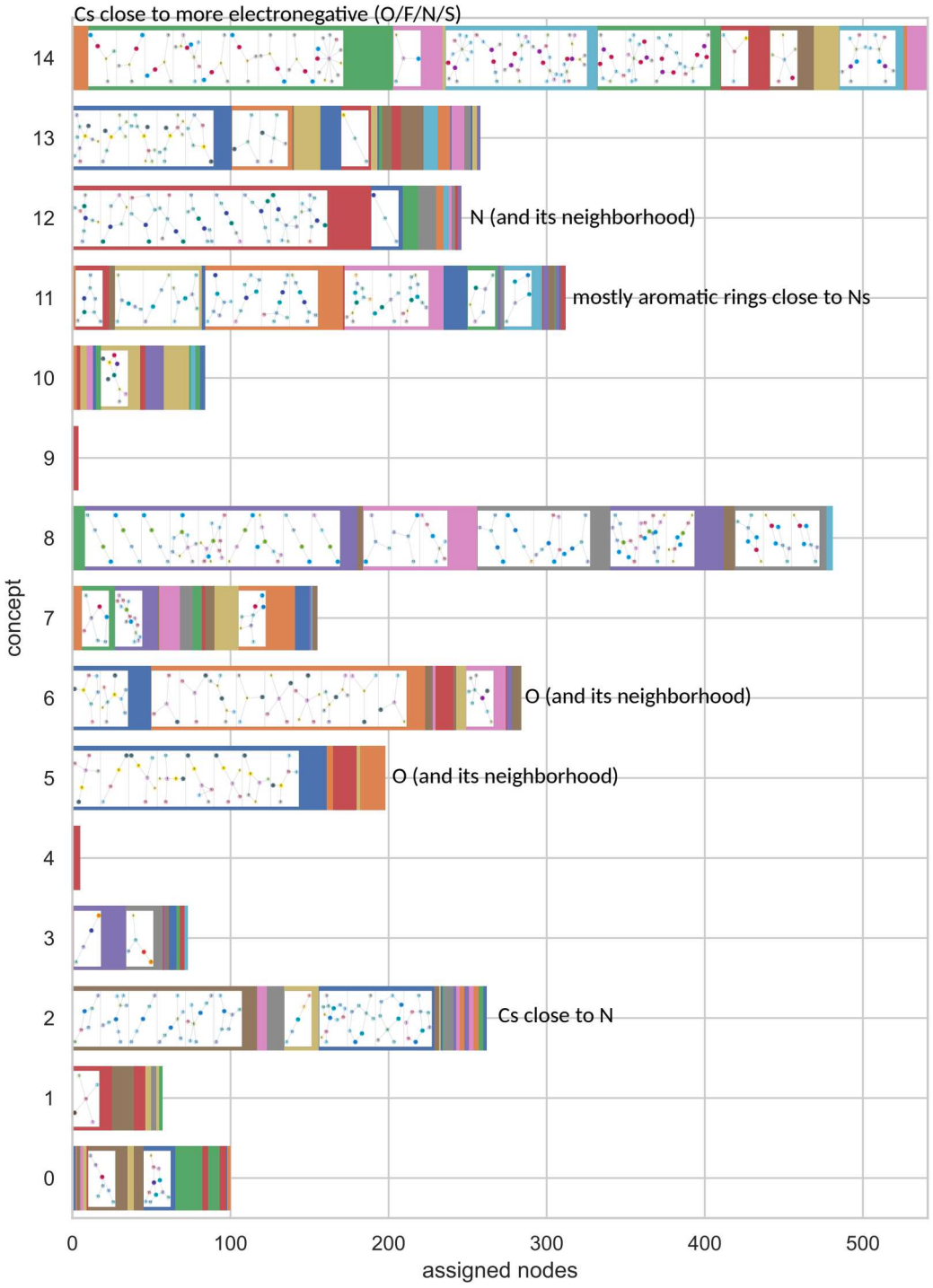


Figure 12: **Subgraphs matched to each concept in the second pooling layer of BBBP and how often they occur**

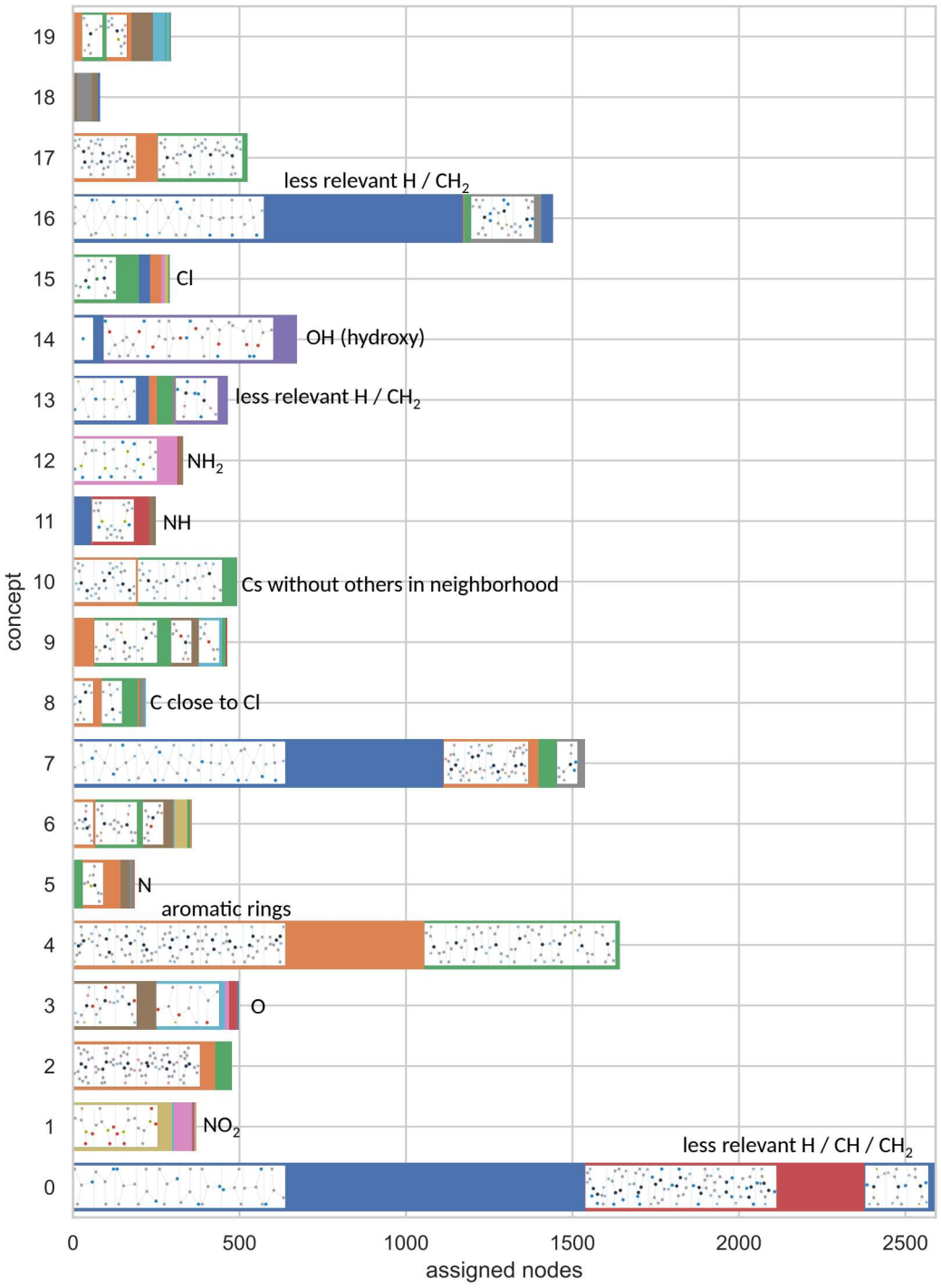


Figure 13: **Subgraphs matched to each concept in the first pooling layer of Mutagenicity and how often they occur**

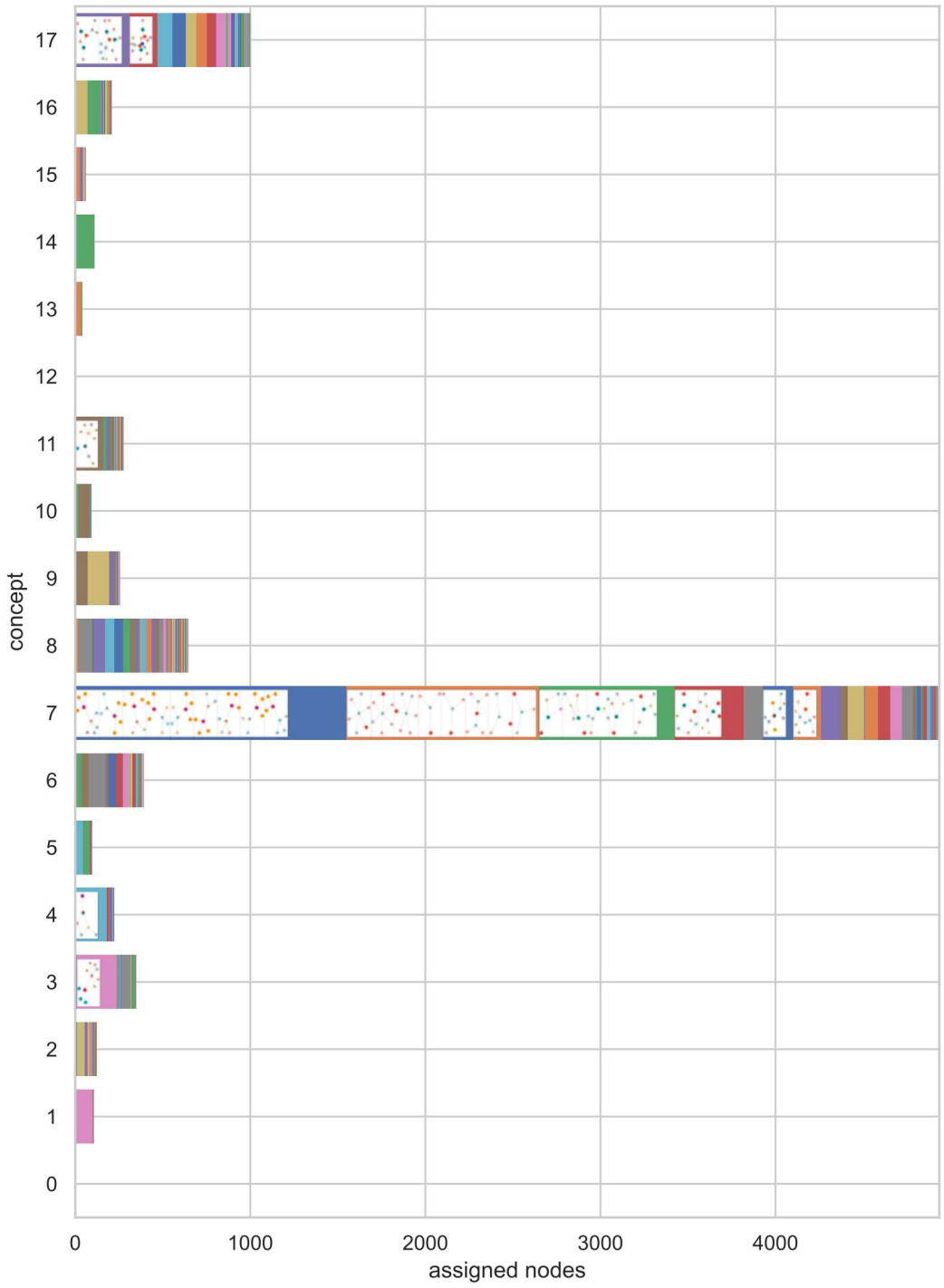


Figure 14: **Subgraphs matched to each concept in the second pooling layer of Mutagenicity and how often they occur**

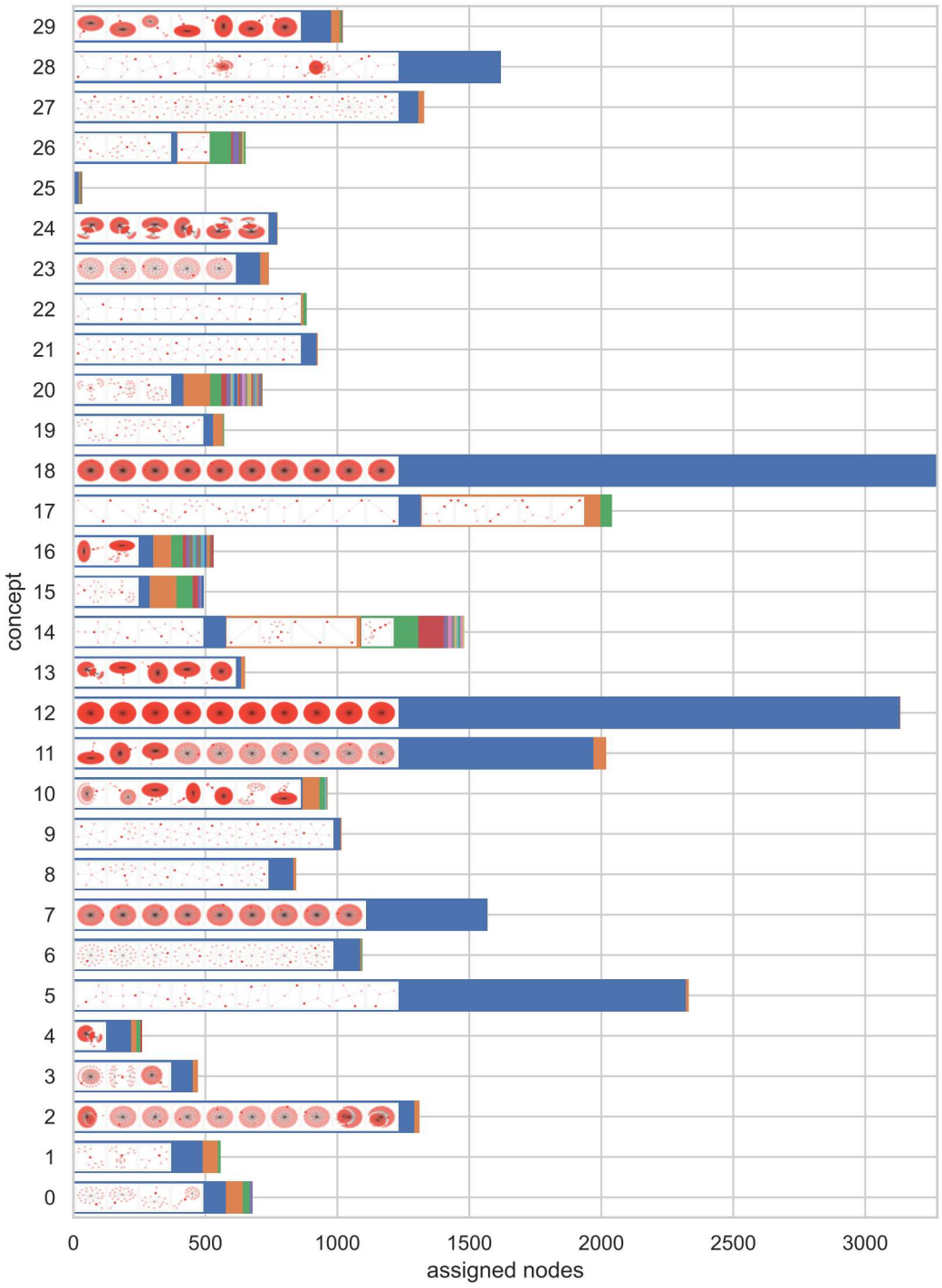


Figure 15: **Subgraphs matched to each concept in the first pooling layer of REDDIT-BINARY and how often they occur.** Generated with 50% of the test data for performance reasons.

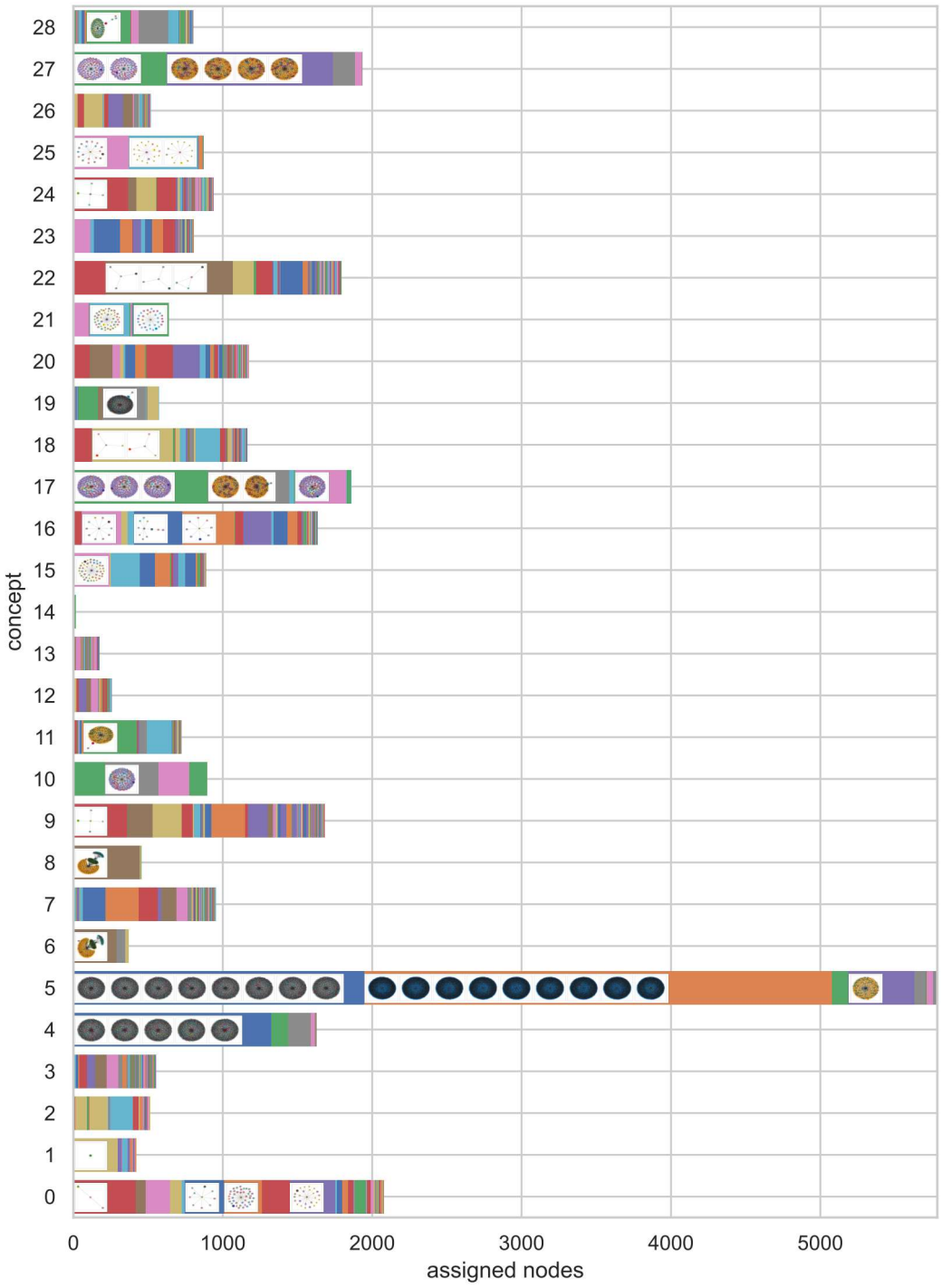


Figure 16: **Subgraphs matched to each concept in the second pooling layer of our REDDIT-BINARY and how often they occur.** Generated with 50% of the test data for performance reasons.