Graph Learning with 1D Convolutions on Random Walks

Jan Toenshoff
RWTH Aachen University
toenshoff@informatik.rwth-aachen.de

Martin Ritzert
RWTH Aachen University
ritzert@informatik.rwth-aachen.de

Hinrikus Wolf
RWTH Aachen University
hinrikus@cs.rwth-aachen.de

Martin Grohe
RWTH Aachen University
grohe@informatik.rwth-aachen.de

Abstract

We propose CRAWL (**C**NNs for **Ra**ndom **W**a**l**ks), a novel neural network architecture for graph learning. It is based on processing sequences of small subgraphs induced by random walks with standard 1D CNNs. Thus, CRAWL is fundamentally different from typical message passing graph neural network architectures. It is inspired by techniques counting small subgraphs, such as the graphlet kernel and motif counting, and combines them with random walk based techniques in a highly efficient and scalable neural architecture. We demonstrate empirically that CRAWL matches or outperforms state-of-the-art GNN architectures across a multitude of benchmark datasets for classification and regression on graphs.

1 Introduction

Graph data is ubiquitous across multiple domains, reaching from cheminformatics and social network analysis to knowledge graphs. Being able to effectively learn on such graph data is thus extremely important. We propose a novel neural network architecture called CRAWL (**C**NNs for **Ra**ndom **W**a**l**ks) that is based on random walks and standard 1D CNNs. Essentially, CRAWL samples a set of random walks and extracts features that fully describe the subgraphs visible within a sliding window over these walks. The walks with the subgraph features are then processed with standard 1D convolutions. We experimentally verify that this approach consistently achieves state-of-the-art performance. For example, CRAWL outperforms all other approaches on the standard graph learning benchmarks MOLPCBA (graph classification; Hu et al., 2020) and ZINC (graph regression; Dwivedi et al., 2020).

The CRAWL architecture was originally motivated from the empirical observation that in many application scenarios random walk based methods perform surprisingly well in comparison with graph neural networks (GNNs). A notable example is node2vec (Grover & Leskovec, 2016) in combination with various classifiers. A second observation is that standard GNNs are not very good at detecting small subgraphs, for example, cycles of length 6 (Morris et al., 2019; Xu et al., 2019).The distribution of such subgraphs in a graph carries relevant information about the structure of a graph, as witnessed by the extensive research on motif detection and counting (e.g. Alon, 2007).

We believe that the key to the strength of CRAWL is a favorable combination of engineering and expressiveness aspects. Even large numbers of random walks can be sampled very efficiently. Once the random walks are available, we can rely on existing highly optimized code for 1D CNNs, which allows us to fully exploit the strengths of modern hardware. Sampling small subgraphs in a sliding window on random walks has the advantage that even in sparse graphs it usually yields meaningful subgraph patterns. In terms of expressiveness, CRAWL detects both the global connectivity structure

Preprint. Under review.

in a graph by sampling longer random walks as well as the full local structure within its window size. The gain in expressiveness compared to GNNs is mainly due to the detailed view on the local structure in the sliding window, which standard message passing GNNs (e.g. Gilmer et al., 2017) do not have. We show that the expressiveness of CRAWL is incomparable to that of GNNs (Theorem 1). In particular, CRAWL even detects features that are not even accessible by higher-order GNNs.

CRAWL empirically outperforms advanced message passing GNN architectures on major benchmark datasets. On the molecular regression dataset ZINC (Dwivedi et al., 2020), CRAWL improves the best results currently listed on the leaderboard by roughly 40% (and 20% compared to the best published approach). CRAWL also places first on the leaderboard for MOLPCBA, a large molecular property prediction dataset from the OGB Project (Hu et al., 2020).

A basic requirement for graph learning methods is their isomorphism invariance, which guarantees that the result of a computation only depends on the structure and not on the specific representation of the input graph. A CRAWL model represents a random variable defined on graphs. This random variable is invariant (in the sense of Maron et al., 2019b), which means that it does not depend on a particular node numbering, but only on the isomorphism type of the input graph. Note that it does not contradict the invariance that on every single random walk that we sample we see the vertices in a specific order and can process the vertices in this order by 1D CNNs.

1.1 Related Work

Over the last few years, message passing GNNs (MPGNNs) have been the dominant type of architecture in all kinds of graph related learning tasks (Wu et al., 2020). Thus, MPGNNs constitute the main baselines in our experiments. Many variants of this architecture exist, such as GCN (Kipf & Welling, 2017), GIN (Xu et al., 2019), GAT (Veličković et al., 2018), GraphSage (Hamilton et al., 2017), and GatedGCN (Bresson & Laurent, 2017). A novel variant of MPGNNs is PNA (Corso et al., 2020) which combines multiple types of local aggregation to improve performance. Another recent advance is DeeperGCN (DGCN) by Li et al. (2020) which is designed for significantly deeper GNNs. PHC-GNNs (Le et al., 2021) are GNNs with complex and hypercomplex feature vectors with learned multiplication strategies.

Multiple extensions to the standard message passing framework have been proposed that strengthen the theoretical expressiveness which otherwise is bounded by the 1-dimensional Weisfeiler-Leman algorithm. With 3WLGNN, Maron et al. (2019a) suggested a higher-order GNN, which is equivalent to the 3-dimensional Weisfeiler-Leman kernel and thus more expressive than standard MPGNNs. In HIMP (Fey et al., 2020), the backbone of a molecule graph is extracted and then two GNNs are run in parallel on the backbone and the full molecule graph. This allows HIMP to detect structural features that are otherwise neglected. Explicit counts of fixed substructures such as cycles or small cliques have been added to the node and edge features by Bouritsas et al. (2020) (GSN). Similarly, Sankar et al. (2017), Lee et al. (2019), and Peng et al. (2020) added the frequencies of motifs, i.e., common connected induced subgraphs, to improve the predictions of GNNs. Sankar et al. (2020) introduce motif-based regularization, a framework that improves multiple MPGNNs. A novel approach with strong empirical performance is GINE+ (Brossard et al., 2020). It is based on GIN and aggregates information from higher-order neighborhoods, allowing it to detect small substructures such as cycles. Combining GINE+ with APPNP (Klicpera et al., 2019), a propagation scheme based on the personalized pagerank, improves the performance even further (Rozemberczki, 2019). Beaini et al. (2020) proposed DGN, which incorporates directional awareness into message passing.

A different way to learn on graph data is to use similarity measures on graphs with graph kernels (Kriege et al., 2020). Graph kernels often count induced subgraphs such as graphlets, label sequences, or subtrees, which relates them conceptually to our approach. The graphlet kernel (Shervashidze et al., 2009) counts the occurrences of all 5-node (or more general k-node) subgraphs. The Weisfeiler-Leman kernel Shervashidze et al. (2011) is based on iterated degree sequences and effectively counts occurrences of local subtrees. The Weisfeiler-Leman algorithm is the traditional yardstick for the expressiveness of GNN architectures.

A few previous approaches utilize either random walks or conventional CNNs in the context of end-to-end graph learning, two concepts our method is also based on. Nikolentzos & Vazirgiannis (2020) propose a differentiable version of the random walk kernel and integrate it into a GNN architecture. In Geerts (2020), the ℓ-walk MPGNN adds random walks directly as features to the

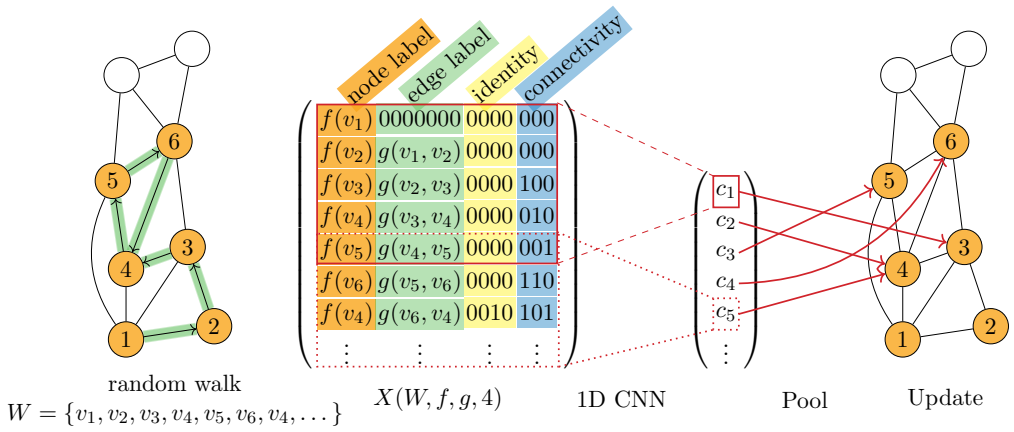

Figure 1: Example of the information flow in a CRAWL layer for a graph with 8 nodes. We sample a walk W and compute the feature matrix X based on node embeddings f, edge embeddings g, and a window size of $s = 4$. To this matrix we apply a 1D CNN with receptive field $r = 5$ and pool the output into the nodes to update their embeddings.

nodes and connects the architecture theoretically to the 2-dimensional Weisfeiler-Leman algorithm. Patchy-SAN (Niepert et al., 2016) normalizes graphs in such a way that they can be interpreted by CNN layers. Zhang et al. (2018) proposed a pooling strategy based on sorting intermediate node embeddings and presented DGCNN which applies a 1D CNN to the sorted embeddings of a graph. Recently, Yuan & Ji (2021) used a 1D CNN layer and attention for neighborhood aggregation to compute node embeddings.

2 Method

CRAWL processes random walks with convolutional neural networks. We initially sample a large enough set of (relatively long) random walks. Each CRAWL layer uses these walks to update a latent node embedding as follows. For each of the walks, the layer constructs features that contain the sequences of node and edge labels that occurred. Additionally, for every position in each walk, the features encode to which of its s predecessors the current node is identical or adjacent. The window size s is a hyperparameter of the model. These walk features are then processed by a 1D CNN. The output of the CNN is an embedding for every position in each random walk. These embeddings are pooled into the nodes, that is, for each node in the graph, we average over the embeddings of all the positions at which it occurs in the random walks. Finally, the CRAWL layer uses a simple MLP to produce the new node embedding from this information. In the full network, each layer uses the embedding produced by the preceding layer as node labels. After the last layer, the node embeddings can be pooled to perform graph level tasks.

Effectively, through the CNN, we extract structural information from many small subgraphs of the size of the CNN's receptive field. Those subgraphs are always connected since they are induced from the nodes of a random walk.

The process of sampling random walks in a graph is not deterministic and therefore the final output of CRAWL is a random variable. However, the output of a trained CRAWL model has low variance such that the inherent randomization does not limit our method's usefulness in real world applications.

2.1 Random Walks

A walk of length $\ell \in \mathbb{N}$ in a graph $G = (V, E)$ is a sequence of nodes $(v_0, \dots, v_\ell) \in V^{\ell+1}$ with $v_{i-1}v_i \in E$ for all $i \in [\ell]$. A random walk in a graph is obtained by starting at some initial node $v_0 \in V$ and then iteratively sampling the next node v_{i+1} randomly from the neighbors $N_G(v_i)$ of the current node v_i. We consider two different random walk strategies: *uniform* and *non-backtracking*. The uniform walks are obtained by sampling the next node uniformly from all neighbors:

$$v_{i+1} \sim \mathcal{U}\big(N_G(v_i)\big).$$

On sparse graphs with nodes of small degree (such as molecules) this walk strategy has a tendency to backtrack often. This slows the traversal of the graph and interferes with the discovery of long-range patterns. The non-backtracking walk strategy addresses this issue by excluding the previous node from the sampling (unless the degree is one):

$$v_{i+1} \sim \mathcal{D}_{\text{NB}}(v_i) \quad \text{with} \quad \mathcal{D}_{\text{NB}}(v_i) = \begin{cases} \mathcal{U}\big(N_G(v_i)\big), & \text{if } i=0 \vee \deg(v_i)=1 \\ \mathcal{U}\big(N_G(v_i) \setminus \{v_{i-1}\}\big), & \text{else.} \end{cases}$$

The choice of the walk strategy is a hyperparameter of CRAWL. In our experiments the non-backtracking strategy usually performs better as shown in Section 3.5.

CRAWL initially samples m random walks $\Omega = \{W_1, \ldots, W_m\}$ of length ℓ from the input graph G. The values for m and ℓ are not fixed hyperparameters of the model but instead can be chosen at runtime. By default, we start one walk at every node, i.e., $m = |V|$. We noted that reducing the number of walks during training can help against overfitting and of course is a way to reduce the memory footprint which is important for large graphs. If we choose to use fewer random walks, we sample $m = p^* \cdot |V|$ starting nodes uniformly at random from the nodes of the graph with chosen probability p^*. We typically choose $\ell \geq 50$, practically ensuring that each node appears multiple times in the walks. For inference, we choose a larger ℓ of up to 150 which improves the predictions.

While, in theory, every layer of CRAWL may use different random walks, we sample the random walks once in the beginning of a run and then make use of the same walks in every layer. This allows us to increase the number of random walks that each layer may work with as the total number of walks is bounded by the GPU memory. This empirically improves stability in training and also the overall performance.

We call contiguous segments $W[i:j] \coloneqq (w_i, \ldots, w_j)$ of a walk $W = (w_0, \ldots, w_\ell)$ *walklets*. The *center* of a walklet $(w_i, \ldots, w_j)$ of even length $j - i$ is the node $w_{(i+j)/2}$. For each walklet $w = (w_i, \ldots, w_j)$, by $G[w]$ we denote the subgraph induced by G on the set $\{w_i, \ldots, w_j\}$. Note that $G[w]$ is connected as it contains all edges $w_k w_{k+1}$ for $i \leq k < j$ and may contain additional edges. Also note that the w_k are not necessarily distinct.

2.2 Walk Features

Based on the walks and a local window size s, we define feature vectors which can then be processed by 1D CNNs. Those feature vectors consist of four parts: one for node features, one for edge features along the walk, and the last two for local structural information. Figure 1 depicts an example of a walk feature matrix and its use in a CRAWL layer.

Given a walk $W \in V^\ell$ of length $\ell - 1$ in a graph $G = (V, E)$, a d-dimensional node embedding $f : V \to \mathbb{R}^d$, a d'-dimensional edge embedding $g : E \to \mathbb{R}^{d'}$, and a local window size $s > 0$ we define the *walk feature matrix* $X(W, f, g, s) \in \mathbb{R}^{\ell \times d_X}$ with feature dimension $d_X = d + d' + s + (s-1)$ as

$$X(W, f, g, s) = (f_W \; g_W \; I_W^s \; A_W^s).$$

For ease of notation, the first dimensions of the matrices f_W, g_W, I_W^s, A_W^s are indexed from 0 to $\ell - 1$. Here, the *node feature sequence* $f_W \in \mathbb{R}^{\ell \times d}$ and the *edge feature sequence* $g_W \in \mathbb{R}^{\ell \times d'}$ are defined as the concatenation of node and edge features, respectively. Formally,

$$(f_W)_{i,\_} = f(v_i) \qquad \text{and} \qquad (g_W)_{i,\_} = \begin{cases} \mathbf{0}, & \text{if } i = 0 \\ g(v_{i-1} v_i), & \text{else.} \end{cases}$$

We define the *local identity relation* $I_W^s \in \{0,1\}^{\ell \times s}$ and the *local adjacency relation* $A_W^s \in \{0,1\}^{\ell \times (s-1)}$ as

$$(I_W^s)_{i,j} = \begin{cases} 1, & \text{if } i - j \geq 0 \wedge v_i = v_{i-j} \\ 0, & \text{else} \end{cases} \quad \text{and}$$

$$(A_W^s)_{i,j} = \begin{cases} 1, & \text{if } i - j \geq 1 \wedge v_i v_{i-j-1} \in E \\ 0, & \text{else.} \end{cases}$$

Intuitively, I_W^s and A_W^s are binary matrices that contain one row for every node v_i in the walk W. The bitstring for v_i in I_W^s encodes which of the s predecessors of v_i in W are identical to v_i, that is,

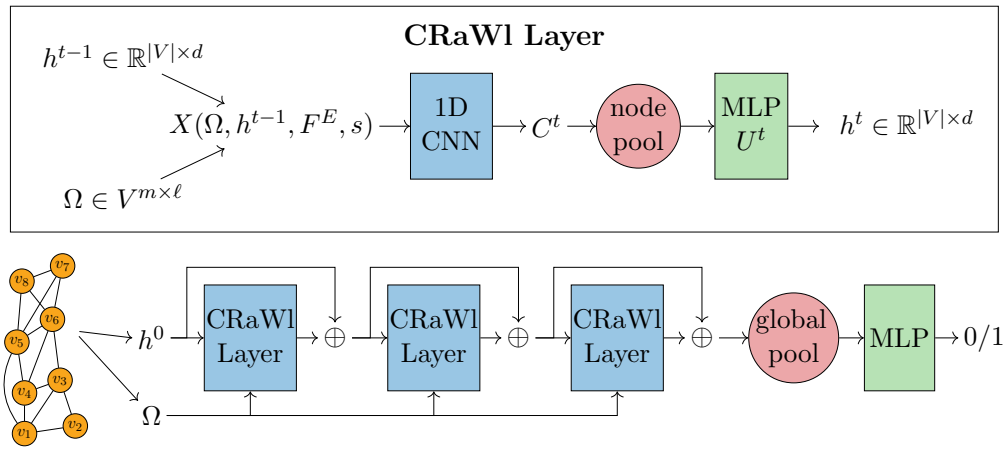

Figure 2: Top: Update procedure of latent node embeddings h^t in a CRAWL layer. Ω is a set of random walks. Bottom: Architecture of a 3-layer CRAWL network as used in the experiments.

where the random walk looped or backtracked. Similarly, A^s_W stores to which of its predecessors v_i has an edge in G. The direct predecessor v_{i-1} must share an edge with v_i and is thus omitted in A^s_W. Note that we do not leverage edge labels of edges not on the walk, only the existence of such edges within the local window is encoded in A^s_W.

For any walklet $w = W[i : i+s]$, the restriction of the walk feature matrix to rows $i, \ldots, i+s$ contains a full description about the induced subgraphs $G[w]$. Hence, when we apply a CNN with receptive field of size at most $s+1$ to the walk feature matrix, the CNN filter has full access to the subgraph induced by the walklet within its scope.

Let $\Omega = \{W_1, \ldots, W_m\}$ be the sampled set of walks. By stacking the individual feature matrices for each walk, we get the *walk feature tensor* $X(\Omega, f, g, s) \in \mathbb{R}^{m \times \ell \times d_X}$ defined as

$$X(\Omega, f, g, s)_i = X(W_i, f, g, s).$$

2.3 CRAWL Layer

A CRAWL network iteratively updates latent embeddings for each node. Let $G = (V, E)$ be a graph with initial node and edge feature maps $F^V : V \to \mathbb{R}^{d_V}$ and $F^E : E \to \mathbb{R}^{d_E}$. The function $h^t : V \to \mathbb{R}^{d_t}$ stores the output of the t-th layer of CRAWL and the initial node features are stored in $h^0 = F^V$. In principle, the size of the output node embedding d_t is an independent hyperparameter for each layer. In practice, we use the same size d for the output node embeddings of all layers for simplicity.

The t-th layer of a CRAWL network constructs the walk feature tensor $X^t = X(\Omega, h^{t-1}, F^E, s)$ using h^{t-1} as its input node embedding and the graph's edge features F^E. This walk feature tensor is then processed by a convolutional network CNN^t based on 1D CNNs. The first dimension of X^t of size m is viewed as the batch dimension. The convolutional filters move along the second dimension (and therefore along each walk) while the third dimension contains the feature channels. Each CNN^t consists of 3 convolutional layers combined with ReLU activations and batch normalization. A detailed description is provided in Appendix B. The stack of operations has a receptive field of $s+1$ and effectively applies an MLP to each subsection of this length in the walk feature matrices. In each CNN^t, we use *Depthwise Separable Convolutions* (Chollet, 2017) for efficiency. Each such CNN network uses $\mathcal{O}(d^2 + sd)$ trainable parameters. Both the time and the memory complexity of applying CNN^t to X^t are therefore in $\mathcal{O}\big(m \cdot \ell \cdot (d^2 + sd)\big)$.

The output of CNN^t is a tensor $C^t \in \mathbb{R}^{m \times (\ell - s) \times d}$. Note that the second dimension is $\ell - s$ instead of ℓ as no padding is used in the convolutions. Through its receptive field, the CNN operates on walklets of size $s+1$ and produces embeddings for those. We pool those embeddings into the nodes of the graph by collecting for each node $v \in V$ all embeddings of walklets centered at v. Let $\Omega = \{W_1, \ldots, W_m\}$ be a set of walks with $W_i = \{v_{i,1}, \ldots, v_{i,\ell}\}$. Then $C^t_{i,j-s/2} \in \mathbb{R}^d$ is the embedding computed by

CNN^t for the walklet $w = W_i[j - \frac{s}{2} : j + \frac{s}{2}]$ centered at $v_{i,j}$. Then, the pooling operation is given as

$$p^t(v) = \underset{(i,j) \in \text{center}(\Omega, s, v)}{\text{mean}} C^t_{i,j-s/2} \qquad \text{with}$$

$$\text{center}(\Omega, s, v) = \left\{ (i,j) \;\middle|\; v_{i,j} = v,\ i \in [m],\ \frac{s}{2} < j < \ell - \frac{s}{2} \right\}.$$

where $\text{center}(\Omega, s, v)$ encodes the walklets of length $s+1$ in which v occurs as center. An illustration of how the output of the CNN is pooled into the nodes of the graph can be found in Figure 1. The output of the pooling step is a vector $p^t(v) \in \mathbb{R}^d$ for each v. This vector is then processed by a trainable MLP U^t with a single hidden layer of dimension $2d$ to compute the next intermediate node embedding $h^t(v)$. Formally, the update procedure of a CRAWL layer is defined by

$$h^t(v) = U^t\big(p^t(v)\big).$$

The upper part of Figure 2 gives an overview over the elements of a CRAWL layer. The runtime of each CRAWL layer is linear in the number of walk steps $m \cdot \ell$ for the CNN and nodes $|V|$ for the final MLP. The initial generation of the random walks is in $\mathcal{O}\big(m \cdot \ell\big)$. Note that m and ℓ are not fixed hyperparameters. They can be chosen freely at runtime and have no effect on the number of trainable parameters.

2.4 Architecture

The architecture we use in the experiments, as illustrated in Figure 2 (bottom), works as follows. The first step for running CRAWL is to compute a set of random walks Ω as described in Section 2.1. We then apply multiple CRAWL layers with residual connections. In each CRAWL layer, we typically choose $s = 8$. After the final CRAWL layer, we apply batch normalization and a ReLU activation to the latent node embeddings before we perform a global pooling step. As pooling we use either sum-pooling or mean-pooling. Finally, a simple feedforward neural network is used to produce a graph-level output which can then be used in classification and regression tasks. In our experiments, we use either an MLP with one hidden layer of dimension d or a single linear layer.

Since CRAWL layers are based on iteratively updating latent node embeddings, they are fully compatible with conventional message passing layers and related techniques such as virtual nodes (Gilmer et al., 2017; Li et al., 2017; Ishiguro et al., 2019). In our experiments, we use virtual nodes whenever this increases validation performance. A detailed explanation of our virtual node layer is provided in the Appendix. Combining CRAWL with message passing layers is left as future work.

We implemented CRAWL in PyTorch (Paszke et al., 2019; Fey & Lenssen, 2019), a public repository is available at GitHub[1]. Crucially, the random walks and the feature matrices are computed entirely on the GPU, increasing speed and reducing the data exchange between CPU and GPU. As a downside of this approach, the current implementation struggles to stay within the available RAM of most GPUs for large graphs such as those occurring in many node classification tasks.

3 Experiments

Recently, two initiatives were launched by Dwivedi et al. (2020) (Benchmarking GNNs) and Hu et al. (2020) (Open Graph Benchmark, OGB) to improve the experimental standards used in graph learning research. Both projects aim to solve common problems of previous experimental settings. Those problems included varying training and evaluation protocols as well as the use of small datasets without standardized splits into training, validation, and test sets. This made the results hard to compare. Both projects introduced novel benchmark datasets with fixed splits and specified training and evaluation procedures. Here, we will use datasets from both projects to evaluate the empirical capabilities of CRAWL. In Appendix A we provide additional results for some of the formerly more common datasets.

3.1 Datasets

From the OGB project, we use the molecular property prediction dataset MOLPCBA with more than 400k molecules. Each of its 128 binary targets states whether or not a molecule is active towards a

[1] `https://github.com/toenshoff/CRaWl`

Table 1: Performance achieved on ZINC, MNIST, CIFAR10, and MOLPCBA. A result marked with "†" indicates that the parameter budget was smaller than for CRAWL and "∗" marks results where no parameter budget was reported.

	METHOD	ZINC TEST MAE	MNIST TEST ACC (%)	CIFAR10 TEST ACC (%)	MOLPCBA TEST AP
LEADERBOARD	GIN	0.526 ± 0.051	96.485 ± 0.252	55.255 ± 1.527	0.2703 ± 0.0023
	GRAPHSAGE	0.398 ± 0.002	97.312 ± 0.097	65.767 ± 0.308	-
	GAT	0.384 ± 0.007	95.535 ± 0.205	64.223 ± 0.455	-
	GCN	0.367 ± 0.011	90.705 ± 0.218	55.710 ± 0.381	0.2483 ± 0.0037
	3WLGNN	0.303 ± 0.068	95.075 ± 0.961	59.175 ± 1.593	-
	GATEDGCN	0.214 ± 0.006	97.340 ± 0.143	67.312 ± 0.311	-
	MPNN	0.145 ± 0.007	97.690 ± 0.220	70.860 ± 0.270	-
	PNA	0.142 ± 0.010	$\mathbf{97.940 \pm 0.120}$	70.350 ± 0.630	0.2838 ± 0.0035
	DGCN&FLAG	-	-	-	0.2842 ± 0.0043
	PHC-GNN	0.164 ± 0.003	-	-	0.2947 ± 0.0026
	GINE+	-	-	-	0.2917 ± 0.0015
	GINE+&APPNP	-	-	-	0.2979 ± 0.0030
OTHER	HIMP	$^{*}0.151 \pm 0.006$	-	-	-
	DGN	$^{\dagger}0.168 \pm 0.003$	-	$\mathbf{72.840 \pm 0.420}$	-
	GSN	$^{*}0.108 \pm 0.018$	-	-	-
OUR	CRAWL	$\mathbf{0.085 \pm 0.004}$	$\mathbf{97.944 \pm 0.050}$	69.013 ± 0.259	$\mathbf{0.2986 \pm 0.0025}$

particular bioassay (a method that quantifies the effect of a substance on a particular kind of living cells or tissues). The dataset is adapted from the MoleculeNet (Wu et al., 2018) and represents molecules as graphs of atoms. It contains multidimensional node and edge features which encode information such as atomic number and chirality. Additionally, it provides a train/val/test split that separates structurally different types of molecules for a more realistic experimental setting. On MOLPCBA, the performance is measured in terms of the average precision (AP).

From the other initiative, started by Dwivedi et al. (2020), we use 4 datasets. The first dataset ZINC is a molecular regression dataset. It is a subset of 12K molecules from the larger ZINC database. The aim is to predict the *constrained solubility*, an important chemical property of molecules. The node label is the atomic number and the edge labels specify the bond type. The datasets CIFAR10 and MNIST are graph datasets derived from the corresponding image classification tasks and contain 60K and 70K graphs, respectively. The original images are modeled as networks of super-pixels. Both datasets are 10-class classification problems. The last dataset CSL is a synthetic dataset containing 150 *Cyclic Skip Link* graphs (Murphy et al., 2019). Those are 4-regular graphs obtained by adding cords of a fixed length to a cycle. The formal definition and an example are provided in the appendix. The aim is to classify the graphs by their isomorphism class. Since all graphs are 4-regular and no node or edge features are provided, this task is unsolvable for most message passing architectures such as standard GNNs.

3.2 Experimental Setting

We adopt the training procedure specified by Dwivedi et al. (2020). In particular, the learning rate is initialized as 10^{-3} and decays with a factor of 0.5 if the performance on the validation set stagnates for 10 epochs. The training stops once the learning rate falls below 10^{-6}. Dwivedi et al. (2020) also specify that networks need to stay within parameter budgets of either 100K or 500K parameters. This ensures a fairer comparison between different methods. For ZINC, we use the larger budget of 500K parameters. For MNIST, CIFAR10 and CSL we build CRAWL models with the smaller budget of 100K since more baseline results are available in the literature. The OGB Project does not specify a standardized training procedure or parameter budgets. For MOLPCBA, we train for 60 epochs and decay the learning rate once with a factor of 0.1 after epoch 50.

During training, we always set the walk length to $\ell = 50$. For evaluation, we use walks of length $\ell = 150$, except for MOLPCBA where we use $\ell = 100$ for efficiency. All hyperparameters and the exact number of trainable parameters are listed in the appendix. There, we also specify the sets of hyperparameters we searched for each dataset.

Table 2: Test accuracy achieved on CSL with and without Laplacian eigenvectors (Dwivedi et al., 2020). As these node features already encode the solution, unsurprisingly most models perform well.

METHOD	CSL TEST ACC (%)	CSL+LAP TEST ACC (%)
GIN	≤ 10.0	99.333 ± 1.333
GRAPHSAGE	≤ 10.0	99.933 ± 0.467
GAT	≤ 10.0	99.933 ± 0.467
GCN	≤ 10.0	$\mathbf{100.000 \pm 0.000}$
3WLGNN	95.700 ± 14.850	30.533 ± 9.863
GATEDGCN	≤ 10.0	99.600 ± 1.083
CRAWL	$\mathbf{100.000 \pm 0.000}$	$\mathbf{100.000 \pm 0.000}$

For each dataset, we train 5 models with different random seeds, except for MOLPCBA where we trained 10 models to meet the submission requirements of the OGB project. We report the mean performance and standard deviation across those models. During inference, the output of each model depends on the sampled random walks. Thus, we evaluate each model on 10 different seeds used for the generation of random walks and take the average of those runs as the model's performance. In the appendix we provide extended results that additionally specify the internal model deviation, that is, the impact of the random walks on the performance. Since this internal model deviation is substantially lower than the differences between the models, they are comparatively insignificant when comparing CRAWL to other models.

3.3 Baselines

We compare the results obtained with CRAWL to a wide range of graph learning methods. We report values that are currently listed on the leaderboard for the benchmark datasets as well as additional results from the literature that are not officially listed yet. Our main baselines are numerous message passing GNN architectures that have been proposed in recent years (see Section 1.1). Additional methods not yet mentioned in Section 1.1 are MPNN by Corso et al. (2020) as well as FLAG (Kong et al., 2020) which was proposed to improve the training of GNNs with adversarial data augmentation.

3.4 Results

Table 1 provides our results on ZINC, MNIST, CIFAR10, and MOLPCBA. On the ZINC dataset, CRAWL achieves an MAE of 0.085. This is approximately a 40% improvement over the current first place (PNA) of the official leaderboard. CRAWL's performance on MNIST dataset is on par with PNA (within standard deviation), which is also the state of the art on this dataset. On CIFAR10, CRAWL achieves the fourth highest accuracy among the eleven compared approaches. For MOLPCBA, we report the baseline results from the leaderboard of the OGB project. On MOLPCBA, CRAWL yields state-of-the-art results and beats all other architectures.

The results on CSL are reported in Table 2. We consider two variants of CSL, the pure task and an easier variant in which node features based on Laplacian eigenvectors are added as suggested by Dwivedi et al. (2020). Without additional node features, CRAWL achieves an accuracy of 100% which indicates that the task is comparatively easy for it. None of the 5 trained CRAWL models misclassified a single graph in the test folds. 3WLGNN is theoretically capable of solving the task without additional node features but unlike CRAWL does not achieve 100% accuracy. Without the Laplacian features that essentially encode the solution, MPGNNs cannot distinguish the 4-regular CSL graphs and achieve at most 10% accuracy. With Laplacian features, all but 3WLGNN achieve very good performance.

Overall, CRAWL performs very well on a variety of datasets across several domains.

3.5 Ablation Study

In an ablation study on the ZINC, MOLPCBA, and CSL datasets, we evaluated the importance of the identity and adjacency encodings in the walk features and the effects of uniform walks and non-backtracking walks. The structural encodings improve CRAWL's performance on all three datasets.

On ZINC and CSL, the structural encodings give a significant benefit on the performance, while on MOLPCBA the improvement is only marginal. Furthermore, non-backtracking walks consistently outperform uniform walks on all three datasets. The detailed results are provided in Appendix C.

4 Expressiveness

In this section, we report on theoretical results comparing the expressiveness of CRAWL with that of other methods. The additional strength of CRAWL is mainly derived from the fact that it detects small subgraphs (of size determined by the window size hyperparameter s) and can sample such subgraphs from a non-uniform, but well-defined, distribution determined by the random walks. In this sense, it is similar to network analysis techniques based on motif detection (Alon, 2007) and graph kernels based on counting subgraphs, such as the graphlet kernel (Shervashidze et al., 2009).

The following results are concerned with the basic expressiveness question which graphs can be distinguished by the various methods, assuming that optimal parameters for the models are available. They do not discuss how such parameters can be learned. This limits the scope of these results, but they still give useful intuition about the different approaches.

It is known that the expressiveness of GNNs corresponds exactly to that of the 1-dimensional Weisfeiler-Leman algorithm (1-WL) (Morris et al., 2019; Xu et al., 2019), in the sense that two graphs are distinguished by 1-WL if and only if they can be distinguished by a GNN. It is also known that higher-dimensional versions of WL characterize the expressiveness of higher-order GNNs (Morris et al., 2019).

Theorem 1. *(1) For every $k \geq 1$ there are graphs that are distinguishable by* CRAWL*, but not by k-WL (and hence not by k-dimensional GNNs). (2) There are graphs that are distinguishable by 1-WL (and hence by GNNs), but not by* CRAWL*.*

We state a precise quantitative version of the theorem and give a proof in Appendix D. Let us just note that for assertion (1) we need a window size s and walk length ℓ quadratic in k. However, the execution cost of CRAWL remains linear in the graph size n, compared to the $\Omega(n^k)$-execution cost for even a single layer of k-dimensional GNN. For assertion (2) of the theorem, we can allow CRAWL to use a window size and path length linear in the size of the graphs. It can also be shown that CRAWL with a window size polynomial in the size of the graphlets is strictly more expressive than graphlet kernels. We omit the precise result, which can be proved similarly to Theorem 1 (1), due to space limitations.

Let us finally remark that the expressiveness of GNNs can be considerably strengthened by adding a random node initialization (Abboud et al., 2020; Sato et al., 2020). The same can be done for CRAWL, but so far the need for such a strengthening (at the cost of a higher variance) did not arise.

5 Conclusion

We have introduced a novel neural network architecture CRAWL for graph learning that is based on random walks and 1D CNNs. Thus, CRAWL is fundamentally different from standard graph neural networks. We demonstrated that this approach works very well across a variety of graph level tasks and is able to outperform state-of-the-art GNNs. By construction, CRAWL can detect arbitrary substructures up to the size of its local window. In particular, on the regular graphs of CSL where pure MPGNNs fail because of the lack of expressiveness, CRAWL is able to extract useful features and solve this task. Future work includes extending the experimental framework to node-level tasks and to motif counting. In both cases, one needs to scale CRAWL to work on individual large graphs instead of many medium sized ones.

CRAWL can be viewed as an attempt to process random walks and the structures they induce with end-to-end neural networks. The strong empirical performance demonstrates the potential of this general approach. However, many variations remain to be explored, including different walk strategies, variations in the walk features, and alternative pooling functions for pooling walklet embeddings into nodes or edges. In view of the incomparability of the expressiveness of GNNs and CRAWL, hybrid approaches that interleave CRAWL layers and GNN layers seem attractive as well. Beyond plain 1D-CNNs, other deep learning architectures for sequential data, such as transformer networks, could be used to process random walks.

Acknowledgements

This work is supported by the German Research Foundation (DFG) under grants GR 1492/16-1 and GRK 2236 UnRAVeL.

References

Abboud, R., Ceylan, İ. İ., Grohe, M., and Lukasiewicz, T. The surprising power of graph neural networks with random node initialization. *arXiv preprint arXiv:2010.01179*, 2020.

Aleliunas, R., Karp, R., Lipton, R., Lovász, L., and Rackoff, C. Random walks, universal traversal sequences, and the complexity of maze problem. In *FOCS79*, pp. 218–223, 1979.

Alon, U. Network motifs: theory and experimental approaches. *Nature Reviews Genetics*, 8:450–461, 2007. doi: 10.1038/nrg2102.

Beaini, D., Passaro, S., Létourneau, V., Hamilton, W. L., Corso, G., and Liò, P. Directional graph networks. *arXiv preprint arXiv:2010.02863*, 2020.

Bouritsas, G., Frasca, F., Zafeiriou, S., and Bronstein, M. M. Improving graph neural network expressivity via subgraph isomorphism counting. *arXiv preprint arXiv:2006.09252*, 2020.

Bresson, X. and Laurent, T. Residual gated graph convnets. *arXiv preprint arXiv:1711.07553*, 2017.

Brossard, R., Frigo, O., and Dehaene, D. Graph convolutions that can finally model local structure. *arXiv preprint arXiv:2011.15069*, 2020.

Cai, J., Fürer, M., and Immerman, N. An optimal lower bound on the number of variables for graph identification. *Combinatorica*, 12:389–410, 1992.

Chollet, F. Xception: Deep learning with depthwise separable convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

Corso, G., Cavalleri, L., Beaini, D., Liò, P., and Veličković, P. Principal neighbourhood aggregation for graph nets. *arXiv preprint arXiv:2004.05718*, 2020.

Du, S. S., Hou, K., Salakhutdinov, R. R., Poczos, B., Wang, R., and Xu, K. Graph neural tangent kernel: Fusing graph neural networks with graph kernels. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 32, pp. 5723–5733. Curran Associates, Inc., 2019. URL `https://proceedings.neurips.cc/paper/2019/file/663fd3c5144fd10bd5ca6611a9a5b92d-Paper.pdf`.

Dwivedi, V. P., Joshi, C. K., Laurent, T., Bengio, Y., and Bresson, X. Benchmarking graph neural networks. *arXiv preprint arXiv:2003.00982*, 2020.

Fey, M. and Lenssen, J. E. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

Fey, M., Yuen, J.-G., and Weichert, F. Hierarchical inter-message passing for learning on molecular graphs. *arXiv preprint arXiv:2006.12179*, 2020.

Geerts, F. Walk message passing neural networks and second-order graph neural networks. *arXiv preprint arXiv:2006.09499*, 2020.

Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. Neural message passing for quantum chemistry. In *Proceedings of the Thirty-Fourth International Conference on Machine Learning (ICML)*, pp. 1263–1272, 2017.

Grohe, M. The logic of graph neural networks. *ArXiv*, 2104.14624 [cs.LG], 2021. URL `https://arxiv.org/abs/2104.14624`.

Grover, A. and Leskovec, J. node2vec: Scalable feature learning for networks. In Krishnapuram, B., Shah, M., Smola, A. J., Aggarwal, C. C., Shen, D., and Rastogi, R. (eds.), *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*, pp. 855–864. ACM, 2016. doi: 10.1145/2939672.2939754.

Hamilton, W. L., Ying, Z., and Leskovec, J. Inductive representation learning on large graphs. pp. 1024–1034, 2017.

Hu, W., Fey, M., Zitnik, M., Dong, Y., Ren, H., Liu, B., Catasta, M., and Leskovec, J. Open graph benchmark: Datasets for machine learning on graphs. *arXiv preprint arXiv:2005.00687*, 2020.

Ishiguro, K., Maeda, S.-i., and Koyama, M. Graph warp module: an auxiliary module for boosting the power of graph neural networks in molecular graph analysis. *arXiv preprint arXiv:1902.01020*, 2019.

Kiefer, S. The Weisfeiler-Leman algorithm: An exploration of its power. *ACM SIGLOG News*, 7(3): 5–27, 2020.

Kipf, T. N. and Welling, M. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations (ICLR)*, 2017.

Klicpera, J., Bojchevski, A., and Günnemann, S. Predict then propagate: Graph neural networks meet personalized pagerank. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL `https://openreview.net/forum?id=H1gL-2A9Ym`.

Kolouri, S., Naderializadeh, N., Rohde, G. K., and Hoffmann, H. Wasserstein embedding for graph learning, 2020.

Kong, K., Li, G., Ding, M., Wu, Z., Zhu, C., Ghanem, B., Taylor, G., and Goldstein, T. Flag: Adversarial data augmentation for graph neural networks, 2020.

Kriege, N. M., Johansson, F. D., and Morris, C. A survey on graph kernels. *Applied Network Science*, 5(1):1–42, 2020.

Le, T., Bertolini, M., Noé, F., and Clevert, D.-A. Parameterized hypercomplex graph neural networks for graph classification. *arXiv preprint arXiv:2103.16584*, 2021.

Lee, J. B., Rossi, R. A., Kong, X., Kim, S., Koh, E., and Rao, A. Graph convolutional networks with motif-based attention. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, pp. 499–508, 2019.

Li, G., Xiong, C., Thabet, A., and Ghanem, B. DeeperGCN: All you need to train deeper GCNs. *arXiv preprint arXiv:2006.07739*, 2020.

Li, J., Cai, D., and He, X. Learning graph-level representation for drug discovery. *arXiv preprint arXiv:1709.03741*, 2017.

Maron, H., Ben-Hamu, H., Serviansky, H., and Lipman, Y. Provably powerful graph networks. pp. 2153–2164, 2019a.

Maron, H., Ben-Hamu, H., Shamir, N., and Lipman, Y. Invariant and equivariant graph networks. In *ICLR*, 2019b.

Morris, C., Ritzert, M., Fey, M., Hamilton, W. L., Lenssen, J. E., Rattan, G., and Grohe, M. Weisfeiler and Leman go neural: Higher-order graph neural networks. In *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence (AAAI)*, pp. 4602–4609, 2019.

Morris, C., Kriege, N. M., Bause, F., Kersting, K., Mutzel, P., and Neumann, M. TUDataset: A collection of benchmark datasets for learning with graphs. In *ICML 2020 Workshop on Graph Representation Learning and Beyond (GRL+ 2020)*, 2020. URL `www.graphlearning.io`.

Murphy, R., Srinivasan, B., Rao, V., and Ribeiro, B. Relational pooling for graph representations. In *International Conference on Machine Learning*, pp. 4663–4673. PMLR, 2019.

Niepert, M., Ahmed, M., and Kutzkov, K. Learning convolutional neural networks for graphs. In *International Conference on Machine Learning*, pp. 2014–2023, 2016.

Nikolentzos, G. and Vazirgiannis, M. Random walk graph neural networks. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H. (eds.), *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. Pytorch: An imperative style, high-performance deep learning library. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 32*, pp. 8024–8035. Curran Associates, Inc., 2019. URL `http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf`.

Peng, H., Li, J., Gong, Q., Ning, Y., Wang, S., and He, L. Motif-matching based subgraph-level attentional convolutional network for graph classification. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 5387–5394, 2020.

Rozemberczki, B. benedekrozemberczki/appnp, 2019. URL `https://github.com/benedekrozemberczki/APPNP`.

Sankar, A., Zhang, X., and Chang, K. C.-C. Motif-based convolutional neural network on graphs. *arXiv preprint arXiv:1711.05697*, 2017.

Sankar, A., Wang, J., Krishnan, A., and Sundaram, H. Beyond localized graph neural networks: An attributed motif regularization framework. *arXiv preprint arXiv:2009.05197*, 2020.

Sato, R., Yamada, M., and Kashima, H. Random features strengthen graph neural networks. *CoRR*, abs/2002.03155, 2020.

Shervashidze, N., Vishwanathan, S., Petri, T., Mehlhorn, K., and Borgwardt, K. Efficient graphlet kernels for large graph comparison. In *Artificial Intelligence and Statistics*, pp. 488–495, 2009.

Shervashidze, N., Schweitzer, P., Van Leeuwen, E. J., Mehlhorn, K., and Borgwardt, K. M. Weisfeiler-Lehman graph kernels. *Journal of Machine Learning Research*, 12(9), 2011.

Veličković, Cucurull, G., Casanova, A., Romero, A., Liò, P., and Bengio, Y. Graph attention networks. 2018.

Wu, Z., Ramsundar, B., Feinberg, E. N., Gomes, J., Geniesse, C., Pappu, A. S., Leswing, K., and Pande, V. MoleculeNet: a benchmark for molecular machine learning. *Chemical science*, 9(2): 513–530, 2018.

Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., and Philip, S. Y. A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems*, 2020.

Xu, K., Hu, W., Leskovec, J., and Jegelka, S. How powerful are graph neural networks? In *Proceedings of the Seventh International Conference on Learning Representations (ICLR)*, 2019.

Yanardag, P. and Vishwanathan, S. Deep graph kernels. In *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*, pp. 1365–1374, 2015.

Yuan, H. and Ji, S. Node2seq: Towards trainable convolutions in graph neural networks. *arXiv preprint arXiv:2101.01849*, 2021.

Zhang, M., Cui, Z., Neumann, M., and Chen, Y. An end-to-end deep learning architecture for graph classification. In *AAAI*, pp. 4438–4445, 2018.

Table 3: Accuracy on Social Datasets.

Method		COLLAB Test Acc	IMDB-MULTI Test Acc	REDDIT-BIN Test Acc
WL-Kernel	(Shervashidze et al., 2011)	78.9 ± 1.9	50.9 ± 3.8	81.0 ± 3.1
WEGL	(Kolouri et al., 2020)	79.8 ± 1.5	52.0 ± 4.1	92.0 ± 0.8
GNTK	(Du et al., 2019)	83.6 ± 1.0	52.8 ± 4.6	-
DGCNN	(Zhang et al., 2018)	73.8 ± 0.5	47.8 ± 0.9	-
3WLGNN	(Maron et al., 2019a)	80.7 ± 1.7	50.5 ± 3.6	-
GIN	(Xu et al., 2019)	80.2 ± 1.9	52.3 ± 2.8	92.4 ± 2.5
GSN	(Bouritsas et al., 2020)	$\mathbf{85.5 \pm 1.2}$	$\mathbf{54.3 \pm 3.3}$	-
CRAWL		$80.40\% \pm 1.50$	$47.77\% \pm 3.87$	$\mathbf{92.75\% \pm 2.16}$

A Extended Results

A.1 Additional Experiments

In this section we evaluate CRAWL on commonly used benchmark datasets from the domain of social networks. We use a subset from the TUDataset (Morris et al., 2020), a list of typically small graph datasets from different domains e.g. chemistry, bioinformatics, and social networks. We focus on three datasets originally proposed by Yanardag & Vishwanathan (2015): COLLAB, a scientific collaboration dataset, IMDB-MULTI, a multiclass dataset of movie collaboration of actors/actresses, and REDDIT-BIN, a balanced binary classification dataset of Reddit users which discussed together in a thread. These datasets do not have any node or edge features and the tasks have to be solved purely with the structure of the graphs.

We stick to the experimental protocol suggested by Xu et al. (2019). Specifically, we perform a 10-fold cross validation. Each dataset is split into 10 stratified folds. We perform 10 training runs where each split is used as test data once, while the remaining 9 are used for training. We then select the epoch with the highest mean test accuracy across all 10 runs. We report this mean test accuracy as the final result. This is not the most realistic setup for simulating real world tasks, since there is no clean split between validation and test data. But in fact, it is the most commonly used experimental setup for these datasets and is mainly justified by the comparatively small number of graphs. Therefore, we adopt the same procedure for the sake of comparability to the previous literature. For COLLAB and IMDB-MULTI we use the same 10-fold split used by Zhang et al. (2018). For REDDIT-BIN we computed our own stratified splits. We also computed separate stratified 10-fold splits for hyperparameter tuning.

We adapt the training procedure of CRAWL towards this setup. Here, the learning rate decays with a factor of 0.5 in fixed intervals. These intervals are chosen to be 20 epochs on COLLAB and REDDIT-BINARY and as 50 epochs on IMDB-MULTI. We train for 200 epochs on COLLAB and REDDIT-BINARY and for 500 epochs on IMDB-MULTI. This ensures a consistent learning rate profile across all 10 runs for each dataset.

Table 3 reports the achieved accuracy of CRAWL and several key baselines on those datasets. For the baselines, we provide the results as reported in the literature. For comparability, we only report values for baselines with the same experimental protocol. On IMDB-MULTI, the smallest of the three datasets, CRAWL yields a slightly lower accuracy than most baselines. On COLLAB, our method performs similarly to standard MPGNN architectures such as GIN. CRAWL outperforms all baselines that report values for REDDIT-BIN. Note that GSN, the method with the best results on COLLAB and IMDB-MULTI, does not scale as well as CRAWL and is infeasible for REDDIT-BIN which contains graphs with several thousand nodes.

A.2 Detailed Results for all Experiments

Table 4 provides the full results from our experimental evaluation. It reports the performance on the train, validation, and test data.

Recall that the output of CRAWL is a random variable. The predictions for a given input graph may vary when different random walks are sampled. To quantify this additional source of randomness,

Table 4: Extended results for CRAWL on all datasets. Note that different metrics are used to measure the performance on the datasets. For each experiment we provide the cross model deviation (CMD) and the internal model deviation (IMD).

Dataset / Model	Metric	Test			Validation			Train	
		Score	CMD	IMD	Score	CMD	IMD	Score	CMD
ZINC	MAE	0.08456	± 0.00352	± 0.00116	0.11398	± 0.00447	± 0.00121	0.04913	± 0.00887
CIFAR10	Acc.	0.69013	± 0.00259	± 0.00158	0.70052	± 0.00307	± 0.00060	0.79180	± 0.01956
MNIST	Acc.	0.97944	± 0.00050	± 0.00055	0.98106	± 0.00110	± 0.00030	0.99044	± 0.00090
CSL	Acc.	1.00000	± 0.00000	± 0.00000	1.00000	± 0.00000	± 0.00000	1.00000	± 0.00000
MOLPCBA	AP	0.29863	± 0.00249	± 0.00055	0.30746	± 0.00195	± 0.00027	0.54889	± 0.01021
COLLAB	Acc.	0.80400	± 0.01497	± 0.00540	-	-	-	0.85827	± 0.00518
IMDB-MULTI	Acc.	0.47767	± 0.03870	± 0.00900	-	-	-	0.44133	± 0.01017
REDDIT-BIN	Acc.	0.92750	± 0.02162	± 0.00450	-	-	-	0.94417	± 0.00702

we measure two deviations for each experiment: The cross model deviation (CMD) and the internal model deviation (IMD). For clarity, let us define these terms formally. For each experiment, we perform $q \in \mathbb{N}$ training runs with different random seeds. Let m_i be the model obtained in the i-th training run with $i \in [q]$. When evaluating (both on test and validation data), we evaluate each model $r \in \mathbb{N}$ times, with different random walks in each evaluation run. Let $p_{i,j} \in \mathbb{R}$ measure the performance achieved by the model m_i in its j-th evaluation run. Note that the unit of $p_{i,j}$ varies between experiments (Accuracy, MAE, ...). We formally define the *internal model deviation* as

$$\text{IMD} = \frac{1}{q} \cdot \sum_{1 \leq i \leq q} \text{STD}\left(\{p_{i,j} \mid 1 \leq j \leq r\}\right),$$

where $\text{STD}(\cdot)$ is the standard deviation of a given distribution. Intuitively, the IMD measures how much the performance of a trained model varies when applying it multiple times to the same input. It quantifies how the model performance depends on the random walks that are sampled during evaluation.

We formally define the *cross model deviation* as

$$\text{CMD} = \text{STD}\left(\left\{\frac{1}{r} \cdot \sum_{1 \leq j \leq r} p_{i,j} \mid 1 \leq i \leq q\right\}\right).$$

The CMD measures the deviation of the average model performance between different training runs. It therefore quantifies how the model performance depends on the random initialization of the network parameters before training.

In the main section, we only reported the CMD for simplicity. Note that the CMD is significantly larger then the IMD across all experiments. Therefore, trained CRAWL models can reliably produce high quality predictions, despite their dependence on randomly sampled walks.

B Model and Setup Details

B.1 Convolution Module

Here, we describe the architecture used for the 1D CNN network CNN^t in each layer t. Let $\text{Conv1D}(d, d', k)$ be a standard 1D convolution with input feature dimension d, output feature dimension d', kernel size k and no bias. This module has $d \cdot d' \cdot k$ trainable parameters. The term scales poorly for larger hidden dimensions d, since the square of this dimension is scaled with an additional factor of k, which we typically set to 9 or more.

To address this issue we leverage *Depthwise Separable Convolutions*, as suggested by Chollet (2017). This method is most commonly applied to 2D data in Computer Vision, but it can also be utilized for 1D convolutions. It decomposes one convolution with kernel size k into two convolutions: The first convolution is a standard 1D convolution with kernel size 1. The second convolution is a depthwise convolution with kernel size k, which convolves each channel individually and therefore only requires $k \cdot d'$ parameters. The second convolution is succeeded by a Batch Norm layer and a ReLU activation function. Note that there is no non-linearity between the two convolutions. These

Table 5: Hyperparameters used in each experiment.

	ZINC	CIFAR10	MNIST	CSL	MOLPCBA	COLLAB	IMDB-M	REDDIT-B
L	3	3	3	2	5	3	3	3
d	147	75	75	90	400	100	100	100
s	8	8	8	8	8	8	8	8
pool	sum	mean	mean	mean	mean	sum	sum	sum
out	mlp	mlp	mlp	mlp	linear	mlp	mlp	mlp
ℓ_{train}	50	50	50	50	50	50	50	50
ℓ_{eval}	150	150	150	150	100	50	50	50
$p*$	1	1	1	1	0.2	1	1	1
walk strat.	nb	nb	nb	nb	nb	nb	nb	nb
r_{val}	2	2	2	5	2	-	-	-
r_{test}	10	10	10	10	10	5	5	5
dropout	0.0	0.0	0.0	0.0	0.25	0.5	0.5	0.5
lr	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001
patience	10	10	10	20	-	-	-	-
batch size	50	50	50	50	100	50	10	10
VN	Yes	No	No	No	Yes	No	No	No

operations effectively simulate a standard convolution with kernel size k but require substantially less memory and runtime.

After the ReLU activation, we apply an additional (standard) convolution with kernel size 1, followed by another ReLU non-linearity. This final convolution increases the expressiveness of our convolution module which could otherwise only learn linearly separable functions. This would limit its ability to distinguish the binary patterns that encode identity and adjacency.

The full stack of operations effectively applies a 2-layer MLP to each sliding window position of the walk feature tensor. Overall, CNN^t is composed of the following operations:
$\text{Conv1D}(d, d', 1) \to \text{Conv1D}^{dw}(d', d', k) \to \text{BatchNorm} \to \text{ReLU} \to \text{Conv1D}(d', d', 1) \to \text{ReLU}$

Here, Conv1D is a standard 1D convolution and Conv1D^{dw} is a depthwise convolution. The total number of parameters of one such module (without the affine transformation of the Batch Norm) is equal to $dd' + kd + {d'}^2$.

B.2 Hyperparameters

Table 5 provides the hyperparameters used in each experiment.

Hyperparameters that define/modify the network architecture:

- The number of layers L.
  We tried out $L \in \{2, 3, 4\}$ on all datasets, except for MOLPCBA, where we searched through $L \in \{3, 5, 7\}$.
- The latent state size d.
  On ZINC, CIFAR10, MNIST and CSL we chose sizes that would roughly use the chosen parameter budgets. On COLLAB, IMDB-MULTI and REDDIT-BIN we set $d = 100$ and for MOLPCBA we chose the largest feasible size for our hardware $(d = 400)$.
- The local window size s.
- The global pooling function (either *mean* or *sum*)
- The architecture of the final output network (either *mlp* or *linear*)
- The number of random walk steps during training (ℓ_{train}) and evaluation (ℓ_{eval}).
- The dropout rate.
  We searched through $\{0.0, 0.25, 0.5\}$. One dropout layer is placed behind the global pooling step.
- Whether or not a virtual node (VN) is used as an intermediate update layer.

Table 6: Number of parameters and runtime for each model in our experiments. The reported times are averaged over all training runs and include the time used to perform a validation run after each training epoch.

MODEL	#PARAM.	TRAIN TIME (H:MM)
ZINC	497 743	0:53
CIFAR10	109 660	6:00
MNIST	109 360	6:00
CSL	104 140	0:01
MOLPCBA	6 115 728	6:22
COLLAB	190 103	1:05
IMDB-MULTI	190 103	1:34
REDDIT-BIN	189 901	2:07

Hyperparameters for the walks and the training procedure:

- The probability of starting a walk from each node during training p^*. We choose $p^* = 1$ by default. On MOLPCBA we set $p^* = 0.2$ to reduce overfitting.
- The walk strategy (either *uniform* (un) or *non-backtracking* (nb))
- The number of evaluation runs with different random walks for validation (r_{val}) and testing (r_{test}).
- The initial learning rate lr was chosen as 0.001 in all experiments.
- The patience for learning rate decay (with a factor of 0.5) is 10 by default.
- The batch size

B.3 Model Size and Runtime

Table 6 provides the number of trainable parameters in each model. Additionally, we report the runtime observed during training. All experiments were run on a machine with 64GB RAM, an Intel Xeon 8160 CPU and an Nvidia Tesla V100 GPU with 16GB GPU memory. The resources were provided by our internal compute cluster.

B.4 Virtual Node

Gilmer et al. (2017), Li et al. (2017), and Ishiguro et al. (2019) suggested the use of a *virtual node* to enhance GNNs for chemical datasets. Intuitively, a special node is inserted into the graph that is connected to all other nodes. This node aggregates the states of all other nodes and uses this information to update its own state. The virtual node has its own distinct update function which is not shared by other nodes. The updated state is then sent back to all nodes in the graph. Effectively, a virtual node allows global information flow after each layer.

Formally, a virtual node updates a latent state $h^t_{vn} \in \mathbb{R}^d$, where h^t_{vn} is computed after the t-th layer and h^0_{vn} is initialized as a zero vector. The update procedure is defined by:

$$h^t_{vn} = U^t_{vn}\left(h^{t-1}_{vn} + \sum_{v \in V} h^t(v)\right)$$
$$\tilde{h}^t(v) = h^t(v) + h^t_{vn}.$$

Here, U^t_{vn} is a trainable MLP and h^t is the latent node embedding computed by the t-th CRAWL layer. $\tilde{h}^t$ is an updated node embedding that is used as the input for the next CRAWL layer instead of h^t. In our experiments, we choose U^t_{vn} to contain a single hidden layer of dimension d. When using a virtual node, we perform this update step after every CRAWL layer, except for the last one.

Note that we view the virtual node as an intermediate update step that is placed between our CRAWL layers to allow for global communication between nodes. No additional node is actually added to

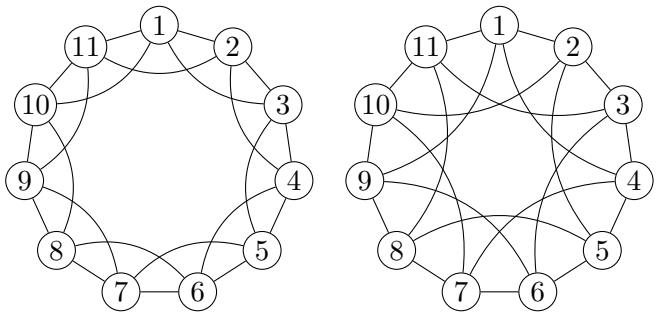

Figure 3: Two cyclic skip-link graphs (see Murphy et al., 2019) with 11 nodes and a skip distance of 2 and 3 respectively.

the graph and, most importantly, the "virtual node" does not occur in the random walks sampled by CRAWL.

B.5 Cross Validation on CSL

Let us briefly discuss the experimental protocol used for the CSL dataset. Unlike the other benchmark datasets provided by Dwivedi et al. (2020), CSL is evaluated with 5-fold cross-validation. We use the 5-fold split Dwivedi et al. (2020) provide in their repository. In each training run, three folds are used for training and one is used for validation and model selection. After training, the remaining fold is used for testing.

Finally, Figure 3 provides an example of two skip-link graphs. The task of CSL is to classify such graphs by their isomorphism class.

C Ablation Study

We perform an ablation study to understand how the key aspects of CRAWL influence the empirical performance. We aim to answer two main questions:

- How useful are the identity and adjacency features we construct for the walks?
- How do different strategies for sampling random walks impact the performance?

Here, we use the ZINC, MOLPCBA, and CSL datasets to answer these questions empirically. We trained multiple versions of CRAWL with varying amounts of structural features used in the walk feature matrices. The simplest version only uses the sequences of node and edge features without any structural information. For ZINC and CSL, we also train intermediate versions using either the identity or the adjacency encoding, but not both. We omit these for MOLPCBA to save computational resources. Finally, we measure the performance of the standard CRAWL architecture, where both encodings are incorporated into the walk feature matrices. For each version, we compute the performance with both walk strategies.

On each dataset, the experimental setup and hyperparameters are identical to those used in the previous experiments on both datasets. In particular, we train five models with different seeds and provide the average performance as well as the standard deviation across models. Note that we repeat the experiment independently for each walk strategy. Switching walk strategies between training and evaluation does not yield good results.

Table 7 reports the performance of each studied version of CRAWL. On ZINC, the networks without any structural encoding yield the worst predictions. Adding either the adjacency or the identity encoding improves the results substantially. The best results are obtained when both encodings are utilized and non-backtracking walks are used. On MOLPCBA, the best performance is also obtained with full structural encodings. However, the improvement over the version without the encodings is only marginal. Again, non-backtracking walks perform significantly better than uniform walks. On

Table 7: Results of our ablation study. Node features F^V, edge features F^E, adjacency encoding A, and identity encoding I. Walk strategies no-backtrack (NB) and uniform (UN).

FEATURES	WALKS	ZINC (MAE)	MOLPCBA (AP)	CSL (ACC)
F^V+F^E	UN	0.19768 ± 0.01159	0.28364 ± 0.00201	0.06000 ± 0.04422
F^V+F^E	NB	0.15475 ± 0.00350	0.29613 ± 0.00209	0.06000 ± 0.04422
F^V+F^E+A	UN	0.10039 ± 0.00514	-	0.97467 ± 0.02587
F^V+F^E+A	NB	0.08656 ± 0.00310	-	0.99933 ± 0.00133
F^V+F^E+I	UN	0.10940 ± 0.00698	-	0.70733 ± 0.07658
F^V+F^E+I	NB	0.09345 ± 0.00219	-	0.97133 ± 0.00859
F^V+F^E+I+A	UN	0.09368 ± 0.00232	0.28522 ± 0.00317	0.96267 ± 0.02037
F^V+F^E+I+A	NB	0.08456 ± 0.00352	0.29863 ± 0.00249	1.00000 ± 0.00000

CSL, the only version to achieve a perfect accuracy of 100% is the one with all structural encodings and non-backtracking walks. Note that the version without any encodings can only guess on CSL since this dataset has no node features (we are not using the Laplacian features here).

Overall, the structural encodings of the walk feature matrices yield a measurable performance increase on all three datasets. However, the margin of the improvement varies significantly and depends on the specific dataset. For some tasks, such as MOLPCBA, CRAWL yields highly competitive results even when only the sequences of node and edge features are considered in the walk feature matrices.

Finally, the non-backtracking walks consistently outperform the uniform walks. This could be attributed to their ability to traverse sparse substructures quickly. On sparse graphs with limited degree, such as molecules, uniform walks will backtrack often. This slows down the traversal of the graph. Each substructure will be traversed less frequently and the average size of the subgraphs induced by the walklets decreases. On the three datasets used here these effects seem to cause a significant loss in performance.

D Theory

In this appendix, we prove Theorem 1 and discuss its context. Let us first recall the setting and introduce some additional notation. Throughout the paper, graphs are undirected and simple (that is, without self-loops and parallel edges).[2] In this appendix, all graphs will be unlabeled. All result can easily be extended to (vertex and edge) labelled graphs. In fact, the (harder) inexpressivity results only become stronger by restricting them to the subclass of unlabelled graphs. We further assume that graphs have no isolated nodes, which enables us to start a random walk from every node. This makes the setup cleaner and avoids tedious case distinctions, but again is no serious restriction.

We denote the edge set of a graph G by $V(G)$ and the node set by $E(G)$. The *order* $|G|$ of G is the number of nodes, that is, $|G| := |V(G)|$. For a set $X \subseteq V(G)$, the *induced subgraph* $G[X]$ is the graph with node set X and edge set $\{vw \in E(G) \mid v, w \in X\}$. A walk of length ℓ in G is a sequence $W = (w_0, \dots, w_\ell) \in V(G)^{\ell+1}$ such that $w_{i-1}w_i \in E(G)$ for $1 \leq i \leq \ell$. The walk is *non-backtracking* if for $1 < i < \ell$ we have $w_{i+1} \neq w_{i-1}$ unless the degree of vertex w_i is 1.

Before we prove the theorem, let us precisely specify what it means that CRAWL distinguishes two graphs. Recall that CRAWL has three (walk related) hyperparameters:

- the *window size* s;
- the *walk length* ℓ;
- the *samples size* m.

Recall furthermore that with every walk $W = (w_0, \dots, w_\ell)$ we associate a *walk feature matrix* $X \in \mathbb{R}^{(\ell+1)\times(d+d'+s+(s-1))}$. For $0 \leq i \leq \ell$, the first d entries of the i-th row of X describe the

[2] It is possible to simulate directed edges and parallel edges through edge labels and loops through node labels, but so far, we have only worked with undirected simple, though possibly labeled graphs.

current embedding of the node w_i, the next d' entries the embedding of the edge $w_{i-1}w_i$ (0 for $i = 0$), the following s entries are indicators for the equalities between w_i and the nodes w_{i-j} for $j = 1, \dots, s$ (1 if $w_i = w_{i-j}$, 0 if $i - j < 0$ or $w_i \neq w_{i-j}$), and the remaining $s - 1$ entries are indicators for the adjacencies between w_i and the nodes w_{i-j} for $j = 2, \dots, s$ (1 if w_i, w_{i-j} are adjacent in G, 0 if $i - j < 0$ or w_i, w_{i-j} are non-adjacent; note that w_i, w_{i-1} are always be adjacent because W is a walk in G). Note that in the unlabelled graphs we consider here the initial node and edge embeddings are the same for all nodes and for all edges, and therefore they do not contribute to the expressivity. As our expressiveness results are based on the initial feature matrices—they carry all information that CRAWL extracts from the graph—we can safely ignore these embeddings and focus on the subgraph features encoded in the last $2s - 1$ columns. For simplicity, we regard X as an $(\ell + 1) \times (2s - 1)$ matrix with only these features in the following. We denote the entries of the matrix X by $X_{i,j}$ and the rows by $X_{i,-}$. So $X_{i,-} = (X_{i,1}, \dots, X_{i,d+d'+2s-1}) \in \{0,1\}^{2s-1}$. We denote the walk feature matrix of a walk W by $X(W)$. It is immediate from the definitions that for walks $W = (w_1, \dots, w_\ell), W' = (w'_1, \dots, w'_\ell)$ in graphs G, G' with feature matrices $X \coloneqq X(W), X' \coloneqq X(W')$, we have:

1. if $X_{i-j,-} = X'_{i-j,-}$ for $j = 0, \dots, s-1$ then the mapping $w_{i-j} \mapsto w'_{i-j}$ for $j = 0, \dots, s$ is an isomorphism from the induced subgraph $G[\{w_{i-j} \mid j = 0, \dots, s\}]$ to the induced subgraph $G'[\{w'_{i-j} \mid j = 0, \dots, s\}]$;
2. if the mapping $w_{i-j} \mapsto w'_{i-j}$ for $j = 0, \dots, 2s-1$ is an isomorphism from the induced subgraph $G[\{w_{i-j} \mid j = 0, \dots, 2s-1\}]$ to the induced subgraph $G'[\{w'_{i-j} \mid j = 0, \dots, 2s-1\}]$, then $X_{i-j,-} = X'_{i-j,-}$ for $j = 0, \dots, s-1$.

The reason that we need to include the vertices $w_{i-2s+1}, \dots, w_{i-s}$ and $w'_{i-2s+1}, \dots, w'_{i-s}$ into the subgraphs in (2) is that row $X_{i-s+1,-}$ of the feature matrix records edges and equalities between w_{i-s+1} and $w_{i-2s+1}, \dots, w_{i-s}$.

For every graph G we denote the distribution of random walks on G starting from a node chosen uniformly at random by $\mathcal{W}(G)$ and $\mathcal{W}_{nb}(G)$ for the non-backtracking walks. We let $\mathcal{X}(G)$ and $\mathcal{X}_{nb}(G))$ be the push-forward distributions on $\{0,1\}^{(\ell+1)\times(2s-1)}$, that is, for every $X \in \{0,1\}^{(\ell+1)\times(2s-1)}$ we let

$$\Pr_{\mathcal{X}(G)}(X) = \Pr_{\mathcal{W}(G)}\big(\{W \mid X(W) = X\}\big)$$

A CRAWL run on G takes m samples from $\mathcal{X}(G)$. So to distinguish two graphs G, G', CRAWL must detect that the distributions $\mathcal{X}(G), \mathcal{X}(G')$ are distinct using m samples.

As a warm-up, let us prove the following simple result.

Theorem 2. *Let G be a cycle of length n and G' the disjoint union of two cycles of length $n/2$. Then G and G' cannot be distinguished by* CRAWL *with window size $s < n/2$ (for any choice of parameters ℓ and m).*

Proof. With a window size smaller than the length of the shortest cycle, the graph CRAWL sees in its window is always a path. Thus for every walk W in either G or G' the feature matrix $X(W)$ only depends on the backtracking pattern of W. This means that $\mathcal{X}(G) = \mathcal{X}(G')$. □

It is worth noting that the graphs G, G' of Theorem 2 can be distinguished by 2-WL (the 2-dimensional Weisfeiler-Leman algorithm), but not by 1-WL.

Proving that two graphs G, G' have identical feature-matrix distributions $\mathcal{X}(G) = \mathcal{X}(G')$ is the ultimate way of proving that they are not distinguishable by CRAWL. Yet for more interesting graphs, we rarely have identical feature-matrix distributions. However, if the distributions are sufficiently close we will still not be able to distinguish them. To quantify closeness, we use the *total variation distance* of the distributions. Recall that the total variation distance between two probability distributions $\mathcal{D}, \mathcal{D}'$ on the same finite sample space Ω is

$$\mathrm{dist}_{TV}(\mathcal{D}, \mathcal{D}') \coloneqq \max_{S \subseteq \Omega} |\Pr_{\mathcal{D}}(S) - \Pr_{\mathcal{D}'}(S)|.$$

It is known that the total variation distance is half the ℓ_1-distance between the distributions, that is,

$$\begin{aligned}\operatorname{dist}_{TV}(\mathcal{D}, \mathcal{D}') &= \frac{1}{2}\|\mathcal{D} - \mathcal{D}\|_1 \\ &= \frac{1}{2}\sum_{\omega\in\Omega} |\Pr_{\mathcal{D}}(\{\omega\}) - \Pr_{\mathcal{D}'}(\{\omega\})|.\end{aligned}$$

Let $\varepsilon > 0$. We say that two graphs G, G' are *ε-indistinguishable* by CRAWL with window size s, walk length ℓ, and sample size m if

$$\operatorname{dist}_{TV}(\mathcal{X}(G), \mathcal{X}(G')) < \frac{\varepsilon}{m}. \tag{1}$$

The rationale behind this definition is that if $\operatorname{dist}_{TV}(\mathcal{X}(G), \mathcal{X}(G')) < \frac{\varepsilon}{m}$ then for every property of feature matrices that CRAWL may want to use to distinguish the graphs, the expected numbers of samples with this property that CRAWL sees in both graphs are close together (assuming ε is small).

Often, we want to make asymptotic statements, where we have two families of graphs $(G_n)_{n\geq 1}$ and $(G'_n)_{n\geq 1}$, typically of order $|G_n| = |G'_n| = \Theta(n)$, and classes S, L, M of functions, such as the class $O(\log n)$ of logarithmic or the class $n^{O(1)}$ of polynomial functions. We say that $(G_n)_{n\geq 1}$ and $(G'_n)_{n\geq 1}$ are *indistinguishable* by CRAWL with window size S, walk length L, and sample size M if for all $\varepsilon > 0$ and all $s \in S, \ell \in L, m \in M$ there is an n such that G_n, G'_n are ε-indistinguishable by CRAWL with window size $s(n)$, walk length $\ell(n)$, and sample size $m(n)$.

We could make similar definitions for distinguishability, but we omit them here and deal with distinguishability in an ad-hoc fashion (in the following subsection).

D.1 Proof of Theorem 1(1)

Here is a precise quantitative version of the first part of Theorem 1.

Theorem 3. *For all $k \geq 1$ there are families of graphs $(G_n)_{n\geq 1}$, $(G'_n)_{n\geq 1}$ of order $|G_n| = |G'_n| = n + O(k)$ that are distinguishable by* CRAWL *with window size $s = O(k^2)$, walk length $s = O(k^2)$, and samples size $m = O(n)$, but not by k-WL (and hence not by k-dimensional GNNs).*

Fortunately, to prove this theorem we do not need to know any details about the Weisfeiler-Leman algorithm (the interested reader is deferred to (Grohe, 2021; Kiefer, 2020)). We can use the following well-known inexpressibility result as a black box.

Theorem 4 (Cai et al. (1992))**.** *For all $k \geq 1$ there are graphs H_k, H'_k such that $|H_k| = |H'_k| = O(k)$, the graphs H_k and H'_k are 3-regular, and k-WL cannot distinguish H_k and H'_k.*

It is a well-known fact that the *cover time* of a connected graph of order n with m edges, that is, the expected time it takes a random walk starting from a random node to visit all nodes of the graph, is bounded from above by $4nm$ (Aleliunas et al., 1979). By Markov's inequality, a path of length $8nm$ visits all nodes with probability at least $1/2$. Sampling several such paths, we can bring the success probability arbitrarily close to 1.

Proof of Theorem 3. Let $k \geq 1$ and let H_k, H'_k be the graphs obtained from the Theorem 4. Let $n_k := |H_k|$. For every $n \geq 1$, we let G_n be the disjoint union of H_k with a path of length n, and let G'_n be defined in the same way from H'_k. Then $|G_n| = |G'_n| = n_k + n + 1$.

Let $m_k = \frac{3}{2}n_k$ be the number of edges of the 3-regular graphs H_k and H'_k, and let $s := 8n_k m_k = O(k^2)$. This will be our window size, and in fact also our walk length: $\ell := s$. Let $\varepsilon > 0$. We choose a sufficiently large $m = m(n) \in O(n)$ to make sure that a sample of m nodes from $V(G_n)$ or $V(G'_n)$ contains sufficiently many nodes in the subset $V(H_k) \subseteq V(G_k)$ resp. $V(H'_k) \subseteq V(G'_k)$. Then, if we sample m paths of length ℓ from $\mathcal{W}(G_n)$, with probability at least $1 - \varepsilon$, one of these paths covers $V(H_k)$. This means that m random walks of length ℓ will detect the subgraphs H_k, and as the window size s is equal to ℓ, these subgraphs will appear in the feature matrix. Since the subgraph H_k does not appear as a subgraph of G'_n, this means that with probability at least $1 - \varepsilon$, CRAWL can distinguish the two graphs. □

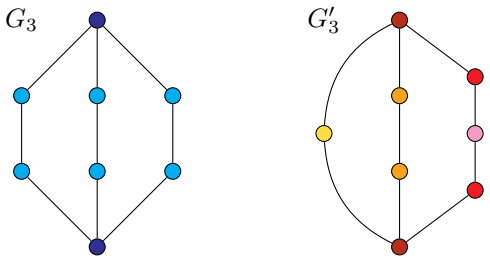

Figure 4: The graphs G_3 and G'_3 in the proof of Theorem 5 with their stable coloring computed by 1-WL

D.2 Proof of Theorem 1(2)

To prove the second part of the theorem, it will be necessary to briefly review the *1-dimensional Weisfeiler-Leman algorithm* (1-*WL)*, which is also known as *color refinement* and as *naive node classification*. The algorithm iteratively computes a partition of the nodes of its input graph. It is convenient to think of the classes of the partition as colors of the nodes. Initially, all nodes have the same color. Then in each iteration step, for all colors c in the current coloring and all nodes v, w of color c, the nodes v and w get different colors in the new coloring if there is some color d such that v and w have different numbers of neighbors of color d. This refinement process is repeated until the coloring is *stable*, that is, any two nodes v, w of the same color c have the same number of neighbors of any color d. We say that 1-WL *distinguishes* two graphs G, G' if, after running the algorithm on the disjoint union $G \uplus G'$ of the two graphs, in the stable coloring of $G \uplus G'$ there is a color c such that G and G' have a different number of nodes of color c.

For the results so far, it has not mattered if we allowed backtracking or not. Here, it makes a big difference. For the non-backtracking version, we obtain a stronger result with an easier proof. The following theorem is a precise quantitative statement of Theorem 1(2).

Theorem 5. *There are families $(G_n)_{n\geq 1}$, $(G'_n)_{n\geq 1}$ of graphs of order $|G_n| = |G'_n| = 3n - 1$ with the following properties.*

1. *For all $n \geq 1$, 1-WL distinguishes G_n and G'_n.*
2. *$(G_n)_{n\geq 1}$, $(G'_n)_{n\geq 1}$ are indistinguishable by the non-backtracking version of* CRAWL *with window size $s(n) = o(n)$ (regardless of the walk length and sample size).*
3. *$(G_n)_{n\geq 1}$ $(G'_n)_{n\geq 1}$ are indistinguishable by* CRAWL *with walk length $\ell(n) = O(n)$, and samples size $m(n) = n^{O(1)}$ (regardless of the window size).*

Proof. The graphs G_n and G'_n both consist of three internally disjoint paths with the same endnodes x and y. In G_n the lengths of all three paths is n. In G'_n, the length of the paths is $n - 1, n, n + 1$ (see Figure 4).

It is easy to see that 1-WL distinguishes the two graphs.

To prove assertion (b), let $s := 2n - 3$. Then the length of the shortest cycle in G_n, G'_n is $s + 2$. Now consider a non-backtracking walk $W = (w_1, \ldots, w_\ell)$ in either G_n or G'_n (of arbitrary length ℓ). Then for all i and j with $i - s \leq j \leq i$ we have $w_i \neq w_j$, and unless $j = i - 1$, there is no edge between w_i and w_j. Thus $X(W) = X(W')$ for all walks W' of the same length ℓ, and since it does not matter which of the two graphs G_n, G'_n the walks are from. It follows that $\mathcal{X}_{nb}(G_n) = \mathcal{X}_{nb}(G'_n)$.

Before we prove (c), we remark that the backtracking version of CRAWL can distinguish $(G_n)_{n\geq 1}$ and $(G'_n)_{n\geq 1}$ with a constant window size 6, walk length $n^{O(1)}$, and samples size $n^{O(1)}$. The reason is that by going back and forth between a node and all its neighbors within its window, CRAWL can distinguish the two degree-3 nodes x, y from the remaining degree-2 nodes. Thus, the feature matrix reflects traversal times between degree-3 nodes, and the distribution of traversal times is different in G_n and G'_n. With sufficiently many samples, CRAWL can detect this.

So, let us turn to the proof that with walks of linear length this is not possible, that is, assertion (c). The reason for this is simple: random walks of length $O(n)$ are very unlikely to traverse a path of length at least $n-1$ from x to y. It is well known that the expected traversal time is $\Theta(n^2)$ (this follows from the analysis of the gambler's ruin problem). However, this does not suffice for us. We need to bound the probability that a path of length $O(n)$ is a traversal. Using a standard, Chernoff type tail bound, it is straightforward to prove that for every constant $c \geq 0$ there is a constant $d \geq 1$ such that the probability that a random walk of length cn in either G_n or G'_n visits both x and y is at most $\exp(-n/d)$. As only walks visiting both x and y can differentiate between the two graphs, this gives us an upper bound of $\exp(-n/d)$ for the total variation distance between $\mathcal{X}(G_n)$ and $\mathcal{X}(G'_n)$. $\square$