Weisfeiler and Leman Return with Graph Transformations

Fabian Jogl, Maximilian Thiessen, and Thomas Gärtner

Research Unit of Machine Learning, TU Wien, Vienna, Austria
{fabian.jogl, maximilian.thiessen, thomas.gaertner}@tuwien.ac.at

Abstract. We propose novel graph transformations that allow standard message passing to achieve state-of-the-art expressiveness and predictive performance. Message passing graph neural networks are known to have limited expressiveness in distinguishing graphs. To mitigate this, one can either change message passing or modify the graphs. Changing message passing is powerful but requires significant changes to existing implementations and cannot easily be combined with other approaches. Modifying the graph requires no changes to the learning algorithm and works directly with off-the-shelf implementations. In this paper, we propose novel graph transformations and compare them to the state-of-the-art. We prove that they are at least as expressive as corresponding message passing algorithms when combined with the Weisfeiler-Leman test or a sufficiently powerful graph neural network. Furthermore, we empirically demonstrate that these transformations lead to competitive results on molecular graph datasets.

Keywords: Graph Neural Networks · Weisfeiler Leman · Expressiveness.

1 Introduction

Message passing graph neural networks (GNNs) have limited expressiveness when it comes to distinguishing graphs [20, 28]. Vanilla GNNs can only distinguish two graphs which the Weisfeiler Leman test (WL) [27] can also distinguish. This means that GNNs are limited in terms of functions on graphs they can express. There are two common approaches of improving the expressiveness of methods that perform message passing on graphs. The first modifies the graphs by a transformation and the second improves the message passing algorithm. Graph transformations that improve the expressiveness might extend vertex features with random features [1, 9, 22]. Other graph transformations selected a set of patterns and for those patterns extend vertex features with rooted homomorphism counts [2], or subgraph isomorphism counts [7]. Some methods that improve message passing are, for example, higher order GNNs [20], Equivariant Subgraph Aggregation Networks [3], Structural Message-Passing Neural Networks [26], and CW Networks [4].

Improving expressiveness via a graph transformation has the advantage that only the graphs need to be transformed before running the learning algorithm.

This allows simple integration into previous implementations and can be easily combined with other existing models. However, improving message passing allows for a lot of flexibility when it comes to designing the algorithm. In contrast, graph transformations are very limited as the resulting algorithm will still rely on message passing. Thus, intuitively it seems that improving message passing should allow us to build more expressive algorithms than by applying just a graph transformation combined with message passing. We show that this is not necessarily the case. We prove that three methods with improved message passing CW Networks (CWN) [4], Equivariant Subgraph Aggregation Networks (ESAN) [3], and δ-k-dimensional WL (δ-k-WL) and its neural networks variants [19] can be reduced to graph transformations. We prove that combining our graph transformations with a sufficiently powerful GNN or WL is at least as expressive as the original algorithm. Additionally, we empirically demonstrate that our methods achieve competitive results on graph datasets.

Contributions. In this work we investigate whether it is possible to replace algorithms with different forms of message passing by a graph transformation. We introduce two novel algorithms that simplify CW Networks [4] and Equivariant Subgraph Aggregation Network [3] to a graph transformation. *Cell encoding* (Section 4) can transform any regular cell complex to a graph and *subgraph bag encoding* (Section 5) can transform any bag of subgraphs into a graph. We prove that these transformation combined with WL or a suitably expressive GNN are at least as expressive as the original methods in distinguishing regular cell complexes or graphs. We identify δ-k-WL [20] (Section 6) and its neural network variants as being an examples of our approach. While they are defined to perform message passing on k-tuples instead of single vertices, they have originally been implemented via a graph transformation plus standard message passing. We prove that implementing δ-k-WL with a graph transformation does not lose expressiveness over directly implementing the algorithm. In Section 7, we show empirically that cell encoding and subgraph bag encoding improves the results of GNNs on graph classification and regression tasks, yielding competitive results to CW Networks and Equivariant Subgraph Aggregation Networks.

2 Related Work

The high-level idea of replacing algorithms that improve message passing by graph transformations, has been proposed in parallel to our work, in a positional paper by Veličković [25]. Much work has been done on building message passing graph neural networks that are more expressive than WL. As we propose the idea of using graph transformations to simplify algorithms that improve message passing, it makes sense to analyze previous works through the lens of this idea. Thus we start with improved message passing algorithms. These methods usually transform the graph to a different structure and then perform message passing on that structure. We consider this transformation the first step of the improved message passing. In the following, we introduce three different types

of such methods. The first operates on k-tuples of nodes, the second on simplical complexes or regular cell complexes, and the third on subgraphs.

The first type of method exchanges messages between k-tuples of nodes. Higher dimensional WL (k-WL) is a generalization of WL and forms a sequence of algorithms that are stronger than WL [8]. Increasing k increases the expressiveness at the cost of the runtime. Morris et al. [20] introduced k-dimensional GNNs which extend the concept of k-WL to GNNs. For a sufficiently powerful neural network, k-GNNs have equal expressiveness as k-WL [20]. However, as k increases, the runtime of k-GNNs grows exponentially. To combat this, Morris et al. [19] introduced (local) δ-k dimensional WL and GNNs. These algorithms use the sparsity of graphs to improve the runtime and expressiveness. The second type of method operates on simplical complexes or a regular cell complexes. The two most prominent examples of this idea are Simplical Networks [5] and CW Networks [4]. Other algorithms that work on these structures are as Simplical Neural Networks [11], Dist2Cycle [18], and Cell Complex Neural Networks [15]. The third type of method decomposes the graph into subgraph, such as Automorphism-based Neural Networks (Autobahn) [24] or Equivariant Subgraph Aggregation Networks (ESAN) [3].

Next we consider algorithms that can be seen as applying a graph transformation. It has been proven that adding random features to vertices improves the expressiveness of GNNs [1, 9, 22]. GNNs with random features are universal, meaning that they can learn any function defined on a graph [1]. However, to do this they sacrifice permutation invariance and equivariance. This means that permuting the graph will change the result of these algorithms. Many permutation invariant approaches extend the graph features by counting patterns. Barceló et al. [2] extend GNNs with rooted homomorphism counts of a set of patterns. They prove multiple interesting theorems relating their GNN and the choice of patterns to the k-WL test. Graph Structural Networks (GSN) [7] introduce a new graph convolution layer which extends messages with subgraph isomorphism counts. While this cannot be directly seen as transforming the graph, it is very similar to adding these subgraph isomorphism counts to the vertex features.

3 Background

In this section, we introduce the concept of *expressiveness* and describe how it can be proven that a new message passing algorithm is more expressive than WL. We say that algorithm A is at least as *expressive* as algorithm B if A can distinguish every pair of graphs or regular cell complexes that B can distinguish. A is *equally expressive* as B, if A is at least as expressive as B and B is at least as expressive as A. A is *more expressive* than B if A is at least as expressive as B and can distinguish more pairs of graphs or cell complexes than B.

The usual first step of developing neural network based algorithms that are more expressive than WL, is to introduce a stronger variant of WL. This variant is used to compare the expressiveness to WL and thus also to GNNs, as GNNs are at most as expressive as WL [28]. Then one usually defines the neural network

and proves that it is equally expressive as the previously introduced WL variant if some conditions on the underlying neural network are met. This makes it possible to relate the expressiveness of the new neural network algorithms to WL and GNNs. For example Cellular Weisfeiler-Leman (CWL) is a variant of WL and CW Networks are the corresponding neural network equivalent [4]. For ESAN, the WL variant is called DSS-WL and the neural network is DSS-GNN. Finally for δ-k-WL the neural network equivalent is δ-k-GNN. As all our theorems focus on to the expressiveness of these algorithms, we will only introduce the WL variants in detail. In what follows, we use $\{\!\{\cdot\}\!\}$ to denote a multiset and $\mathcal{N}_G(v)$ to denote the neighbours of vertex v in graph G. In each iteration of a colour refinement algorithm, each vertex is assigned a colour which is a refinement of its colour in the previous iteration. We say a colouring is stable, if performing another iteration of the colouring algorithm does not refine the colouring.

4 Cell Encoding

CW Networks [4] perform message passing on regular cell complexes instead of graphs. By lifting graphs to regular cell complexes they can be enriched with more structure, yielding more expressive algorithms. We propose *cell encoding* which transforms any regular cell complex to graph. This allows any GNN to operate on regular cell complexes, and allows us to increase the expressiveness of GNNs by first lifting graphs to regular cell complexes and then transforming them back to graphs via cell encoding. The results of this section appeared already in [17].

4.1 CW Networks

In this section we briefly introduce CW Networks, for more information consider Appendix A. Bodnar et al. [4] generalized the message passing paradigm from graphs to regular cell complexes. Regular cell complexes generalize the simplical complexes used by [6]. A regular cell complex X is a topological space consisting of subspaces $\{X_\sigma\}_{\sigma \in P_X}$ called cells together with an indexing set P_X. This indexing set encodes all topological information about X and can be used to define a boundary relation $\prec$ between cells. This boundary relation can then be leveraged to define adjacencies between cells. Cellular Weisfeiler Leman (CWL) performs message passing on cells. In each iteration of CWL the algorithm computes a colouring for each cell depending on the colours of neighbouring cells in the previous iteration. Similar to WL two regular cell complexes are not isomorphic if at some iteration the colour histograms of all cells are different for the two complexes.

To apply the concept of regular cell complexes to graphs, Bodnar et al. [4] define the concept of a *cellular lifting map*, a function f that transforms a graph to a regular cell complex such that two graphs G_1, G_2 are isomorphic if and only if $f(G_1), f(G_2)$ are isomorphic. They prove that a class of lifting maps called *skeleton preserving lifting maps* together with CWL are at least as expressive as

WL. Typically, such lifting maps create cells out of vertices, together with cells that encode other structures such as induced cycles or cliques. Figure 1 shows an example of this, the original graph (left) is turned into a cell complex (center) where the vertices are 0-dimensional cells, edges are 1-dimensional cells, and induced cycles are 2-dimensional cells. Bodnar et al. [4] define CW Networks which combine neural networks with cellular message passing, similar to graph neural networks with message passing. CW Networks can be made equally expressive as CWL. Thus, by lifting graphs to cell complexes and then using a CW Network one can obtain algorithms that are strictly more expressive than WL.

4.2 Graph Transformation

We propose *cell encoding*, a novel algorithm that transforms a regular cell complex X to a graph G_X. A similar construction for a type of regular cell complexes called simplical complexes is already known to the topology community [13]. We show that cell encoding combined with WL is at least as expressive as CWL in distinguishing regular cell complexes. With this, we can perform message passing on graphs instead of cell complexes while keeping the expressiveness guarantees from CWL. However, this approach is not limited to cell complexes obtained with a cellular lifting map. Indeed, any cell complex can be transformed into a graph while ensuring that WL is as expressive least as CWL.

Definition 1 (Cell Encoding). *Given a regular cell complex X with a finite indexing set P_X, cell encoding transforms P_X into a graph $G_X = (V_X, E_X)$ with vertex features. Where*

$$V_X = P_X,$$

$$E_X = \{\{\tau, \delta\} \mid \tau, \delta \in P_X,\ \tau \prec \delta \text{ or } \delta \prec \tau\} \cup \{\{\tau, \delta\} \mid \exists \sigma \in P_X,\ \tau \prec \sigma,\ \delta \prec \sigma\}.$$

For unlabeled regular cell complexes we introduce a feature that encodes the dimension of each cell. For labeled regular cell complexes we extend the features correspondingly.

Encoding the dimension of a cell in vertex features can be done via one-hot encoding and we use it to distinguish between cells of different dimensions.

Theorem 1. *Cell encoding together with WL is as least as expressive as CWL.*

While cell encoding together with WL is as expressive as CWL, this does not mean that it yields exactly the same result. When CWL passes messages via upper adjacent cells, it adds the colour of the higher dimensional cell to the message. This is not something covered by our transformation. This does not impact the expressiveness of the method, but still might lead to better results on practical applications. Running cell encoding takes linear time with respect to the number of cells and their adjacencies. From runtime point of view, a single iteration of running WL on a graph obtained by cell encoding is slightly more efficient than CWL on the corresponding regular cell complex: each vertex corresponds to a cell and each edge to a message passed in CWL. However, as described above our method passes *less* messages, leading to it being slightly more efficient.

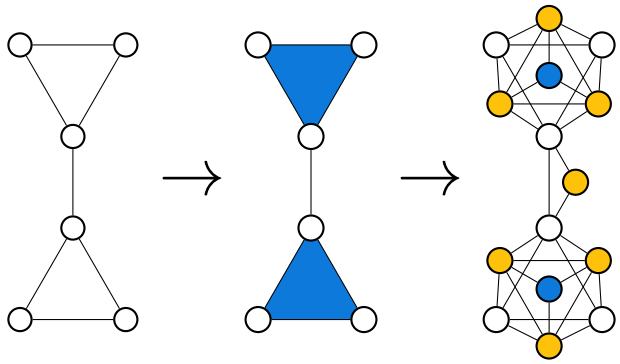

Fig. 1. Left: a graph. Center: a regular cell complex built from the graph by lifting all induced cycles to 2-dimensional cells (blue) via k-IC. Right: a graph obtained from the cell complex via cell encoding or directly from the original graph via cellular ring encoding. Vertices that existed in the original graph are colored white. Yellow vertices represent edges in the original graph and blue vertices represent induced cycles in the original graph. (This image is partly based on an image by [4])

Proof Sketch. (Full proof in Appendix A). We prove that every pair of vertices assigned the same colour by WL imply that the underlying cells will be assigned the same colour by CWL. We show this by induction on the iterations of CWL. The base case directly follows from the fact that graphs obtained by applying CRE to a regular cell complex have the same number of vertices as the underlying cell complex has cells. In the induction step, the properties of a stable WL colouring together with the vertex features encoding the dimension of cells means we can distinguish between vertices that correspond to cells of different dimensions. This allows us to show that if cell encoding together with WL cannot distinguish a pair of regular cell complexes then neither can CWL. □

4.3 Cellular Ring Encoding

We have seen that we can use cell encoding to transform any regular cell complexes to a graph. In this section, we show that cell encoding can be used to build more expressive GNNs. Bodnar et al. [4] present cellular lifting maps that when combined with CWL yield algorithms strictly more expressive than WL. One of these lifting maps is k-IC that transforms every vertex, edge, and induced cycle (IC) of length up to k into a cell. Note that $k \geq 3$ is a hyperparamter that needs to be set separately. Combining k-IC with cell encoding gives us *cellular ring encoding* (CRE). CRE transforms a graph into another graph with vertex features. An example of CRE can be seen in Figure 1.

Proposition 1. *CRE together with WL is more expressive than WL.*

Proof. k-IC has been shown to be strictly more expressive than WL when combined with CWL [4]. By Theorem 1 it follows that combining CRE with WL is strictly more expressive than just WL. □

Since the graph neural network GIN [28] can be made as expressive as WL, it follows that combining CRE with GIN is more expressive than WL.

5 Subgraph Bag Encoding

With CW Networks we have seen an algorithm that performs message passing on cell complexes. Equivariant Subgraph Aggregation Networks [3] (ESAN) perform message passing on a collection of subgraphs (subgraph bags). To apply this to graphs, a policy first transforms the graphs to subgraph bags. Depending on the policy, this can yield algorithms that are strictly more expressive than WL. As there are no restrictions on the policy, this is a very general method that can be adapted closely to the specific problem. We propose *subgraph bag encoding* which transforms any bag of subgraphs to a graph.

5.1 Equivariant Subgraph Aggregation Networks

To analyze the theoretical expressiveness of this approach Bevilacqua et al. [3] present DSS-WL, a stronger variant of WL. DSS-WL takes a graph G and a policy π of computing the subgraph bags. In each iteration the algorithm computes a colour $c_{v,S}$ for each node v in each subgraph S.

1. The policy π is applied to G to obtain the bag of subgraphs. If G has vertex features then the initial colour of a vertex are set to its features. Otherwise, all vertices are assigned the same colour.
2. Given the colour $c^t_{v,S}$ of vertex v in subgraph $S \in \pi(G)$ in iteration t. The colour in the next iteration is defined by $c^{t+1}_{v,S} = \text{HASH}\left(c^t_{v,S}, N^t_{v,S}, C^t_v, M^t_v\right)$. Where $N^t_{v,S} = \left\{\!\left\{ c^t_{x,S} \mid x \in \mathcal{N}_S(v) \right\}\!\right\}$ is the multiset of colours of neighbors of v in subgraph S, $C^t_v = \left\{\!\left\{ c^t_{v,S'} \mid S' \in \pi(G_i) \text{ and } v \in V(S') \right\}\!\right\}$ is the multiset of v's colour across the different subgraphs, and $M^t_v = \left\{\!\left\{ C^t_x \mid x \in \mathcal{N}_{G_i}(v) \right\}\!\right\}$ is the multiset of all C^t_x where x is a neighbor of v in the original graph G.
3. After each iteration a colour c_S is assigned to each subgraph S that encodes the multiset of node colours in S. Two graphs are non isomorphic if the multiset of these colours for the two graphs diverge. If the colour refinement algorithm converges on all subgraphs, then the test is inconclusive.

There exist policies that make DSS-WL strictly more powerful than WL and message passing graph neural networks [3]. One such policy computes 3-*ego-networks*, that is the induced k-hop neighbourhood for each node in the graph.

5.2 Graph Transformation

We propose *subgraph bag encoding* (SBE), an algorithm that for a given policy π transforms a graph G to G_π. We prove that SBE combined with WL is at least as expressive as DSS-WL (or DSS-GNN) in distinguishing pairs of graphs. An example of SBE can be found in Figure 2. For each vertex v in a subgraph

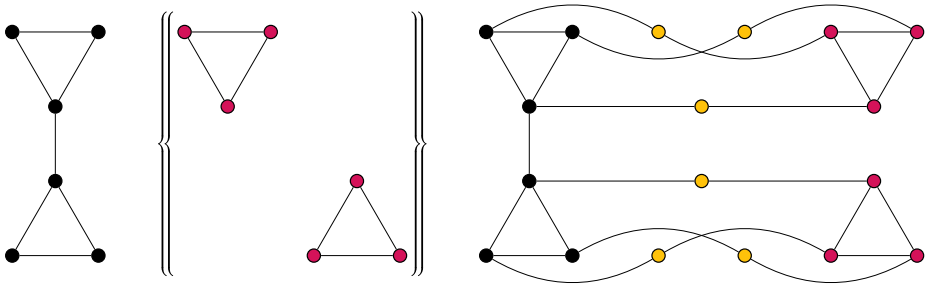

Fig. 2. A graph (left), the graph transformed into bags of subgraphs (center) by a policy and the result of applying subgraph bag encoding to the graph and bags of subgraphs (right). Black vertices represent the original graph, red vertices the subgraphs and yellow vertices the separator vertices.

$S \in \pi(G)$, SBE creates a unique vertex v_S and connects them according to $E(S)$. For any vertex $v \in V(G)$, it creates a vertex v_G and a separator vertex v'. All vertices of the form v_G are connected as defined by $E(G)$. Separator vertices connect vertices in different subgraphs that stem from the same vertex in the original graph.

Definition 2 (Subgraph Bag Encoding). *Given a graph G and policy π we define $G_\pi = (V_\pi, E_\pi)$ to be the graph obtained by subgraph bag encoding. Where*

$$V_\pi = \{v' \mid v \in V(G)\} \cup \bigcup_{S \in \pi(G) \cup G} V(S),$$

$$E_\pi = \bigcup_{S \in \pi(G) \cup G} E(S) \cup \{\{v', v_S\} \mid S \in \pi(G) \cup G \text{ and } v \in V(S)\}.$$

For unlabeled graphs we introduce a vertex feature that encodes whether it is a separator vertex, was created from a subgraph $S \in \pi(G)$, or was created from G. For labeled graphs we extend the features correspondingly.

Theorem 2. *SBE together with WL is at least as expressive as DSS-WL.*

Proof Sketch. (Full proof in Appendix A). The proof follows a similar structure as the proof of Theorem 1. We assume two arbitrary graphs G, H, a policy π such that WL cannot distinguish G_π, H_π. We show that DSS-WL cannot distinguish G, H by induction on the iterations t of DSS-WL. For each iteration we show that if a vertex in subgraph $S \in \pi(G)$ is assigned the same colour by WL as a vertex in subgraph $T \in \pi(H)$, then they are also assigned the same colour by DSS-WL. The base case follows from the fact that all vertices are initialized with the same colours by WL and DSS-WL, excluding type encoding. In the induction step, the we use the fact that separator vertices have features that distinguish them from subgraph vertices. Then we argue that the colour of a separator vertex encodes the colours of that vertex among the different subgraphs. □

Note that Theorem 2 does not put any requirements on the policy. This allows maximal flexibility when it comes to developing new policy while still being able to profit from subgraph bag encoding. SBE runs in linear time with respect to the size of all subgraphs computed by the policy plus the size of the original graph. A single iteration of WL on a graph created by SBE is slightly less efficient than ESAN on the corresponding bags of subgraphs as the separator nodes add additional overheard. However, we think that in a follow up work we be able to proof the theorem without using separator nodes.

6 δ-k Graph Transformation

The next algorithm, δ-k-dimensional Weisfeiler-Leman (δ-k-WL) [19] performs message passing on k-tuples of vertices and is very similar to k-WL [8]. However, only δ-k-WL distinguishes between global and local neighborhoods. The neural network equivalent of δ-k-WL [19] is an example of our approach. We show that implementing the algorithm with a graph transformation is at least as expressive as directly implementing the algorithm. Due to a lack of space, we defer this part to Appendix C.

7 Experiments

Models. In this section, we investigate the empirical performance of cell encoding and subgraph bag encoding. We compare our methods against the CW Network CIN [4] and the Equivariat Subgraph Aggregation Network DSS-GNN [3]. Similar to [3, 4] we use the Graph Isomorphism Network (GIN) [28] as our GNN. As GIN with suitable parameters is equally expressive as WL, this means that GIN with CRE or SBE can be more expressive than WL. We combine GIN with methods obtained via cell encoding and subgraph bag encoding. CIN performs message passing on cell complexes constructed with the cellular lifting map k-IC. Analogously, we use the cell encoding method cellular ring encoding (CRE). For subgraph bag encoding we use a policy that transforms a graph into a set of 3-ego-networks, that is a set of induced 3-hop neighborhoods for each vertex. We use the same policy for DSS-GNN. We also include GIN and a multilayer perceptron (MLP) as baselines. The baseline MLP performs a mean pooling operation over the whole graph and then applies a MLP to the resulting feature vector, completely ignoring the graph structure and edge features. Note that we do not compare the δ-k Graph Transformation against δ-k-GNN, as δ-k-GNN was implemented via a method that is an example of our approach.

Datasets. We evaluate on the graph regression dataset `ZINC` [14, 23][1], and on two graph classification datasets `ogbg-molhiv` [16] and `ogbg-moltox21` [16][2].

[1] `ZINC` is accessible via pytorch geometric `https://pytorch-geometric.readthedocs.io/`

[2] Both ogb datasets are accessible via the ogb python package `https://ogb.stanford.edu`

Table 1. ROC-AUC on `ogbg-molhiv` (bigger is better)

Algorithm	Test ROC-AUC	Validation ROC-AUC
MLP	0.62 ± 0.013	0.651 ± 0.005
GIN	0.766 ± 0.014	0.823 ± 0.007
CIN	0.772 ± 0.014	0.829 ± 0.014
GIN + CRE	0.779 ± 0.017	0.833 ± 0.007
DSS-GNN	0.774 ± 0.017	0.847 ± 0.011
GIN + SBE	0.737 ± 0.013	0.831 ± 0.010

Table 2. ROC-AUC on `ogbg-moltox21` (bigger is better)

Algorithm	Test ROC-AUC	Validation ROC-AUC
MLP	0.613 ± 0.003	0.586 ± 0.001
GIN	0.749 ± 0.005	0.788 ± 0.004
CIN	0.754 ± 0.006	0.803 ± 0.004
GIN + CRE	0.750 ± 0.007	0.804 ± 0.006
DSS-GNN	0.771 ± 0.009	0.824 ± 0.005
GIN + SBE	0.762 ± 0.006	0.814 ± 0.004

For `ZINC` we use the commonly used smaller variant that contains 12000 molecular graphs, it has been used for example by [2, 3, 7]. `ogbg-molhiv` contains 41127 graphs and `ogbg-moltox21` contains 7831 graphs. Both `ogbg-molhiv` and `ogbg-moltox21` are binary graph classification problems. However, the prior has only one task while the latter has 12 tasks, meaning that 12 predictions need to be performed simultaneously. As defined in the open graph benchmark (ogb) [16] we use ROC-AUC to evaluate the results on `ogbg-molhiv` and `ogbg-moltox21`. As is common we use mean average error for `ZINC`.

Setup. Our setup[3] is based on [10]. We evaluate each method on each of the three datasets. All datasets supply a train, validation and test split which we use and keep fixed for all algorithms. We tune the hyperparameters on the validation set. The used hypereparameter grids and more information about our setup can be found in Appendix D. We try 20 parameter combinations per method and dataset. During training we use a learning rate of 10^{-3}, whenever the model has not improved on the validation set for 20 epochs we lower the learning rate by 50%. The training stops when the learning rate dips below 10^{-5} or after at most 300 epochs. The only exception is that we train CIN on the `ogbg-molhiv` dataset for only 100 epochs due to how long the training takes. For evaluation, we use the metric obtained in the epoch with the best validation performance and report the average and standard deviation over 10 separate training runs with different random seeds.

Results. The results can be found in Table 1, 2, and 3. GIN+CRE and CIN obtain similar results on all three datasets and outperform GIN on two. On two of the three datasets GIN+SBE performs much better than the GIN and MLP baselines and competitively to DSS-GNN. Only on `ogbg-molhiv` it performs slightly worse than DSS-GNN and GIN. On `ZINC` both graph transformations show strong improvements over GIN.

[3] Code can be found at `https://github.com/ocatias/WL_Return`

Table 3. Mean average error (MAE) on `ZINC` (smaller is better)

Algorithm	Test MAE	Validation MAE
MLP	1.441 ± 0.001	1.317 ± 0.001
GIN	0.250 ± 0.007	0.264 ± 0.004
CIN	0.083 ± 0.005	0.087 ± 0.005
GIN + CRE	0.090 ± 0.005	0.097 ± 0.005
DSS-GNN	0.101 ± 0.004	0.118 ± 0.006
GIN + SBE	0.103 ± 0.006	0.133 ± 0.006

8 Conclusion

We have proposed three graph transformations, that can grant any GNN some of the advantages of CW Networks, ESAN, or (local) δ-k-WL. For a sufficiently powerful GNN, these transformations provably improve the expressiveness. Instead of requiring changes to existing models only the preprocessing of graphs needs to be adapted. Additionally, we can adapt any GNN to operate on regular cell complexes or bags of subgraphs. For cell encoding and subgraph bag encoding, we have demonstrated empirically that such an approach can improve the quality of predictions on graph classification and regression datasets. While our graph transformations often yield strong improvements over just an MPNN, this is not always the case. However, due to how simple it is to use this approach this is not a big problem: all that is required is to change the preprocessing of the graph datasets. With this many possible algorithms can be switched in and out quickly with little downside.

More generally, we have shown that graph transformations together with standard message passing can be used to implement improved message passing. We have also identified δ-k-WL and its variants as an example of this graph transformation approach that has been used in the past. Note that our graph transformations are not restricted to be only used with message passing algorithms. As long as they are combined with an algorithm that is at least as expressive as WL the theorems hold and give expressiveness guarantees.

The idea of graph transformations to improve the expressiveness of GNNs has limitations caused by the focus on expressiveness, as there is no direct connection between a GNN being more expressive and achieving better predictive results. As a consequence, the fact that a graph transformation increases the expressiveness of an MPNN alone is not useful: there is a trivial graph transformation that yields the same expressiveness as any algorithm A that is based on WL or computes something similar to graph isomorphism classes. For this one simply runs A on the input graph G and returns a new graph with a single node whose features encode the output of A on G. Combining this algorithm with an WL or a suitably expressive MPNN is guaranteed to be as expressive as A. However, it is unlikely to yield good predictive results as it compresses the graph into a single vector before the learning algorithm is applied. In the future, we intend to formalize this idea to avoid such issues and investigate how far we can improve message

passing with graph transformations. We conjecture that any GNN that passes messages between pairs of objects along paths that are fixed before the message passing is performed can be implemented as a graph transformation plus an MPNN.

References

1. Abboud, R., Ceylan, İ.İ., Grohe, M., Lukasiewicz, T.: The surprising power of graph neural networks with random node initialization. In: IJCAI (2021)
2. Barceló, P., Geerts, F., Reutter, J.L., Ryschkov, M.: Graph neural networks with local graph parameters. In: NeurIPS (2021)
3. Bevilacqua, B., Frasca, F., Lim, D., Srinivasan, B., Cai, C., Balamurugan, G., Bronstein, M., Maron, H.: Equivariant subgraph aggregation networks. In: ICLR (2021)
4. Bodnar, C., Frasca, F., Otter, N., Wang, Y.G., Liò, P., Montúfar, G., Bronstein, M.: Weisfeiler and Lehman go cellular: CW networks. In: NeurIPS (2021)
5. Bodnar, C., Frasca, F., Wang, Y.G., Otter, N., Montufar, G., Lio, P., Bronstein, M.: Weisfeiler and Lehman go topological: Message passing simplicial networks (03 2021)
6. Bodnar, C., Frasca, F., Wang, Y.G., Otter, N., Montúfar, G., Liò, P., Bronstein, M.: Weisfeiler and Lehman go topological: Message passing simplicial networks. In: ICML (2021)
7. Bouritsas, G., Frasca, F., Zafeiriou, S., Bronstein, M.M.: Improving graph neural network expressivity via subgraph isomorphism counting. arXiv preprint arXiv:2006.09252, earlier version appeared in Graph Representation Learning and Beyond (GRL+) Workshop at ICML (2020)
8. Cai, J., Fürer, M., Immerman, N.: An Optimal Lower Bound on the Number of Variables for Graph Identification. Combinatorica **12**, 389–410 (1992)
9. Dasoulas, G., Dos Santos, L., Scaman, K., Virmaux, A.: Coloring graph neural networks for node disambiguation. In: IJCAI (2020)
10. Dwivedi, V.P., Joshi, C.K., Laurent, T., Bengio, Y., Bresson, X.: Benchmarking graph neural networks. arXiv preprint arXiv:2003.00982 (2020)
11. Ebli, S., Defferrard, M., Spreemann, G.: Simplicial neural networks. In: NeurIPS (2020)
12. Fey, M., Lenssen, J.E.: Fast graph representation learning with PyTorch Geometric. In: ICLR Workshop on Representation Learning on Graphs and Manifolds (2019)
13. Grigor, A., Yan, Muranov, Y., Yau, S.T.: Graphs associated with simplicial complexes. Homology, Homotopy and Applications **16** (01 2014)
14. Gómez-Bombarelli, R., Wei, J.N., Duvenaud, D., Hernández-Lobato, J.M., Sánchez-Lengeling, B., Sheberla, D., Aguilera-Iparraguirre, J., Hirzel, T.D., Adams, R.P., Aspuru-Guzik, A.: Automatic chemical design using a data-driven continuous representation of molecules. ACS Central Science **4**(2), 268–276 (2018)
15. Hajij, M., Istvan, K., Zamzmi, G.: Cell complex neural networks. In: Topological Data Analysis and Beyond Workshop at NeurIPS (2020)
16. Hu, W., Fey, M., Zitnik, M., Dong, Y., Ren, H., Liu, B., Catasta, M., Leskovec, J.: Open Graph Benchmark: Datasets for machine learning on graphs. In: NeurIPS (2020)
17. Jogl, F., Thiessen, M., Gärtner, T.: Reducing learning on cell complexes to graphs. In: Workshop on Geometrical and Topological Representation Learning at ICLR (2022)

18. Keros, A.D., Nanda, V., Subr, K.: Dist2cycle: A simplicial neural network for homology localization. In: AAAI (2022)
19. Morris, C., Rattan, G., Mutzel, P.: Weisfeiler and Leman go sparse: Towards scalable higher-order graph embeddings. arXiv: Data Structures and Algorithms (2020)
20. Morris, C., Ritzert, M., Fey, M., Hamilton, W., Lenssen, J., Rattan, G., Grohe, M.: Weisfeiler and Leman go neural: Higher-order graph neural networks. In: AAAI (2019)
21. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., Chintala, S.: PyTorch: An imperative style, high-performance deep learning library. In: NeurIPS (2019)
22. Sato, R., Yamada, M., Kashima, H.: Random features strengthen graph neural networks. In: SDM (2021)
23. Sterling, T., Irwin, J.J.: Zinc 15 – ligand discovery for everyone. Journal of Chemical Information and Modeling **55**(11), 2324–2337 (2015)
24. Thiede, E., Zhou, W., Kondor, R.: Autobahn: Automorphism-based Graph Neural Nets. In: NeurIPS (2021)
25. Veličković, P.: Message passing all the way up. In: ICLR Workshop on Geometrical and Topological Representation Learning (2022)
26. Vignac, C., Loukas, A., Frossard, P.: Building powerful and equivariant graph neural networks with structural message-passing. In: NeurIPS (2020)
27. Weisfeiler, B., Leman, A.: The reduction of a graph to canonical form and the algebra which appears therein. Nauchno-Technicheskaya Informatsia **9** (1968)
28. Xu, K., Hu, W., Leskovec, J., Jegelka, S.: How powerful are graph neural networks? In: ICLR (2019)

A Proof for Cell Encoding

Before we prove Theorem 1, we introduce necessary definitions. Bodnar et al. [4] generalized the message passing paradigm from graphs to regular cell complexes. Regular cell complexes generalize the simplical complexes used by [6]. A regular cell complex X is a topological space with a partition of subspaces $\{X_\sigma\}_{\sigma \in P_X}$ called cells with an indexing set P_X. The subspace of each cell defines its dimension. This indexing set encodes all topological information about X and can be used to define a boundary relation $\prec$ between cells.

Definition 3 (Bodnar et al. [4]). *For cells σ, τ the boundary relation $\sigma \prec \tau$ holds if and only if $X_\sigma \subset X_\tau$ and there is no cell δ such that $X_\sigma \subset X_\delta \subset X_\tau$.*

This boundary relation can then be used to define adjacencies between cells.

Definition 4 (Bodnar et al. [4]). *For a regular cell complex X and a cell $\sigma \in P_X$, we define:*

1. *The boundary adjacent cells $\mathcal{B}(\sigma) = \{\tau \mid \tau \prec \sigma\}$. These are the lower-dimensional cells on the boundary of σ. For instance, the boundary cells of an edge are its vertices.*
2. *The co-boundary adjacent cell $\mathcal{C}(\sigma) = \{\tau \mid \sigma \prec \tau\}$. These are the higher-dimensional cells with σ on their boundary. For instance, the co-boundary cells of a vertex are the edges it is part of.*
3. *The upper adjacent cells $\mathcal{N}_\uparrow(\sigma) = \{\tau \mid \exists \delta$ such that $\sigma \prec \delta$ and $\tau \prec \delta\}$. These are the cells of the same dimension as σ that are on the boundary of the same higher-dimensional cell as σ. The typical graph adjacencies between vertices are the canonical example here.*

To WL generalize to regular cell complexes, Bodnar et al. define how to collect the colours of neighbouring cells.

Definition 5 (Bodnar et al. [4]). *For any cells $\sigma, \tau \in P_X$ we define $\mathcal{C}(\sigma, \tau) = \mathcal{C}(\sigma) \cap C(\tau)$.*

Definition 6 (Bodnar et al. [4]). *Let c be a celulular colouring of X with c_σ denoting the colour assigned to cell $\sigma \in P_X$. We define the following multi-sets of colours:*

1. *The colours of the boundary cells of $\sigma : c_\mathcal{B}(\sigma) = \{\!\{c_\tau \mid \tau \in \mathcal{B}(\sigma)\}\!\}$.*
2. *The upper adjacent colours of $\sigma : c_\uparrow(\sigma) = \{\!\{(c_\tau, c_\delta) \mid \tau \in \mathcal{N}_\uparrow(\sigma)$ and $\delta \in \mathcal{C}(\sigma, \tau)\}\!\}$.*

Cellular Weisfeiler Leman (CWL) performs message passing on cells. In each iteration of CWL the algorithm computes a colouring for each cell depending on the colours of neighbouring cells in the previous iteration. We define the neighbourhood of a cell σ via $c_\mathcal{B}(\sigma)$ and $c_\uparrow(\sigma)$. Similar to WL two regular cell complexes are not isomorphic if at some iteration the colour histograms of the cells are different for the two complexes.

1. Given a regular cell complex X, all cells are initialised with the same colour.
2. Given the colour c^t_σ of cell σ at iteration t, we compute the colour of cell σ at the next iteration c^{t+1}_σ by injectively mapping the multi-sets of colours belonging to the adjacent cells of σ using an injective HASH function: $c^{t+1}_\sigma = \text{HASH}\left(c^t_\sigma, c^t_{\mathcal{B}}(\sigma), c^t_{\uparrow}(\sigma)\right)$.
3. The algorithm stops when a stable colouring is reached. Two cell complexes are considered non-isomorphic if their colour histograms are different. Otherwise, the test is inconclusive.

To apply the concept of regular cell complexes to graphs, Bodnar et al. [4] define the concept of a *cellular lifting map*, a function f that transforms a graph to a regular cell complex such that two graphs G_1, G_2 are isomorphic if and only if $f(G_1), f(G_2)$ are isomorphic. They prove that lifting maps called *skeleton preserving lifting maps* together with CWL are at least as expressive as WL. Typically, lifting maps create cells out of vertices, together with cells that encode other structures such as induced cycles or cliques. They define CW Networks which combine neural networks with cellular message passing, similar to graph neural networks with message passing. With a sufficiently powerful neural network CW Networks are equally expressive as CWL. Thus, by lifting graphs to cell complexes and then using a CW Network one can obtain algorithms that are strictly more expressive than WL.

We prove that Theorem 1 holds.

Proof. Let P_X, P_Y be the indexing sets of two regular cell complexes. Let G_X and H_Y be the graphs obtained by applying cell encoding to P_X and P_Y. We use π to denote the stable colouring obtained by WL to G_X, H_Y and c^t to denote the colour obtained by CWL on P_X, P_Y after iteration t. Thus π_σ denotes the colours assigned to *vertex* σ by WL and c^t_σ denotes the colour assigned to *cell* σ by CWL. We assume that WL with cell encoding cannot distinguish G_X and H_Y. From this we show that for every iteration $t \geq 0$ of CWL it holds that:

For all $\tau \in V(G_X), \sigma \in V(H_Y)$ with $\pi_\tau = \pi_\sigma$ it holds that $c^t_\tau = c^t_\sigma$.

Note that this is equivalent to showing that if WL with cell encoding cannot distinguish G_X and H_Y then CWL cannot distinguish P_X and P_Y. When WL with cell encoding cannot distinguish G_X and H_Y, then we know that for every vertex in G_X there is a vertex in H_Y that are assigned the same color by WL. The statement then implies that there is a bijective mapping from cells of P_X to P_Y such that they are assigned the same colour by CWL which means that the histogram of colours is the same for both graphs. From this it follows that CWL cannot distinguish P_X, P_Y. We show that this statement holds by induction on the iteration t of CWL.

The proof will make use of the fact that π is a stable colouring. This means that any two vertices $p \in V(G_X)$ and $q \in V(H_Y)$ assigned the same colour $\pi_p = \pi_q$ would be assigned the same colour in any additional iteration of WL. This implies that the multiset of colours of neighbors of p is equivalent to the multiset of colours of neighbors of q. Thus, there exists a bijective function $\alpha : \mathcal{N}_{G_X}(p) \to \mathcal{N}_{H_Y}(q)$ such that for any $x \in \mathcal{N}_{G_X}(p)$ it holds that $\pi_x = \pi_{\alpha(x)}$.

Base case: We show that the statement holds for $t = 0$. CWL initializes all cells to the same colour. Thus, all we need to show is that P_X and P_Y have the same number of cells. The number of vertices in G_X and H_Y is equal to the number of cells in P_X and P_Y, respectively. Since WL cannot distinguish G_X and H_Y we know that they must have the same number of vertices.

Induction hypothesis: We assume that the statements holds for n.

Induction step: We show that the statements hold for $n+1$. Let $\tau \in V(G_X), \sigma \in V(H_Y)$ be arbitrary vertices with $\pi_\tau = \pi_\sigma$. We need to show that $c_\tau^{n+1} = c_\sigma^{n+1}$. By the definition of CWL we know that $c_\tau^{n+1} = \text{HASH}\left(c_\tau^n, c_{\mathcal{B}}^n(\tau), c_\uparrow^n(\tau)\right)$ and $c_\sigma^{n+1} = \text{HASH}\left(c_\sigma^n, c_{\mathcal{B}}^n(\sigma), c_\uparrow^n(\sigma)\right)$. We will show that the inputs into the two hash functions are equal for c_τ^{n+1} and c_σ^{n+1}.

First, we show that $c_\tau^n = c_\sigma^n$. This immediately follows from the assumption $\pi_\tau = \pi_\sigma$ and the induction hypothesis.

Next, we want to show that $c_{\mathcal{B}}^n(\tau) = c_{\mathcal{B}}^n(\sigma)$. The assumption $\pi_\tau = \pi_\sigma$ implies that there exists a bijective function $\alpha : \mathcal{N}_{G_X}(\tau) \to \mathcal{N}_{H_Y}(\sigma)$ such that for any $x \in N_{G_X}(\tau)$ it holds that $\pi_x = \pi_{\alpha(x)}$. Since the initial colours encode the dimensions of the cell, this function mus also respect the dimension of the cell meaning it only maps vertices to vertices whose cells have the same dimensions. This implies that for every cell μ with $\mu \prec \tau$ we know that there exists a cell $\nu = \alpha(\mu)$ with $\nu \prec \sigma$ such that $\pi_\mu = \pi_\nu$. With the induction hypothesis it follows that $c_\mu^n = c_\nu^n$. Observe, that $\mathcal{B}(\tau)$ contains only cells μ with $\mu \prec \tau$. Analogously, $\mathcal{B}(\sigma)$ contains cells ν with $\nu \prec \sigma$. Thus $c_{\mathcal{B}}^n(\tau) = c_{\mathcal{B}}^n(\sigma)$.

Finally, we need to show that $c_\uparrow^n(\tau) = c_\uparrow^n(\sigma)$. By definition we know that $c_\uparrow^n(\tau) = \left\{\!\!\left\{(c_\mu^n, c_\delta^n) \mid \mu \in \mathcal{N}_\uparrow(\tau) \text{ and } \delta \in \mathcal{C}(\tau, \mu)\right\}\!\!\right\}$. We can rewrite this as $c_\uparrow^n(\tau) = \left\{\!\!\left\{(c_\mu^n, c_\delta^n) \mid \tau \prec \delta \text{ and } \mu \prec \delta\right\}\!\!\right\}$. We will make use of the bijective function α defined in the paragraph above. We know that for any cell δ with $\tau \prec \delta$ there exists a vertex δ adjacent to τ such that $\pi_\delta = \pi_{\alpha(\delta)}$. This implies two things: first by using the induction hypothesis we know that $c_\delta^n = c_{\alpha(\delta)}^n$. Secondly, there exist a bijective function $\beta_\delta : \mathcal{N}_{G_X}(\delta) \to \mathcal{N}_{H_Y}(\alpha(\delta))$ that has the same properties as α. That is, for any $x \in \mathcal{N}_{G_X}(\delta)$ it holds that $\pi_x = \pi_{\beta_\delta(x)}$.

We can now put all of this together. The existence of α means that for any cell δ with $\tau \prec \delta$ there is a cell $\alpha(\delta) \in P_Y$ such that $c_\delta^n = c_{\alpha(\delta)}^n$. Next, with the existence of β_δ it follows that for each cell μ with $\mu \prec \delta$ there exists a cell $\beta_\delta(\mu) \in P_Y$ such that $c_\mu^n = c_{\beta_\delta(\mu)}^n$. With the fact that α and $\beta_\delta(\mu)$ are bijective, it follows that $c_\uparrow^n(\tau) = c_\uparrow^n(\sigma)$. This proves the induction step and concludes the proof of Theorem 1. □

B Proof for Subgraph Bag Encoding

We prove that Theorem 2 holds.

Proof. Let G, H be two graphs and π a policy. We use $\tau_{v,S}$ to denote the colouring of vertex v_S obtained by applying WL to G_π, and τ_v to denote the colour of vertex v. We use $c_{v,S}^t$ to denote the colour of vertex v in subgraph S obtained in

the t-th iteration of DSS-WL. We assume that WL cannot distinguish G_π, H_π. We prove that for any $t \geq 0$ it holds that

$$\forall v \in V(G), w \in V(H), S \in \pi(G), T \in \pi(H) : \tau_{v,S} = \tau_{w,T} \rightarrow c^t_{v,S} = c^t_{w,T}$$

by induction on the iteration t of DSS-WL. Note that proving this statement is equivalent to the theorem: As the graphs cannot be distinguished by WL we know that there exists a bijection $\alpha : V(G_\pi) \rightarrow V(H_\pi)$ that maps vertices of the same colours together. Since the additional features let us distinguish vertices v from v_S, this means that α always maps a vertex v_S to a vertex w_T such that they are assigned the same colour. From the statement then follows that the graphs are assigned the same colours by DSS-WL.

Base case: Any vertices $v_S \in V(G_\pi), w_T \in V(H_\pi)$ are assigned the same colour by WL as their counterpart $v \in V(S), w \in V(T)$ by DSS-WL. Thus from $\tau_{v,S} = \tau_{w,T}$ it follows that $c^0_{v,S} = c^0_{w,T}$.

Induction hypothesis: We assume the statement holds for n.

Induction step: Let $v \in V(G), w \in V(H), S \in \pi(G), T \in \pi(H)$. We assume that $\tau_{v,S} = \tau_{w,T}$ holds and want to show $c^{n+1}_{v,S} = c^{n+1}_{w,T}$. For this, we show that the colour obtained by the hash function is the same, as in both cases the same values get put into the function. Thus, we need to show that $c^n_{v,S} = c^n_{w,T}$, $N^n_{v,S} = N^n_{w,T}$, $C^n_v = C^n_w$, and $M^n_v = M^n_w$. From the induction hypothesis and $\tau_{v,S} = \tau_{w,T}$ it immediately follows that $c^n_{v,S} = c^n_{w,T}$.

We want to show that $N^n_{v,S} = N^n_{w,Z}$. From $\tau_{v,S} = \tau_{w,Z}$ and the fact that τ is a stable colouring we know there exists a bijection $\beta : \mathcal{N}_{G_\pi}(v_S) \rightarrow \mathcal{N}_{H_\pi}(w_Z)$ where for any $x \in \mathcal{N}_{G_\pi}(v_S)$ it holds that $\tau_x = \tau_{\beta(x)}$. Additionally, all neighbors of v_S are either separator vertices or neighbors in the subgraph S and the features allow us to distinguish between those two types. Thus β maps vertices from $V(S)$ to vertices from $V(Z)$. Note that these are exactly the vertices whose colours are in $N^n_{v,S}, N^n_{w,Z}$. Thus, by combining this with the induction hypothesis we obtain that $N^n_{v,S} = N^n_{w,Z}$.

We want to show that $C^n_v = C^n_w$. We use the function β defined in the previous paragraph. Due to the additional features, β must map a separator vertex v' to a separator vertex w' meaning that $\tau_{v'} = \tau_{w'}$. As the colouring is stable, this means that there exists a bijective function γ mapping neighbors of v' to neighbors of w' such that they are assigned the same colour by τ. As these neighborhoods correspond to $c_{v,S'}, c_{w,T'}$ for all $S' \in \pi(G), T' \in \pi(H)$, we can combine this with the induction hypothesis to obtain that $C^n_v = \left\{\!\left\{ c^n_{v,S'} \mid S' \in \pi(G) \text{ and } v \in V(S') \right\}\!\right\} = \left\{\!\left\{ c^n_{w,T'} \mid T' \in \pi(H) \text{ and } w \in V(T') \right\}\!\right\} = C^n_w$. Note that v' also has a neighbor v_G, that is a vertex from the graph G, which was *not* created from a policy. However, as this vertex is assigned a different feature γ does not map such a vertex to a vertex created by the policy.

Finally, we want to show $M^n_v = M^n_w$. Recall that $M^n_v = \left\{\!\left\{ C^n_x \mid x \in \mathcal{N}_G(v) \right\}\!\right\}$. In the previous paragraph we have already argued that $\tau_{v'} = \tau_{w'}$. Thus we know that $\tau_{v,G} = \tau_{w,H}$ as the colouring is stable and they are assigned a feature that is unique among the neighbours of v' and w'. Thus there exists a bijective function $\sigma : \mathcal{N}_G(v) \rightarrow \mathcal{N}_H(w)$ such that for any neighbor $x \in \mathcal{N}_G(v)$ it holds that

$\tau_{x,G} = \tau_{\sigma(x),H}$. We want to show that for any such neighbor $x \in N_G(v)$ it holds that $C^n_x = C^n_{\sigma(x)}$. Let x be one such vertex. As $\tau_{x,G} = \tau_{\sigma(x),H}$ it follows that the separator vertices assigned to x and $\sigma(x)$ have the same colour: $\tau_x = \tau_{\sigma(x)}$. From here we can use the same argument as in the previous paragraph to obtain that $C^n_x = C^n_{\sigma(x)}$. This concludes the proof of the induction hypothesis and shows that the theorem holds.

□

C δ-k Graph Transformation

We start by introducing the relevant variants of (local) δ-k-WL [19]. Then we define our graph transformations and prove that they are at least as expressive as the original algorithms.

C.1 δ-k-dimensional Weisfeiler-Leman

The algorithm starts by assigning to each k-tuple a colour that encodes its isomorphism class, meaning that two tuples are assigned the same colour if and only if there exists an isomorphism between the labeled subgraphs induced by the tuples. In each iteration of δ-k-WL, each k-tuple is assigned a colour that depends on its previous colour and the colours of its neighbours: Let $\mathbf{v}$ be a k-tuple of vertices from graph G and j be in $\{1, \dots, n\}$. We use $\phi_j(\mathbf{v}, w)$ to denote the tuple obtained by replacing the j-th vertex in $\mathbf{v}$ by the vertex w. For a tuple $\mathbf{w} = \phi_j(\mathbf{v}, w)$, we say that $\mathbf{v}$ and $\mathbf{w}$ are j-neighbours. We use v_j to denote the j-th vertex in $\mathbf{v}$, that is the vertex that gets replaced by w to obtain $\mathbf{w}$. If v_j and w are neighbors in G, then $\mathbf{v}$ and $\mathbf{w}$ are local j-neighbours, otherwise they are global j-neighbours. The function $\text{adj}(\mathbf{v}, \mathbf{w})$ returns 'L' if $\mathbf{v}$ and $\mathbf{w}$ are local j-neighbours and 'G' otherwise. We can define δ-k-WL:

1. Assign to each k-tuple a colour that encodes its isomorphism class
2. Given the colour $c^t_{\mathbf{v}}$ of tuple $\mathbf{v}$ in iteration t. The colour in the next iteration is defined by $c^{t+1}_{\mathbf{v}} = \left(c^t_{\mathbf{v}}, M^t_{\delta,\bar{\delta}}(\mathbf{v})\right)$ where

$$M^t_{\delta,\bar{\delta}}(\mathbf{v}) = \Bigg(\left\{\!\!\left\{ \left(c^n_{\phi_1(\mathbf{v},w)}, \text{adj}(\mathbf{v}, \phi_1(\mathbf{v}, w))\right) \mid w \in V(G) \right\}\!\!\right\}, \dots, \left\{\!\!\left\{ \left(c^n_{\phi_k(\mathbf{v},w)}, \text{adj}(\mathbf{v}, \phi_k(\mathbf{v}, w))\right) \mid w \in V(G) \right\}\!\!\right\} \Bigg).$$

3. The algorithm stops when a stable colouring is reached. Two graphs are non-isomorphic if their histogram of colours is different. Otherwise the test is inconclusive.

The δ-k-WL algorithm is strictly more expressive than k-WL. However, as it passes messages between global and local neighbors, it has the same runtime complexity as k-WL. To remedy this, Morris et al. [20] propose two additional

variants of δ-k-WL that make only limited use of global neighbourhoods. Local δ-k-Weisfeiler-Leman (δ-k-LWL) only considers local j-neighborhoods. For this $M^t_{\delta,\bar{\delta}}(\mathbf{v})$ gets replaced by

$$M^t_\delta(\mathbf{v}) = \left(\left\{\!\left\{ c_{\phi_1(\mathbf{v},w)|w\in\mathcal{N}_G(v_1)} \right\}\!\right\}, \ldots, \left\{\!\left\{ c_{\phi_k(\mathbf{v},w)|w\in\mathcal{N}_G(v_k)} \right\}\!\right\} \right).$$

The authors proved that δ-k-WL is at least as expressive as δ-k-LWL. Another variant, δ-k-LWL$^+$ uses a similar aggregation function as $M^t_\delta(\mathbf{v})$. However, it supplements the colour of a local j-neighbour by the *number* of j-neighbours (including global neighbours) that have the same colour as the local neighbour. This means that δ-k-LWL$^+$ uses only some global information. Interestingly, δ-k-LWL$^+$ is equally expressive as δ-k-WL. There exist equally expressive neural network variant for all δ-k-WL variants.

C.2 Graph Transformation

We present *δ-k graph transformation*, an algorithm that transforms a graph to another graph such that applying WL or a suitably expressive GNN to the result graph is equally expressive as δ-k-WL. Note that a very similar construction was used by Morris et al. [19] to implement neural network variants of δ-k-WL.

Definition 7 (δ-k Graph Transformation). *Let $G = (V, E)$ be a graph and $k \geq 1$ an integer. We denote by T_L (T_G) the sets containing every triplet (x, y, j) for which the k-tuples x and y are local (respectively global) j-neighbors. Then applying a δ-k graph transformation to G returns the graph $G_{\delta,k} = (V_{\delta,k}, E_{\delta,k})$ where*

$$V_{\delta,k} = \{x \mid x \in V(G)^k\} \cup \{l_{xyj} \mid (x, y, j) \in T_L\} \cup \{g_{xyj} \mid (x, y, j) \in T_G\},$$

$$E_{\delta,k} = \{\{x, l_{xyj}\}, \{y, l_{xyj}\} \mid (x, y, j) \in T_L\} \cup \{\{x, g_{xyj}\}, \{y, g_{xyj}\} \mid (x, y, j) \in T_G\}.$$

For any vertex x corresponding to a tuple we add a vertex feature that encodes its isomorphism type and allows us to distinguish it from l and g vertices. To all l_{xyj} (or g_{xyj}) vertices we add a feature that allows us to distinguish it from all g (respectively l) vertices and all vertices corresponding to tuples. Additionally, we add a feature that encodes j.

The l and g vertices and their features allow us to distinguish whether two tuples are global or local j-neighbors, and keep track of the value of j. These nodes are not strictly necessary if one uses edge features. For this one removes all edges, and all g and l vertices. Then one simply adds an edge between two vertices if their corresponding tuples are j-neighbors for any value of j. The edge features then encode j and whether they are global or local neighbours. This works because for any two different tuples there exists at most one value of j for which they are j-neighbours. Thus at most one edge needs to be created between any two vertices.

For the purpose of GNNs, the features must be padded to have the same length for all vertices. Additionally, it might be useful to use one-hot encoding for j and the type of vertex.

Theorem 3. *δ-k graph transformation together with WL is at least as expressive as δ-k-WL.*

Proof. Let G and H be two graphs, we need to prove that if WL cannot distinguish $G_{\delta,k}$ and $H_{\delta,k}$ then δ-k-WL cannot distinguish G and H. We use π_v to denote the colour of vertex v encoding a k-tuple obtained by applying WL to $G_{\delta,k}$ or $H_{\delta,k}$. We use $k^t_{\mathbf{v}}$ to denote the colour of the corresponding k-tuple $\mathbf{v}$ computed in the t-th iteration of δ-k-WL on G or H. We assume that WL cannot distinguish $G_{\delta,k}, H_{\delta,k}$ and show the stronger statement

$$\forall x \in V(G)^k, y \in V(H)^k : \pi_x = \pi_y \rightarrow c^t_{\mathbf{x}} = c^t_{\mathbf{y}}$$

for all $t \geq 0$ by induction on the iteration t of δ-k-WL.

Base case: Let x and y be arbitrary k-tuples. The statement follows from the fact that both the initial features of x, y and $c^0_{\mathbf{x}}, c^0_{\mathbf{y}}$ encode the isomorphism class of x and y.

Induction hypothesis: We assume the statement holds for n.

Induction step: Let x and y be arbitrary k-tuples. We need to show that $\pi_x = \pi_y \rightarrow c^{n+1}_{\mathbf{x}} = c^{n+1}_{\mathbf{y}}$. We assume that $\pi_x = \pi_y$. From the definition of δ-k-WL it follows that $c^{n+1}_{\mathbf{x}} = \left(c^n_{\mathbf{x}}, M^n_{\delta,\bar{\delta}}(\mathbf{x})\right)$ and $c^{n+1}_{\mathbf{y}} = \left(c^n_{\mathbf{y}}, M^n_{\delta,\bar{\delta}}(\mathbf{y})\right)$. From the assumption that $\pi_x = \pi_y$ and the induction hypothesis it follows that $c^n_{\mathbf{x}} = c^n_{\mathbf{y}}$. Next, we want to show that $M^n_{\delta,\bar{\delta}}(\mathbf{x}) = M^n_{\delta,\bar{\delta}}(\mathbf{y})$. By definition we know that

$$M^n_{\delta,\bar{\delta}}(\mathbf{x}) = \Bigg(\left\{\!\!\left\{ \left(c^n_{\phi_1(\mathbf{x},w)}, \mathrm{adj}(x, \phi_1(\mathbf{x}, w))\right) \mid w \in V(G) \right\}\!\!\right\}, \ldots, \\ \left\{\!\!\left\{ \left(c^n_{\phi_k(\mathbf{x},w)}, \mathrm{adj}(\mathbf{x}, \phi_k(\mathbf{x}, w))\right) \mid w \in V(G) \right\}\!\!\right\} \Bigg).$$

We prove that $M^n_{\delta,\bar{\delta}}(\mathbf{x}) = M^n_{\delta,\bar{\delta}}(\mathbf{y})$ by showing that for any $j \in [1, \ldots, k]$ it holds that

$$\begin{aligned} &\left\{\!\!\left\{ \left(c^n_{\phi_j(\mathbf{x},w)}, \mathrm{adj}(\mathbf{x}, \phi_j(\mathbf{x}, w))\right) \mid w \in V(G) \right\}\!\!\right\} \\ =&\left\{\!\!\left\{ \left(c^n_{\phi_j(\mathbf{y},w)}, \mathrm{adj}(\mathbf{y}, \phi_j(\mathbf{y}, w))\right) \mid w \in V(H) \right\}\!\!\right\}. \end{aligned} \tag{1}$$

As $\pi_x = \pi_y$ we know that there exists a a bijective function $\alpha : \mathcal{N}_{G_{\delta,k}}(x) \rightarrow \mathcal{N}_{H_{\delta,k}}(y)$ such that for any neighbor p of x it holds that $\pi_p = \pi_{\alpha(p)}$. Recall that all neighbors of x and y are l and g vertices, and that the features of l and g vertices encode their type and the j of the j-neighborhood that connects the tuples. As WL is a colour refinement algorithm it follows that α only maps l to l vertices and g to g vertices. Additionally, α only maps one vertex to another

if they have the same j. Due to the colouring being stable and x and y having the same colour it follows that

$$\begin{aligned}&\{\!\{ \left(\pi_{\phi_j(x,w)}, \mathrm{adj}(x, \phi_j(x,w))\right) \mid w \in V(G) \}\!\}\\ =&\{\!\{ \left(\pi_{\alpha(\phi_j(x,w))}, \mathrm{adj}(x, \alpha(\phi_j(x,w)))\right) \mid w \in V(G) \}\!\}\\ =&\{\!\{ \left(\pi_{\phi_j(y,w)}, \mathrm{adj}(y, \phi_j(y,w))\right) \mid w \in V(H) \}\!\}.\end{aligned}$$

By applying the induction hypothesis we obtain that Equation 1 holds. This shows that the induction step is true and concludes the proof of Theorem 3. □

As δ-k-WL is equally expressive as δ-k-LWL$^+$ and at least as expressive as k-WL and δ-k-LWL, this result also generalizes to those algorithms. However, the big advantages of δ-k-LWL and δ-k-LWL$^+$ over δ-k-WL is that they only make very restricted use of global j-neighbourhoods. This means that δ-k graph transformation is not an efficient alternative as it uses all global neighbourhoods. However, the graph transformation can be adapted into *local δ-k graph transformation* which only makes uses of local neighbourhoods.

Definition 8 (Local δ-k Graph Transformation). *Let $G = (V, E)$ be a graph and $k \geq 1$ an integer. Then applying a local δ-k graph transformation to G returns the graph $G_{\delta,k} = (V_{\delta,k}, E_{\delta,k})$ where*

$$\begin{aligned}V_{\delta,k} =&\{x \mid x \in V(G)^k\} \cup\\ &\{l_{xyj} \mid x, y \in V(G)^k, j \in [1, \ldots, k], x \text{ and } y \text{ are local } j\text{-neighbours}\},\end{aligned}$$

$$E_{\delta,k} = \{\{x, l_{xyj}\}, \{y, l_{xyj}\} \mid x, y \in V(G)^k, j \in [1, \ldots, k], x \text{ and } y \text{ are local } j\text{-neighbours}\}.$$

For any vertex x corresponding to a tuple we add a vertex feature that encodes its isomorphism type and a feature that allows us to distinguish it from l vertices. To all l_{xyj} vertices we add a feature that allows us to distinguish it from all vertices corresponding to tuples. Additionally, we add a feature that encodes j.

Theorem 4. *Local δ-k graph transformation together with WL is at least as expressive as δ-k-LWL.*

Similar to δ-k graph transformation, we can avoid using l vertices if we use edge features. Next, we prove Theorem 4.

Proof. Let G and H be two graphs, we need to prove that if WL cannot distinguish $G_{\delta,k}$ and $H_{\delta,k}$ then δ-k-LWL cannot distinguish G and H. We use π_v to denote the colour of vertex v encoding a k-tuple obtained by applying WL to $G_{\delta,k}$ or $H_{\delta,k}$. We use $k^t_{\mathbf{v}}$ to denote the colour of the corresponding k-tuple $\mathbf{v}$ computed in the t-th iteration of δ-k-LWL on G or H. We assume that WL cannot distinguish $G_{\delta,k}, H_{\delta,k}$ and show the stronger statement

$$\forall x \in V(G)^k, y \in V(H)^k : \pi_x = \pi_y \rightarrow c^t_{\mathbf{x}} = c^t_{\mathbf{y}}$$

for all $t \geq 0$ by induction on the iteration t of δ-k-LWL.

Base case: Let x and y be arbitrary k-tuples. The statement follows from the fact that both the initial features of x, y and $c^0_{\mathbf{x}}, c^0_{\mathbf{y}}$ encode the isomorphism class of x and y.

Induction hypothesis: We assume the statement holds for $t = n$.

Induction step: Let x and y be arbitrary k-tuples. We need to show that $\pi_x = \pi_y \rightarrow c^{n+1}_{\mathbf{x}} = c^{n+1}_{\mathbf{y}}$. We assume that $\pi_x = \pi_y$. From the definition of δ-k-LWL it follows that $c^{n+1}_{\mathbf{x}} = (c^n_{\mathbf{x}}, M^n_\delta(\mathbf{x}))$ and $c^{n+1}_{\mathbf{y}} = \left(c^n_{\mathbf{y}}, M^n_\delta(\mathbf{y})\right)$. From the assumption that $\pi_x = \pi_y$ and the induction hypothesis it follows that $c^n_{\mathbf{x}} = c^n_{\mathbf{y}}$. Next, we want to show that $M^n_\delta(\mathbf{x}) = M^n_\delta(\mathbf{y})$. By definition we know that

$$M^n_\delta(\mathbf{x}) = \Bigg(\left\{\!\!\left\{ \left(c^n_{\phi_1(\mathbf{x},w)}\right) \mid w \in \mathcal{N}_G(x_1) \right\}\!\!\right\}, \ldots, \left\{\!\!\left\{ \left(c^n_{\phi_k(\mathbf{x},w)}\right) \mid w \in \mathcal{N}_G(x_k) \right\}\!\!\right\} \Bigg).$$

We prove that $M^n_\delta(\mathbf{x}) = M^n_\delta(\mathbf{y})$ by showing that for any $j \in [1, \ldots, k]$ it holds that

$$\begin{aligned} &\left\{\!\!\left\{ \left(c^n_{\phi_j(\mathbf{x},w)}\right) \mid w \in \mathcal{N}_G(x_j) \right\}\!\!\right\} \\ =&\left\{\!\!\left\{ \left(c^n_{\phi_1(\mathbf{y},w)}\right) \mid w \in \mathcal{N}_H(y_j) \right\}\!\!\right\}. \end{aligned} \tag{2}$$

As $\pi_x = \pi_y$ we know that there exists a a bijective function $\alpha : \mathcal{N}_{G_{\delta,k}}(x) \rightarrow \mathcal{N}_{G_{\delta,k}}$ such that for any neighbor p of x it holds that $\pi_p = \pi_{\alpha(p)}$. Recall that all neighbors of x and y are l vertices. As the feature of an l vertex encodes j and WL is a colour refinement algorithm, we know that α only maps two vertices to another if they have the same j. Due to the colouring being stable and x and y having the same colour it follows that

$$\begin{aligned} &\left\{\!\!\left\{ \left(\pi_{\phi_j(x,w)}\right) \mid w \in \mathcal{N}_G(x_j) \right\}\!\!\right\} \\ =&\left\{\!\!\left\{ \left(\pi_{\alpha(\phi_j(x,w))}\right) \mid w \in \mathcal{N}_G(x_j) \right\}\!\!\right\} \\ =&\left\{\!\!\left\{ \left(\pi_{\phi_j(y,w)}\right) \mid w \in \mathcal{N}_H(y_j) \right\}\!\!\right\}. \end{aligned}$$

By applying the induction hypothesis we obtain that Equation 2 holds. This shows that the induction step is true and concludes the proof of Theorem 4. □

As δ-k-LWL$^+$ makes use of counting global neighbors with the same colour it is not obvious whether it is possible to create a similar algorithm by just combining a graph transformation and WL. This shows one of the limits of our idea: while we are able to create a transformation with the same expressiveness, we are unable to build one that is equally sparse.

Suppose we apply the graph transformation to a graph with N vertices. Then, the δ-k graph transformation creates, $N^k + kN^{k+1}/2$ vertices and kN^{k+1} edges. It thus runs in time $\mathcal{O}(N^{k+1})$. By using edge features, the δ-k graph transformation only creates N^k vertices and kN^{k+1} edges. The local δ-k graph transformation creates less vertices and edges, depending on the sparsity of the

graph. Running a single iteration of WL on a graph created by the (local) δ-k graph transformations is computationally more expensive than the corresponding WL variant. This is due to the l and g vertices causing more messages to be passed. However, if one uses edge features l and g are unnecessary and the runtime complexity will be the same as for the corresponding WL variant.

D Experimental Details

The neural network models are implemented in Python with PyTorch [21] and PyTorch Geometric [12]. The code to compute cellular ring encoding is based on [4] and uses graph-tool[4] to compute induced cycles in the graphs. The code to compute subgraph bags is based on [3]. We use Weights & Biases[5] to keep track of the experiments.

All models except CIN were trained on systems with an NVIDIA GeForce RTX 3080 GPUs. CIN was trained on a system with an NVIDIA GeForce GTX TITAN X GPU, and a system with an GTX 1080 TI GPU.

D.1 Type Pooling

In all our new transformation algorithm vertices get assigned additional features that encode the type of a vertex. In cell encoding, the features encode the dimension of the cell that corresponds to the vertex. In subgraph bag encoding, the features encode whether a vertex is a separator vertex, in the original graph, or in a subgraph. Inspired by a pooling operation by Bodnar et al. [4] we propose *type pooling*. The idea is that vertices with different types contain different information and have varying importance. Thus instead of pooling all vertices in a graph, we instead pool all vertices of a type. To get a single output for the entire graph, we then apply a multilayer perceptron with a ReLU activation function to each pooled vector and then sum the results. Formally, let T be the set of types and let H_t with $t \in T$ be the set of all representations of vertices of type t, and MLP_t be a multilayer perceptron with activation function. Then type pooling computes

$$\text{Type-Pool} = \sum_{t \in T} \mathrm{MLP}_t \left(\sum_{h \in H_t} h \right).$$

D.2 Hyperparameters

For all methods we tune the number of layers, dropout probability, and embedding dimension (or hidden layer size for MLP). For GIN+CRE and CIN, we tune the size of the largest cycle to lift. For GIN+CRE and GIN+SBE we also tune whether to use type pooling (see Appendix D.1). Additionally, for GIN+CRE we tune whether to aggregate edge and vertex features to higher dimensional cells. The hyperparameters can be found in Table 4.

[4] `https://graph-tool.skewed.de/`
[5] `https://wandb.ai/`

Table 4. Hyperparameter grids for all models

Parameter	Models	Possible Values
Number of layers	All	[2,3,4,5]
Embedding Dimension / Hidden Layer Size (MLP)	All	[32, 64, 128, 256, 300]
Dropout probability	All	[0, 0.5]
Ring size	GIN+CRE, CIN	[Yes, No]
Type pooling	GIN+CRE, GIN+SBE	[Yes, No]
Aggregate vertex features	GIN+CRE	[Yes, No]
Aggregate edge features	GIN+CRE	[Yes, No]