data

Article

The Impact of Global Structural Information in Graph Neural Networks Applications

Davide Buffelli * and **Fabio Vandin**

Department of Information Engineering, University of Padova, 35131 Padova, Italy; fabio.vandin@unipd.it
* Correspondence: davide.buffelli@unipd.it

Abstract: Graph Neural Networks (GNNs) rely on the graph structure to define an aggregation strategy where each node updates its representation by combining information from its neighbours. A known limitation of GNNs is that, as the number of layers increases, information gets *smoothed* and *squashed* and node embeddings become indistinguishable, negatively affecting performance. Therefore, practical GNN models employ few layers and only leverage the graph structure in terms of limited, small neighbourhoods around each node. Inevitably, practical GNNs do not capture information depending on the global structure of the graph. While there have been several works studying the limitations and expressivity of GNNs, the question of whether practical applications on graph structured data require global structural knowledge or not remains unanswered. In this work, we empirically address this question by giving access to *global* information to several GNN models, and observing the impact it has on downstream performance. Our results show that global information can in fact provide significant benefits for common graph-related tasks. We further identify a novel regularization strategy that leads to an average accuracy improvement of more than 5% on all considered tasks.

Keywords: graph neural networks; graph representation learning; deep learning; representation learning; graphs

check for updates

Citation: Buffelli, D.; Vandin, F. The Impact of Global Structural Information in Graph Neural Networks Applications. *Data* **2022**, *7*, 10. https://doi.org/10.3390/data7010010

Academic Editor: Giuseppe Ciaburro

Received: 29 November 2021
Accepted: 10 January 2022
Published: 13 January 2022

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Copyright: © 2022 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

Graph Neural Networks (GNNs) [1] are deep learning models for graph structured data, which achieve state-of-the-art results for many graph-related tasks. Most popular GNNs fall into the message-passing framework [2] and are denoted as Message Passing Neural Networks (MPNNs). (In this paper, we use the terms GNN and MPNN interchangeably.) MPNNs have become increasingly popular thanks to their simplicity, extensibility, and empirical effectiveness. MPNNs adopt a message passing mechanism where, at each layer, every node receives a message from its 1-hop neighbours. The incoming messages for each node are aggregated in a permutation-invariant fashion and used to update the node's representation by the means of a learnable function (usually implemented with a neural network). The final node representations (also referred to as *node embeddings*) are then used to perform some graph-related downstream task, for example graph classification or node classification. Empirically, the best results are obtained when the message passing procedure is repeated a relatively small number of times (typical numbers are 2 to 5), as a higher number of layers leads to over-smoothing [3] and over-squashing [4]. Thus, *practical* GNNs are only leveraging the graph structure in the form of small neighbourhoods around each node. A direct consequence of this limitation is that GNNs are not capable of accessing, or extracting, information that depends on the whole structure of the graph (e.g., random walk probabilities [5]).

In this work, we are interested in studying the consequences of the over-smoothing and over-squashing issues. In more detail, we are interested in understanding whether global information (i.e., information that depends on the *whole* structure of the graph, and that cannot be recovered by just focusing on local neighbourhoods) is important for GNNs and their practical applications.

In fact, there is an ongoing debate in the GNN research community on whether it is needed to have "deep" GNNs [6], or if most tasks of interest only require access to local neighbourhoods. We tackle this question directly at its root, and address the overlooked aspect of whether *global* structural information is useful for GNN models, by studying if *global* structural information is important in practical scenarios. In more detail, we introduce three different ways to provide GNN models with *global* structural information, and study how they affect the performance of state-of-the-art MPNNs on common graph related tasks. The three strategies to include *global* structural information we consider are: (i) providing the model direct access to the adjacency matrix, (ii) providing the model direct access to random walks with restart coefficients, and (iii) combining (ii) with a regularization term which enforces the role of the information extracted by random walks with restart. These methods are introduced to study the impact of global information, and are not meant to be used as practical strategies to improve the performance of GNNs. On the latter aspect, we show that the sole use of our regularization term provides significant gains in performance while being easily and efficiently applicable to any GNN model. The use of random walks with restart is also supported by a theoretical contribution which proves they can increase the ability of GNNs in distinguishing non-isomorphic graphs.

Our Contribution

Previous studies on the capabilities and limitations of GNNs have focused on the relation between GNNs and the Weisfeiler–Leman (WL) algorithm [7] to study the *theoretical* expressiveness of these models (e.g., [8]), or on how to alleviate the over-smoothing and over-squashing issues (e.g., [3,4,9]). There are, however, no empirical studies on the practical impact of *global* information (i.e., information that depends on the *whole* structure of the graph) in MPNNs.

We assess whether providing *global* information regarding the whole graph structure has a significant impact on the performance of state-of-the-art MPNNs. In this regard, our contributions are threefold.

- We propose and formalize three different types of *global* structural information "injection". We test how the injection of *global* structural information impacts the performance of six GNN architectures (GCN [10], Graphsage [11], and GAT [12] for node-level tasks; GCN with global readout, DiffPool [13] and k-GNN [8] for graph-level tasks) on both transductive and inductive tasks. Results show that the injection of *global* structural information significantly impacts current state-of-the-art models on common graph-related tasks.
- As we discuss later in the paper, injecting *global* structural information can be impractical. We then identify a novel and practical regularization strategy, called RWRReg, based on random walks with restart [14]. RWRReg maintains the permutation-invariance of GNN models, and leads to an average 5% increase in accuracy on both node classification and graph classification.
- We introduce a theoretical result proving that the information extracted by random walks with restart can "speed up" the 1-Weisfeiler–Leman (1-WL) algorithm [7]. In more detail, we show that, by constructing an initial coloring based on random walks with restart probabilities, the 1-WL algorithm always terminates in one iteration. Given the known relationship between GNNs and the 1-WL algorithm, this result shows that providing information obtained from random walks with restart to GNN models can improve their *practical* ability of distinguishing non-isomorphic graphs.

2. Preliminaries

In this section, we introduce the notation we use throughout the paper, and provide a brief introduction to GNNs and random walks with restart (RWR; also known as Personalized PageRank [14]).

2.1. Notation

We use uppercase bold letters for matrices ($\boldsymbol{M}$), and lowercase bold letters for vectors ($\boldsymbol{v}$). We use plain letters with subscript indices to refer to a specific element of a matrix ($M_{i,j}$), or of a vector (v_i). We refer to the vector containing the i-th row of a matrix with the subscript "$i,:$" ($\boldsymbol{M}_{i,:}$), while we refer to the i-th column with the subscript "$:,i$" ($\boldsymbol{M}_{:,i}$).

A graph $\mathcal{G} = (\mathcal{V}, E)$, where $\mathcal{V} = \{1, .., n\}$ is the set of nodes and $E \subseteq \mathcal{V} \times \mathcal{V}$ is the set of edges, is represented by a tuple $(\boldsymbol{X}, \boldsymbol{A})$. $\boldsymbol{X}$ is an $n \times d$ matrix, where the i-th row contains the d-dimensional feature vector of the i-th node, and $\boldsymbol{A}$ is the $n \times n$ adjacency matrix. For the sake of clarity, we restrict our presentation to undirected graphs, but similar concepts can be applied to directed graphs.

2.2. Graph Neural Networks

In graph representation learning, the goal is to learn a vector representation (also referred to as *node embedding*) for each node that can then be used to effectively perform downstream tasks. The message-passing framework [2], to which most GNNs belong, is based on the following procedure: each node receives messages from its neighbours, *aggregates* them, and *updates* its representation based on the aggregated messages and its previous representation. For a node v, with neighbours $\mathcal{N}_v$, we can represent the operations at the ℓ-th layer of message-passing as follows:

$$\begin{aligned}\mathbf{m}^{(v,\ell)} &= \text{AGGREGATE}(\{\mathbf{H}_u^{(\ell)} \; \forall u \in \mathcal{N}_v\}) \\ \mathbf{H}_v^{(\ell+1)} &= \text{UPDATE}(\mathbf{H}_v^{(\ell)}, \mathbf{m}^{(v,\ell)})\end{aligned}$$

where $\mathbf{H}^{(\ell)}$ is a matrix where the i-th row contains the representation of node i at layer ℓ, AGGREGATE is a permutation invariant function (e.g., average or sum) that takes as input the set of representations of the neighbours and aggregates them into a message $\mathbf{m}$, and UPDATE is usually a learnable function implemented with a neural network. The initial representation (at layer 0) is defined as $\mathbf{H}^{(0)} = \mathbf{X}$. As such, after k message-passing iterations, the representation of a node v depends on its k-hop neighbourhood (i.e., all the nodes at a distance of at most k from v). The GNNs proposed in literature differ on how they implement the AGGREGATE and UPDATE functions [1,10–12].

2.3. Random Walks with Restart

A RWR [14] for node i returns a vector $\boldsymbol{r}^{(i)}$ of size n which satisfies the following equation:

$$\boldsymbol{r}^{(i)} = (1-c)\boldsymbol{W}\boldsymbol{r}^{(i)} + c\boldsymbol{e}^{(i)}$$

where $\boldsymbol{e}^{(i)}$ is a vector where the i-th element is 1 and all the others are 0, c is the restart probability, and $\boldsymbol{W}$ is the transition matrix of the random walk. The restart probability c defines the probability that the walk "jumps" back to the starting node (a common value for c, used in many libraries, is 0.15). The RWR vector can be computed using the power iteration method, and over the year a large number of methods have been developed for its efficient and practical computation, or approximation, even for large scale graphs (e.g., [15,16]). Elements of $\boldsymbol{r}^{(i)}$ capture the relative relationships between nodes [16], and the RWR vectors capture the global structure of the graph [17,18].

3. Random Walks with Restart and the Weisfeiler–Leman Algorithm

We provide analytical evidence that RWR can significantly empower MPNNs by proving a connection with the 1-Weisfeiler–Leman (1-WL) algorithm [7].

The 1-WL algorithm is a well known method for testing the isomorphism of two graphs. The 1-WL algorithm uses an iterative coloring, or relabeling, scheme, in which all nodes are initially assigned the same label (e.g., the value 1). It then iteratively refines the color of each node by aggregating the multiset of colors in its neighborhood with the use of a hash function. At every iteration, the feature representation of a graph is the histogram of resulting node colors. If, at a certain iteration of this process, two graphs have a different feature representation, then the two graphs are not isomorphic. (For a more detailed description of the 1-WL algorithm, we refer the reader to [7,19].)

It is known that not all non-isomorphic graphs are distinguishable by the 1-WL algorithm, and that n iterations are enough to distinguish two graphs of n vertices which are distinguishable by the 1-WL algorithm. There is a tight connection between 1-WL and MPNNs [10,20]. In particular, graphs that can be distinguished in k iterations by the 1-WL algorithm can be distinguished by *certain* GNNs in k message passing iterations [8]. This implies that, when using a GNN that can theoretically achieve the distinguishing power of the 1-WL algorithm, if such GNN is deployed with k' layers, it will not be able to distinguish graphs that are distinguishable by the 1-WL algorithm with $k'' > k'$ iterations.

Here, we prove that graphs that are distinguishable by 1-WL in k iterations have different feature representations extracted by RWR of length k, and hence if we use the RWR feature representations as initial coloring for the 1-WL algorithm, then the algorithm will always finish in one iteration. Given a graph $G = (V, E)$, we define its *k-step RWR representation* as the set of vectors $\mathbf{r}_v = [r_{v,u_1}, \ldots, r_{v,u_n}]$, $v \in V$, where each entry $r_{v,u}$ is the probability that a RWR of length k starting in v ends in $u \in V$.

Proposition 1. *Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be two non-isomorphic graphs for which the 1-WL algorithm terminates with the correct answer after k iterations and starting from the labelling of all 1's. Then, the k-step RWR representations of G_1 and G_2 are different.*

The proof can be found in Appendix A. Since k iterations of the 1-WL algorithm are performed by MPNNs of depth k, but, in practice, MPNNs are limited to small depths; this result shows that RWR can empower MPNNs with relevant information that is discarded in practice.

We further provide an empirical analysis of RWR and their capability of encapsulating global information in Appendix F.

4. Injecting Global Information in MPNNs

To test if MPNNs are missing on important information that is encoded in the structure of a graph, we inject global structural information into existing MPNN models, and test how the performance of these models changes in several graph-related tasks. Intuitively, based on a model's performance when injected with different types of global structural information, we can understand if this additional knowledge can improve performance on the considered tasks. In the rest of this section, we present the types of global structural information injection that we consider, and the models chosen for our experimental evaluation.

4.1. Types of Global Structural Information Injection

We consider three different types of global structural information injection described below. The injection strategies presented in this section are not designed for *practical* use, as the scope of these strategies is to help us understand the importance of global structural information. At this point, our objective is to study the impact of global structural information that is not accessible to GNN models. We discuss scalability and practical aspects in Section 6.

Adjacency Matrix. We provide GNNs with direct access to the adjacency matrix by concatenating each node's adjacency matrix row to its feature vector. This explicitly empowers the GNN model with the connectivity of each node, and allows for higher level structural reasoning when considering a neighbourhood (the model will have access to the connectivity of the whole neighbourhood when aggregating messages from neighbouring nodes). In more detail, the row of the adjacency matrix for a specific node pinpoints the position of the node in the graph (i.e., it acts as a kind of positional encoding), and during the message passing procedure, when a node aggregates information from its neighbours, it allows the network to get a more precise positioning of the node in the graph.

Random Walk with Restart (RWR) Matrix. We perform RWR [14] from each node v, thus obtaining a n-dimensional vector that gives a score of how much v is "related" to every other node in the graph. For every node, we concatenate its vector of RWR coefficients to its feature vector. The choice of RWR is motivated by their capability to capture the relevance between two nodes [16] and the global structure of a graph [17,18], and by the possibility to modulate the exploration of long-range dependencies by changing the restart probability. Intuitively, if a RWR starting at node v is very likely to visit a node u (e.g., there are multiple paths that connect the two), then there will be a high score in the RWR vector for v at position u. This gives the GNN model higher level information about the global structure of the graph, and, again, it allows for high level reasoning on neighbourhood connectivity.

RWR Matrix + RWR Regularization. Together with the addition of the RWR score vector to the feature vector of each node, we also introduce a regularization term based on RWR that pushes nodes with mutually high RWR scores to have embeddings that are close to each other (independently of how far they are in the graph). Let $\boldsymbol{S}$ be the $n \times n$ matrix with the RWR scores. We define the RWRReg (Random Walk with Restart Regularization) loss as follows:

$$\mathcal{L}_{RWRReg} = \sum_{i,j \in V} S_{i,j} ||\boldsymbol{H}_{i,:} - \boldsymbol{H}_{j,:}||^2$$

where $\boldsymbol{H}$ is a matrix of size $n \times d$ containing d-dimensional node embeddings that are in between message-passing layers (see Appendix B for the exact point in which $\boldsymbol{H}$ is considered for each model). With this approach, the loss function used to train the model becomes: $\mathcal{L} = \mathcal{L}_{original} + \lambda \mathcal{L}_{RWRReg}$, where $\mathcal{L}_{original}$ is the original loss function for each model, and λ is a balancing term. In Appendix E, we show how to compute the RWRReg term efficiently using GPUs. We expect this type of information injection to have the highest impact on performance of the models on downstream tasks.

4.2. Choice of Models

In order to test the effect of the different types of global structural information injection and to obtain results that are indicative of the whole class of MPNNs models, we conceptually identify four different categories of MPNNs from which we select representative models.

Simple Aggregation Models

Such models utilize a "simple" aggregation strategy, where each node receives messages (e.g., feature vectors) from its neighbours, aggregates them by assigning the same "importance" to each neighbour (e.g., by averaging their messages), and uses the aggregated messages to update its embedding vector. As a representative, we choose GCN [10], one of the fundamental and widely used GNNs models. We also consider GraphSage [11], as it represents a different computation strategy where a set of neighborhood aggregation functions are learned, and a sampling approach is used for defining fixed size neighbourhoods.

Attention Models

Several models have used an attention mechanism in a GNN scenario [12,21–23]. These methods differ from the previous category as they use an attention mechanism to assign a different "weight", or "importance", to each neighbour. As a representative, we focus on GAT [12], the first to present an attention mechanism over nodes for the aggregation phase, and one of the best performing models on several datasets. Furthermore, it can be used in an inductive scenario.

Pooling Techniques

Pooling on graphs is a very challenging task, since it has to take into account the underlying graph structure. At a high level, pooling methods provide a coarsened version of the input graph by combining groups of nodes into clusters. Among the methods that have been proposed for differentiable pooling on graphs [13,24–27], we choose DiffPool [13] for its strong empirical results. Furthermore, it can learn to dynamically adjust the number of clusters (the number is a hyperparameter, but the network can learn to use fewer clusters if necessary).

Beyond WL

Morris et al. [8] prove that message-passing GNNs cannot be more powerful than the 1-WL algorithm, and propose k-GNNs, which rely on a *subgraph message-passing* mechanism and are proven to be as powerful as the k-WL algorithm. Another approach that goes beyond the WL algorithm was proposed by Murphy et al. [28]. Both models are computationally intractable in their initial theoretical formulation, so approximations are needed. As a representative, we choose k-GNNs, to test if subgraph message-passing is affected by additional global structural information.

5. Evaluation of the Injection of Global Structural Information

We now present our framework for evaluating the effects of the injection of global structural information into GNNs and the results of our experiments. Code for our method can be found at: https://github.com/DavideBuffelli/RWRReg, 9 January 2022. We consider one *transductive* task (node classification) and two *inductive* tasks (graph classification, and triangle counting). We use each architecture for the task that better suits its design: GCN, GraphSage, and GAT for node classification, and DiffPool and k-GNN for graph classification. We add an adapted version of GCN for graph classification, as a common strategy for this task is to deploy a node-level GNN, and then apply a *readout* function to combine node embeddings into a global graph embedding vector.

With regard to datasets, for node classification, we considered the three most used benchmarking datasets in literature: Cora, Citeseer, and Pubmed [29]. Analogously, for graph classification, we chose three frequently used datasets: ENZYMES, PROTEINS, and D&D [30]. Dataset statistics can be found in Appendix C.

For all the considered models, we take the hyperparameters from the implementations released by the authors. The only parameter tuned using the validation set is the balancing term λ when RWRReg is applied. We found that the RWRReg loss tends to be larger than the Cross Entropy loss for prediction, and the best values for λ lie in the range $[10^{-9}, 10^{-6}]$. For all the RWR-based techniques, we used a restart probability of 0.15 (we use 0.15 as it is a common default value used in many papers and software libraries). The effects of different restart probabilities are explored in Section 6.) Detailed information on our implementations can be found in Appendix B.

5.1. Node Classification

For each dataset, we follow the approach that has been widely adopted in literature: we take 20 labeled nodes per class as training set, 500 nodes as validation set, and 1000 nodes for testing. Most authors have used the train/validation/test split defined by [31]. Since we want to test the general effect of the injection of global structural information, we differ from this approach and we do not rely on a single split. We perform 100 runs, where at each run we randomly sample 20 nodes per class for training, 500 random nodes for validation, and 1000 random nodes for testing. We then report mean and standard deviation for the accuracy on the test set over these 100 runs.

Results are summarized in Table 1, where we observe that the simple addition of RWR features to the feature vector of each node is sufficient to give a performance gain (up to 2%). The RWRReg term then significantly increments the gain (up to 7.5%). These results show that, perhaps surprisingly, even for the task of node classification global structural information is important.

Table 1. Node classification accuracy results of different models with added Adjacency matrix features (AD), RWR features (RWR), and RWR features + RWR Regularization (RWR + RWRReg).

Model	Structural Information		Dataset	
		Cora	Pubmed	Citeseer
GCN	none	0.799 ± 0.029	0.776 ± 0.022	0.663 ± 0.095
	AD	0.806 ± 0.035	0.779 ± 0.070	0.653 ± 0.104
	RWR	0.817 ± 0.025	0.782 ± 0.042	0.665 ± 0.098
	RWR + RWRReg	**0.842 ± 0.026**	**0.811 ± 0.037**	**0.690 ± 0.102**
GraphSage	none	0.806 ± 0.017	0.807 ± 0.016	0.681 ± 0.021
	AD	0.803 ± 0.014	0.803 ± 0.013	0.688 ± 0.020
	RWR	0.816 ± 0.014	0.807 ± 0.015	0.693 ± 0.019
	RWR + RWRReg	**0.837 ± 0.015**	**0.820 ± 0.010**	**0.728 ± 0.020**
GAT	none	0.815 ± 0.021	0.804 ± 0.011	0.664 ± 0.008
	AD	0.823 ± 0.019	0.796 ± 0.014	0.672 ± 0.017
	RWR	0.833 ± 0.020	0.811 ± 0.009	0.686 ± 0.009
	RWR + RWRReg	**0.848 ± 0.019**	**0.828 ± 0.010**	**0.701 ± 0.011**

5.2. Graph Classification

Following the approach from [8,13], we use 10-fold cross validation, and report mean and standard deviation of the accuracy on graph classification. Results are summarized in Table 2. The performance gains given by the injection of global structural information are even more apparent than for the node classification task. Intuitively, this is explained by the fact that the global structure of the nodes in a graph is important for distinguishing different graphs. Most notably, the addition of the adjacency features is sufficient to give a large performance boost (up to 11%).

Table 2. Graph classification accuracy results of different models with added Adjacency matrix features (AD), RWR features (RWR), and RWR features + RWR Regularization (RWR + RWRReg).

Model	Structural Information		Dataset	
		ENZYMES	D&D	PROTEINS
GCN	none	0.570 ± 0.052	0.755 ± 0.028	0.740 ± 0.035
	AD	0.591 ± 0.076	0.779 ± 0.022	0.775 ± 0.042
	RWR	0.584 ± 0.055	0.775 ± 0.023	0.784 ± 0.034
	RWR + RWRReg	**0.616 ± 0.065**	**0.790 ± 0.023**	**0.795 ± 0.032**
DiffPool	none	0.661 ± 0.031	0.793 ± 0.022	0.813 ± 0.017
	AD	0.711 ± 0.027	0.837 ± 0.020	0.821 ± 0.039
	RWR	0.687 ± 0.025	0.824 ± 0.028	0.783 ± 0.043
	RWR + RWRReg	**0.721 ± 0.039**	**0.840 ± 0.024**	**0.834 ± 0.038**
k-GNN	none	0.515 ± 0.111	0.756 ± 0.021	0.763 ± 0.043
	AD	0.572 ± 0.063	0.778 ± 0.020	0.751 ± 0.034
	RWR	**0.573 ± 0.077**	**0.794 ± 0.022**	0.781 ± 0.028
	RWR + RWRReg	0.571 ± 0.080	0.786 ± 0.021	**0.785 ± 0.026**

Surprisingly, models like DiffPool and *k*-GNN show an important difference in accuracy (up to 10%) when there is injection of structural information, meaning that even the most advanced methods suffer from the inability to exploit global structural information.

5.3. Counting Triangles

The TRIANGLES dataset [32] is composed of randomly generated graphs, where the task is to count the number of triangles contained in each graph. This is a hard task for GNNs and, as in [32], we use node degrees as node features to impose some structural information in the network. The TRIANGLES dataset has a test set with 10,000 graphs, of which half are similar in size to the ones in the training and validation sets (4–25 nodes), and half are bigger (up to 100 nodes). This permits an evaluation of a model's capability generalization to graphs of unseen sizes.

For this regression task, we use a three layer GCN, and we minimize the Mean Squared Error (MSE) loss (more details can be found in Appendix B). Table 3 presents MSE results on the test dataset as a whole and on the two splits separately. We see that the addition of RWR features and of RWRReg provides significant benefits (up to 19% improvements), especially when the model has to generalize to graphs of unseen sizes, while the addition of adjacency features leads to overfitting (we provide more details in Appendix D).

Table 3. Mean Squared Error of GCN with different types of global structural information injection on the TRIANGLES dataset.

Model	TRIANGLES Test Set		
	Global	Small	Large
GCN	2.290	1.311	3.608
GCN-AD	4.746	1.162	5.971
GCN-RWR	2.044	**1.101**	2.988
GCN-RWR + RWRReg	**2.029**	1.166	**2.893**

6. Practical Aspects

From the results shown in Section 5, it would be tempting to propose the addition of adjacency matrix information or RWR information into node feature vectors as a strategy to improve the performance of GNN models. However, the benefits introduced by such a strategy come at a high cost: adding n features increases the input size of $n \times n$ elements (which is prohibitive for large graphs). Furthermore, all the considered models have a weight matrix at each layer that depends on the feature dimension, which means we are also increasing the number of parameters at the first layer by $n \times d^{(1)}$ (where $d^{(1)}$ is the dimension of the feature vector for each node after the first GNN layer). In this section, we propose a practical way to take advantage of the injection of global structural information without increasing the number of parameters, and controlling the memory consumption during training.

6.1. RWRReg

From Section 5, the use of RWR coefficients as additional features coupled with the additional RWRReg term is the strategy that provides the highest performance improvement on all tasks. As discussed at the beginning of this section, the addition of RWR coefficients can be problematic, and hence we study the impact of using **only** the RWRReg term. We consider the same settings and tasks presented in Section 5, and results are shown in Table 4. The results show that the sole addition of the RWRReg term increases the performance of the considered models by more than 5%. At the same time, RWRReg (i) does not increase the input size or the number of parameters, (ii) does not require additional operations at inference time, (iii) does not require additional supervision (it is in fact a *self-supervised* objective), (iv) maintains the permutation invariance of MPNN models, and (v) there is a vast literature on efficient methods for computing RWR, even for web-scale graphs (e.g., [15,33,34]). Hence, the only downside of RWRReg is the storage of the RWR matrix during training on very large graphs.

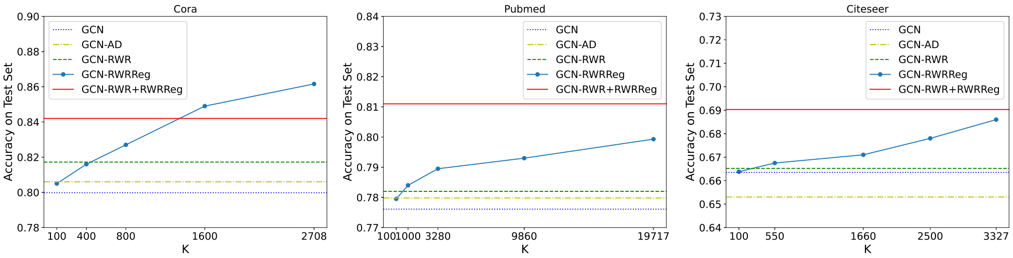
Table 4. Results for the addition of *only* the RWRReg term to existing models on node classification (accuracy), graph classification (accuracy), and triangle counting (MSE—lower is better).

Model	**Regularization**	**Dataset**		
			Node Classification	
		Cora	**Pubmed**	**Citeseer**
GCN	none RWRReg	0.799 ± 0.029 $\mathbf{0.861 \pm 0.025}$	0.776 ± 0.022 $\mathbf{0.799 \pm 0.034}$	0.663 ± 0.095 $\mathbf{0.686 \pm 0.096}$
GraphSage	none RWRReg	0.806 ± 0.017 $\mathbf{0.841 \pm 0.016}$	0.807 ± 0.016 $\mathbf{0.818 \pm 0.017}$	0.681 ± 0.021 $\mathbf{0.721 \pm 0.021}$
GAT	none RWRReg	0.815 ± 0.021 $\mathbf{0.824 \pm 0.022}$	0.804 ± 0.011 $\mathbf{0.811 \pm 0.013}$	0.664 ± 0.008 $\mathbf{0.702 \pm 0.013}$
			Graph Classification	
		ENZYMES	D&D	PROTEINS
GCN	none RWRReg	0.570 ± 0.052 $\mathbf{0.621 \pm 0.041}$	0.755 ± 0.028 $\mathbf{0.786 \pm 0.024}$	0.740 ± 0.035 $\mathbf{0.785 \pm 0.036}$
DiffPool	none RWRReg	0.661 ± 0.031 $\mathbf{0.733 \pm 0.032}$	0.793 ± 0.022 $\mathbf{0.822 \pm 0.025}$	0.813 ± 0.017 $\mathbf{0.820 \pm 0.038}$
k-GNN	none RWRReg	0.515 ± 0.111 $\mathbf{0.582 \pm 0.075}$	0.756 ± 0.021 $\mathbf{0.787 \pm 0.022}$	0.763 ± 0.043 $\mathbf{0.780 \pm 0.028}$
			Triangles Test Set	
		Global	Small	Large
GCN	none RWRReg	2.290 **2.187**	1.311 **1.282**	3.608 **3.014**

6.2. Sparsification of the RWR Matrix

To tackle the issue of storing in memory large RWR matrices, we explore how the sparsification of the RWR matrix affects the regularization of the model. In particular, we apply a *top-K* strategy: for each node, we only keep the K highest RWR weights. This approach can further take advantage of existing efficient methods to directly compute only the top-K RWR weights (e.g., [33–36]). As an example, TopPPR [33] provides guarantees on the precision of the returned scores, and requires only 15 seconds to retrieve the top-500 scores on a billion edge graph.

Figure 1 shows how different values of K impact performance on node classification (which usually is the task with the largest graphs). We can see that the addition of the RWRReg term is always beneficial. Furthermore, by taking the *top*-$\frac{n}{2}$, we can reduce the number of entries in the RWR matrix of $\frac{n^2}{2}$ elements, while still obtaining an average 3.2% increment on the accuracy of the model. This strategy then allows the selection of the value of K that best suits the available memory, while still obtaining a high performing model (better than GCN without global structural information injection).

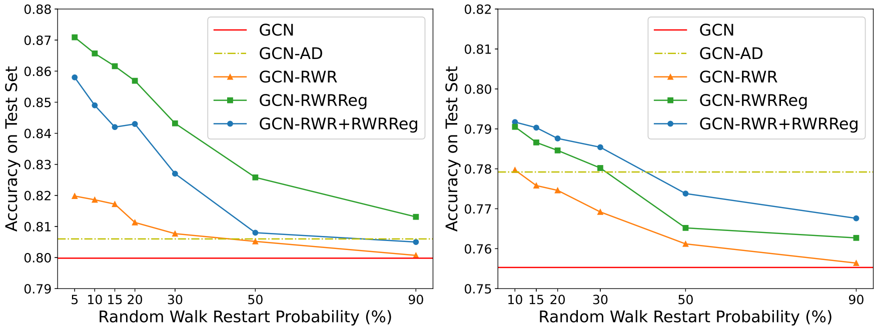


(**a**) (**b**) (**c**)

Figure 1. Performance of GCN on node classification for different values of K when trained with RWRReg with *Top-K* sparsification of the RWR matrix on the following datasets: (**a**) Cora, (**b**) Pubmed, (**c**) Citeseer.

6.3. Impact of RWR Restart Probability

The use of RWR requires to set the restart probability parameter. We show how performance changes with different restart probabilities. Intuitively, higher restart probabilities might put more much focus on close nodes, as the random walker with frequent return to the starting node. On the other side, lower probabilities allow for more long-range exploration, but may get "trapped" into densely connected subgraphs. Intuitively, we would expect lower probabilities to provide more information that is not already available to practical GNNs, and hence lead to higher performance. Figure 2 summarises how the accuracy on node classification (side (a)) and graph classification (side (b)) changes with different restart probabilities. (We did not go below 5% for Cora, and 10% for D&D for stability reasons in the computation of the RWR coefficients.) In accordance with our intuition, higher restart probabilities focus on close nodes (and less on distant nodes), and produce lower accuracies. Furthermore, we notice how injecting RWR information is never detrimental to the performance of the model without any injection.



(**a**) (**b**)

Figure 2. Accuracy on Cora (**a**), and on D&D (**b**), of GCN without and with the injection of structural information, and for different restart probabilities of RWR.

7. Related Work

The field of GNNs has become extremely vast; for a thorough review, we refer the reader to a recent survey on the subject [1]. To the best of our knowledge, there are no studies that test if global information regarding the whole graph can significantly impact MPNNs on real-world tasks. However, there are some works that are conceptually related to our approach.

Several works have taken advantage of RWR in the context of MPNNs. Klicpera et al. [37] use RWR to create a new (weighted) adjacency matrix where message passing is performed. Li et al. [3] use random walks in a co-training scenario to add new nodes for the MPNNs' training set. Ying et al. [38] and Zhang et al. [39] use random walks to define aggregation neighbourhoods that are not confined to a fixed distance. Abu-El-Haija et al. [40,41] use powers of the adjacency matrix, which can be considered as random walk statistics, to define neighbourhoods of different scales. Zhuang and Ma [42] use random walks to define the positive pointwise mutual information (PPMI) matrix and then use it in place of the adjacency matrix in the MPNN formulation. Klicpera et al. [43] use a diffusion strategy based on RWR instead of aggregating information from neighbours. This last work has recently been extended by [44] to scale to large graphs using RWRs to sample neighbourhoods. We remark how the aforementioned works focus on creating novel MPNN models, while we are interested in studying the impact of global structural information (which MPNNs do not have access to).

Gao et al. [45] and Ref. [46] uses regularization techniques to enforce that the embeddings of neighbouring nodes should be close to each other. The first uses Conditional Random Fields, while the second uses a regularization term based on the graph Laplacian. Both approaches only focus on 1-hop neighbours and do not take global information into account.

With regard to the study of the capabilities and weaknesses of GNNs, Refs. [3,47] study the over-smoothing problem that appears in Deep-GCN architectures, while Refs. [8,20] characterize the relation to the Weisfeiler–Leman algorithm. Other works have expressed the similarity with distributed computing [48,49], and the alignment with particular algorithmic structures [50]. These important contributions have advanced our understanding of the capabilities of GNNs, but they do not analyze or quantify the impact of *global* structural information.

Our RWRReg term relies on the computation of the RWR coefficients for every node (for computing the loss function). When dealing with large graphs, there is a vast literature on fast approximations of RWR scores [15,16,33,34,51,52].

Recent work [53] has shown that *anonymous random walks* (i.e., random walks where the global identities of nodes are not known) of fixed length starting at node u are sufficient to reconstruct the local neighborhood within a fixed distance of a node u [53]. Subsequently, anonymous random walks have been introduced in the context of learning graph representations [54]. Such results are complementary to ours, since they assume access to the distribution of *entire walks* of a given length, while our RWR representation only stores information on the probability of ending in a given node. In addition, such works do not provide a connection between RWR and 1-WL.

8. Conclusions

Whether *global* structural information (i.e., information that depends on the structure of the whole graph) is needed in GNNs for common tasks on graph-structured data is an open question. In this work, we tackle this question directly at its root. In particular, we identify three strategies to inject *global* structural information into MPNN models, and we quantify their impact on popular downstream tasks. Our experiments show that the additional information significantly boosts the performance of all considered state-of-the-art models, highlighting and quantifying the importance that *global* structural information can have on common MPNN applications. We further discuss a novel practical regularization technique based on RWR, which leads to an average improvement of 5% on all models, and is supported by a novel connection between RWR and the 1-Weisfeiler–Leman algorithm.

Author Contributions: Conceptualization, D.B. and F.V.; methodology, D.B. and F.V.; software, D.B.; validation, D.B. and F.V.; formal analysis, D.B. and F.V.; investigation, D.B.; resources, F.V.; data curation, D.B.; writing—original draft preparation, D.B. and F.V.; writing—review and editing, D.B. and F.V.; visualization, D.B.; supervision, F.V.; project administration, F.V.; funding acquisition, F.V. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported, in part, by MIUR of Italy, under PRIN Project No. 20174LF3T8 AHeAD and Grant L. 232 (Dipartimenti di Eccellenza), and by the University of Padova under project “SID 2020: RATED-X”.

Data Availability Statement: Publicly available datasets were analyzed in this study. This data can be found here: https://linqs.soe.ucsc.edu/data (access date: 12 February 2021), https://chrsmrrs.github.io/datasets/ (access date: 12 February 2021).

Conflicts of Interest: The authors declare no conflict of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript, or in the decision to publish the results.

Appendix A. Proof of Proposition 1

Given a graph $G = (V, E)$, we define its *k-step RWR representation* as the set of vectors $\mathbf{r}_v = [r_{v,u_1}, \dots, r_{v,u_n}]$, $v \in V$, where each entry $r_{v,u}$ describes the probability that an RWR of length k starting in v ends in u.

Proposition A1. *Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be two non-isomorphic graphs for which the 1-WL algorithm terminates with the correct answer after k iterations and starting from the labelling of all 1's. Then, the k-step RWR representations of G_1 and G_2 are different.*

Proof. Consider the WL algorithm with initial labeling given by all 1's. It's easy to see that (i) after k iterations the label of a node v corresponds to the information regarding the degree distribution of the neighborhood of distance $\leq k$ from v and (ii) in iteration $i \leq k$, the degrees of nodes at distance i from v are included in the label of v. In fact, after the first iteration, two nodes have the same colour if they have the same degree, as the colour of each node is given by the multiset of the colours of its neighbours (and we start with initial labeling given by all 1's). After the second colour refinement iteration, two nodes have the same colour if they had the same colour after the first iteration (i.e., have the same degree), and the multisets containing the colours (degrees) of their neighbours are the same. In general, after the k-th iteration, two nodes have the same colour if they had the same colour in iteration $k-1$, and the multiset containing the degrees of the neighbours at distance k is the same for the two nodes. Hence, two nodes that have different colours after a certain iteration will have different colours in all the successive iterations. Furthermore, the colour after the k-th iteration depends on the colour at the previous iteration (which "encodes" the distribution of degree of neighbours up to distance $k-1$ included), and the multiset of the degrees of neighbours at distance k.

Given two non-isomorphic graphs G_1 and G_2, if the WL algorithm terminates with the correct answer starting from the all 1's labelling in k iterations, it means that there is no *matching* between vertices in V_1 and vertices in V_2 such that matched vertices have the same degree distribution for neighborhoods at distance exactly k. Equivalently, any matching M that minimizes the number of matched vertices with different degree distribution has at least one such pair. Now consider one such matching M, and let $v \in V_1$ and $w \in V_2$ be vertices matched in M with different degree distributions for neighborhoods at distance exactly k. Since v and w have different degree distributions at distance k, the number of choices for paths of length k starting from v and w must be different (since the number of choices for the k-th edge on the path is different). Therefore, there must be at least a node $u \in V_1$ and a node $z \in V_2$ that are matched by M but for which the number of paths of length k from v to u is different from the number of paths of length k from w to z. Since $r_{v,u}$ is proportional to the number of paths of length k from v to u, we have that $r_{v,u} \neq r_{w,z}$ that is $\mathbf{r}_v \neq \mathbf{r}_w$. Thus, the *k-step RWR representation* of G_1 and G_2 are different. □

Appendix B. Model Implementation Details

We present here a detailed description of the implementations of the models we use in our experimental section. Whenever possible, we started from the official implementation of the authors of each model. Table A1 contains links to the implementations we used as a starting point for the code for our experiments.

Table A1. Starting model implementations.

Model	Implementation	Access Date
GCN *(for node classification)*	github.com/tkipf/pygcn	2 February 2021
GCN *(for graph classification)* GCN *(for triangle counting)*	github.com/bknyaz/graph_nn	2 February 2021
GraphSage	github.com/williamleif/graphsage-simple	10 February 2021
GAT	github.com/Diego999/pyGAT	13 February 2021
DiffPool	github.com/RexYing/diffpool	15 February 2021
k-GNN	github.com/chrsmrrs/k-gnn	15 February 2021

Training Details

With regard to the training procedure, we have that all models are trained with early stopping on the validation set (stopping the training if the validation loss does not decrease for a certain amount of epochs), and unless explicitly specified, we use Cross Entropy as loss function for all the classification tasks.

For the task of graph classification, we zero-pad the feature vectors of each node to make them all the same length when we inject structural information into the node feature vectors.

For the task of triangle counting, we follow [32] and use the one-hot representation of node degrees as node feature vectors to impose some structural information in the network.

Computing Infrastructure

The experiments were run on a GPU cluster with 7 Nvidia 1080Ti, and on a CPU cluster (when the memory consumption was too big to fit in the GPUs) equipped with 8 cpus 12-Core Intel Xeon Gold 5118 @2.30 GHz, with 1.5 Tb of RAM.

In the rest of this section, we go through each model used in our experiments, specifying architecture, hyperparameters, and the position of the node embeddings used for RWRReg.

Appendix B.1. GCN (Node Classification)

We use a two layer architecture. The first layer outputs a 16-dimensional embedding vector for each node, and passes it through a ReLu activation, before applying dropout [55], with probability 0.5. The second layer outputs a c-dimensional embedding vector for each node, where c is the number of output classes and these vectors are passed through *Softmax* to get the output probabilities for each class. An additional L2-loss is added with a balancing term of 0.0005. The model is trained using the Adam optimizer [56] with a learning rate of 0.01.

We apply the RWRReg on the 16-dimensional node embeddings after the first layer.

Appendix B.2. GCN (Graph Classification)

We first have two GCN layers, each one generating a 128-dimensional embedding vector for each node. Then, we apply *max*-pooling on the features of the nodes and pass the pooled 128-dimensional vector to a two-layer feed-forward neural network with 256 neurons at the first layer and c at the last one, where c is the number of output classes. A ReLu activation is applied in between the two feed-forward layers, and *Softmax* is applied after the last layer. Dropout [55] is applied in between the last GCN layer and the feed-forward layer, and in between the feedforward layers (after ReLu), in both cases with probability of 0.1. The model is trained using the Adam optimizer [56] with a learning rate of 0.0005.

We apply the RWRReg on the 128-dimensional node embeddings after the last GCN layer.

Appendix B.3. GCN (Counting Triangles)

We first have three GCN layers, each one generating a 64-dimensional embedding vector for each node. Then, we apply *max*-pooling on the features of the nodes and pass the pooled 64-dimensional vector to a one-layer feed-forward neural network with one neuron. Dropout [55] is applied in between the last GCN layer and the feed-forward layer with probability of 0.1. The model is trained by minimizing the mean squared error (MSE) and is optimized using the Adam optimizer [56] with a learning rate of 0.005.

We apply the RWRReg on the 64-dimensional node embeddings after the last GCN layer.

Appendix B.4. GraphSage

We use a two layer architecture. For Cora, we sample five nodes per-neighbourhood at the first layer and 5 at the second, while, on the other datasets, we sample 10 nodes per-neighbourhood at the first layer and 25 at the second. Both layers are composed of *mean-aggregators* (i.e., we take the mean of the feature vectors of the nodes in the sampled neighbourhood) that output a 128-dimensional embedding vector per node. After the second layer, these embeddings are multiplied by a learnable matrix with size 128 × c, where c is the number of output classes, giving thus a c-dimensional vector per-node. These vectors are passed through *Softmax* to get the output probabilities for each class. The model is optimized using Stochastic Gradient Descent with a learning rate of 0.7.

We apply the RWRReg on the 128-dimensional node embeddings after the second aggregation layer.

Appendix B.5. GAT

We use a two layer architecture. The first layer uses an 8-headed attention mechanism that outputs an 8-dimensional embedding vector per-node. LeakyReLu is set with slope $\alpha = 0.2$. Dropout [55] (with probability of 0.6) is applied after both layers. The second layer outputs a c-dimensional vector for each node, where c is the number of classes, and before passing each vector through *Softmax* to obtain the output predictions, the vectors are passed through an Elu activation [57]. An additional L2-loss is added with a balancing term of 0.0005. The model is optimized using Adam [56] with a learning rate of 0.005.

We apply the RWRReg on the 8-dimensional node embeddings after the first attention layer. A particular note needs to be made for the training of GATs: we found that naively implementing the RWRReg term on the node embeddings in between two layers brings to an exploding loss as the RWRReg term grows exponentially at each epoch. We believe this happens because the attention mechanism in GATs allows the network to infer that certain close nodes, even 1-hop neighbours, might not be important to a specific node and so they should not be embedded close to each other. This clearly goes in contrast with the RWRReg loss, since 1-hop neighbours always have a high score. We solved this issue by using the attention weights to scale the RWR coefficients at each epoch (we make sure that gradients are not calculated for this operation as we only use them for scaling). This way, the RWRReg penalizations are in accordance with the attention mechanism, and are still encoding long-range dependencies.

Appendix B.6. DiffPool

We use a 1-pooling architecture. The initial node feature matrix is passed through two (one to obtain the assignment matrix and one for node embeddings) 3-layer GCN, where each layer outputs a 20-dimensional vector per-node. Pooling is then applied, where the number of clusters is set as 10% of the number of nodes in the graph, and then another 3-layer GCN is applied to the pooled node features. Batch normalization [58] is added in between every GCN layer. The final graph embedding is passed through a 2-layer MLP with a final *Softmax* activation. An additional L2-loss is added with a balancing term of 10^{-7}, together with two pooling-specific losses. The first enforces the intuition that nodes that are close to each other should be pooled together and is defined as: $\mathcal{L}_{LP} = \|\boldsymbol{A}^{(l)}, \boldsymbol{S}^{(l)\intercal}\boldsymbol{S}^{(l)}\|_F$, where $\|\cdot\|_F$ is the Frobenius norm, and $\boldsymbol{S}^{(l)}$ is the assignment matrix at layer l. The second one encourages the cluster assignment to be close to a one-hot vector, and is defined as: $\mathcal{L}_E = \frac{1}{n}\sum_{i=1}^{n} H(\boldsymbol{S}_{i,:})$, where H is the entropy function. However, in the implementation available online, the authors do not make use of these additional losses. We follow the latter implementation. The model is optimized using Adam [56] with a learning rate of 0.001.

We apply the RWRReg on the 20-dimensional node embeddings after the first 3-layer GCN (before pooling). We tried applying it also after pooling on the coarsened graph, but the fact that this graph could change during training yields to poor results.

Appendix B.7. k-GNN

We use the hierarchical 1-2-3-GNN architecture (which is the one showing the highest empirical results). First, a 1-GNN is applied to obtain node embeddings, then these embeddings are used as initial values for the 2 GNN (1-2-GNN). The embeddings of the 2-GNN are then used as initial values for the 3-GNN (1-2-3-GNN). The 1-GNN applies three graph convolutions, while 2-GNN and the 3-GNN apply two graph convolutions. Each convolution outputs a 64-dimensional vector and is followed by an Elu activation [57]. For each k, node features are then globally averaged and the final vectors are concatenated and passed through a three layer MLP. The first layer outputs a 64-dimensional vector, while the second outputs a 3two-dimensional vector, and the third outputs a c-dimensional vector, where c is the number of output classes. To obtain the final output probabilities for each class, *log(Softmax)* is applied, and the negative log likelihood is used as loss function. After the first and the second MLP layers an Elu activation [57] is applied, furthermore, after the first MLP layer dropout [55] is applied with probability 0.5. The model is optimized using Adam [56] with a learning rate of 0.01, and a decaying learning rate schedule based on validation results (with minimum value of 10^{-5}).

We apply the RWRReg on the 64-dimensional node embeddings after the 1-GNN. We were not able to apply it also after the 2-GNN and the 3-GNN, as it would cause out-of-memory issues with our computing resources.

Appendix C. Datasets

We briefly present here some additional details about the datasets used for our experimental section. Table A2 summarizes the datasets for node classification, while Table A3 presents information about the datasets for graph classification and triangle counting. The node classification datasets are available at https://linqs.soe.ucsc.edu/data (access date: 12 February 2021), while the graph classification and the triangle counting at https://chrsmrrs.github.io/datasets/ (access date: 12 February 2021).

Table A2. Node classification dataset statistics.

Dataset	Nodes	Edges	Classes	Features	Label Rate
Cora	2708	5429	7	1433	0.052
Pubmed	19,717	44,338	3	500	0.003
Citeseer	3327	4732	6	3703	0.036

Table A3. Graph classification and triangle counting dataset statistics.

Dataset	Graphs	Classes	Avg. # Nodes	Avg. # Edges
ENZYMES	600	6	32.63	62.14
D&D	1178	2	284.32	715.66
PROTEINS	1113	2	39.1	72.82
TRIANGLES	45,000	10	20.85	32.74

Appendix D. Adjacency Matrix Features Lead to Bad Generalization on the Triangle Counting Task

We present additional details about the overfitting behaviour of GCN on the triangle counting task when injected with adjacency matrix information. In Figure A1, we plot the evolution of the MSE on the training and test set over the training epochs. GCN-AD reaches the lowest error on the training set, while the highest on the test set, thus confirming its overfitting behaviour. We can observe that, after 6 epochs, GCN-AD is already the model presenting the lowest training loss, and it remains so until the end. Furthermore, we can notice how the test loss presents a growing trend, which is in contrast to the other models.

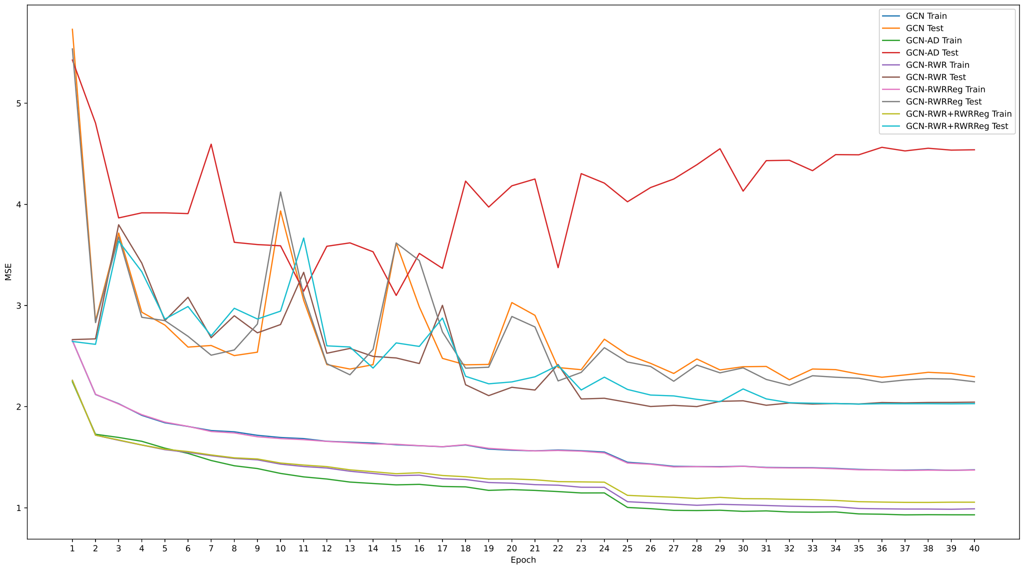


Figure A1. Training and test losses of GCN with different structural information injection on the triangle counting task.

Appendix E. Fast Implementation of the Random Walk with Restart Regularization

Let $\boldsymbol{H}$ be the matrix containing the node embeddings, and $\boldsymbol{S}$ be the matrix with the RWR statistics. We are interested in the following quantity

$$\mathcal{L}_{RWRReg} = \sum_{i,j} \boldsymbol{S}_{i,j} ||\boldsymbol{H}_{i,:} - \boldsymbol{H}_{j,:}||^2$$

To calculate it in a fast way (specially when using GPUs), we use the following procedure. Let us first define the following matrices:

$$\begin{aligned}
&\hat{\boldsymbol{S}} = n \times n \text{ symmetric matrix with } \hat{\boldsymbol{S}}_{i,j} = \begin{cases} \boldsymbol{S}_{i,j} + \boldsymbol{S}_{j,i} & \text{for } i \neq j \\ \boldsymbol{S}_{i,j} & \text{for } i = j \end{cases} \\
&\boldsymbol{D} = n \times n \text{ diagonal matrix with } D_{i,i} = \sum_j \hat{\boldsymbol{S}}_{i,j} \\
&\boldsymbol{\Delta} = \boldsymbol{D} - \hat{\boldsymbol{S}}
\end{aligned}$$

We then have

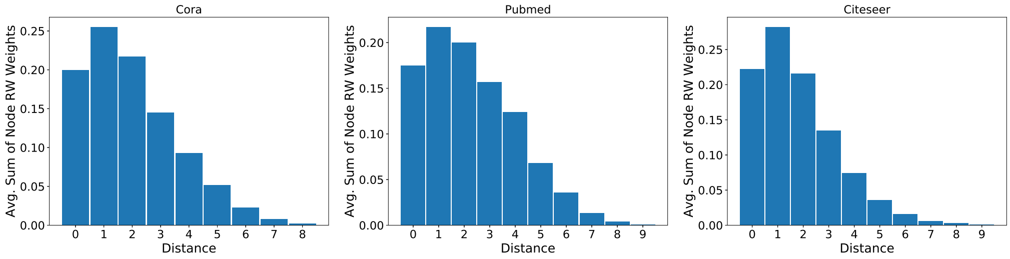
$$\mathcal{L}_{RWRReg} = \sum_{i,j} \boldsymbol{S}_{i,j} ||\boldsymbol{H}_{i,:} - \boldsymbol{H}_{j,:}||^2 = \sum_i \boldsymbol{H}_{:,i}^{\intercal} \boldsymbol{\Delta} \boldsymbol{H}_{:,i} = Tr(\boldsymbol{H}^{\intercal} \boldsymbol{\Delta} \boldsymbol{H})$$

where $Tr(\cdot)$ is the trace of the matrix. Note that $\boldsymbol{H}_{:,i}^{\intercal}$ is the i-th column of $\boldsymbol{H}$, transposed, so its size is $1 \times n$.

Appendix F. Empirical Analysis of the Random Walk with Restart Matrix

We now analyse the RWR matrix to justify the use of RWR for the encoding of global structural information. We consider the three node classification datasets (see Section 5 of the paper), as this is the task with the largest input graphs, and hence where this kind of information seems more relevant.

We first consider the distribution of the RWR (we consider RWR, with a restart probability of 0.15, as done for the experimental evaluation of our proposed technique) weights at different distances from a given node. In particular, for each node, we take the sum of the weights assigned to the 1-hop neighbours, the 2-hop neighbours, and so on. We then take the average, over all nodes, of the sum of the RWR weights at each hop. We discard nodes that belong to connected components with diameter ≤ 4, and we only plot the values for the distances that have an average sum of weights higher than 0.001. Plots are shown in Figure A2. We notice that the RWR matrix contains information that goes beyond the immediate neighbourhood of a node. In fact, we see that approximately 90% of the weights are contained within the 6-hop neighbourhood, with a significant portion that is not contained in the 2-hop neighbourhood usually accessed by MPNN models.



(**a**) (**b**) (**c**)

Figure A2. Average distribution of the RWR weights at different distances for the following node classification datasets: (**a**) Cora, (**b**) Pubmed, (**c**) Citeseer. Distance zero indicates the weight that a node assigns to itself.

Next, we analyse if RWR captures some non-trivial relationships between nodes. In particular, we investigate if there are nodes that are far from the starting node, but receive a higher weight than some closer nodes. To quantify this property, we use the Kendall Tau-b (We use the Tau-b version because the elements in the sequences we analyze are not all distinct) measure [59]. In more detail, for each node v, we consider the sequence $rw^{(v)}$ where the i-th element is the weight that the RWR from node v has assigned to node i: $rw^{(v)}[i] = S_{v,i}$. We then define the sequence $drw^{(v)}$ such that $drw^{(v)}[j] = dist(v, f_{sort\_weights}(j, rw^{(v)}))$, where $dist(x, y)$ is the shortest path distance between node x and node y, and $f_{sort\_weights}(j, rw^{(v)})$ is the node with the j-th highest RWR weight in $rw^{(v)}$. Intuitively, if the RWR matrix is not capable of capturing a non-trivial relationship, we would have that $drw^{(v)}$ is a sorted list (with repetitions). By comparing $drw^{(v)}$ with its sorted version with the Kendall Tau-b rank, we obtain a value between 1 and -1, where 1 means that the two sequences are identical, and -1 means that one is the reverse of the other. Table A4 presents the results, averaged over all nodes, on the node classification datasets. These results show that, while there is a strong relation between the information provided by RWR and the distance between nodes, there is information in the RWR that is not captured by shortest path distances.

Table A4. Average and standard deviation, over all nodes, of Kendall Tau-b values measuring the non-trivial relationships between nodes captured by the RWR weights.

Dataset	Average Kendall Tau-b
Cora	0.729 ± 0.082
Pubmed	0.631 ± 0.057
Citeseer	0.722 ± 0.171

As an example of the non-trivial relationships encoded by RWR, Figure A3 presents a $drw^{(v)}$ sequence taken from a node in Cora. This sequence obtains a Kendall Tau-b value of 0.591. We can observe that, for distances greater than 1, we already have some non-trivial relationships. In fact, we observe some nodes at distance 3 that receive a larger weight than nodes at distance 2. There are many other interesting non-trivial relationships; for example, we notice that some nodes at distance 7, and some at distance 11, obtain a higher weight than some nodes at distance 5.

```
$drw^{(1000)}$ = [1, 1, 1, 2, 2, 2, 2, 2, 3, 2, 3, 2, 2, 2, 2, 3, 2, 3, 2, 2, 3, 2, 2, 3, 2, 2, 2, 3, 2, 3, 3, 2, 2, 3, 3, 4, 3, 4,
3, 3, 3, 3, 3, 3, 4, 3, 3, 4, 3, 3, 4, 3, 3, 4, 3, 3, 3, 3, 4, 4, 3, 3, 3, 4, 3, 3, 3, 3, 4, 3, 4, 3, 3, 3, 4, 4, 3, 4, 4, 4, 4, 4,
4, 4, 3, 3, 3, 4, 4, 3, 3, 4, 3, 3, 4, 4, 3, 4, 3, 4, 3, 4, 4, 4, 4, 4, 5, 5, 3, 4, 4, 4, 4, 4, 4, 4, 4, 3, 4, 4, 5, 4, 4, 4, 6, 3,
4, 4, 5, 4, 4, 5, 4, 5, 4, 4, 4, 4, 4, 3, 4, 4, 4, 4, 4, 5, 5, 4, 4, 4, 4, 5, 4, 4, 4, 4, 4, 5, 5, 5, 4, 5, 4, 5, 5, 4, 4, 4, 5, 4,
4, 4, 4, 5, 5, 4, 4, 4, 5, 4, 4, 5, 4, 4, 5, 4, 4, 5, 5, 5, 5, 5, 4, 5, 5, 4, 5, 5, 4, 5, 4, 4, 5, 4, 5, 5, 5, 5, 5, 4, 5, 4, 5, 5,
4, 4, 5, 4, 5, 5, 4, 5, 5, 5, 4, 4, 5, 5, 5, 4, 5, 4, 5, 5, 5, 5, 4, 5, 5, 5, 6, 5, 5, 5, 4, 5, 5, 5, 4, 4, 5, 5, 5, 4, 5, 5, 5, 4,
4, 5, 5, 5, 5, 4, 5, 5, 5, 4, 5, 4, 5, 4, 5, 4, 5, 4, 5, 5, 6, 4, 4, 4, 5, 5, 5, 4, 5, 4, 5, 5, 5, 6, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5,
5, 5, 5, 5, 5, 5, 5, 5, 6, 5, 4, 5, 5, 4, 6, 5, 4, 5, 5, 5, 4, 5, 5, 5, 6, 4, 4, 5, 5, 5, 4, 5, 5, 5, 5, 5, 5, 5, 4, 5, 6, 5, 5, 5,
6, 5, 4, 5, 5, 5, 4, 5, 4, 5, 4, 5, 4, 4, 5, 6, 6, 5, 4, 5, 5, 5, 5, 5, 5, 5, 5, 5, 4, 5, 5, 6, 6, 5, 5, 4, 5, 5, 5, 5, 6, 5, 6, 5,
6, 5, 4, 5, 5, 5, 5, 5, 5, 6, 6, 5, 5, 5, 5, 6, 5, 5, 5, 5, 5, 6, 5, 5, 6, 6, 6, 6, 5, 5, 5, 5, 4, 6, 6, 5, 4, 4, 4, 4, 6, 5, 5, 5,
5, 5, 5, 5, 5, 6, 5, 5, 6, 5, 5, 5, 4, 6, 5, 5, 5, 5, 5, 6, 5, 5, 6, 6, 6, 5, 6, 6, 6, 5, 6, 5, 5, 5, 4, 5, 5, 6, 5, 5, 6, 4, 5, 5,
5, 6, 5, 6, 5, 4, 5, 6, 5, 4, 6, 5, 6, 5, 6, 4, 6, 5, 5, 5, 5, 6, 5, 6, 5, 5, 5, 6, 5, 6, 4, 6, 6, 5, 5, 6, 5, 5, 5, 5, 5, 5, 5, 5,
5, 5, 4, 6, 5, 5, 5, 6, 6, 6, 6, 6, 4, 6, 5, 6, 5, 5, 6, 5, 5, 6, 5, 6, 5, 5, 5, 5, 7, 5, 5, 5, 5, 5, 5, 4, 5, 6, 5, 5, 6, 4, 6, 5,
5, 5, 5, 5, 5, 6, 5, 6, 6, 5, 5, 6, 5, 5, 4, 6, 5, 5, 7, 5, 6, 6, 5, 5, 5, 6, 6, 5, 5, 5, 5, 5, 5, 5, 5, 5, 6, 5, 6, 5, 5, 5, 5, 5,
5, 6, 5, 6, 6, 5, 6, 5, 6, 5, 6, 6, 5, 5, 5, 7, 5, 6, 5, 6, 5, 6, 6, 6, 5, 5, 5, 5, 5, 6, 4, 5, 6, 6, 6, 5, 5, 6, 5, 6, 5, 5, 5, 6,
5, 5, 6, 6, 4, 6, 5, 6, 5, 5, 6, 5, 5, 6, 5, 5, 6, 5, 6, 6, 4, 6, 6, 7, 5, 5, 5, 6, 6, 5, 6, 5, 6, 6, 5, 4, 5, 4, 6, 6, 6, 5, 5, 5,
6, 7, 6, 5, 4, 6, 5, 6, 6, 5, 4, 6, 5, 5, 5, 7, 5, 6, 5, 5, 4, 6, 5, 5, 6, 4, 5, 6, 5, 6, 5, 6, 6, 5, 5, 6, 7, 6, 6, 6, 6, 5, 6, 5,
5, 5, 5, 7, 5, 5, 5, 6, 5, 5, 5, 6, 6, 6, 5, 5, 6, 7, 5, 5, 6, 4, 6, 5, 6, 6, 4, 6, 6, 6, 6, 5, 6, 5, 6, 6, 5, 5, 6, 6, 6, 6, 5, 6,
7, 5, 5, 7, 6, 6, 6, 6, 5, 7, 6, 4, 5, 5, 6, 6, 5, 5, 5, 6, 6, 5, 5, 6, 7, 5, 5, 6, 5, 7, 5, 5, 6, 6, 7, 6, 6, 7, 6, 6, 6, 6, 6, 5,
5, 4, 5, 6, 5, 6, 7, 6, 6, 6, 6, 6, 6, 6, 4, 7, 6, 4, 4, 5, 6, 6, 6, 4, 6, 7, 6, 7, 6, 6, 5, 5, 4, 6, 6, 6, 5, 5, 6, 7, 6, 5, 4, 6,
7, 6, 4, 6, 5, 7, 6, 7, 5, 7, 6, 6, 4, 4, 5, 6, 4, 6, 6, 7, 6, 6, 6, 7, 5, 6, 6, 4, 6, 6, 6, 6, 5, 6, 5, 5, 7, 6, 7, 5, 7, 6, 6, 6,
5, 7, 6, 7, 6, 5, 6, 7, 6, 6, 5, 7, 7, 4, 6, 6, 5, 6, 4, 6, 6, 6, 5, 6, 6, 6, 6, 7, 4, 6, 5, 4, 6, 5, 6, 7, 7, 6, 6, 6, 4, 6, 6, 6,
6, 4, 4, 6, 4, 5, 4, 6, 7, 6, 6, 6, 6, 5, 6, 6, 6, 6, 5, 6, 5, 4, 4, 6, 6, 4, 5, 6, 7, 6, 6, 6, 5, 6, 4, 4, 5, 5, 4, 5, 7, 7, 4, 7,
6, 6, 7, 7, 7, 7, 5, 5, 6, 4, 5, 5, 6, 6, 6, 5, 6, 4, 7, 6, 6, 7, 6, 6, 6, 6, 4, 5, 6, 6, 7, 7, 4, 6, 6, 6, 6, 4, 4, 4, 5, 7, 5, 5,
6, 5, 5, 6, 7, 4, 6, 7, 6, 4, 6, 5, 5, 6, 5, 6, 6, 6, 6, 6, 5, 6, 6, 6, 5, 8, 6, 7, 6, 7, 6, 6, 6, 6, 6, 6, 6, 6, 6, 4, 6, 6, 6, 6,
6, 5, 6, 4, 5, 6, 6, 7, 4, 6, 6, 6, 6, 6, 8, 5, 7, 7, 4, 4, 6, 6, 6, 7, 6, 6, 6, 6, 6, 5, 6, 7, 6, 6, 4, 7, 5, 4, 6, 6, 4, 4, 4, 4,
7, 6, 6, 6, 6, 6, 6, 6, 7, 5, 7, 6, 4, 6, 4, 5, 6, 6, 7, 4, 4, 4, 6, 4, 7, 6, 7, 6, 5, 6, 7, 7, 4, 6, 7, 5, 4, 6, 6, 5, 6, 5, 6, 7,
4, 4, 4, 5, 6, 4, 7, 4, 4, 6, 6, 6, 6, 6, 7, 7, 6, 7, 5, 7, 6, 7, 6, 7, 6, 6, 4, 6, 6, 6, 6, 6, 6, 6, 8, 6, 5, 5, 7, 4, 5, 4, 6, 8,
8, 6, 7, 6, 7, 7, 4, 7, 6, 7, 7, 5, 7, 4, 6, 4, 6, 6, 6, 7, 7, 4, 7, 7, 7, 6, 6, 6, 6, 7, 6, 6, 6, 4, 6, 4, 7, 4, 6, 7, 6, 6, 5, 6,
4, 6, 4, 4, 4, 6, 7, 5, 4, 5, 6, 5, 4, 7, 4, 5, 7, 6, 6, 7, 6, 7, 6, 4, 4, 4, 7, 4, 4, 5, 7, 8, 4, 6, 5, 5, 6, 6, 4, 6, 6, 7, 6, 6,
7, 6, 7, 4, 7, 7, 5, 6, 6, 5, 4, 7, 6, 7, 7, 7, 6, 8, 7, 7, 6, 6, 5, 7, 6, 7, 7, 7, 8, 4, 7, 4, 7, 5, 7, 4, 6, 6, 5, 4, 4, 7, 4, 6,
4, 4, 4, 7, 7, 5, 5, 8, 6, 6, 5, 5, 6, 6, 6, 5, 6, 6, 7, 6, 6, 6, 5, 7, 7, 6, 5, 6, 7, 7, 4, 7, 6, 5, 4, 5, 6, 4, 6, 7, 8, 7, 7, 6,
6, 7, 6, 7, 7, 7, 5, 7, 7, 5, 6, 6, 6, 8, 6, 7, 6, 7, 8, 6, 6, 6, 5, 7, 6, 6, 7, 5, 6, 5, 6, 4, 5, 7, 4, 7, 6, 7, 6, 4, 7, 5, 7, 4,
7, 7, 7, 4, 7, 8, 6, 6, 5, 5, 6, 4, 7, 5, 6, 7, 6, 6, 6, 4, 7, 6, 6, 7, 7, 5, 7, 5, 7, 7, 6, 8, 6, 7, 6, 7, 5, 4, 6, 6, 6, 6, 7, 4,
6, 7, 6, 4, 9, 7, 6, 5, 6, 6, 4, 5, 6, 7, 5, 6, 6, 5, 6, 7, 6, 6, 8, 4, 8, 6, 9, 6, 6, 5, 6, 5, 6, 6, 6, 6, 6, 6, 7, 5, 6, 6, 7, 7,
5, 7, 8, 5, 6, 6, 7, 6, 4, 6, 6, 7, 7, 7, 7, 5, 7, 5, 4, 7, 5, 7, 7, 6, 6, 6, 7, 8, 7, 4, 10, 5, 7, 7, 6, 6, 8, 6, 6, 6, 7, 4, 7, 8,
5, 7, 7, 7, 7, 5, 5, 7, 5, 6, 6, 6, 5, 6, 5, 5, 5, 7, 5, 4, 5, 5, 6, 4, 6, 5, 6, 8, 4, 4, 6, 5, 5, 8, 7, 5, 7, 7, 7, 5, 8, 7, 6, 8,
5, 8, 7, 6, 7, 6, 7, 6, 8, 6, 8, 7, 7, 5, 6, 6, 6, 5, 7, 5, 5, 6, 5, 6, 7, 7, 7, 5, 7, 6, 7, 5, 6, 5, 6, 8, 7, 7, 6, 7, 5, 6, 6, 7,
6, 6, 7, 7, 7, 8, 5, 7, 8, 7, 5, 7, 6, 6, 7, 5, 5, 7, 7, 6, 6, 7, 5, 8, 6, 7, 8, 6, 8, 9, 7, 7, 5, 8, 8, 7, 7, 5, 5, 7, 5, 5, 9, 7,
6, 6, 7, 7, 6, 5, 8, 5, 10, 10, 7, 6, 8, 5, 6, 7, 6, 8, 5, 7, 6, 5, 7, 5, 5, 7, 7, 6, 5, 8, 6, 7, 5, 8, 8, 5, 5, 6, 7, 6, 6, 7, 5,
6, 8, 6, 7, 7, 5, 8, 9, 6, 7, 5, 7, 8, 6, 7, 5, 5, 7, 7, 5, 7, 7, 5, 7, 4, 6, 5, 7, 7, 7, 6, 7, 6, 7, 7, 7, 6, 8, 6, 5, 5, 6, 6, 6,
6, 8, 7, 7, 8, 5, 6, 8, 8, 8, 9, 5, 8, 8, 7, 8, 7, 5, 5, 6, 5, 7, 6, 8, 7, 8, 7, 9, 5, 7, 7, 5, 8, 5, 5, 7, 6, 6, 5, 9, 6, 7, 6, 6,
5, 7, 7, 6, 7, 8, 5, 7, 8, 7, 5, 7, 7, 8, 9, 7, 7, 5, 8, 7, 7, 8, 8, 5, 9, 5, 6, 6, 8, 7, 6, 10, 5, 8, 6, 8, 7, 7, 7, 6, 7, 6, 6, 5,
7, 6, 6, 5, 5, 7, 8, 8, 7, 5, 7, 7, 8, 6, 8, 8, 6, 7, 6, 6, 6, 6, 6, 8, 8, 8, 8, 6, 5, 11, 8, 7, 8, 8, 7, 8, 9, 7, 6, 6, 8, 8, 8, 9,
6, 7, 6, 5, 6, 5, 6, 8, 8, 6, 7, 7, 8, 8, 8, 7, 8, 8, 9, 6, 9, 8, 6, 7, 8, 7, 7, 5, 8, 7, 7, 7, 7, 10, 8, 7, 7, 9, 7, 8, 8, 8, 9, 8,
5, 7, 7, 7, 8, 8, 5, 7, 6, 7, 7, 7, 8, 6, 7, 7, 7, 5, 8, 7, 10, 8, 8, 8, 8, 8, 8, 8, 5, 7, 5, 11, 9, 6, 5, 6, 7, 8, 8, 8, 8, 7, 9,
8, 5, 8, 7, 7, 8, 8, 7, 6, 7, 8, 6, 6, 7, 7, 8, 6, 6, 5, 10, 6, 10, 8, 5, 9, 8, 5, 8, 6, 8, 8, 8, 6, 6, 8, 8, 9, 9, 7, 6, 7, 8, 8,
8, 9, 7, 10, 8, 5, 6, 8, 6, 8, 9, 9, 6, 6, 6, 7, 7, 5, 6, 5, 10, 8, 5, 9, 7, 9, 8, 9, 8, 8, 8, 7, 10, 10, 5, 6, 6, 8, 8, 8, 8, 5, 6,
8, 8, 8, 9, 5, 8, 8, 9, 8, 6, 8, 7, 11, 6, 8, 9, 11, 6, 8, 5, 8, 6, 12, 8, 5, 8, 7, 7, 6, 8, 9, 9, 6, 8, 6, 8, 9, 8, 7, 9, 8, 9, 9,
7, 11, 8, 9, 10, 10, 8, 10, 9, 8, 8, 9, 7, 5, 10, 9, 9, 8, 7, 8, 5, 6, 7, 8, 5, 6, 6, 10, 8, 8, 6, 9, 7, 11, 5, 8, 8, 7, 10, 8, 8,
8, 5, 9, 8, 6, 8, 8, 9, 8, 8, 8, 9, 8, 11, 8, 8, 11, 8, 6, 9, 9, 6, 8, 5, 8, 8, 8, 6, 8, 6, 7, 11, 6, 7, 7, 9, 6, 8, 8, 9, 6, 6, 9,
9, 11, 10, 9, 8, 9, 9, 10, 10, 10, 7, 9, 8, 8, 6, 7, 8, 9, 7, 6, 6, 8, 10, 9, 10, 6, 8, 9, 8, 10, 11, 9, 10, 10, 11, 6, 11, 11, 8,
9, 7, 7, 8, 8, 10, 8, 9, 10, 13, 9, 8, 9, 7, 9, 8, 11, 7, 9, 10, 9, 9, 12, 8, 8, 9, 9, 11, 9, 11, 9, 12, 10, 11, 11]
```

Figure A3. $drw^{(v)}$ sequence for the 1000-th node in Cora.

References

1. Wu, Z.; Pan, S.; Chen, F.; Long, G.; Zhang, C.; Yu, P.S. A Comprehensive Survey on Graph Neural Networks. *arXiv* **2019**, arXiv:1901.00596.
2. Gilmer, J.; Schoenholz, S.S.; Riley, P.F.; Vinyals, O.; Dahl, G.E. Neural Message Passing for Quantum Chemistry. In Proceedings of the International Conference on Machine Learning, Sydney, Australia, 6–11 August 2017.
3. Li, Q.; Han, Z.; Wu, X. Deeper Insights Into Graph Convolutional Networks for Semi-Supervised Learning. In Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, Palo Alto, CA, USA, 7–8 February 2018.
4. Alon, U.; Yahav, E. On the Bottleneck of Graph Neural Networks and its Practical Implications. In Proceedings of the International Conference on Learning Representations, Vienna, Austria, 4 May 2021.
5. Masuda, N.; Porter, M.A.; Lambiotte, R. Random walks and diffusion on networks. *Phys. Rep.* **2017**, *716–717*, 1–58. [CrossRef]
6. Bronstein, M. Do We Need Deep Graph Neural Networks? Available online: https://towardsdatascience.com/do-we-need-deep-graph-neural-networks-be62d3ec5c59 (accessed on 17 November 2021).
7. Weisfeiler, B.; Leman, A. A reduction of a graph to a canonical form and an algebra arising during this reduction. *Nauchno-Tech. Inf.* **1968**, *2*, 12–16.
8. Morris, C.; Ritzert, M.; Fey, M.; Hamilton, W.L.; Lenssen, J.E.; Rattan, G.; Grohe, M. Weisfeiler and Leman Go Neural: Higher-Order Graph Neural Networks. In Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence, Honolulu, HI, USA, 27 January–1 February 2019.
9. Li, G.; Müller, M.; Ghanem, B.; Koltun, V. Training Graph Neural Networks with 1000 layers. In Proceedings of the International Conference on Machine Learning (ICML), Online, 18–24 July 2021.
10. Kipf, T.N.; Welling, M. Semi-Supervised Classification with Graph Convolutional Networks. In Proceedings of the International Conference on Learning Representations, Toulon, France, 24–26 April 2017.
11. Hamilton, W.L.; Ying, R.; Leskovec, J. Inductive Representation Learning on Large Graphs. In Proceedings of the Conference on Neural Information Processing Systems, Long Beach, CA, USA, 4–9 December 2017.
12. Veličković, P.; Cucurull, G.; Casanova, A.; Romero, A.; Liò, P.; Bengio, Y. Graph Attention Networks. In Proceedings of the International Conference on Learning Representations, Vancouver, BC, Canada, 30 April–3 May 2018.
13. Ying, Z.; You, J.; Morris, C.; Ren, X.; Hamilton, W.L.; Leskovec, J. Hierarchical Graph Representation Learning with Differentiable Pooling. In Proceedings of the Conference on Neural Information Processing Systems, Montréal, QC, Canada, 6–14 December 2018.
14. Page, L.; Brin, S.; Motwani, R.; Winograd, T. The PageRank citation ranking: Bringing order to the Web. In Proceedings of the 7th International World Wide Web Conference (WWW), Brisbane, Australia, 14–18 April 1998 .
15. Lofgren, P. Efficient Algorithms for Personalized PageRank. *arXiv* **2015**, arXiv:1512.04633.
16. Tong, H.; Faloutsos, C.; Pan, J. Fast Random Walk with Restart and Its Applications. In Proceedings of the International Conference on Data Mining, Hong Kong, China, 18–22 December 2006.

17. Jin, W.; Jung, J.; Kang, U. Supervised and extended restart in random walks for ranking and link prediction in networks. *PLoS ONE* **2019**, *14*, e0213857. [CrossRef] [PubMed]
18. He, J.; Li, M.; Zhang, H.J.; Tong, H.; Zhang, C. Manifold-Ranking Based Image Retrieval. In Proceedings of the 12th Annual ACM International Conference on Multimedia, New York, NY, USA, 10–16 October 2004; Association for Computing Machinery: New York, NY, USA, 2004; pp. 9–16. [CrossRef]
19. Shervashidze, N.; Schweitzer, P.; Leeuwen, E.J.; Mehlhorn, K.; Borgwardt, K.M. Weisfeiler-Lehman graph kernels. *J. Mach. Learn. Res.* **2011**, *12*, 2539–2561.
20. Xu, K.; Hu, W.; Leskovec, J.; Jegelka, S. How Powerful are Graph Neural Networks? In Proceedings of the 7th International Conference on Learning Representations (ICLR), New Orleans, LA, USA, 6–9 May 2019.
21. Lee, J.B.; Rossi, R.A.; Kim, S.; Ahmed, N.K.; Koh, E. Attention Models in Graphs: A Survey. *arXiv* **2018**, arXiv:1807.07984.
22. Lee, J.B.; Rossi, R.A.; Kong, X. Graph Classification using Structural Attention. In Proceedings of the International Conference on Knowledge Discovery and Data Mining, London, UK, 19–23 August 2018.
23. Zhang, J.; Shi, X.; Xie, J.; Ma, H.; King, I.; Yeung, D.Y. GaAN: Gated Attention Networks for Learning on Large and Spatiotemporal Graphs. In Proceedings of the Uncertainty in Artificial Intelligence, Monterey, CA, USA, 6–10 August 2018.
24. Cangea, C.; Veličković, P.; Jovanović, N.; Kipf, T.; Liò, P. Towards Sparse Hierarchical Graph Classifiers. In Proceedings of the NeurIPS Workshop on Relational Representation Learning, Montréal, QC, Canada, 3–8 December 2018.
25. Diehl, F.; Brunner, T.; Truong Le, M.; Knoll, A. Towards Graph Pooling by Edge Contraction. In Proceedings of the ICML Workshop on Learning and Reasoning with Graph-Structured Data, Long Beach, CA, USA, 9–15 June 2019.
26. Gao, H.; Ji, S. Graph U-Nets. 2019. Available online: http://xxx.lanl.gov/abs/1905.05178 (accessed on 20 February 2021).
27. Lee, J.; Lee, I.; Kang, J. Self-Attention Graph Pooling. In Proceedings of the International Conference on Machine Learning, Long Beach, CA, USA, 10–15 June 2019.
28. Murphy, R.L.; Srinivasan, B.; Rao, V.A.; Ribeiro, B. Relational Pooling for Graph Representations. In Proceedings of the International Conference on Machine Learning, Long Beach, CA, USA, 10–15 June 2019.
29. Sen, P.; Namata, G.; Bilgic, M.; Getoor, L.; Galligher, B.; Eliassi-Rad, T. Collective Classification in Network Data. *AI Mag.* **2008**, *29*, 93. [CrossRef]
30. Kersting, K.; Kriege, N.M.; Morris, C.; Mutzel, P.; Neumann, M. Benchmark Data Sets for Graph Kernels. 2016. Available online: https://ls11-www.cs.tu-dortmund.de/staff/morris/graphkerneldatasets#citing_this_website (accessed on 10 February 2021).
31. Yang, Z.; Cohen, W.W.; Salakhutdinov, R. Revisiting Semi-Supervised Learning with Graph Embeddings. In Proceedings of the International Conference on Machine Learning, New York, NY, USA, 19–24 June 2016.
32. Knyazev, B.; Taylor, G.; Amer, M. Understanding Attention in Graph Neural Networks. In Proceedings of the ICLR RLGM Workshop, New Orleans, LA, USA, 6–9 May 2019 .
33. Wei, Z.; He, X.; Xiao, X.; Wang, S.; Shang, S.; Wen, J. TopPPR: Top-k Personalized PageRank Queries with Precision Guarantees on Large Graphs. In Proceedings of the 2018 International Conference on Management of Data, Houston, TX, USA, 10–15 June 2018; pp. 441–456. [CrossRef]
34. Wang, S.; Yang, R.; Wang, R.; Xiao, X.; Wei, Z.; Lin, W.; Yang, Y.; Tang, N. Efficient Algorithms for Approximate Single-Source Personalized PageRank Queries. *ACM Transact. Data. Syst.* **2019**, *44*, 1–37. [CrossRef]
35. Lofgren, P.; Banerjee, S.; Goel, A. Personalized PageRank Estimation and Search: A Bidirectional Approach. In Proceedings of the Ninth ACM International Conference on Web Search and Data Mining, San Francisco, CA, USA, 22–25 February 2016; Association for Computing Machinery, New York, NY, USA, 22–25 February 2016; pp. 163–172. [CrossRef]
36. Wang, S.; Yang, R.; Xiao, X.; Wei, Z.; Yang, Y. FORA: Simple and Effective Approximate Single-Source Personalized PageRank. In Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Halifax, NS, Canada, 13–17 August 2017; pp. 505–514.
37. Klicpera, J.; Weißenberger, S.; Günnemann, S. Diffusion Improves Graph Learning. In Proceedings of the Conference on Neural Information Processing Systems, Online, 8–14 December 2019.
38. Ying, R.; He, R.; Chen, K.; Eksombatchai, P.; Hamilton, W.L.; Leskovec, J. Graph Convolutional Neural Networks for Web-Scale Recommender Systems. In Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, London, UK, 19 July 2018; pp. 974–983. [CrossRef]
39. Zhang, C.; Song, D.; Huang, C.; Swami, A.; Chawla, N.V. Heterogeneous Graph Neural Network. In Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, Anchorage, AK, USA, 4–8 August 2019; pp. 793–803. [CrossRef]
40. Abu-El-Haija, S.; Kapoor, A.; Perozzi, B.; Lee, J. N-GCN: Multi-scale Graph Convolution for Semi-supervised Node Classification. In Proceedings of the Conference on Uncertainty in Artificial Intelligence, Monterey, CA, USA, 6–10 August 2018.
41. Abu-El-Haija, S.; Perozzi, B.; Kapoor, A.; Harutyunyan, H.; Alipourfard, N.; Lerman, K.; Steeg, G.V.; Galstyan, A. MixHop: Higher-Order Graph Convolution Architectures via Sparsified Neighborhood Mixing. In Proceedings of the International Conference on Machine Learning, Long Beach, CA, USA, 10–15 June 2019.
42. Zhuang, C.; Ma, Q. Dual Graph Convolutional Networks for Graph-Based Semi-Supervised Classification. In Proceedings of the 2018 World Wide Web Conference, Lyon, France, 23–27 April 2018; pp. 499–508. [CrossRef]
43. Klicpera, J.; Bojchevski, A.; Günnemann, S. Predict then Propagate: Graph Neural Networks meet Personalized PageRank. In Proceedings of the International Conference on Learning Representations, New Orleans, LA, USA, 6–9 May 2019.

44. Bojchevski, A.; Klicpera, J.; Perozzi, B.; Kapoor, A.; Blais, M.J.; Rozemberczki, B.; Lukasik, M.; Gunnemann, S. Scaling Graph Neural Networks with Approximate PageRank. In Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, Online, 6–10 July 2020; pp. 2464–2473.
45. Gao, H.; Pei, J.; Huang, H. Conditional Random Field Enhanced Graph Convolutional Neural Networks. In Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, Anchorage, AK, USA, 4–8 August 2019; pp. 276–284. [CrossRef]
46. Jiang, B.; Lin, D. Graph Laplacian Regularized Graph Convolutional Networks for Semi-supervised Learning. *arXiv* **2018**, arXiv:1809.09839.
47. Xu, K.; Li, C.; Tian, Y.; Sonobe, T.; Ichi Kawarabayashi, K.; Jegelka, S. Representation Learning on Graphs with Jumping Knowledge Networks. In Proceedings of the International Conference on Machine Learning, Stockholm, Sweden, 10–15 July 2018.
48. Sato, R.; Yamada, M.; Kashima, H. Approximation Ratios of Graph Neural Networks for Combinatorial Problems. In Proceedings of the Conference on Neural Information Processing Systems, Vancouver, BC, Canada, 8–14 December 2019.
49. Loukas, A. What graph neural networks cannot learn: Depth vs width. In Proceedings of the International Conference on Learning Representations (ICLR), Online, 26 April–1 May 2020 .
50. Xu, K.; Li, J.; Zhang, M.; Du, S.S.; Ichi Kawarabayashi, K.; Jegelka, S. What Can Neural Networks Reason About? In Proceedings of the International Conference on Learning Representations (ICLR), Online, 26 April–1 May 2020.
51. Andersen, R.; Chung, F.; Lang, K. Local Graph Partitioning using PageRank Vectors. In Proceedings of the 2006 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06), Berkeley, CA, USA, 21–24 October 2006. [CrossRef]
52. Bahmani, B.; Chowdhury, A.; Goel, A. Fast incremental and personalized PageRank. *Proc. VLDB Endow.* **2010**, *4*, 173–184. [CrossRef]
53. Micali, S.; Zhu, Z.A. Reconstructing markov processes from independent and anonymous experiments. *Discret. Appl. Math.* **2016**, *200*, 108–122. [CrossRef]
54. Ivanov, S.; Burnaev, E. Anonymous walk embeddings. *arXiv* **2018**, arXiv:1805.11921.
55. Srivastava, N.; Hinton, G.; Krizhevsky, A.; Sutskever, I.; Salakhutdinov, R. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *J. Mach. Learn. Res.* **2014**, *15*, 1929–1958.
56. Kingma, D.P.; Ba, J. Adam: A Method for Stochastic Optimization. In Proceedings of the International Conference on Learning Representations, San Diego, CA, USA, 7–9 May 2015.
57. Clevert, D.A.; Unterthiner, T.; Hochreiter, S. Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs). In Proceedings of the International Conference on Learning Representations, San Juan, PR, USA, 4–6 May 2016.
58. Ioffe, S.; Szegedy, C. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In Proceedings of the International Conference on Learning Representations, San Diego, CA, USA, 7–9 May 2015.
59. Kendall, M.G. The Treatment of Ties in Ranking Problems. *Biometrika* **1945**, *33*, 239–251. [CrossRef] [PubMed]