EDGNN: A SIMPLE AND POWERFUL GNN FOR DIRECTED LABELED GRAPHS

Guillaume Jaume[*]
IBM Research, Zurich
EPFL, Lausanne
gja@zurich.ibm.com

An-phi Nguyen[*]
IBM Research, Zurich
ETH, Zurich
uye@zurich.ibm.com

María Rodríguez Martínez
IBM Research, Zurich
mrm@zurich.ibm.com

Jean-Philippe Thiran
EPFL, Lausanne
jean-philippe.thiran@epfl.ch

Maria Gabrani
IBM Research, Zurich
mga@zurich.ibm.com

ABSTRACT

The ability of a graph neural network (GNN) to leverage both the graph topology and graph labels is fundamental to building discriminative node and graph embeddings. Building on previous work, we theoretically show that edGNN, our model for directed labeled graphs, is as powerful as the Weisfeiler–Lehman algorithm for graph isomorphism. Our experiments support our theoretical findings, confirming that graph neural networks can be used effectively for inference problems on directed graphs with both node and edge labels. Code available at https://github.com/guillaumejaume/edGNN.

1 INTRODUCTION

In recent years, much work has been devoted to extending deep-learning models to graphs, *e.g.*, Scarselli et al. (2009); Bruna et al. (2014); Li et al. (2016); Defferrard et al. (2016); Kipf & Welling (2017); Hamilton et al. (2017a;b); Veličković et al. (2017); Ying et al. (2018). Gilmer et al. (2017) formulated numerous such models within their proposed Message Passing Neural Network (MPNN) framework. In this model, the state of a vertex is represented by a feature vector. The state is then updated iteratively via a two-step strategy: For each vertex (i) in the aggregation step, the feature vectors of the neighboring vertices are combined into a single vector via a differentiable operator, *e.g.*, a sum. Then, (ii) in the update step, a new state is computed by applying another differentiable operator, *e.g.*, a one-layer perceptron, to the current state of the vertex and to the aggregate vector from the previous step.

In two recent works, Xu et al. (2019) and Morris et al. (2018), independently proved that certain formulations of MPNNs are *as powerful as* the Weisfeiler–Lehman (WL) algorithm for graph isomorphism (Weisfeiler & Leman (1968)). In practice, this means that there exist MPNNs able to learn *unique* representations for (*almost*) *all* undirected node-labeled graphs, which is desirable for such tasks as node or graph classification. Note that previous work has drawn a parallel between the WL test and the MPNN, *e.g.*, Hamilton et al. (2017a); Jin et al. (2017); Lei et al. (2017).

In this paper, we extend the above-mentioned results to *directed* graphs with labels for both nodes and *edges*. In particular, by extending the theoretical framework provided by Morris et al. (2018), we show that there exist MPNNs as powerful as the (one-dimensional) WL algorithm for directed labeled graphs. Although this problem has already been addressed, *e.g.*, Li et al. (2016); Niepert et al. (2016); Simonovsky & Komodakis (2017); Beck et al. (2018); Schlichtkrull et al. (2018), we present a theoretically-grounded GNN formulation for directed labeled graphs. We experimentally corroborate our theoretical results by comparing our model, edGNN, against state-of-the-art models for node and graph classification.

---

[*]equal contribution

2 THEORETICAL FRAMEWORK

2.1 NOTATION AND SETUP

As our work is an extension of Morris et al. (2018), we maintain a notation consistent to theirs in order to more easily reference their results.

A graph G is a pair (V, E), where V is the set of vertices, E is the set of edges and the directed edge $(u, v) \in E$ for $u, v \in V$ is an edge starting in u and ending in v. We will denote the vertex and the edge sets of G as V_G and E_G, respectively.

We are interested in graphs with both node and edge labels. We therefore assume that, given a graph G, there exists a vertex-labeling function $l_V : V_G \to \mathcal{X}$ and an edge-labeling function $l_E : E_G \to \mathcal{Z}$ that assign to each vertex and edge of G a label from *countable* sets $\mathcal{X}$ and $\mathcal{Z}$. For the rest of this paper, we will refer to graphs with node and edge labels simply as *labeled graphs*.

For each vertex $v \in V$, we can define the neighborhood $\mathcal{N}(v) := \{u \in V \mid (v, u) \in E \lor (u, v) \in E\}$. As we are dealing with directed graphs, we distinguish between incoming neighbors $\mathcal{N}^I(v) := \{u \in V \mid (u, v) \in E\}$ and outgoing neighbors $\mathcal{N}^O(v) := \{u \in V \mid (v, u) \in E\}$. Naturally, $\mathcal{N}(v) = \mathcal{N}^I(v) \cup \mathcal{N}^O(v)$. The cardinalities $|\mathcal{N}(v)|$, $|\mathcal{N}^I(v)|$ and $|\mathcal{N}^O(v)|$ of the neighborhoods are referred to as the *degree*, the *in-degree* and the *out-degree* of vertex v, respectively.

Definition 2.1. *Two labeled directed graphs G and H are* isomorphic *if there exists a bijection* $f : V_G \to V_H$ *such that* $(u, v) \in E_G$ *if and only if* $(f(u), f(v)) \in E_H$ *with* $l_{V_G}(v) = l_{V_H}(f(v))$ *and* $l_{E_G}(u, v) = l_{E_H}(f(u), f(v))$.

2.1.1 THE WEISFEILER–LEHMAN ALGORITHM

The Weisfeiler–Lehman (WL) test (Weisfeiler & Leman (1968)) is an algorithm to distinguish whether two graphs are non-isomorphic. We present the test in its one-dimensional variant, also known as the *naive vertex refinement*. We will start by presenting the WL test on node-labeled graphs, and later discuss its extension to directed labeled graphs.

At initialization, the vertices are labeled consistently with the vertex-labeling function l_V. We call this the *initial coloring* of the graph and we denote it as $c_l^{(0)}(v) := l_V(v), \ \forall v \in V$. The algorithm then proceeds in a recursive fashion. At iteration t, new labels are computed for each vertex from the current labels of the vertex itself and its neighbors, *i.e.*,

$$c_l^{(t)}(v) = g\Big(c_l^{(t-1)}(v), \big\{\big\{ c_l^{(t-1)}(u) : u \in \mathcal{N}(v) \big\}\big\} \Big), \tag{1}$$

where g is an injective hashing function, and $\big\{\big\{ \cdot \big\}\big\}$ denotes a *multiset*, *i.e.*, a generalization of a set that allows elements to be repeated. Each iteration is performed in parallel for the two graphs to be tested, G and H. If at some iteration t, the number of vertices assigned to a label $l \in \mathcal{X}$ differs for the two graphs, then the algorithm stops, concluding that the two graphs are not isomorphic. Otherwise, the algorithm will stop whenever a *stable coloring* is achieved, *i.e.*, whenever $c_l^{(t)}(v_G) = c_l^{(t)}(v_H)$ for all $t \geq T$ and for any pair (v_G, v_H) with $v_G \in V_G$, $v_H \in V_H$, and $c_l^{(T)}(v_G) = c_l^{(T)}(v_H)$. This is guaranteed to happen at most after $T = \max\{|V_G|, |V_H|\}$ iterations. In this case, G and H are considered isomorphic.

Despite a few corner cases, the WL test is able to distinguish a wide class of graphs (Cai et al. (1992)). Moreover, the most efficient implementation of the algorithm has a runtime complexity which is quasi-linear in the number of vertices (Grohe et al. (2017)).

The extension of the WL test to a directed graph with edge labels is straightforward (Grohe et al. (2017); Orsini et al. (2016)). During the recursive step, for each vertex v, we need to include the in-degrees and out-degrees of v separately in the hashing function *with respect to each* edge label.

Let us denote an edge label as $e \in \mathcal{Z}$. For each vertex v, we then define $n_v^I(e) := |\{u \in \mathcal{N}^I(v) \mid l_E(u, v) = e\}|$ as the number of edges incoming to v with label e. Similarly, $n_v^O(e)$ is defined for outgoing edges. Then, Eq. (1) can be adapted for labeled directed graphs in the follow-

ing way:

$$
c_l^{(t)}(v) = g\Big(c_l^{(t-1)}(v), \big\{\{c_l^{(t-1)}(u) : u \in \mathcal{N}(v)\}\big\},
$$
$$
\big\{\big(n_v^I(e), e\big) : \exists(u,v) \in E \text{ with } l_E(u,v) = e\big\}, \tag{2}
$$
$$
\big\{\big(n_v^O(e), e\big) : \exists(v,u) \in E \text{ with } l_E(v,u) = e\big\}\Big).
$$

2.2 GRAPH NEURAL NETWORKS

Graph neural networks architectures implement a neighborhood aggregation strategy. Similar to Morris et al. (2018), our work focuses on the GNN model presented by Hamilton et al. (2017a), which implements this strategy with the node update function

$$
f^{(t)}(v) = \sigma\Big(f^{(t-1)}(v)W_1^{(t)} + \sum_{u \in \mathcal{N}(v)} f^{(t-1)}(u)W_2^{(t)}\Big), \tag{3}
$$

where $f^{(t)}(v) \in \mathbb{R}^{1 \times d^{(t)}}$ is the $d^{(t)}$-dimensional node representation, or node embedding, at time step t of the vertex v, and $W_1^{(t)}, W_2^{(t)} \in \mathbb{R}^{d^{(t-1)} \times d^{(t)}}$ are weight matrices. The initial representation $f^{(0)}(v)$ is *consistent* with the vertex-labeling function l_V, *i.e.*, $f^{(0)}(v) = f^{(0)}(u)$ if and only if $l_V(v) = l_V(u)$ for all $v, u \in V$. Morris et al. (2018) showed that there exists a sequence $(W_1^{(t)}, W_2^{(t)})$ such that the GNN model in Eq. (3) is as powerful as the WL test.

2.2.1 EXTENSION TO DIRECTED LABELED GRAPHS

The extension of Eq. (3) to directed labeled graphs follows the WL test extension. We simply need to augment the equation with embeddings for the labeled edges with incoming and outgoing edges considered separately, *i.e.*,

$$
f^{(t)}(v) = \sigma\Big(f^{(t-1)}(v)W_1^{(t)} + \sum_{u \in \mathcal{N}(v)} f^{(t-1)}(u)W_2^{(t)} +
$$
$$
+ \sum_{(u,v) \in E} f_E\big(u,v,l_E(u,v)\big)W_3^{(t)} + \sum_{(v,u) \in E} f_E\big(v,u,l_E(v,u)\big)W_4^{(t)}\Big), \tag{4}
$$

where $f_E\big(v,u,l_E(v,u)\big) \in \mathbb{R}^{1 \times d_E}$ is the d_E-dimensional embedding of the edge (v,u) with label $l_E(v,u)$. The embeddings f_E should be defined such that $\sum_{(u,v) \in E} f_E\big(u,v,l_E(u,v)\big) = \sum_{(u,v') \in E} f_E\big(u,v',l_E(u,v')\big)$ if and only if $\big\{\big(n_v^I(e), e\big) : \exists(u,v) \in E \text{ with } l_E(u,v) = e\big\} = \big\{\big(n_{v'}^I(e), e\big) : \exists(u,v') \in E \text{ with } l_E(u,v') = e\big\}$. The same should hold for outgoing edges. For practical applications, this can be achieved by using one-hot encodings of the edge labels.

We can now directly extend the theorems presented in Morris et al. (2018).

Theorem 2.1 (Theorem 1 in Morris et al. (2018)). *Let G be a directed labeled graph. Then for all $t \geq 0$ and for all choices of initial colorings $f^{(0)}$ consistent with l_V and of edge embeddings f_E consistent with l_E, and weights $W_1^{(t)}, W_2^{(t)}, W_3^{(t)}, W_4^{(t)}$*

$$
c_l^{(t)}(v) = c_l^{(t)}(u) \Rightarrow f^{(t)}(v) = f^{(t)}(u) \quad \forall u,v \in V \tag{5}
$$

with $c_l^{(t)}$ and $f^{(t)}$ defined in Eqs. (2) and (4), respectively.

Morris et al. (2018) prove this theorem by induction. The proof is essentially the same for our extended case. In fact, as neither the labels l_E nor the embeddings f_E change over the iterations, there is no need to include them in the induction step.

Theorem 2.2 (Theorem 2 in Morris et al. (2018)). *Let G be a directed labeled graph with finite vertex degree. Then there exists a sequence $(W_1^{(t)}, W_2^{(t)}, W_3^{(t)}, W_4^{(t)})$ with $t \geq 0$ such that*

$$
c_l^{(t)}(v) = c_l^{(t)}(u) \Leftrightarrow f^{(t)}(v) = f^{(t)}(u) \quad \forall u,v \in V \tag{6}
$$

The proof is provided in the supplemental material. Note that we specifically require the graph to have finite vertex degree. However, this is not a strong assumption in real-world applications.

2.2.2 GRAPH CLASSIFICATION

For graph classification tasks, we need a representation f_G of the graph. We build it from the node representations $f^{(t)}(v)$ following the formulation of Xu et al. (2019):

$$f_G = \text{Concat}\Big(\Big\{ \sum_{v \in V_G} f^{(t)}(v) \ \Big| \ t = 0, \dots, T \Big\}\Big). \tag{7}$$

Note that, although T should theoretically be at least $|V_G|$ (Section 2.1.1), only a few layers (*i.e.*, iterations) are used in practice to update the node representations. Finally, a linear classifier is applied to the graph representation to perform the classification.

3 EXPERIMENTS

3.1 DATASETS AND BASELINES

We benchmark our algorithm edGNN on graph and node classification tasks. For graph classification tasks, we benchmark our model against the Subgraph Matching Kernel (CSM) (Kriege & Mutzel (2012)), Weisfeiler–Lehman Shortest Path Kernel (Shervashidze et al. (2011)) and R-GCN (Schlichtkrull et al. (2018)) . As R-GCN only defines how to build node embeddings, we reuse the formulation of Xu et al. (2019) to build a graph-level representation. For node classification tasks, we compare our model against R-GCN (Schlichtkrull et al. (2018)), RDF2Vec (Ristoski & Paulheim (2016)) and WL (Shervashidze et al. (2011); De Vries & De Rooij (2015)) on the AIFB and MUTAG dataset (Ristoski et al. (2016)). Dataset statistics as well as training details are reported in the Appendix.

3.2 RESULTS AND DISCUSSION

The left-hand table in Table 1 reports results for the graph classification tasks. Our provably powerful model, edGNN, reaches state-of-the-art performance on each dataset showing its discriminative power over other approaches. In the case of graph-kernel-based methods, we conjecture that the lower performances are due to the use of less powerful models, which could lead to their underfitting. Both CSM and WLSP focus on specific subgraph features (*e.g.*, shortest paths), which may not be sufficient to fully characterize a graph in the datasets we analyzed. However, as we have no information about the training accuracies for these models, we cannot confirm this hypothesis. In the case of R-GCNs, as there is no theoretical result proving that they can or *cannot* be equivalent to the WL test, the consistently low performance might merely be because the R-GCN formulation is not appropriate for the particular learning setting of our experiments (*e.g.*, relatively small datasets). For node classification (right-hand table in Table 1), edGNN achieves comparable performance with the state-of-the-art without outperforming it. This does not contradict our theoretical findings. In fact, the power of a learnable model does not guarantee its generalization nor that the best model can be learned. However, it is true that, *in the best-case scenario*, a more powerful model should perform better than a less powerful one, as shown by the results regarding the best-learned edGNN model. An interesting future research direction would be the study of all the proposed models fitting the MPNN framework in order to understand whether they can be as powerful as the WL test or whether, instead, they introduce a particular bias.

Table 1: Graph (left) and node (right) classification results in accuracy averaged over ten runs. Results are expressed as percentages. For graph classification, following prior art, we performed 10-fold cross validation. For node classification, we used the split provided by Ristoski & Paulheim (2016).

Model	MUTAG	PTC FM	PTC FR	PTC MM	PTC MR	Model	AIFB	MUTAG
CSM	85.4	63.8	65.5	63.3	58.1	WL	80.5	**80.9**
WLSP	85.4	60.4	65.7	66.6	65.7	RDF2Vec	88.9	67.2
R-GCN	94.7	82.0	86.9	85.4	84.9	R-GCN	**95.8**	73.2
edGNN (avg)	**98.9**	**86.0**	**87.7**	**86.2**	**85.7**	edGNN (avg)	91.1	80.0
edGNN (max)	99.4	87.6	88.8	87.9	87.6	edGNN (max)	97.2	85.3

REFERENCES

Daniel Beck, Gholamreza Haffari, and Trevor Cohn. Graph-to-Sequence Learning using Gated Graph Neural Networks. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*, pp. 273—-283, 2018.

Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann Lecun. Spectral Networks and Deep Locally Connected Networks on Graphs. In *International Conference on Learning Representations (ICLR)*, 2014.

Jin Yi Cai, Martin Fürer, and Neil Immerman. An optimal lower bound on the number of variables for graph identification. *Combinatorica*, 1992.

Gerben Klaas Dirk De Vries and Steven De Rooij. Substructure counting graph kernels for machine learning from RDF data. *Journal of Web Semantics*, 2015.

Asim Kumar Debnath, Rosa L. Lopez de Compadre, Gargi Debnath, Alan J. Shusterman, and Corwin Hansch. Structure-Activity Relationship of Mutagenic Aromatic and Heteroaromatic Nitro Componds. Correlation with Modelcular Orbital Energies and Hydrophobicity. *J. Med. Chem*, 34:786–797, 1991.

Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering. In *Advances in neural information processing systems (NIPS)*, 2016.

Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural Message Passing for Quantum Chemistry. In *International Conference on Machine Learning (ICML)*, pp. 1273–1272, 2017.

Martin Grohe, Kristian Kersting, Martin Mladenov, and Pascal Schweitzer. Color Refinement and its Applications. 2017.

William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive Representation Learning on Large Graphs. In *Advances in neural information processing systems (NIPS)*, 2017a.

William L. Hamilton, Rex Ying, and Jure Leskovec. Representation Learning on Graphs: Methods and Applications. In *IEEE Data Engineering Bulletin*, 2017b. ISBN 1476-4687.

C Helma, R D King, S Kramer, and A Srinivasan. The Predictive Toxicology Challenge 2000-2001. *Bioinformatics (Oxford, England)*, 19(1):1179–82, 2003.

Wengong Jin, Connor W. Coley, Regina Barzilay, and Tommi Jaakkola. Predicting Organic Reaction Outcomes with Weisfeiler-Lehman Network. In *Advances in neural information processing systems (NIPS)*, 2017.

Thomas N Kipf and Max Welling. Semi supervised classification with graph convolutional networks. In *International Conference on Learning Representations*, 2017.

Nils Kriege and Petra Mutzel. Subgraph Matching Kernels for Attributed Graphs. In *International Conference on Machine Learning (ICML)*, 2012.

Tao Lei, Wengong Jin, Regina Barzilay, and Tommi Jaakkola. Deriving Neural Architectures from Sequence and Graph Kernels. *International Conference on Machine Learning (ICML)*, 2017.

Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. Gated Graph Sequence Neural Networks. In *International Conference on Learning Representations (ICLR)*, 2016.

Christopher Morris, Martin Ritzert, Matthias Fey, William L. Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and Leman Go Neural: Higher-order Graph Neural Networks. In *33rd AAAI Conference on Artificial Intelligence (AAAI)*, 2018.

Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. Learning Convolutional Neural Networks for Graphs. In *International Conference on Machine Learning (ICML)*, pp. 2014–2023, 2016.

Francesco Orsini, Paolo Frasconi, and Luc De Raedt. Graph invariant kernels. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2016.

E. M. Patterson and Seth Warner. *Modern Algebra (Prentice-Hall, Inc., 1965), two volumes, 806 pp.*, volume 15. Cambridge University Press, dec 1967. URL `http://www.journals.cambridge.org/abstract{_}S0013091500012098`.

Petar Ristoski and Heiko Paulheim. RDF2Vec: RDF graph embeddings for data mining. In *Lecture Notes in Computer Science*, 2016.

Petar Ristoski, Gerben Klaas Dirk De Vries, and Heiko Paulheim. A collection of benchmark datasets for systematic evaluations of machine learning on the semantic web. In *Lecture Notes in Computer Science*, 2016.

Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 2009.

Michael Schlichtkrull, Thomas N. Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov, and Max Welling. Modeling Relational Data with Graph Convolutional Networks. In *Extended Semantic Web Conference*, 2018.

Nino Shervashidze, Pascal Schweitzer, Erik Jan van Leeuwen, Kurt Mehlhorn, and Karsten M Borgwardt. Weisfeiler-Lehman Graph Kernels. *Journal of Machine Learning Research*, 12:2539–2561, 2011.

Martin Simonovsky and Nikos Komodakis. Dynamic edge-conditioned filters in convolutional neural networks on graphs. In *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph Attention Networks. *International Conference on Learning Representations (ICLR)*, 2017.

B Yu Weisfeiler and A A Leman. A reduction of a graph to a canonical form and an algebra arising during this reduction. In *Nauchno-Technicheskaya Informatsia , 2(9)*, pp. 2–16, 1968.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How Powerful are Graph Neural Networks? In *International Conference on Learning Representations (ICLR)*, 2019.

Rex Ying, Jiaxuan You, Christopher Morris, Xiang Ren, William L. Hamilton, and Jure Leskovec. Hierarchical Graph Representation Learning with Differentiable Pooling. In *Neural Information Processing Systems (NIPS)*, 2018.

4 APPENDIX

4.1 PROOF OF THEOREM 2.2

For a given vertex v, we define $E_v^I := \{l_E(u, v) : \exists (u, v) \in E\}$ as the set of labels of edges incoming into the vertex v. We then define $L_v^I := \text{Concat}\left(\{\left(n_v^I(e), e\right) : \forall e \in E_v^I\}\right)$. That is, for each vertex v, we create a label by concatenating all the labels of the incoming edges together with their multiplicities $n_v^I(e)$. Similarly, we define E_v^O and L_v^O for the outgoing edges. Note that the pairs $\left(n_v^I(e), e\right)$ and $\left(n_v^O(e), e\right)$ take values in $\mathcal{L} := \mathbb{N} \times \mathcal{Z}$. Therefore, L_v^I (or L_v^O, respectively) can take values in $\mathcal{L}^{|E_v^I|}$ (or $\mathcal{L}^{|E_v^O|}$, respectively), *i.e.*, the Cartesian product of $|E_v^I|$ (or $|E_v^O|$, respectively) copies of the set $\mathcal{L}$.

For all vertices v, we can construct a function $h_v : \mathcal{X} \times \mathcal{L}^{|E_v^I|} \times \mathcal{L}^{|E_v^O|} \to \mathcal{Y}_v$ that *bijectively* maps a tuple $\left(c_l^{(t-1)}(v), L_v^I, L_v^O\right)$ to a label $y \in \mathcal{Y}_v$.

Note that, as we are considering graphs with finite vertex degree, $|\mathcal{N}^I(v)|$ and $|\mathcal{N}^O(v)|$ (and, consequently, $|E_v^I|$ and $|E_v^O|$) are finite. Therefore, $\mathcal{X} \times \mathcal{L}^{|E_v^I|} \times \mathcal{L}^{|E_v^O|}$ is a *countable* set because the finite Cartesian product of countable sets is itself countable. Thus, as we built the function h_v to be bijective, the sets $\mathcal{Y}_v$, and their countable union $\mathcal{Y} := \bigcup_{v \in V} \mathcal{Y}_v$, are also countable (for results on countable sets, refer for example to Patterson & Warner (1967)).

We can then construct an injective hash function g' such that

$$g'\left(y, \{c_l^{(t-1)}(u) : u \in \mathcal{N}(v)\}\right) = g\left(c_l^{(t-1)}(v), \{c_l^{(t-1)}(u) : u \in \mathcal{N}(v)\}, L_v^I, L_v^O\right). \tag{8}$$

where the right-hand side is the relabeling function defined in Eq. (2).

These constructions highlight the fact that an iteration of the WL algorithm on a directed labeled graph is the same as performing an iteration of the WL algorithm on a undirected node-only-labeled graph, where node labels take values in an appropriately augmented label set $\mathcal{Y}$.

The same equivalence can be highlighted between the GNN update functions in Eqs. (3) and (4). In fact, Eq. (4) can be rewritten as

$$f^{(t)}(v) = \sigma\left(f_{\mathcal{Y}}^{(t-1)}(v)W_{1,3,4}^{(t)} + \sum_{u \in \mathcal{N}(v)} f^{(t-1)}(u)W_2^{(t)}\right), \tag{9}$$

where $f_{\mathcal{Y}}^{(t)}(v) \in \mathbb{R}^{1 \times (d^{(t)} + 2d_E)}$ is the embedding resulting from the (horizontal) concatenation of $f^{(t)}(v)$, $\sum_{(u,v) \in E} f_E\left(u, v, l_E(u, v)\right)$, and $\sum_{(v,u) \in E} f_E\left(v, u, l_E(v, u)\right)$, whereas $W_{1,3,4}^{(t)}$ is the (vertical) concatenation of $W_1^{(t)}$, $W_3^{(t)}$, and $W_4^{(t)}$.

The reformulations presented in Eqs. (8) and (9) allow us to treat our problem as one of undirected graphs with labels only for nodes. We can therefore prove this theorem by directly using the proof of Theorem 2 in Morris et al. (2018). $\square$

4.2 DETAILS OF TRAINING AND MORE RESULTS

4.3 INITIALIZATION

We initialize the node and edge features with a one-hot encoding of their input label. For node classification, we use the in-degree as input label of the nodes. We also report results with learned embeddings (emb) instead of a one-hot encoding. To model the outgoing edges of the node classification graphs, we create new artificial relations by reversing each directed edge. We also report results without reversing the edges (reg).

4.3.1 GRAPH CLASSIFICATION

The tasks predict the mutagenicity (Debnath et al. (1991); Kriege & Mutzel (2012)) (MUTAG) and the toxicity (Helma et al. (2003)) of chemical compounds (PTC).

We ran our graph classification experiments with a batch size of 32 and a learning rate of 10^{-3} with a weight decay of 5×10^{-4}. We then performed a parameter search over the number of layers and node embedding size. The best performance was reached by using 3 layers and 64 hidden units. When the system was trained with learned embeddings, the initial embedding size was set to the number of hidden units (*i.e.*, 64). We used a ReLu activation at each layer without dropout. The system was trained for at most 40 epochs with early stopping with respect to the validation set cross-entropy loss.

R-GCN for graph classification was also trained with a batch size of 32, a learning rate of 10^{-3} with a weight decay of 5×10^{-4}. We used 3 layers and 64 hidden units with learned initial embeddings for the nodes (instead of a one-hot encoding for edGNN). We used a basis decomposition with the number of basis set to the number of relations. Results for CSM (Kriege & Mutzel (2012)) and WLSP (Shervashidze et al. (2011)) are based on the re-implementation of Kriege & Mutzel (2012). All our experiments were performed with 10-fold cross validation as in Kriege & Mutzel (2012).

Dataset statistics for the graph classification tasks are shown in Table 2, whereas Table 3 shows the accuracy results together with the standard deviation.

Table 2: Graph classification dataset statistics with the number of graphs (Graphs), the number of classes (Classes), the average number of nodes per graph (Avg nodes), the average number of edges per graph (Avg edges), the number of node labels (Node labels) and the number of edge labels (Edge labels).

Dataset	Graphs	Classes	Avg nodes	Avg edges	Node labels	Edge labels
MUTAG	188	2	17.9	19.8	6	3
PTC FM	349	2	14.1	14.5	18	4
PTC FR	351	2	14.6	15.0	19	4
PTC MM	336	2	14.0	14.3	20	4
PTC MR	344	2	14.3	14.7	18	4

Table 3: Graph classification results in accuracy obtained with 10-fold cross validation. Results are expressed as percentages. edGNN is compared with the Subgraph Matching Kernel (CSM) (Kriege & Mutzel (2012)), Weisfeiler–Lehman Shortest Path Kernel (Shervashidze et al. (2011)) and R-GCN (Schlichtkrull et al. (2018)).

Model	MUTAG	PTC FM	PTC FR	PTC MM	PTC MR
CSM	85.4 ± 1.2	63.8 ± 1.0	65.5 ± 1.4	63.3 ± 1.7	58.1 ± 1.6
WLSP	85.4 ± 1.2	60.4 ± 1.32	65.7 ± 1.3	66.6 ± 1.16	65.7 ± 1.3
R-GCN	94.7 ± 0.7	82.0 ± 2.2	86.9 ± 1.7	85.4 ± 2.6	84.9 ± 1.7
edGNN (avg)	$\mathbf{98.9 \pm 0.4}$	$\mathbf{86.0 \pm 0.9}$	$\mathbf{87.7 \pm 0.9}$	$\mathbf{86.2 \pm 1.0}$	$\mathbf{85.7 \pm 1.2}$
edGNN (max)	$\mathbf{99.4}$	$\mathbf{87.6}$	$\mathbf{88.8}$	$\mathbf{87.9}$	$\mathbf{87.6}$
edGNN (emb)	90.8 ± 1.8	70.8 ± 2.7	77.7 ± 1.5	76.2 ± 3.5	72.2 ± 2.7

4.3.2 NODE CLASSIFICATION

The node classification experiments were run with a learning rate of 5×10^{-3} without weight decay. We used dropout 0.5 on each layer with a ReLu activation. The best performance was achieved by using 2 layers and 64 hidden units. The maximum number of epochs was set to 400 with early stopping with respect to the validation set cross-entropy loss. Results with R-GCN (Schlichtkrull et al. (2018)), RDF2Vec (Ristoski & Paulheim (2016)) and WL (Shervashidze et al. (2011)) are based on the re-implementation of Schlichtkrull et al. (2018).

Table 4: Node classification dataset statistics with the number of classes (Classes), the total number of nodes (Nodes), the total number of edges (Edges) and the number of edge labels (Edge labels).

Dataset	Classes	Nodes	Edges	Edge labels
AIFB	4	8,285	29,043	45
MUTAG	2	23,644	74,227	23

Table 5: Node classification results in accuracy averaged over ten runs. Results are expressed as percentages. edGNN is compared with WL (De Vries & De Rooij (2015)), RDF2Vec (Ristoski & Paulheim (2016)) and R-GCN (Schlichtkrull et al. (2018)).

Model	AIFB	MUTAG
WL	80.5 ± 0.0	$\mathbf{80.9 \pm 0.0}$
RDF2Vec	88.9 ± 0.0	67.2 ± 1.2
R-GCN	$\mathbf{95.8 \pm 0.6}$	73.2 ± 0.5
edGNN (avg)	91.1 ± 2.4	80.0 ± 3.2
edGNN (max)	$\mathbf{97.2}$	$\mathbf{85.3}$
edGNN (emb)	91.1 ± 1.7	77.2 ± 2.6
edGNN (reg)	89.4 ± 1.7	80.4 ± 3.4