machine learning & knowledge extraction

MDPI

Article

Beyond Weisfeiler–Lehman with Local Ego-Network Encodings †

Nurudin Alvarez-Gonzalez [1,*], Andreas Kaltenbrunner [2,3] and Vicenç Gómez [1,*]

[1] Department of Information and Communications Technologies, Universitat Pompeu Fabra, 08018 Barcelona, Spain
[2] Internet Interdisciplinary Institute, Universitat Oberta de Catalunya, 08018 Barcelona, Spain; kaltenbrunner@gmail.com
[3] ISI Foundation, 10126 Turin, Italy
* Correspondence: nuralgon@gmail.com (N.A.-G.); vicen.gomez@upf.edu (V.G.)

† This paper is an extended version of our previous work published in the *non-archival* Extended Abstract track in the First Learning on Graphs Conference (LoG 2022), Virtual, 9–12 December 2022.

Abstract: Identifying similar network structures is key to capturing graph isomorphisms and learning representations that exploit structural information encoded in graph data. This work shows that ego networks can produce a structural encoding scheme for arbitrary graphs with greater expressivity than the Weisfeiler–Lehman (1-WL) test. We introduce IGEL, a preprocessing step to produce features that augment node representations by encoding ego networks into sparse vectors that enrich message passing (MP) graph neural networks (GNNs) beyond 1-WL expressivity. We formally describe the relation between IGEL and 1-WL, and characterize its expressive power and limitations. Experiments show that IGEL matches the empirical expressivity of state-of-the-art methods on isomorphism detection while improving performance on nine GNN architectures and six graph machine learning tasks.

Keywords: graph neural networks; graph representation learning; Weisfeiler–Lehman; graph isomorphism; GNN expressivity; ego networks

check for updates

Citation: Alvarez-Gonzalez, N.; Kaltenbrunner, A.; Gómez, V. Beyond Weisfeiler–Lehman with Local Ego-Network Encodings. *Mach. Learn. Knowl. Extr.* **2023**, *5*, 1234–1265. https://doi.org/10.3390/make5040063

Academic Editor: Simon Tjoa

Received: 1 August 2023
Revised: 10 September 2023
Accepted: 17 September 2023
Published: 22 September 2023

Copyright: © 2023 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

Novel approaches for representation learning on graph-structured data have appeared in recent years [1]. Graph neural networks can efficiently learn representations that depend both on the graph structure and node and edge features from large-scale graph data sets. The most popular choice of architecture is the message passing graph neural network (MP-GNN). MP-GNNs represent nodes by repeatedly aggregating feature 'messages' from their neighbors.

Despite being successfully applied in a wide variety of domains [2–6], there is a limit on the representational power of MP-GNNs provided by the computationally efficient Weisfeiler–Lehman (1-WL) test [7] for checking graph isomorphisms [8,9]. Establishing this connection has lead to a better theoretical understanding of the performance of MP-GNNs and many possible generalizations at the price of additional computational cost [10–14].

To improve the expressivity of MP-GNNs, recent methods have extended the vanilla message passing mechanism is various ways. For example, using higher order k-vertex tuples [9] leading to k-WL generalizations, introducing relative positioning information for network vertices [15], propagating messages beyond direct neighborhoods [16], using concepts from algebraic topology [17], or combining subgraph information in different ways [18–24]. Similarly, provably powerful graph networks (PPGN) [25] have been proposed as an architecture with 3-WL expressivity guarantees, at the cost of quadratic memory and cubic time complexities with respect to the number of nodes. More recently, Balcilar

et al. [26] proposed a novel MP-GNN with linear time and memory complexity, but theoretically more powerful than the 1-WL test, and experimentally as powerful as 3-WL by leveraging a preprocessing, spectral decomposition step, with cubic worst-case time complexity. All aforementioned approaches improve expressivity by extending MP-GNN architectures, often evaluating on standarized benchmarks [27–29]. However, identifying the optimal approach on novel domains requires costly architecture searches.

In this work, we show that a simple encoding of local ego networks is a possible solution to these shortcomings. We present **IGEL**, an **I**nductive **G**raph **E**ncoding of **L**ocal ego network subgraphs allowing MP-GNN and deep neural network (DNN) models to go beyond 1-WL expressivity without modifying existing model architectures. IGEL produces inductive representations that can be introduced into MP-GNN models. IGEL reframes capturing 1-WL information irrespective of model architecture as a preprocessing step that simply extends node/edge attributes. Our main contributions in this paper are:

C1 We present a novel structural encoding scheme for graphs, describing its relationship with existing graph representations and MP-GNNs.

C2 We formally show that the proposed encoding has more expressive power than the 1-WL test, and identify expressivity upper bounds for graphs that match subgraph GNN state-of-the-art methods.

C3 We experimentally assess the performance of nine model architectures enriched with our proposed method on six tasks and thirteen graph data sets and find that it consistently improves downstream model performance.

We structure the paper as follows: In Section 2, we introduce our notation and the required background, including relevant works extending MP-GNNs beyond 1-WL. Then, we describe IGEL in Section 3. We analyze IGEL's expressivity in Section 4 and evaluate its performance experimentally in Section 5. Finally, we discuss our findings and summarize our results in Section 6.

2. Notation and Related Work

Given a graph $G = (V, E)$, we define $n = |V|$ and $m = |E|$, $d_G(v)$ is the degree of a node v in G, and $d_{\texttt{max}}$ is the maximum degree. For $u, v \in V$, $l_G(u, v)$ is their shortest distance, and $\texttt{diam}(G) = \max(l_G(u, v) | u, v \in V)$ is the diameter of G. Double brackets $\{\!\{\cdot\}\!\}$ denote a lexicographically-ordered multi-set, $\mathcal{E}_v^\alpha \subseteq G$ is the α-depth ego network centered on v, and $\mathcal{N}_G^\alpha(v)$ is the set of neighbors of v in G up to distance α, i.e., $\mathcal{N}_G^\alpha(v) = \{u \mid u \in V \wedge l_G(u, v) \leq \alpha\}$.

2.1. Message Passing Graph Neural Networks

Graph neural networks are deep learning architectures for learning representations on graphs. The most popular choice of architecture is the message passing graph neural network (MP-GNN). In MP-GNNs, there is a direct correspondence between the connectivity of network layers and the structure of the input graph. Because of this, the representation (embedding) of each node depends directly on its neighbors and only indirectly on more distant nodes.

Each layer of an MP-GNN computes an embedding for a node by iteratively aggregating its attributes and the attributes of their neighboring nodes. Aggregation is expressed via two parametrized functions: MSG, which represents the computation of joint information for a vertex and a given neighbor, and UPDATE, a pooling operation over messages that produces a vertex representation. Let $h_v^0 \in \mathbb{R}^w$ denote an initial w-dimensional feature vector associated with $v \in V$. Each i-th GNN layer computes the i-th message passing step, such that μ_v^i is the multi-set of messages received by v:

$$\mu_v^i = \left\{\!\!\left\{ \text{MSG}_G^i(h_u^{i-1}) \mid \underset{u \neq v}{\forall} u \in \mathcal{N}_G^1(v) \right\}\!\!\right\}, \tag{1}$$

and h_v^i is the output of a permutation-invariant UPDATE function over the message multi-set and the previous vertex state:

$$h_v^i = \text{UPDATE}_G^i\left(\mu_v^i, h_v^{i-1}\right). \tag{2}$$

For machine learning tasks that require graph-level embeddings, an additional parameterized function dubbed READOUT produces a graph-level representation r_G^i, pooling all vertex representations at step i:

$$r_G^i = \text{READOUT}\left(\left\{\!\!\left\{h_v^i \mid \forall\, v \in V\right\}\!\!\right\}\right). \tag{3}$$

The functions MSG, UPDATE, and READOUT are differentiable with respect to their parameters, which are optimized during learning via gradient descent. The choice of functions gives rise to a broad variety of GNN architectures [30–34]. However, it has been shown that all MP-GNNs defined by MSG, UPDATE, and READOUT are at most as expressive as the 1-WL test when distinguishing non-isomorphic graphs [8].

2.2. Expressivity of Weisfeiler–Lehman and MATLANG

The classic Weisfeiler–Lehman algorithm (1-WL), also known as color refinement, is shown in Algorithm 1. The algorithm starts with an initial color assignment to each vertex c_v^0 according to their degree and proceeds updating the assignment at each iteration.

Algorithm 1 1-WL (Color refinement).

Input: $G = (V, E)$
1: $c_v^0 := \texttt{hash}(\{\!\{d_G(v)\}\!\})\ \forall\, v \in V$
2: **do**
3: $\quad c_v^{i+1} := \texttt{hash}(\{\!\{c_u^i :\ \forall\, u \in \mathcal{N}_G^1(v)\}\!\})$
4: **while** $c_v^i \neq c_v^{i-1}$
Output: $c_v^i : V \mapsto \mathbb{N}$

The update aggregates, for each node v, its color c_v^i and the colors of its neigbors c_u^i, then hashes this multi-set of colors, mapping it into a new color to be used in the next iteration c_v^{i+1}. The algorithm converges to a stable color assignment, which can be used to test graphs for isomorphism. Note that neighbor aggregation in 1-WL can be understood as a message passing step, with the `hash` operation being analogous to UPDATE steps in MP-GNNs.

Two graphs, G_1, G_2, are not isomorphic if they are distinguishable (that is, their stable color assignments do not match). However, if they are not distinguishable (that is, their stable color assignments match), they are likely to be isomorphic [35]. To reduce the likelihood of false positives when color assignments match, one can consider k-tuples of vertices instead of single vertices, leading to higher order variants of the WL test (denoted k-WL) which assign colors to k-vertex tuples. In this case, G_1 and G_2 are said to be k-WL equivalent, denoted $G_1 \equiv_{k-\text{WL}} G_2$, if their stable assignments are not distinguishable. For more details on the algorithm, we refer to [14,36]. k-WL tests are more expressive than their $(k-1)$-WL counterparts for $k > 2$, with the exception of 2-WL, which is known to be as expressive as the 1-WL color refinement test [10].

Among the various characterizations of k-WL expressivity, the relationship with matrix query languages defined by MATLANG [11] has also found applications for graph representation learning [26]. MATLANG is a language of operations on matrices, where sentences are formed by sequences of operations. There exists a subset of MATLANG—$\mathbf{ML}_1$—that is as expressive as the 1-WL test. Another subset—$\mathbf{ML}_2$—is strictly more expressive than the 1-WL test but less expressive than the 3-WL test. Finally, there exists another subset $\mathbf{ML}_3$ that is as expressive as the 3-WL test [13]. We provide additional technical details on

MATLANG and its sub-languages in section E, where we analyze the relation between IGEL and MATLANG.

2.3. Graph Neural Networks Beyond 1-WL

Recently, approaches have been proposed for improving the expressivity of MP-GNNs. Here, we focus on subgraph and substructure GNNs, which are most closely related to IGEL. For an overview on augmented message passing methods, see [37] and Appendix I.

***k*-hop MP-GNNs** (*k*-hop) [16] propagate messages beyond immediate vertex neighbors, effectively using ego network information in the vertex representation. Neighborhood subgraphs are extracted, and message passing occurs on each subgraph, with an exponential cost on the number of hops k both at preprocessing and at each iteration (epoch). In contrast, IGEL only requires a preprocessing step that can be cached once computed.

Distance encoding GNNs (**DE-GNN**) [38] also propose to improve MP-GNN by using extra node features by encoding distances to a subset of p nodes. The features obtained by DE-GNN are similar to IGEL when conditioning the subset to size $p = 1$ and using a distance encoding function with $k = \alpha$. However, these features are not strictly equivalent to IGEL, as node degrees within the ego network can be smaller than in the full graph.

Graph Substructure networks (**GSNs**) [19] incorporate topological features by counting local substructures (such as the presence of cliques or cycles). GSNs require expert knowledge on what features are relevant for a given task with modifications to MP-GNN architectures. In contrast, IGEL reaches comparable performance using a general encoding for ego networks and without altering the original message passing mechanism.

GNNML3 [26] performs message passing in spectral domain with a custom frequency profile. While this approach achieves good performance on graph classification, it requires an expensive preprocessing step for computing the eigendecomposition of the graph Laplacian and $\mathcal{O}(k)$-order tensors to achieve k-WL expressiveness with cubic time complexity.

More recently, a series of methods formulate the problem of representing vertices or graphs as aggregations over subgraphs. The subgraph information is pooled or introduced during message passing at an additional cost that varies depending on each architecture. Consequently, they require generating the subgraphs (or effectively replicating the nodes of every subgraph of interest) and pay an additional overhead due to the aggregation.

These approaches include identity-aware GNNs (**ID-GNNs**) [39], which embed each node while incorporating identity information in the GNN and apply rounds of heterogeneous message passing; nested GNNs (**NGNNs**) [20], which perform a two-level GNN using rooted subgraphs and consider a graph as a bag of subgraphs; GNN-as-Kernel (**GNN-AK**) [21], which follows a similar idea but introduces additional positional and contextual embeddings during aggregation; equivariant subgraph aggregation networks (**ESAN**) [22] encode graphs as bags of subgraphs and show that such an encoding can lead to a better expressive power; shortest-path neural networks (**SPNNs**) [40], which represent nodes by aggregating over sets of neighbors at the same distance; and subgraph union networks (**SUN**) [23], which unify and generalize previous subgraph GNN architectures and connect them to invariant graph networks [41]. Compared to all these methods, IGEL only relies on an initial preprocessing step based on distances and degrees without having to run additional message passing iterations or modify the architecture of the GNN.

Interestingly, recent work [42] showed that the implicit encoding of the pairwise distance between nodes plus the degree information which can be extracted via aggregation are fundamental to provide a theoretical justification of **ESAN**. Furthermore, the work on **SUN** [23] showed that node-based subgraph GNNs are provably as powerful as the 3-WL test. This result is aligned with recent analyses on the hierarchies of model expressivity [43].

In this work, we directly consider distances and degrees in the ego network, explicitly providing the structural information encoded by more expressive GNN architectures. In contrast to previous work, IGEL aims to be a *minimal* yet *expressive* representation of network structures *without learning* that is amenable to formal analysis, as shown in Section 4. This connects ego network properties to subgraph GNNs, corroborating the 3-WL upper bound

of **SUN** and the expressivity analysis of **ESAN**—to which IGEL is strongly related in its ego networks policy (EGO+). Furthermore, these relationships may explain the comparable empirical performance of IGEL to the state of the art, as shown in Section 5.

3. Local Ego-Network Encodings

The idea behind the IGEL encoding is to represent each vertex v by compactly encoding its corresponding ego network $\mathcal{E}_v^\alpha$ at depth α. The choice of encoding consists of a histogram of vertex degrees at distance $d \leq \alpha$, for each vertex in $\mathcal{E}_v^\alpha$. Essentially, IGEL runs a breadth-first traversal up to depth α, counting the number of times the same degree appears at distance $d \leq \alpha$. We postulate that such a simple encoding is sufficiently expressive and at the same time computationally tractable to be considered as vertex features.

Figure 1 shows an illustrative example for $\alpha = 2$. In this example, the green node is encoded using a (sparse) vector of size $5 \times 3 = 15$, since the maximum degree at depth 2 is 5 and there are three distances considered: $0, 1, 2$. The ego network contains six nodes, and all (distance, degree) pairs occur once, except for degree 3 at distance 2, which occurs twice. Algorithm 2 describes the steps to produce the IGEL encoding iteratively.

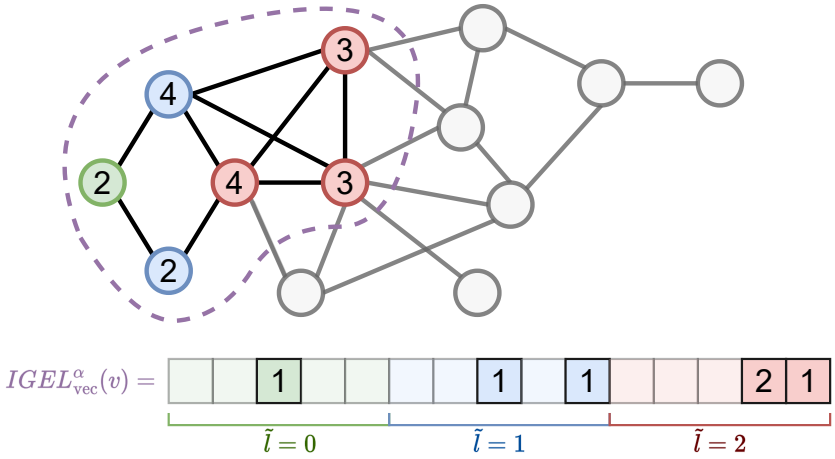


Figure 1. IGEL encoding of the green vertex. Dashed region denotes $\mathcal{E}_v^\alpha(\alpha = 2)$. The green vertex is at distance 0, blue vertices at 1, and red vertices at 2. Labels show degrees in $\mathcal{E}_v^\alpha$. The frequency of distance degree $(\tilde{l}, \tilde{d})$ tuples forming $\text{IGEL}^\alpha_{\texttt{vec}}(v)$ is: $\{(0,2):1, (1,2):1, (1,4):1, (2,3):2, (2,4):1\}$.

Algorithm 2 IGEL Encoding.

Input: $G = (V, E), \alpha : \mathbb{N}$
1: $e_v^0 := \{\!\{(0, d_G(v))\}\!\}\ \forall\, v \in V$
2: **for** $i := 1; i \mathrel{+}= 1$ **until** $i = \alpha$ **do**
3: $\quad e_v^i := \bigcup (e_v^{i-1}, \{\!\{(i, d_{\mathcal{E}_G^\alpha(v)}(u))\ \forall u \in \mathcal{N}_G^\alpha(v) \mid l_G(u,v) = i\}\!\})$
4: **end for**
Output: $e_v^\alpha : V \mapsto \{\!\{(\mathbb{N}, \mathbb{N})\}\!\}$

Similarly to 1-WL, information of the neighbors of each vertex is aggregated at each iteration. However, 1-WL does not preserve distance information in the encoding due to the hashing step. Instead of hashing the degrees into equivalence classes, IGEL keeps track of the distance at which a degree is found, generating a more expressive encoding.

The cost of such additional expressiveness is a computational complexity that grows exponentially in the number of iterations. More precisely, the time complexity follows $\mathcal{O}(n \cdot \min(m, (d_{\texttt{max}})^\alpha))$, with $\mathcal{O}(n \cdot m)$ when $\alpha \geq \texttt{diam}(G)$. In Appendix F, we provide a possible breadth-first search (BFS) implementation that is embarrassingly parallel and thus can be computed over p processors following $\mathcal{O}(n \cdot \min(m, (d_{\texttt{max}})^\alpha)/p)$ time complexity.

The encoding produced by Algorithm 1 can be described as a multi-set of path length and degree pairs $(\tilde{l}, \tilde{d})$ in the α-depth ego graph of v, $\mathcal{E}_v^\alpha$:

$$e_v^\alpha = \left\{\!\!\left\{ \left(l_{\mathcal{E}_v^\alpha}(u, v), d_{\mathcal{E}_v^\alpha}(u) \right) \middle| \; \forall \, u \in \mathcal{N}_G^\alpha(v) \right\}\!\!\right\}, \tag{4}$$

which also results in exponential space complexity. However, e_v^α can be represented as a (sparse) vector $\textsc{Igel}_{\texttt{vec}}^\alpha(v)$, where the i-th index contains the frequency of path length and degree pairs $(\tilde{l}, \tilde{d})$, as shown in Figure 1:

$$\textsc{Igel}_{\texttt{vec}}^\alpha(v)_i = \left| \{\!\{ (\tilde{l}, \tilde{d}) \in e_v^\alpha \text{ s.t. } f(\tilde{l}, \tilde{d}) = i \}\!\} \right|, \tag{5}$$

which has linear space complexity $\mathcal{O}(\alpha \cdot n \cdot d_{\texttt{max}})$, conservatively assuming every node requires $d_{\texttt{max}}$ parameters at every α depth from the center of the ego network, where $d_{\texttt{max}} = \mathcal{O}(n)$ in the worst case when a vertex is fully connected. Note that in practice, we may normalize raw counts by, e.g., applying log1p-normalization, and for real-world graphs, $d_{\texttt{max}} \ll n$, where the probability of larger degrees often decays exponentially [44]. Finally, complexity can be further reduced by making use of sparse vector implementations.

We finish this section with two remarks. First, the produced encodings can differ only for values of $\alpha < \texttt{diam}(G)$, since otherwise the ego network is the same for all nodes, $\mathcal{E}_v^\alpha = \mathcal{E}_v^{\alpha+1} = G, \forall v$, and thus the resulting encodings, i.e., $e_v^\alpha = e_v^{\alpha+1}$, for $\alpha \geq \texttt{diam}(G)$. Second, the sparse formulation in Equation (5) can be understood as an inductive analogue of Weisfeiler–Lehman graph kernels [45], as we explore in Appendix B.

4. Which Graphs Are IGEL-Distinguishable?

We now present results about the expressive power of IGEL, extending previous preliminary results on 1-WL that had been presented in [46]. We discuss the increased expressivity of IGEL with respect to 1-WL, and identify upper-bounds on the expressivity on graphs that are also indistinguishable under MATLANG and the 3-WL test. We assess expressivity by studying whether two graphs can be distinguished by comparing the encodings obtained by the k-WL test and the IGEL encodings for a given value of α. Similarly to the definition of k-WL equivalence, we say that $G_1 = (V_1, E_1)$ and $G_2 = (V_1, E_1)$ are IGEL-equivalent if the sorted multi-set of node representations is the same for G_1 and G_2:

$$G_1 \equiv_{\text{IGEL}}^\alpha G_2 \iff \{\!\{ e_{v_1}^\alpha : \forall v_1 \in V_1 \}\!\} = \{\!\{ e_{v_2}^\alpha : \forall v_2 \in V_2 \}\!\}.$$

4.1. Distinguishability on 1-WL Equivalent Graphs

We first show IGEL is more powerful than 1-WL, following Lemmas 1 and 2:

Lemma 1. IGEL *is at least as expressive as 1-WL. For two graphs, G_1, G_2, which are distinguished by 1-WL in k iterations ($G_1 \not\equiv_{\text{1-WL}} G_2$) it also holds that $G_1 \not\equiv_{\text{IGEL}}^\alpha G_2$ for $\alpha = k + 1$. If* IGEL *does not distinguish two graphs G_1' and G_2', 1-WL also does not distinguish them: $G_1' \equiv_{\text{IGEL}}^\alpha G_2' \Rightarrow G_1' \equiv_{\text{1-WL}} G_2'$.*

Lemma 2. *There exist at least two non-isomorphic graphs, G_1, G_2, that* IGEL *can distinguish but that 1-WL cannot distinguish; i.e., $G_1 \not\equiv_{\text{IGEL}}^\alpha G_2$ while $G_1 \equiv_{\text{1-WL}} G_2$.*

First, we formally prove Lemma 1, i.e., that IGEL is at least as expressive as 1-WL. For this, we consider a variant of 1-WL which removes the hashing step. This modification can only increase the expressive power of 1-WL and makes it possible to directly compare such (possibly more expressive) 1-WL encodings with the encodings generated by IGEL. Intuitively, after k color refinement iterations, 1-WL considers nodes at k hops from each node, which is equivalent to running IGEL with $\alpha = k + 1$, i.e., using ego networks that include information of *all* nodes that 1-WL would visit.

Proof of Lemma 1. For convenience, let $c_v^{i+1} = \{\{c_v^i; c_u^i \; \forall \; u \in \mathcal{N}_G^1(v) \mid u \neq v\}\}$ be a recursive definition of Algorithm 1 where hashing is removed and $c_v^0 = \{\{d_G(v)\}\}$. Since the hash is no longer computed, the nested multi-sets contain strictly the same or more information as in the traditional 1-WL algorithm.

For IGEL to be less expressive than 1-WL, it must hold that there exist two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ such that $G_1 \not\equiv_{\text{1-WL}} G_2$ while $G_1 \equiv^{\alpha}_{\text{IGEL}} G_2$.

Let k be the minimum number of color refinement iterations such that $\exists\, v_1 \in V_1$ and $\forall\, v_2 \in V_2, c_{v_1}^k \neq c_{v_2}^k$. We define an equally or more expressive variant of the 1-WL test 1-WL* where hashing is removed, such that $c_{v_1}^k = \{\{\{...\{\{d_G(v_1)\}\}, \{\{d_G(u) \forall u \in \mathcal{N}_{G_1}^1(v_1)\}\}...\}\}\}$, nested up to depth k. To avoid nesting, the multi-set of nested degree multi-sets can be rewritten as the union of degree multi-sets by introducing an indicator variable for the iteration number where a degree is found:

$$
\begin{aligned}
c_{v_1}^k = &\left\{\!\!\left\{ (0, d_G(v_1)) \right\}\!\!\right\} \bigcup \\
&\left\{\!\!\left\{ (1, d_G(v_1)); (1, d_G(u)) \; \forall \; u \in \mathcal{N}_G^1(v_1) \right\}\!\!\right\} \bigcup \\
&\left\{\!\!\left\{ (2, d_G(v_1)); (2, d_G(u)) \; \forall \; u \in \mathcal{N}_G^1(v_1); (2, d_G(w)) \; \forall \; w \in \mathcal{N}_G^1(u) \right\}\!\!\right\} \bigcup ...
\end{aligned}
$$

At each step i, we introduce information about nodes up to distance i of v_1. Furthermore, by construction, nodes will be visited on every subsequent iteration—i.e., for $c_{v_1}^2$, we will observe $(2, d_G(v_1))$ exactly $d_G(v_1) + 1$ times, as all its $d_G(v_1)$ neighbors $u \in \mathcal{N}_G^1(v)$ encode the degree of v_1 in c_u^1. The flattened representation provided by 1-WL* is still equally or more expressive than 1-WL, as it removes hashing and keeps track of the iteration at which a degree is found.

Let IGEL-W be a less expressive version of IGEL that does not include edges between nodes at $k+1$ hops of the ego network root. Now, consider the case in which $c_{v_1}^k \neq c_{v_2}^k$ from 1-WL*, and let $\alpha = k+1$ so that IGEL-W considers degrees by counting edges found at k to $k+1$ hops of v_1 and v_2. Assume that $G_1 \equiv^{\alpha}_{\text{IGEL-W}} G_2$. By construction, this means that $\{\{e_{v_1}^{\alpha} : \forall\, v_1 \in V_1\}\} = \{\{e_{v_2}^{\alpha} : \forall\, v_2 \in V_2\}\}$. This implies that all degrees and iteration counts match as per the distance indicator variable at which the degrees are found, so $c_{v_1}^k = c_{v_2}^k$ which contradicts the assumption $c_{v_1}^k \neq c_{v_2}^k$ and therefore implies that also $G_1 \equiv_{\text{1-WL}^*} G_2$. Thus, $G_1 \equiv^{\alpha}_{\text{IGEL-W}} G_2 \Rightarrow G_1 \equiv_{\text{1-WL}^*} G_2$ for $\alpha = k+1$ and also $G_1 \not\equiv_{\text{1-WL}^*} G_2 \Rightarrow G_1 \not\equiv^{\alpha}_{\text{IGEL-W}} G_2$. Therefore, by extension IGEL is at least as expressive as 1-WL. □

To prove Lemma 2, we show graphs that IGEL can distinguish despite being undistinguishable by 1-WL and the MATLANG sub-languages $\mathbf{ML}_1$ and $\mathbf{ML}_2$. In Section 4.1.1, we provide an example where IGEL distinguishes 1-WL/$\mathbf{ML}_1$ equivalent graphs, while Section 4.1.2 shows that IGEL also distinguishes graphs that are known to be distinguishable in the strictly more expressive $\mathbf{ML}_2$ language.

4.1.1. $\mathbf{ML}_1$/1-WL Expressivity: Decalin and Bicyclopentyl

Decalin and Bycyclopentyl (in Figure 2) are two molecules whose graph representations are not distinguishable by 1-WL despite their simplicity. The graphs are non-isomorphic, but 1-WL identifies 3 equivalence classes in both graphs: central nodes with degree 3 (purple), their neighbors (blue), and peripheral nodes farthest from the center (green).

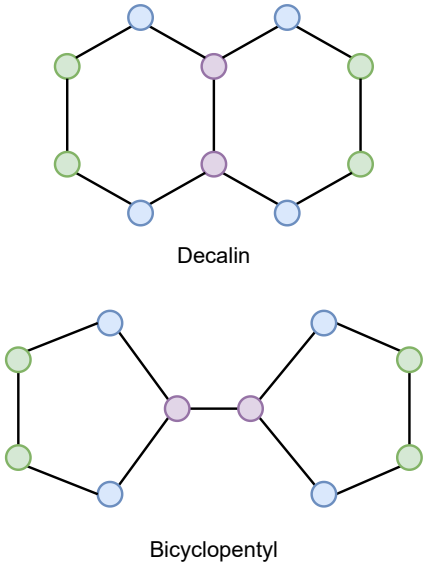


Figure 2. Decalin (top) and Bicyclopentyl (bottom). 1-WL (Algorithm 1) produces equivalent colorings in both graphs; hence, they are 1-WL equivalent. The colorings match between central nodes (purple), their immediate neighbors (blue), and peripheral nodes farthest from the center (green).

Figure 2 shows the resulting IGEL encoding for the central node (in purple) using $\alpha = 1$ (top) and $\alpha = 2$ (bottom). For $\alpha = 1$, the encoding field of IGEL is too narrow to identify substructures that distinguish the two graphs (Figure 3, top). However, for $\alpha = 2$ the representations of central nodes differ between the two graphs (Figure 3, bottom). In this example, any value of $\alpha \geq 2$ can distinguish between the graphs.

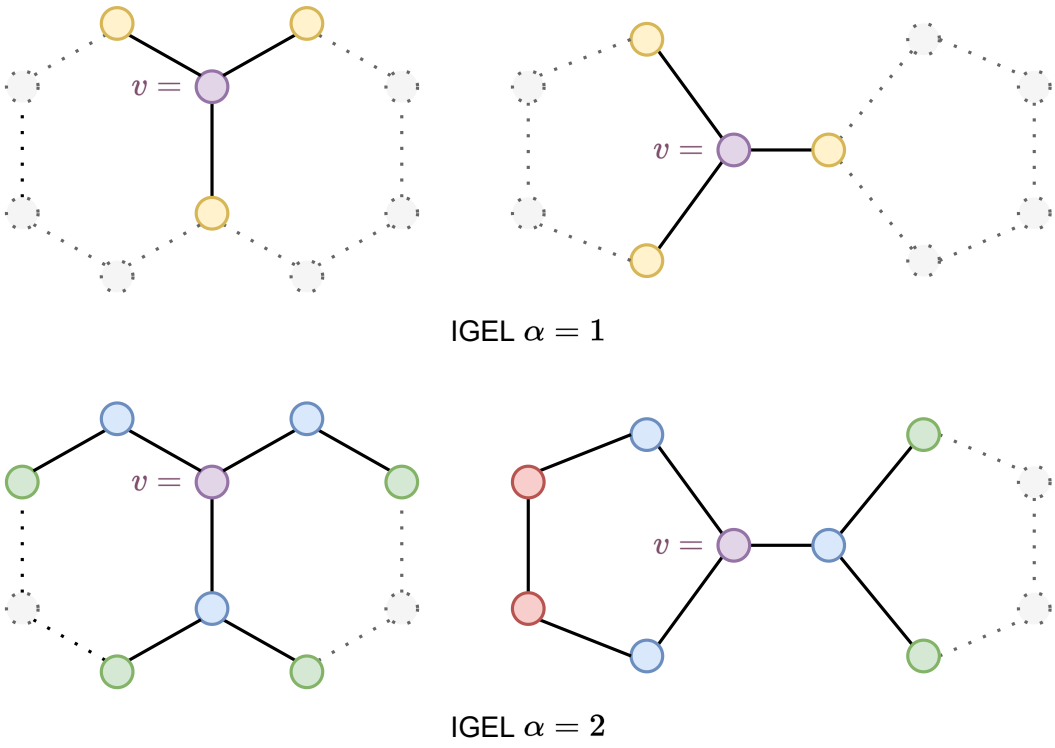


Figure 3. IGEL encodings ($\alpha \in \{1, 2\}$) for Decalin and Bicyclopentyl computed for purple vertices (v). Dotted sections are not encoded. Colors denote different $(\tilde{l}, \tilde{d})$ tuples. IGEL$(\alpha = 2)$ distinguishes the graphs since Decalin nodes at distance 2 from v have degree 1 (green) while their Bicyclopentyl counterparts have degrees 1 (green) and 2 (red).

4.1.2. $\mathbf{ML}_2$ Expressivity: Cospectral 4-Regular Graphs

IGEL can also distinguish $\mathbf{ML}_2$-equivalent graphs. Recall that $\mathbf{ML}_2$ is strictly more expressive than 1-WL, as described in Section 2.2. It is known that d-regular graphs of the same cardinality are indistinguishable by the 1-WL test in Algorithm 1 and that co-spectral graphs cannot be distinguished in $\mathbf{ML}_2$:

Definition 1. *$G = (V, E)$ is d-regular with $d \in \mathbb{N}$ if $\forall\, v \in V, d_G(v) = d$.*

Remark 1. *For any pair of n-vertex d-regular graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, $G_1 \equiv_{1-WL} G_2$ (see [10], Example 3.5.2, p. 81).*

Definition 2. *Two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are co-spectral if their adjacency matrices have the same multi-set of eigenvalues.*

Remark 2. *For any pair of n-vertex co-spectral graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, $G_1 \equiv_{\mathbf{ML}_2} G_2$ (see [13], Proposition 5.1).*

In contrast to 1-WL, we can find examples of non-isomorphic d-regular graphs that IGEL can distinguish, as the generated encodings will differ for any pair of graphs whose sets of sorted degree sequences do not match at any path length less than α. Furthermore, we can find examples of co-spectral graphs that can be distinguished by IGEL encodings. In both cases, the intuition is that the ego network encoding generated by IGEL discards the edges that connect nodes beyond the subgraph. Consequently, the generated encoding will depend on the actual connectivity at the boundary of the ego network and provide IGEL with increased expressivity compared to other methods.

Figure 4 shows two co-spectral 4-regular graphs taken from [47], and the structures obtained using IGEL encodings with $\alpha = 1$ on each graph. The 1-WL test assigns a single color to all nodes and stabilizes after one iteration. Likewise, any $\mathbf{ML}_2$ sentences executed as operations on the adjacency matrices of both graphs produce equal results. However, IGEL identifies four different structures (denoted a, b, c, d). Since the IGEL encodings between both graphs do not match, they are distinguishable. This is the case for any value of $\alpha \geq 1$.

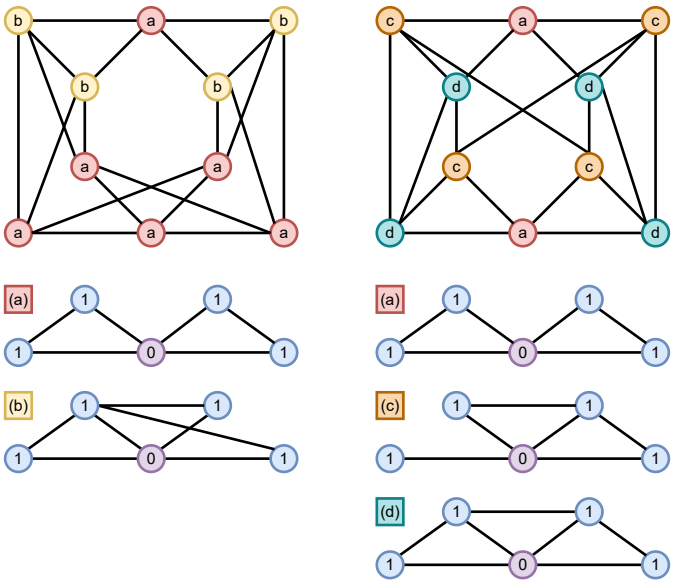


Figure 4. IGEL encodings for two co-spectral 4-regular graphs from [47]. IGEL distinguishes 4 kinds of structures within the graphs (associated with every node as a, b, c, and d). The two graphs can be distinguished since the encoded structures and their frequencies do not match.

4.2. Indistinguishability on Strongly Regular Graphs

We identify an upper bound on the expressive power of IGEL: non-isomorphic **S**trongly **R**egular **G**raphs with equal parameters. In this case, two non-isomorphic graphs are not indistinguishable by IGEL.

Definition 3. *A n-vertex d-regular graph is strongly regular—denoted* $\texttt{SRG}(n, d, \lambda, \gamma)$*—if all adjacent vertices have* λ *vertices in common and all non-adjacent vertices have* γ *vertices in common.*

Remark 3. *For any* $G = \texttt{SRG}(n, d, \lambda, \gamma)$, $\texttt{diam}(G) \leq 2$ *[48].*

Theorem 1. IGEL *cannot distinguish two SRGs when n, d, and* λ *are the same, and between any value of* γ *(equal or not).*

Formally, given $G_1 = \texttt{SRG}(n_1, d_1, \lambda_1, \gamma_1)$, $G_2 = \texttt{SRG}(n_2, d_2, \lambda_2, \gamma_2)$*:*
— $G_1 \equiv^1_{\text{IGEL}} G_2 \iff n_1 = n_2 \wedge d_1 = d_2 \wedge \lambda_1 = \lambda_2$*;*
— $G_1 \equiv^2_{\text{IGEL}} G_2 \iff n_1 = n_2 \wedge d_1 = d_2$*;*
— no values of γ *can be distinguished by* $\equiv^1_{\text{IGEL}}$ *or* $\equiv^2_{\text{IGEL}}$*.*

Proof. Recall that for any graph G, IGEL encodings are equal for all $\alpha \geq \texttt{diam}(G)$ and per Remark 3, SRGs have diameters of two or less. Let $G = \texttt{SRG}(n, d, \lambda, \gamma)$, only $\alpha \in \{1, 2\}$ produce different encodings for nodes of G. By construction, $\forall v$ in G, v has encoding e^α_v, so the encoding of G is $e^\alpha_G = \{\!\{e^\alpha_v\}\!\}^n$. Furthermore, $\{\!\{e^1_v\}\!\}^n$ only encodes n, d, and λ, and $\{\!\{e^2_v\}\!\}^n$ only encodes n and d by expanding e^α_v in Algorithm 2:

- Let $\alpha = 1$: $\forall\, v \in V, \mathcal{E}^1_v = (V', E')$ *s.t.* $V' = \mathcal{N}^1_G(v)$. Since G is d-regular, v is the center of $\mathcal{E}^1_v$, and has d-neighbors. By definition, d neighbors of v have λ shared neighbors with v each, plus an edge with v, and $\mathcal{E}^1_v$ does not include γ edges beyond its neighbors. Thus, for SRGs G_1, G_2 where $n_1 = n_2$, $d_1 = d_2$, and $\lambda_1 = \lambda_2$, $e^1_{G_1} = e^1_{G_2} = \{\!\{e^1_v\}\!\}^n$ where

$$e^1_v = \left\{\!\!\left\{ (0, d) \right\}\!\!\right\} \bigcup \left\{\!\!\left\{ (1, \lambda + 1) \right\}\!\!\right\}^d.$$

- Let $\alpha = 2$: $\forall\, v \in V, \mathcal{E}^2_v = G$ as $\forall\, u \in V, u \in \mathcal{N}^2_G(v)$ when $\texttt{diam}(G) \leq 2$. G is d-regular, so $\forall\, v \in V, d = d_{\mathcal{E}^2_v}(v) = d_G(v)$. Thus, for any SRGs G_1, G_2 s.t. $n_1 = n_2$ and $d_1 = d_2$, $e^2_{G_1} = e^2_{G_1} = \{\!\{e^2_v\}\!\}^n$ where

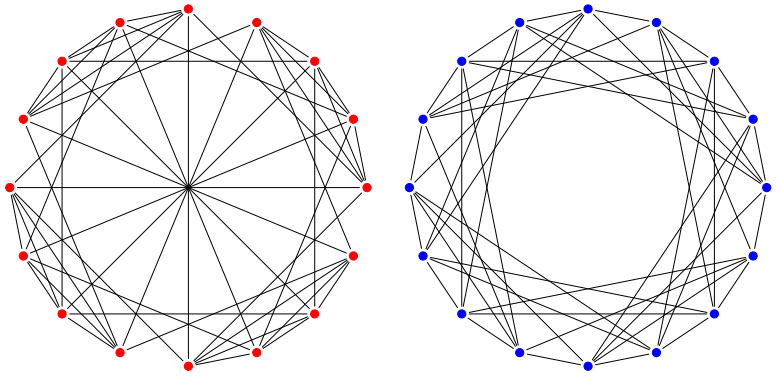
$$e^2_v = \left\{\!\!\left\{ (0, d) \right\}\!\!\right\} \bigcup \left\{\!\!\left\{ (1, d) \right\}\!\!\right\}^d \bigcup \left\{\!\!\left\{ (2, d) \right\}\!\!\right\}^{n-d-1}$$

Thus, IGEL with $\alpha \in \{1, 2\}$ can only distinguish between *different* values of n, d and λ. □

4.3. Expressivity Implications

As expected from Theorem 1, IGEL cannot distinguish the Shrikhande and 4×4 Rook graphs (shown in Figure 5), which are known to be $\mathbf{ML}_3$-equivalent graphs [26,49] with $\texttt{SRG}(16, 6, 2, 2)$ parameters despite not being isomorphic.

Our findings show that IGEL is a powerful permutation-equivariant representation (see Lemma A1), capable of distinguishing 1-WL equivalent graphs such as Figure 4—which as cospectral graphs are known to be expressable in strictly more powerful MATLANG sub-languages than 1-WL [13]. Furthermore, in Appendix D, we connect IGEL to SPNNs [40] and show that IGEL is strictly more expressive than SPNNs on unattributed graphs. Finally, we note that the upper bound on strongly regular graphs is a hard ceiling on expressivity since SRGs are commonly known to be indistinguishable by 3-WL [14,21,23,43,49].



(**a**) 4×4 Rook Graph. (**b**) Shrikhande Graph.

Figure 5. 3-WL equivalent 4×4 Rook ((**a**), red) and Shrikhande ((**b**), blue) graphs are indistinguishable by IGEL as they are non-isomorphic SRGs with parameters $\mathtt{SRG}(16, 6, 2, 2)$.

IGEL formally reaches an expressivity upper bound on SRGs, distinguishing SRGs with different values of n, d, and λ. These results are similar with subgraph methods implemented within MP-GNN architectures, such as nested GNNs [20] and GNN-AK [21], which are known to be not less powerful than 3-WL, and the ESAN framework when leveraging ego networks with root-node flags as a subgraph sampling policy (EGO+), which is as powerful as 3-WL on SRGs [22]. However, in contrast to these methods, IGEL cannot distinguish graphs with different values of γ. Furthermore, in Section 5, we study IGEL in a series of empirical tasks, finding that the expressivity difference that IGEL exhibits on SRGs does not have significant implications in downstream tasks.

Summarizing, IGEL distinguishes non-isomorphic graphs that are undistinguishable by the 1-WL test. Furthermore, Lemma 2 shows that IGEL can distinguish any graphs that 1-WL can distinguish. We derive a precise expressivity upper bound for IGEL on SRGs, showing that IGEL cannot distinguish SRGs with equal parameters or between values of γ. Overall, the expressive power of IGEL on SRGs is similar to other state-of-the-art methods, including k-hop GNNs [16], GSNs [19], NGNNs [20], GNN-AK [21], and ESAN [22].

5. Empirical Validation

We evaluate IGEL as a sparse local ego network encoding following Equation (5), extending vertex/edge attributes on six experimental tasks: graph classification, graph isomorphism detection, graphlet counting, graph regression, link prediction, and vertex classification. With our experiments, we seek to evaluate the following empirical questions:

Q1. Does IGEL improve MP-GNN performance on standard graph-level tasks?
Q2. Can we empirically validate our results on the expressive power of IGEL compared to 1-WL?
Q3. Are IGEL encodings appropriate features to learn on unattributed graphs?
Q4. How do GNN models compare with more traditional neural network models when they are enriched with IGEL features?

5.1. Overview of the Experiments

For graph classification, isomorphism detection, and graphlet counting, we reproduce the benchmark proposed by [26] on eight graph data sets. For each task and data set, we introduce IGEL as vertex/edge attributes, and compare the performance of including or excluding IGEL on several GNN architectures—including linear and MLP baselines (without message passing), GCNs [31], GATs [33], GINs [8], Chebnets [30], and GNNML3 [26]. We measure whether IGEL improves inductive MP-GNN performance while validating our theoretical expressivity analysis (**Q1** and **Q2**). We also evaluate on the ZINC-12K and PATTERN data sets from benchmarking GNNs [50] to test IGEL on larger, real-world data sets.

For link prediction, we experiment on two unattributed social network graphs. We train self-supervised embeddings on IGEL encodings and compare them against standard transductive vertex embeddings. *Transductive* methods require that all nodes in the graph are known at training and inference time, while *inductive* methods can be applied to unseen nodes, edges, and graphs. Inductive methods may be applied in transductive settings but not vice-versa. Since IGEL is label and permutation invariant, its output is an inductive representation. We detail the self-supervised embedding approach in Appendix H. We compare our results against strong vertex embedding models, namely DeepWalk [51] and Node2Vec [52], seeking to validate IGEL as a theoretically grounded structural feature extractor in unattributed graphs (**Q3**).

For vertex classification, we train using IGEL encodings and vertex attributes as inputs on DNN models without message passing on an inductive protein-to-protein interaction (PPI) multi-label classification problem. We evaluate the impact of introducing IGEL on top of vertex attributes and compare the performance of IGEL-inclusive models with MP-GNNs (**Q4**).

5.2. Experimental Methodology

On graph-level tasks, we introduce IGEL encodings concatenated to existing vertex features, and also introduce IGEL as edge-level features representing an edge as the element-wise product of node-level IGEL encodings at each end of the edge into the best performing model configurations found by [26] without any hyper-parameter tuning (e.g., number of layers, hidden units, choice pooling and activation functions). We evaluate performance differences with and without IGEL on each task, data set and model on 10 independent runs, measuring statistical significance of the differences through paired t-tests. On benchmark data sets, we reuse the best reported GIN-AK$^+$ baseline from [21] and simply introduce IGEL as additional node features with $\alpha \in \{1, 2\}$, with no hyper-parameter changes.

On vertex and edge-level tasks, we report best performing configurations after hyper-parameter search. Each configuration is evaluated on five independent runs. Our results are compared against strong standard baselines from the literature, and we provide a breakdown of the best-performing hyper-parameters found in Appendix A.

5.3. Results and Notation

The following formatting denotes significant (as per paired t-tests) **positive (in bold)**, *negative (in italic)*, and insignificant differences (no formatting) after introducing IGEL, with the best results per task/data set underlined.

5.4. Graph Classification: TU Graphs

Table 1 shows graph classification results for TU molecule data sets [28]. In each data set, nodes represent atoms and edges represent their atomic bonds. The graphs contain no edge features while, node features are a one-hot encoded vector of the atom represented by that node. We evaluate differences in mean accuracies with and without IGEL through paired t-tests, denoting significance intervals of $p < 0.01$ as * and $p < 0.0001$ as $^\diamond$.

Table 1. Per-model classification accuracy metrics on TU data sets. Each cell shows the average accuracy of the model and data set in that row and column, with IGEL (left) and without IGEL (right).

Model	Enzymes	Mutag	Proteins	PTC
MLP	**41.10 > 26.18** $^\diamond$	**87.61 > 84.61** $^\diamond$	75.43 ~ 75.01	**64.59 > 62.79** $^\diamond$
GCN	**54.48 > 48.60** $^\diamond$	**89.61 > 85.42** $^\diamond$	**75.67 > 74.50** *	65.76 ~ 65.21
GAT	54.88 ~ 54.95	**90.00 > 86.14** $^\diamond$	**73.44 > 70.51** $^\diamond$	66.29 ~ 66.29
GIN	**54.77 > 53.44** *	89.56 ~ 88.33	**73.32 > 72.05** $^\diamond$	61.44 ~ 60.21
Chebnet	61.88 ~ 62.23	**91.44 > 88.33** $^\diamond$	**74.30 > 66.94** $^\diamond$	64.79 ~ 63.87
GNNML3	*61.42 < 62.79* $^\diamond$	**92.50 > 91.47** *	**75.54 > 62.32** $^\diamond$	*64.26 < 66.10* $^\diamond$

Our results show that IGEL in the Mutag and Proteins data sets improves the performance of all MP-GNN models, including GNNML3, contributing to answer **Q1.** By introducing IGEL in those data sets, MP-GNN models reach similar performance to GNNML3. Introducing IGEL achieves this at $\mathcal{O}(n \cdot \min(m, (d_{\texttt{max}})^{\alpha}))$ preprocessing costs compared to $\mathcal{O}(n^3)$ worst-case eigen-decomposition costs associated with GNNML3's spectral supports.

Additionally, since IGEL is an inductive method, the worst-case $\mathcal{O}(n \cdot (d_{\texttt{max}})^{\alpha})$ when $\alpha < \texttt{diam}(G)$ cost is only required when the graph is first processed. Afterwards, encodings can be reused, recomputing them for nodes neighboring new nodes or updated edges as given by α. This contrasts with GNNML3's spectral supports, which are computed on the adjacency matrix and would require a full recalculation when nodes or edges change.

On the Enzymes and PTC data sets, results are mixed: for all models other than GNNML3, IGEL either significantly improves accuracy (on MLPNet, GCN, and GIN on Enzymes), or does not have a negative impact on performance. GAT outperforms GNNML3 in PTC, while GNNML3 is the best performing model on Enzymes. Additionally, GNNML3 performance degrades when IGEL is introduced on the Enzymes and PTC data sets. We believe this degradation may be caused by overfitting due to a lack of additional parameter tuning, as GNNML3 models are deeper in Enzymes and PTC (four GNNML3 layers) when compared Mutag and Proteins (three and two GNNML3 layers respectively). It may be possible to improve GNNML3 performance with IGEL by re-tuning model parameters, but due to computational constraints we do not test this hypothesis. Nevertheless, we observe that all models improve in at least two different data sets after introducing IGEL without hyper-parameter tuning, which we believe indicates our results are a conservative lower-bound on model performance.

We also compare the best IGEL results from Table 1 with state-of-the-art methods improving expressivity. Table 2 summarizes the reported results for k-hop GNNs [16], GSNs [19], nested GNNs [20], ID-GNNs [39], GNN-AK [21], and ESAN [22]. When we compare IGEL and the best performing baseline for every data set, none of the differences are statistically significant ($p > 0.01$) except for ID-GNN in Proteins (where $p = 0.009$).

Table 2. Mean ± stddev of best IGEL configuration and state-of-the-art results reported on [16,19–22,39] with best performing baselines underlined.

Model	Mutag	Proteins	PTC
IGEL (ours)	92.5 ± 1.2	75.7 ± 0.3	66.3 ± 1.3
***k*-hop [16]** †	*87.9 ± 1.2* ◇	75.3 ± 0.4	—
GSN [19] †	92.2 ± 7.5	76.6 ± 5.0	68.2 ± 7.2
NGNN [20] †	87.9 ± 8.2	74.2 ± 3.7	—
ID-GNN [39] †	$\underline{93.0 \pm 5.6}$	$\underline{77.9 \pm 2.4}$ *	62.5 ± 5.3
GNN-AK [21] †	91.7 ± 7.0	77.1 ± 5.7	67.7 ± 8.8
ESAN [22] †	91.1 ± 7.0	76.7 ± 4.1	$\underline{69.2 \pm 6.5}$

†: Results as reported by [16,19–22,39].

Overall, our results show that incorporating the IGEL encodings in a vanilla GNN yields comparable performance to state-of-the-art methods (**Q2**).

5.5. Graph Isomorphism Detection

Table 3 shows isomorphism detection results on two data sets: Graph8c (as described in Appendix A), and EXP [53]. On the Graph8c data set, we identify isomorphisms by counting the number of graph pairs for which randomly initialized MP-GNN models produce equivalent outputs. Equivalence is measured by the Manhattan distance between graph on 100 independent initialization runs further described in Appendix A. The EXP data set contains 1-WL equivalent pairs of graph, and the objective is to identify whether they are isomorphic or not. We report model accuracies on the binary classification task of distinguishing non-isomorphic graphs that are 1-WL equivalent.

Table 3. Graph isomorphism detection results. The IGEL column denotes whether IGEL is used or not in the configuration. For Graph8c, we describe graph pairs erroneously detected as isomorphic. For EXP classify, we show the accuracy of distinguishing non-isomorphic graphs in a binary classification task.

Model	+ IGEL	Graph8c (#Errors)	EXP Class. (Accuracy)
Linear	No	6.242M	50%
	Yes	**1571**	**97.25%**
MLP	No	293K	50%
	Yes	**1487**	**100%**
GCN	No	4196	50%
	Yes	**5**	**100%**
GAT	No	1827	50%
	Yes	**5**	**100%**
GIN	No	571	50%
	Yes	**5**	**100%**
Chebnet	No	44	50%
	Yes	**1**	**100%**
GNNML3	No	0	100%
	Yes	0	100%

On Graph8c, introducing IGEL significantly reduces the amount of graph pairs erroneously identified as isomorphic for all MP-GNN models. Furthermore, IGEL allows a linear baseline employing a sum readout over input feature vectors, then projecting onto a 10-component space to identify all but 1571 non-isomorphic pairs compared to GCNs (4196 errors) or GATs (1827 errors) that can be identified without IGEL.

We also find that all Graph8c graphs can be distinguished if the IGEL encodings for $\alpha = 1$ and $\alpha = 2$ are concatenated. We do not study the expressivity of concatenating combinations of α in this work, but based on our results we hypothesize it produces strictly more expressive representations.

On EXP, introducing IGEL is sufficient to correctly identify all non-isomorphic graphs for all standard MP-GNN models, as well as the MLP baseline. Furthermore, the linear baseline can reach 97.25% classification accuracy with IGEL despite only computing a global sum readout before a single-output fully connected layer. Results on Graph8c and EXP validate our theoretical claims that IGEL is more expressive than 1-WL and can distinguish graphs that would be indistinguishable under 1-WL, answering **Q2**.

We also evaluate IGEL on the SR25 data set (described in Appendix A), which contains 15 strongly regular 25 vertex non-isomorphic graphs known to be indistinguishable by 3-WL where we can empirically validate Theorem 1. In [26], it was shown that all models in our benchmark are unable to distinguish any of the 105 non-isomorphic graph pairs in SR25. Introducing IGEL does not improve distinguishability—as expected from Theorem 1.

5.6. Graphlet Counting

We evaluate IGEL on a graphlet counting regression task, training a model to minimize mean squared error (MSE) on the normalized graphlet counts. Counts are normalized by the standard deviation of counts in the training set, as in [26].

In Table 4, we show the results of introducing IGEL in five graphlet counting tasks on the RandomGraph data set [54]. We identify 3-stars, triangles, tailed triangles and 4-cycle graphlets, as shown in Figure 6, plus a custom structure with 1-WL expressiveness proposed in [26] to evaluate GNNML3. We highlight statistically significant differences when introducing IGEL ($p < 0.0001$).

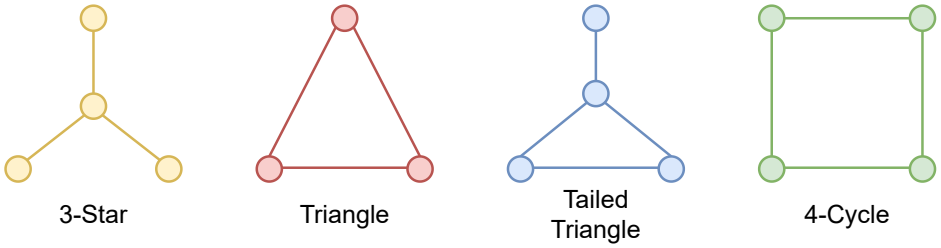


Figure 6. Graphlet types in the counting task.

Table 4. Graphlet counting results. On every cell, we show mean test set MSE error (lower is better), with stat. sig ($p < 0.0001$) results **highlighted** and best results per task underlined. For comparison, we also report strong literature results from two subgraph GNNs: GNN-AK$^+$ using a GIN base [21] and SUN [23].

Model	+ IGEL	Star	Triangle	Tailed Tri.	4-Cycle	Custom
Linear	No	1.60×10^{-1}	3.41×10^{-1}	2.82×10^{-1}	2.03×10^{-1}	5.11×10^{-1}
	Yes	4.23×10^{-3}	4.38×10^{-3}	1.85×10^{-2}	1.36×10^{-1}	5.25×10^{-2}
MLP	No	$\underline{2.66 \times 10^{-6}}$	2.56×10^{-1}	1.60×10^{-1}	1.18×10^{-1}	4.54×10^{-1}
	Yes	8.31×10^{-5}	$\underline{5.69 \times 10^{-5}}$	$\underline{5.57 \times 10^{-5}}$	7.64×10^{-2}	$\underline{2.34 \times 10^{-4}}$
GCN	No	4.72×10^{-4}	2.42×10^{-1}	1.35×10^{-1}	1.11×10^{-1}	1.54×10^{-3}
	Yes	8.26×10^{-4}	1.25×10^{-3}	4.15×10^{-3}	7.32×10^{-2}	1.17×10^{-3}
GAT	No	4.15×10^{-4}	2.35×10^{-1}	1.28×10^{-1}	1.11×10^{-1}	2.85×10^{-3}
	Yes	4.52×10^{-4}	6.22×10^{-4}	7.77×10^{-4}	7.33×10^{-2}	6.66×10^{-4}
GIN	No	3.17×10^{-4}	2.26×10^{-1}	1.22×10^{-1}	1.11×10^{-1}	2.69×10^{-3}
	Yes	6.09×10^{-4}	1.03×10^{-3}	2.72×10^{-3}	6.98×10^{-2}	2.18×10^{-3}
Chebnet	No	5.79×10^{-4}	1.71×10^{-1}	1.12×10^{-1}	8.95×10^{-2}	2.06×10^{-3}
	Yes	3.81×10^{-3}	7.88×10^{-4}	2.10×10^{-3}	7.90×10^{-2}	2.05×10^{-3}
GNNML3	No	8.90×10^{-5}	2.36×10^{-4}	2.91×10^{-4}	$\underline{6.82 \times 10^{-4}}$	9.86×10^{-4}
	Yes	9.29×10^{-4}	2.19×10^{-4}	4.23×10^{-4}	6.98×10^{-4}	4.17×10^{-4}
GIN-AK$^+$ [21]	No	1.6×10^{-2}	1.1×10^{-2}	1.0×10^{-2}	1.1×10^{-2}	—
SUN [23]	No	6.0×10^{-3}	8.0×10^{-3}	8.0×10^{-3}	1.1×10^{-2}	—

Introducing IGEL improves the ability of 1-WL GNNs to recognize triangles, tailed triangles, and custom 1-WL graphlets from [26]. Stars can be identified by all baselines, and introducing IGEL only produces statistically significant differences on the linear baseline. Interestingly, IGEL on the linear model produces results outperforming MP-GNNs without IGEL for star, triangle, tailed triangle, and custom 1-WL graphlets.

By introducing IGEL on the MLP baseline, it obtains the best performance (lower MSE) on the triangle, tailed-triangle, and custom 1-WL graphlets, even when compared to GNNML3 *and* subgraph GNNs—including when IGEL encodings are input to GNNML3.

Results on linear and MLP baselines are interesting as neither baseline uses message passing, indicating that raw IGEL encodings may be sufficient to identify certain graph structures in simple linear models. For all graphlets except 4-cycles, introducing IGEL outperforms or matches GNNML3 performance at lower preprocessing and model training/inference costs—without the need for costly eigen decomposition or message passing, answering **Q1** and **Q2.** IGEL moderately improves performance counting 4-cycle graphlets, but the results are not competitive when compared to GNNML3.

5.7. Benchmark Results with Subgraph GNNs

Given the favorable performance of IGEL compared to related subgraph aware methods such as NGNN and ESAN, as shown in Table 2, we also explore introducing IGEL on a subgraph GNN, namely GNN-AK proposed by [21]. We follow a similar experimental approach as in previous experiments, reproducing the results of GNN-AK using GINs [8] with edge-feature support [55] (GINs are the best performing base model in three out of

the four data sets, as reported in [21]) as the base GNN on two real-world benchmark data sets: ZINC-12K (as a graph regression task minimizing mean squared error ↓, where lower is better) and PATTERN (as a graph classification task, where higher accuracy ↑ is better) from benchmarking GNNs [50].

Without performing any additional hyper-parameter tuning or architecture search, we evaluate the impact of introducing IGEL with $\alpha \in \{1, 2\}$ on the best performing model configuration and code published by the authors of GNN-AK [21]. Furthermore, we also evaluate the setting in which subgraph information is not used to assess whether IGEL can provide comparable performance to GNN-AK without changes to the network architecture. Table 5 summarizes our results.

Table 5. Mean and standard deviations of the evaluation metrics on the real-world benchmark data sets in combination with GNN-AK [21], highlighting **positive** and *negative* stat. sig, ($p < 0.05$) results when IGEL is added with best per-data set results underlined.

Model	+ IGEL	ZINC-12K (Mean Squared Error, ↓)	PATTERN (Accuracy, ↑)
GIN	No	0.155 ± 0.003	85.692 ± 0.042
	Yes	0.155 ± 0.005	**86.711 ± 0.009**
GIN-AK$^+$	No	0.086 ± 0.002	86.877 ± 0.006
	Yes	**0.078 ± 0.003**	*86.737 ± 0.062*

↓: lower is better; ↑: higher is better.

IGEL maintains or improves performance in both cases when introduced on a GIN, but only in the case of PATTERN we find a statistically significant difference ($p < 0.05$). When introducing IGEL on a GIN-AK model, we find statistically significant improvements on ZINC-12K. Introducing IGEL on GIN-AK$^+$ in PATTERN produces unstable losses on the validation set, with model performance showing larger variations across epochs. We believe that this instability might explain the loss in performance and that further hyper-parameter tuning and regularization (e.g., tuning dropout to avoid overfitting on specific IGEL features) could result in improved model performance.

Finally, we note that despite our constrained setup, introducing IGEL is also interesting from a runtime and memory standpoint. In particular, introducing IGEL on a GIN for PATTERN yields performance only -0.2% worse than its GNN-AK (86.711 vs. 86.877), while executing 3.87 times faster (62.21 s vs. 240.91 s per iteration) and requiring 20.51 times less memory (26.2 GB vs. 1.3 GB). This is in line with our theoretical analysis in Section 3, as IGEL can be computed once as a preprocessing step and then introduces a negligible cost on the input size of the first layer, which is amortized in multi-layer GNNs. Together with our results on graph classification, isomorphism detection, and graphlet counting, our experiments show that IGEL is also an efficient way of introducing subgraph information without architecture modifications in large real-world data sets.

5.8. Link Prediction

We also test IGEL on a link prediction task, following the approach of [52] to compare with well-known transductive node embedding methods on the Facebook and ArXiv AstroPhysics data sets [56]. We mode the task as follows: for each graph, we generate negative examples (non-existing edges) by sampling random unconnected node pairs. Positive examples (existing edges) are obtained by removing half of the edges, keeping the pruned graph connected after edge removals. Both sets of vertex pairs are chosen to have the same cardinality. Note that keeping the graph connected is not required for IGEL—it is required by transductive methods, which fail to learn meaningful representations on disconnected graph components. We learn *self-supervised* IGEL embeddings with a DeepWalk [51] approach (described in Appendix H) and model the link prediction task as a logistic regression problem whose input is the representation of an edge—as the element-

wise product of IGEL embeddings of vertices at each end of an edge without fine-tuning, which is the best edge representation reported by [52].

In Table 6, we report AUC results averaged on five independent executions and compared against previous reported baselines. In this case, we perform hyper-parameter search, with results described on Appendix A. We provide additional details on the unsupervised parameters in our code repositories also referenced in Appendix A.

Table 6. Area under the ROC curve (AUROC) link prediction results on Facebook and AP-arXiv. Embeddings learned on IGEL encodings outperform transductive methods. IGEL stddevs < 0.005.

Method	Facebook	arXiv
DeepWalk [51]	0.968	0.934
node2vec [52]	0.968	0.937
IGEL ($\alpha = 2$)	**0.976**	**0.984**

IGEL with $\alpha = 2$ significantly outperforms standard transductive methods on both data sets. This is despite the fact that we compare against methods that are aware of node identities and that several vertices can share the same IGEL encodings. Furthermore, IGEL is an inductive method that may be used on unseen nodes and edges, unlike transductive methods DeepWalk or node2vec. We do not explore the inductive setting as it would unfairly favor IGEL and cannot be directly applied to DeepWalk or Node2vec.

Additionally, IGEL significantly underperforms when $\alpha = 1$. We believe this might be caused by the fact that when $\alpha < 2$, it is not possible for the model to assess whether two vertices are neighbors based on their IGEL representation. Overall, our link prediction results show that IGEL encodings can be used as a potentially inductive feature generation approach in unattributed networks, without degrading performance when compared to standard vertex embedding methods—answering **Q3**.

5.9. Vertex Classification

In light of our graph-level results on graphlet counting, we evaluate to which extent a vertex classification task can be solved by leveraging IGEL structural features without message passing (**Q4**). We introduce IGEL in a DNN model and evaluate against several MP-GNN baselines. Our comparison includes supervised baselines proposed by GraphSAGE [32], LCGL [34], and GAT [33] on a multi-label classification task in a protein-to-protein interaction (PPI) [32] data set. The aim is to predict 121 binary labels, given graph data where every vertex has 50 attributes. We tune the parameters of a multi-layer perceptron (MLP) whose input features are either IGEL, vertex attributes, or both through randomized grid search—and we provide a detailed description of the grid-search parameters in Appendix A.

Table 7 shows Micro-F1 scores averaged over five independent runs. Introducing IGEL alongside vertex attributes in an MLP can outperform standard MP-GNNs like GraphSAGE or LGCL despite not using message passing—and thus only having access to the attributes of a vertex without additional context from its neighbors, answering **Q4**. Furthermore, even though IGEL underperforms when compared with GAT, the results reported by [33] use a three-layer GAT model propagating messages through 3-hops, while we observe the best IGEL performance with $\alpha = 1$. We believe that the structural information at 1-hop captured by IGEL might be sufficient to improve performance on tasks where local information is critical, potentially reducing the hops required by downstream models (e.g., GATs).

Table 7. Multi-label class. Micro-F1 scores on PPI. IGEL plus node features as input for an MLP outperforms LGCL and GraphSAGE. IGEL stddevs < 0.01.

Method		PPI
Only Features (MLP, ours)		0.558
GraphSAGE-GCN [32]		0.500
GraphSAGE-mean [32]		0.598
GraphSAGE-LSTM [32]		0.612
GraphSAGE-pool [32]		0.600
GraphSAGE (no sampling) [33]		0.768
LGCL [34]		0.772
IGEL ($\alpha = 1$)	Graph Only	0.736
	Graph + Feats	*0.850*
IGEL ($\alpha = 2$)	Graph Only	0.506
	Graph + Feats	0.741
Const-GAT [33]		0.934
GAT [33]		**0.973**

6. Discussion and Conclusions

IGEL is a novel and simple vertex representation algorithm that increases the expressive power of MP-GNNs beyond the 1-WL test. Empirically, we found IGEL can be used as a vertex/edge feature extractor on graph-, edge-, and node-level settings. On four different graph-level tasks, IGEL significantly improves the performance of nine graph representation models without requiring architectural modifications—including Linear and MLP baselines, GCNs [57], GATs [33], GINs [8], ChebNets [30], GNNML3 [26], and GNN-AK [21]. We introduce IGEL without performing hyper-parameter search on an existing baseline, which suggests that IGEL encodings are informative and can be introduced in a model without costly architecture search.

Although structure-aware message passing [16,20,21,23], substructure counts [19], identity [39], and subgraph pooling [22] may also be combined with existing MP-GNN architectures, IGEL reaches comparable performance as related models while simply augmenting the set of vertex-level feature without tuning model hyper-parameters. IGEL consistently improves performance on five data sets for graph classification: one data set for graph regression, two data sets for isomorphism detection, and five different graphlet structures in a graphlet counting task. Furthermore, even though MP-GNNs with learnable subgraph representations are expected to be more expressive since they can freely learn structural characteristics according to optimization objectives, our results show that introducing IGEL produces comparable results on three different domains and improves the performance of a strong GNN-AK$^+$ baseline on ZINC-12K. Additionally, introducing IGEL on a GIN model in the PATTERN data set achieves 99.8% of the performance of a strong GIN-AK$^+$ baseline at 3.87 times lower memory costs. The computational efficiency is a key benefit of IGEL as it only requires a single preprocessing step that extends vertex/edge attributes and can be cached during training and inference.

On link prediction tasks evaluated in two different graph data sets, IGEL-based DeepWalk [51] embeddings outperformed transductive methods based on embedding node identities such as DeepWalk [51] and Node2vec [52]. Finally, IGEL with $\alpha = 1$ outperformed certain MP-GNN architectures like GraphSAGE [32] on a protein–protein interaction node-classification task despite being used as an additional input to a DNN without message passing. More powerful MP-GNNs—namely a three-layer GAT [33]—outperformed the IGEL-enriched DNN model, albeit at potentially higher computational costs due to the increased depth of the model.

A fundamental aspect in our analysis of the expressivity of IGEL is that the connectivity of nodes is different depending on whether they are analyzed as part of the subgraph (ego network) or within the entire input graph. In particular, edges at the boundary of the ego network are a subset of the edges in the input graph. IGEL exploits this idea in combination with a simple encoding based on frequencies of degrees and distances. This is a novel idea, which allows us to connect the expressive power of IGEL with other analyses like the 1-WL test, as well as Weisfeiler–Lehman kernels (see Appendix B), shortest-path neural networks (see Appendix D), and MATLANG (see Appendix E). Furthermore, the ego network formulation allows us to identify an upper-bound on expressivity in strongly regular graphs—matching recent findings on the expressivity of subgraph GNNs.

Although we have presented IGEL on unattributed graphs, the principle underlying its encoding can be also applied to labelled or attributed graphs. Appendix G outlines possible extensions in this direction. In Appendix G.3, we also connect IGEL with k-hop GNNs and GNN-AK—drawing a direct link between subgraph GNNs and our proposed encoding.

Overall, our results show that IGEL can be efficiently used to enrich network representations on a variety of tasks and data sets, which we believe is an attractive baseline in expressivity-related tasks. This opens up interesting future research directions by showing that explicit network encodings like IGEL can perform competitively compared to more complex learnable representations while being more computationally efficient.

Author Contributions: N.A.-G.: Conceptualization, Methodology, Software, Investigation, Formal analysis, Writing—Original Draft; A.K.: Validation, Supervision, Writing—Review & Editing; V.G.: Resources, Validation, Supervision, Writing—Review & Editing. All authors have read and agreed to the published version of the manuscript.

Funding: This work has been co-funded by MCIN/AEI/10.13039/501100011033 under the Maria de Maeztu Units of Excellence Programme (CEX2021-001195-M). This publication is part of the action CNS2022-136178 financed by MCIN/AEI/10.13039/501100011033 and by the European Union Next Generation EU/PRTR.

Institutional Review Board Statement: This article does not contain any studies with human participants or animals performed by any of the authors.

Informed Consent Statement: Not applicable.

Data Availability Statement: This work uses publicly available graph data sets. We produced no new data sets, and access the data sets in our empirical evaluation through open-source libraries. We provide dataset details and links to code repositories and artifacts in Appendix A.1 in Appendix A.

Acknowledgments: We thank the anonymous reviewers of the MAKE journal and the First Learning on Graphs Conference (LoG 2022) for their comments that helped improve the article.

Conflicts of Interest: The authors declare no conflicts of interest.

Appendix A. Additional Settings and Results

Appendix A.1. Data Set Details

We summarize all graph data sets used in the experimental validation of IGEL in Table A4. To provide an overview of each data set and its usage, we indicate the average cardinality of the graphs in the data set (Avg. n), the average number of edges per graph (Avg. m), and the total number of graphs per data set. We describe the task, loosely grouped in graph classification, non-isomorphism detection, graph regression, link prediction and vertex classification. We also provide the shape of the output for every problem and describe the split regime for training/validation/test sets when relevant.

Reproducibility —We provide three code repositories with our code and changes to the original benchmarks, including our modeling scripts, metadata, and experimental results: https://github.com/nur-ag/IGEL, https://github.com/nur-ag/gnn-matlang and https://github.com/nur-ag/IGEL-GNN-AK (all accessed on 16 September 2023).

Appendix A.2. Hyper-Parameters and Experiment Details

Graph Level Experiments. We reproduce the benchmark of [26] without modifying model hyper-parameters for the tasks of graph classification, graph isomorphism detection, and graphlet counting. For classification tasks, the six models in Table 2 are trained on binary/categorical cross-entropy objectives depending on the task. For Graph isomorphism detection, we train GNNs as binary classification models on the binary classification task on EXP [53], and identify isomorphisms by counting the number of graph pairs for which randomly initialized MP-GNN models produce equivalent outputs on Graph8c—Simple 8 vertices graphs from: http://users.cecs.anu.edu.au/~bdm/data/graphs.html (accessed on 16 September 2023). This evaluation procedure means models are not trained but simply initialized, following the approach of [26]. We also evaluate on the SR25 dataset and find IGEL encodings cannot distinguish the 105 Strongly Regular graph pairs with parameters $\texttt{SRG}(25, 12, 5, 6)$ from: http://users.cecs.anu.edu.au/~bdm/data/graphs.html (accessed on 16 September 2023). For the graphlet counting regression task on the RandomGraph data set [54], we train models to minimize mean squared error (MSE) on the normalized graphlet counts for five types of graphlets.

On all tasks, we experiment with $\alpha \in \{1, 2\}$ and optionally introduce a preliminary linear transformation layer to reduce the dimensionality of IGEL encodings. For every setup, we execute the same configuration 10 times with different seeds and compare runs introducing IGEL or not by measuring whether differences on the target metric (e.g., accuracy or MSE) are statistically significant, as shown in Tables 1 and 2. In Table A1, we provide the value of α that was used in our experimental results. Our results show that the choice of α depends on both the task and model type. We believe these results may be applicable to subgraph-based MP-GNNs, and will explore how different settings, graph sizes, and downstream models interact with α in future work.

Table A1. Values of α used when introducing IGEL in the best reported configuration for graphlet counting and graph classification tasks. The table is broken down by graphlet types (upper section) and graph classification tasks on the TU data sets (bottom section).

	Chebnet	GAT	GCN	GIN	GNNML3	Linear	MLP
Star	2	1	2	1	1	2	1
Tailed Triangle	1	1	1	1	2	1	1
Triangle	1	1	1	1	1	1	1
4-Cycle	2	1	1	1	1	1	1
Custom Graphlet	2	1	1	1	2	2	2
Enzymes	1	2	2	1	2	2	2
Mutag	1	1	1	1	1	1	2
Proteins	2	2	2	1	2	1	1
PTC	1	1	2	1	1	2	2

Vertex and Edge-level Experiments. In this section we break down the best performing hyper-parameters on the edge- (link prediction) and vertex-level (node classification) experiments.

Link Prediction—The best performing hyperparameter configuration on the Facebook graph including $\alpha = 2$, learning $t = 256$ component vectors with $e = 10$ walks per node, each of length $s = 150$ and $p = 8$ negative samples per positive for the self-supervised negative sampling. Respectively, on the arXiv citation graph, we find the best configuration at $\alpha = 2$, $t = 256$, $e = 2$, $s = 100$ and $p = 9$.

Node Classification—In the node classification experiment, we analyze both encoding distances $\alpha \in \{1, 2\}$. Other IGEL hyper-parameters are fixed after a small greedy search based on the best configurations in the link prediction experiments. For the MLP model, we perform a greedy architecture search, including number of hidden units, activation

functions and depth. Our results show scores averaged over five different seeded runs with the same configuration obtained from hyperparameter search.

The best performing hyperparameter configuration on the node classification is found with $\alpha = 2$ on $t = 256$ length embedding vectors, concatenated with node features as the input layer for 1000 epochs in a three-layer MLP using ELU activations with a learning rate of 0.005. Additionally, we apply 100 epoch patience for early stopping, monitoring the F1 score on the validation set.

Reproducibility—We provide a replication folder in the code repository for the exact configurations used to run the experiments.

Appendix B. Relationship to Weisfeiler–Lehman Graph Kernels

The Weisfeiler–Lehman algorithm has inspired the design of graph kernels for graph classification, as proposed by [45,58]. In particular, Weisfeiler–Lehman graph kernels [45] produce representations of graphs based on the labels resulting of evaluating several iterations of the 1-WL test described in Algorithm 1. The resulting vector representation resembles IGEL encodings, particularly in vector form—$\text{IGEL}^{\alpha}_{\texttt{vec}}(v)$. In the case of WL kernels, the `hash` function maps sorted label multi-sets in terms of their position in lexicographic order within the iteration. An iteration of the algorithm is illustrated in Figure A1.

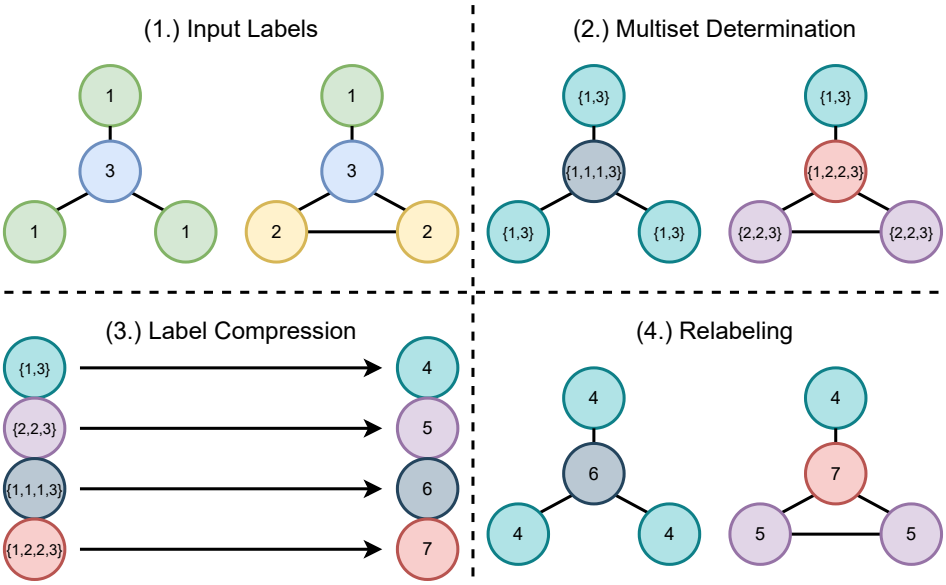


Figure A1. One iteration of the Weisfeiler–Lehman graph kernel where input labels are node degrees. Given an input labeling, one iteration of the kernel computes a new set of labels based on the sorted multi-sets produced by aggregating the labels of each node's neighbors.

After k iterations, graphs are represented by counting the frequency of distinct labels c^i_v that were observed at each iteration $1 \leq i \leq k$. The resulting vector representation can be used to compare whether two graphs are similar and as an input to graph classification models.

However, WL graph kernels can suffer from generalization problems, as the label compression step assigns different labels to multi-sets that differ on a single element within the multi-set. If a given graph contains a previously unseen label, each iteration will produce previously unseen multi-sets, propagating at each iteration step and potentially harming model performance. Recent works generalize certain iteration steps of WL graph kernels to address these limitations, introducing topological information [59] or Wasserstein distances between node attribute to derive labels [60]. IGEL can be understood as another work in that direction, removing the hashing step altogether and simply relying on target structural features—path length and distance tuples—that can be inductively computed in unseen or mutating graphs.

Appendix C. IGEL Is Permutation Equivariant

Lemma A1. *Given any $v \in V$ for $G = (V, E)$ and given a permuted graph $G' = (V', E')$ of G produced by a permutation of node labels $\pi : V \to V'$ such that $\forall v \in V \Leftrightarrow \pi(v) \in V'$, $\forall (u, v) \in E \Leftrightarrow (\pi(u), \pi(v)) \in E'$.*

The IGEL *representation is permutation equivariant at the graph level*

$$\pi(\{\{e^{\alpha}_{v_1}, \ldots, e^{\alpha}_{v_n}\}\}) = \{\{e^{\alpha}_{\pi(v_1)}, \ldots, e^{\alpha}_{\pi(v_n)}\}\}.$$

The IGEL *representation is permutation invariant at the node level*

$$e^{\alpha}_{v} = e^{\alpha}_{\pi(v)}, \forall v \in G.$$

Proof. Note that e^{α}_{v} in Algorithm 2 can be expressed recursively as

$$e^{\alpha}_{v} = \left\{\!\!\left\{ \left(l_{\mathcal{E}^{\alpha}_{v}}(u, v), d_{\mathcal{E}^{\alpha}_{v}}(u) \right) \middle| \; \forall \, u \in \mathcal{N}^{\alpha}_{G}(v) \right\}\!\!\right\}.$$

Since IGEL only relies on node distances $l_G(\cdot, \cdot)$ and degree nodes $d_G(\cdot)$, and both $l_G(\cdot, \cdot)$ and $d_G(\cdot)$ are permutation invariant (and the node level) and equivariant (at the graph level) functions, the IGEL representation is permutation equivariant at the graph level, and permutation invariant at the node level. □

Appendix D. Connecting IGEL and SPNNs

In this section, we connect IGEL with the shortest-path neural network (**SPNN**) model [40] and show that IGEL is strictly more expressive than SPNNs.

Proposition A1. IGEL *is strictly more expressive than SPNNs on unattributed graphs.*

Proof. First, we show that IGEL encodings contain at least the same the information as SPNNs in unattributed graphs. Let $\mathcal{L}^{k}_{G}(v) = \{u | u \in V \wedge l_G(u, v) = k\}$ be the nodes in G exactly at distance k of v. In [40], the embedding of v at layer $t+1$ ($\mathbf{h}^{t+1}_{v}$) in a k-depth SPNN is defined based on the following aggregation over the 1, ..., k-distance neighborhoods:

$$(1 + \epsilon) \cdot \mathbf{h}^{t}_{v} + \sum_{i=1}^{k} \psi_i \sum_{u \in \mathcal{L}^{i}_{G}(v)} \mathbf{h}^{t}_{u}, \tag{A1}$$

where ϵ and ψ_i are learnable parameters that modulate aggregation of node embeddings for node v and node embeddings for nodes at a distance i, respectively.

For unattributed graphs, the information captured by Equation (A1) is equivalent to counting the frequency of all node distances to node v in $\mathcal{E}^{\alpha}_{v}$. This can be written as a reduced IGEL representation following Equation (4):

$$\texttt{SPNN}^{\alpha}_{v} = \left\{\!\!\left\{ l_{\mathcal{E}^{\alpha}_{v}}(u, v) \middle| \; \forall \, u \in \mathcal{N}^{\alpha}_{G}(v) \right\}\!\!\right\}. \tag{A2}$$

This representation captures an encoding that only considers *distances*. While $\texttt{SPNN}^{\alpha=1}_{v}$ can be used to compute the degree of $v \in V$ in the entire graph G, it does not capture the degree of any node u only within the ego network $\mathcal{E}^{\alpha}_{v}$. Thus, by definition, the IGEL encoding of Equation (4), contains *at least* all the necessary information to construct Equation (A2)—showing that IGEL is at least as expensive as SPNNs in unattributed graphs.

Second, we show that there exist unattributed graphs that IGEL can distinguish but that cannot be distinguished by SPNNs. Figure A2 shows an example. Thus, IGEL is strictly more expressive than SPNNs. □

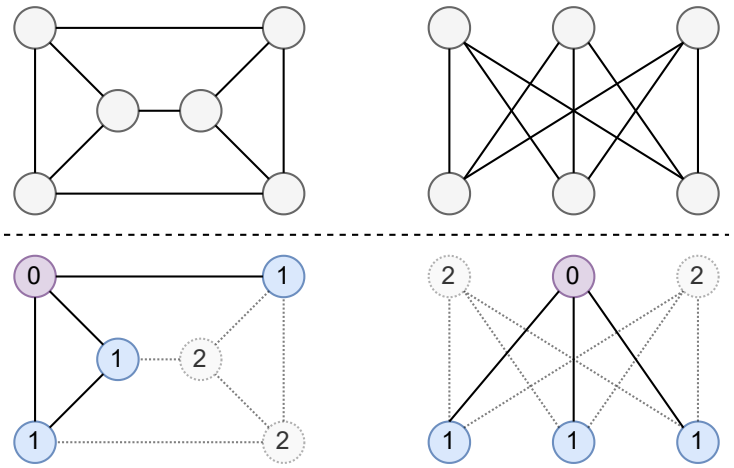


Figure A2. IGEL encodings for two SPNN-indistinguishable graphs [40] from [61] (top). IGEL with $\alpha = 1$ distinguishes both graphs (bottom) as all ego networks form tailed triangles on the right graph and stars on the left. However, SPNNs (as well as 1-WL) fail to distinguish them as all ego network roots (purple) have three adjacent nodes (blue) and two non-adjacent nodes at 2-hops (dotted).

Appendix E. IGEL and MATLANG

A natural question is to explore how IGEL relates to MATLANG—whether IGEL can be expressed in $\mathbf{ML}_1$, $\mathbf{ML}_2$, or $\mathbf{ML}_3$, and whether certain MATLANG operations (and hence languages containing them) can be reduced to IGEL. We focus on the former to evaluate whether IGEL can be computed for a given node as an expression in MATLANG.

Appendix E.1. MATLANG *Sub-Languages*

As introduced in Section 1, MATLANG is a language of operations on matrices. MATLANG sentences are formed by sequences of operations containing matrix multiplication $(\cdot)$, addition $(+)$, transpose $(^\intercal)$, element-wise multiplication $(\odot)$, column vector of ones $(\mathbf{1})$, vector diagonalization $(\texttt{diag})$, matrix trace $(\texttt{tr})$, scalar-matrix/vector multiplication $(\times)$, and element-wise function application (f). Given MATLANG's operations $\mathbf{ML} = \{\cdot, +, ^\intercal, \odot, \mathbf{1}, \texttt{diag}, \texttt{tr}, \times, f\}$, [13] shows that a language containing $\mathbf{ML}_1 = \{\cdot, ^\intercal, \mathbf{1}, \texttt{diag}\}$ is as powerful as the 1-WL test, $\mathbf{ML}_2 = \{\cdot, ^\intercal, \mathbf{1}, \texttt{diag}, \texttt{tr}\}$ is strictly more expressive than the 1-WL test, but less expressive than the 3-WL test, and $\mathbf{ML}_3 = \{\cdot, ^\intercal, \mathbf{1}, \texttt{diag}, \texttt{tr}, \odot\}$ is as powerful as the 3-WL test. For $\mathbf{ML}_1, \mathbf{ML}_2, \mathbf{ML}_3$, enriching the language with $\{+, \times, f\}$ has no impact on expressivity.

Appendix E.2. Can IGEL *Be Represented in* MATLANG*?*

Let $A \in \{0,1\}^{n \times n}$ be the adjacency matrix of $G = (V, E)$ where $n = |V|$. Our objective is to find a sequence of operations in $\mathbf{ML} = \{\cdot, +, ^\intercal, \odot, \mathbf{1}, \texttt{diag}, \texttt{tr}, \times, f\}$ such that when applied to A, we can express the same information as $\text{IGEL}^{\alpha}_{\texttt{vec}}(v)$ for all $v \in V$. To compute the IGEL encoding of v for a given α, we must write $\mathbf{ML}$ sentences to compute:

(a) $\mathcal{E}^{\alpha}_{v} = (V', E')$, the ego network of v in G at depth α.
(b) The degrees of every node in $\mathcal{E}^{\alpha}_{v}$.
(c) The shortest path lengths from every node in $\mathcal{E}^{\alpha}_{v}$ to v.

Let $A^{\alpha}_{v} \in \{0,1\}^{n \times n}$ denote the adjacency matrix of $\mathcal{E}^{\alpha}_{v}$. Provided we can compute A^{α}_{v} from A, computing the degree is a trivial operation in $\mathbf{ML}_1$ as we only need to compute the matrix-vector product $(\cdot)$ of A^{α}_{v} with the vector of ones $(\mathbf{1})$. Recall that $d_{\mathcal{E}^{\alpha}_{G}(v)}(u)$ is the degree of vertex $u \in \mathcal{E}^{\alpha}_{v}$, and let $(\cdot)_i$ denote the i-th index in the resulting vector:

$$d_{\mathcal{E}^{\alpha}_{G}(v)}(u) = (A^{\alpha}_{v} \cdot \mathbf{1})_u$$

Furthermore, if we can find A_v^α, it is also possible to find $A_v^k \; \forall \; k \in \{1, ..., \alpha\}$. Thus, we can compute the distance of every node to v by progressively expanding k, mapping the degrees of nodes to the value of k being processed by means of unary function application (f), and then computing the minimum value. Let ξ denote a distance vector containing entries for all vertices found at distance k of v:

$$\xi_{\mathcal{E}_G^k(v)} = A_v^\alpha \cdot \mathbf{1} f_{\mathtt{k}}$$

where

$$f_{\mathtt{k}}(x) = \begin{cases} k+1 & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}$$

The distance $l_{\mathcal{E}_v^\alpha}(u)$ between a given node u to v in $\mathcal{E}_v^\alpha$ can then be computed in terms of $\mathbf{ML}_1$ operations that may be applied recursively:

$$l_{\mathcal{E}_v^\alpha}(u) = \left(\xi_{\mathcal{E}_G^\alpha(v)} - l_{\mathcal{E}_v^{\alpha-1}}(u)\right) f_{\mathtt{min}}^\alpha$$

where $f_{\mathtt{min}}^\alpha$ is a unary function that retrieves the minimum length from having computed $\alpha - l_{\mathcal{E}_v^{\alpha-1}}(u)$:

$$f_{\mathtt{min}}^\alpha(x) = \begin{cases} \alpha & \text{if } x = \alpha \\ \alpha - x & \text{if } x < \alpha \wedge x > 0 \\ 0 & \text{otherwise} \end{cases}$$

Thus, computing the distance is also possible in $\mathbf{ML}_1$ given A_v^α. However, in order to compute A_v^α, we must be able to extract a subset of V contained in $\mathcal{E}_v^\alpha$. Following the approach of [13] (see Proposition 7.2), we may leverage a node indicator vector $\mathbf{1}_{V'} \in \{0,1\}^n$ for $V' \subseteq V$ where $\mathbf{1}_{V'v} = 1$ when $v \in V'$ and 0 otherwise. We can then compute A_v^α via a diagonal mask matrix $M \in \{0,1\}^{n \times n}$ such that $M_{i,i} = 1$ when $v_i \in V'$, and $M_{i,i} = 0$ when $v_i \notin V'$, computed as $M = \mathbf{1}_{V'} \cdot \mathbf{1}_{V'}^\intercal$. Finding A_v^α involves

$$A_v^\alpha = M \cdot A \cdot M \tag{A3}$$

It follows from Equation (A3) that if we are provided M, it is possible to compute A_v^α in $\mathbf{ML}_1$. However, since indicator vectors are not part of **ML**, it is not possible to extract the ego subgraph for a given node v. As such IGEL cannot be expressed within MATLANG unless indicator vectors are introduced.

A natural question is whether it is possible to express IGEL with an operator that requires no knowledge of V' unlike indicator vectors, which require computing $\mathcal{E}_v^\alpha = (V', E')$ beforehand. One possible approach is to only allow indicator vectors for single vertices, encoding any $V' \subseteq V$ only if $|V'| = 1$. We denote single-vertex indicator vectors as $\texttt{one-hot}_v$—an operation that represents the one-hot encoding of v. Note that for any $V' \subseteq V$, its indicator vector $\mathbf{1}_{V'}$ can be computed as the sum of one-hot encoding vectors: $\mathbf{1}_{V'} = \sum_{v \in V'} \texttt{one-hot}_v$. Thus, introducing `one-hot` is as expressive as introducing indicator vectors.

We now express IGEL in terms of the `one-hot` operation. Consider the following $\mathbf{ML}_1$ expression where $\mathbf{Z}_v^0 = \texttt{one-hot}_v$:

$$\mathbf{Z}_v^{i+1} = \left(A \cdot \mathbf{Z}_v^i\right) f_{\mathtt{bin}}$$

For $\mathbf{Z}_v^1$, we obtain an indicator vector containing the neighbors of v—matching $\mathcal{N}_G^1(v)$—which is binarized (mapping non-zero values to 1—e.g., applying $f_{\mathtt{bin}}$ returns 0 when x is 0, and 1 otherwise). Furthermore, when computed recursively for α steps, $\mathbf{Z}_v^\alpha$

matches the indicator vector of $\mathcal{N}_G^\alpha(v)$, which can trivially be used to compute $\xi_{\mathcal{E}_G^k(v)}$. We can then find M as used in Equation (A3)

$$M = \left(\texttt{diag}\left(\mathbf{Z}_v^\alpha\right)\right) f_{\texttt{bin}}$$

Thus, introducing $\texttt{one-hot}_v$ is sufficient to express IGEL within MATLANG as $\mathbf{ML}_1$. Given our results in Sections 4 and 5, introducing $\texttt{one-hot}_v$ in MATLANG produces matrix languages that are at least as expressive as IGEL. We leave the study of the expressivity of MATLANG after introducing a transductive operation such as $\texttt{one-hot}_v$ as future work.

Appendix F. Implementing IGEL through Breadth-First Search

The idea behind the IGEL encoding is to represent each vertex v by compactly encoding its corresponding ego network $\mathcal{E}_v^\alpha$ at depth α. The choice of encoding consists of a histogram of vertex degrees at distance $d \leq \alpha$, for each vertex in $\mathcal{E}_v^\alpha$. Essentially, IGEL runs a breadth-first traversal up to depth α, counting the number of times the same degree appears at distance $d \leq \alpha$.

The algorithm shown in Algorithm 2 showcases IGEL and its relationship to the 1-WL test. However, in a practical setting, it might be preferable to implement IGEL through breadth-first search (BFS). In Algorithm A1, we show one such implementation that fits the time and space complexity described in Section 3.

Algorithm A1 IGEL Encoding (BFS).

Input: $v \in V, \alpha \in \mathbb{N}$

```
toVisit := [ ]                                   ▷ Queue of nodes to visit.
degrees := { }                                   ▷ Mapping of nodes to their degrees.
distances := {v : 0}                             ▷ Mapping of nodes to their distance to v
while toVisit ≠ ∅ do
    u := toVisit.dequeue()
    currentDistance := distances[u]
    currentDegree := 0
    for w ∈ u.neighbors() do
        if w ∉ distances then
            distances[w] := currentDistance + 1
                                                 ▷ w is a new node 1-hop further from v.
        end if
        if distances[w] ≤ α then
            currentDegree := currentDegree + 1
                                                 ▷ Count edges only within α-hops.
            if w ∉ degrees then                  ▷ Enqueue if w has not been visited.
                toVisit.append(w)
            end if
        end if
    end for
    degrees[u] := currentDegree
                                                 ▷ u is now visited: we know its degree and distance to v.
end while
e_v^α = {{(distances[u], degrees[u]) ∀ u ∈ degrees.keys()}}
                                                 ▷ Produce the multi-set of (distance, degree) pairs for visited nodes.
```

Output: $e_v^\alpha : (\mathbb{N}, \mathbb{N}) \to \mathbb{N}$

Due to how we structure BFS to count degrees and distances in a single pass, each edge is processed twice—once for each node at end of the edge. It must be noted that when processing every $v \in V$, the time complexity is $\mathcal{O}(n \cdot \min(m, (d_{\texttt{max}})^\alpha))$. However, the BFS

implementation is also embarrassingly parallel, which means that, as noted in Section 3, it can be distributed over p processors with $\mathcal{O}(n \cdot \min(m, (d_{\texttt{max}})^{\alpha})/p)$ time complexity.

Appendix G. Extending IGEL beyond Unattributed Graphs

In this appendix, we explore extensions to IGEL beyond unattributed graphs. In particular, we describe how minor modifications would allow IGEL to leverage label and attribute information, connecting IGEL with state-of-the-art MP-GNN representations.

Appendix G.1. Application to Labelled Graphs

Labeled graphs are defined given $G = (V, E, \mathcal{L})$, where $\mathcal{L}_G(v) : V \mapsto \mathcal{L}$ is a mapping function assigning a label $L \in \mathcal{L}$ to each vertex. A standard way of applying the 1-WL test to labeled graphs is to replace $d_G(v)$ in the first step of Algorithm 1 with $\mathcal{L}_G(v)$, such that the initial coloring is given by node labels. Since IGEL retains a similar structure, the same modification can be introduced in Equation (4). Some applications might require both degree and label information, achievable through a mapping $C_G(v) : V \mapsto \mathbb{N}$ that combines $\mathcal{L}_G(v)$ and $d_G(v)$ given a label indexing function $I : \mathcal{L} \mapsto \{0, ..., |\mathcal{L}| - 1\}$:

$$C_G(v) = d_G(v) \cdot |\mathcal{L}| + I(\mathcal{L}_G(v))$$

Applying $\mathcal{L}_G(v)$ or $C_G(v)$ only requires adjusting the shape of the vector representation, whose cardinality will depend on the size of the label set (e.g., $|\mathcal{L}|$), or the combination of degrees and $\mathcal{L}$ given by $d_{\texttt{max}} \times |\mathcal{L}|$.

Appendix G.2. Application to Attributed Graphs

IGEL can also be applied to attributed graphs of the form $G = (V, E, \mathbf{X})$, where $\mathbf{X} \in \mathbb{R}^{n \times w}$. Following Algorithm 2, IGEL relies on two discrete functions to represent structural features—the path length between vertices $l_G(u, v)$ and vertex degrees $d_G(v)$. However, in an attributed graph, it may be desirable to consider the attributes of a node besides degree to represent it with respect to its attributes and the attributes of its neighbors. To extend the representation to node attributes, $d_G(v)$ may be replaced by a bucketed similarity function $\phi(u, v) : (V \times V) \mapsto B$ applicable for v and any $u \in \mathcal{E}_G^{\alpha}(v)$, whose output is discretized into $b \in B \subseteq \mathbb{N}$ buckets $1 \leq b \leq |B|$. A straightforward implementation of $\phi_{\texttt{cos}}$ is to compute the cosine similarity ($|\cdot|$) between attribute vectors and remaps the $[-1, 1]$ interval to discrete buckets by rounding:

$$\phi_{\texttt{cos}}(u, v) : \left\lfloor \Big(|B| - 1 \Big) \cdot \frac{(\mathbf{X}_u | \cdot | \mathbf{X}_v) + 1}{2} \right\rceil$$

ϕ-IGEL is a generalization over the structural representation function in Algorithm 2 to take also source vertex v. Thus, unattributed and unlabeled IGEL can be understood as the case in which $\phi(u, v) = d_G(u)$, such that $|B| = d_{\texttt{max}}$.

Furthermore, by introducing the bucketing transformation in ϕ-IGEL, we are de-facto providing a simple mechanism to control the size of IGEL encoding vectors. By implementing $\phi(u, v) = \lfloor d_G(u)/b \rfloor$ or introducing non-linear transformations such as $\phi(u, v) = \lfloor \log(d_G(u))/b \rfloor$ with $b \in \{1, ..., d_{\texttt{max}}\}$, it is possible to compress $\text{IGEL}^{\alpha}_{\texttt{vec}}(v)$ into the $t = (\alpha + 1) \times b$ denser components of a t-dimensional vector.

Appendix G.3. IGEL *and Subgraph MP-GNNs*

Another natural extension is to apply the distance-based aggregation schema to message passing GNNs. This can be achieved by expressing message passing in terms not just

of immediate neighbors of v but from nodes at every distance $1 \leq \tilde{l} \leq \alpha$. Equation (1) then becomes

$$\mu_v^{i,\tilde{l}} = \left\{\!\!\left\{ \textsc{Msg}_G^{i,\tilde{l}}(h_u^{i-1}) \mid \forall\, u \in \mathcal{N}_G^{\alpha}(v) \;\wedge\; l_G(u,v) = \tilde{l} \right\}\!\!\right\}$$

The update step will need to pool over the messages passed onto v:

$$h_v^i = \textsc{Update}_G^i\left(\mu_v^{i,0}, \mu_v^{i,1}, ..., \mu_v^{i,\alpha}, h_v^{i-1}\right)$$

This formulation matches the general form of k-hop GNNs [16] presented in Section 2.3. Furthermore, introducing distance and degree signals in the message passing can yield models analogous to GNN-AK [21]—which explicitly embed the distance of a neighboring node when representing the ego network root during message passing. As such, IGEL directly connects the expressivity of k-hop GNNs with the 1-WL algorithm and provides a formal framework to explore the expressivity of higher order MP-GNN architectures.

Appendix H. Self-Supervised IGEL

We provide additional context for the application of IGEL as a representational input for methods such as DeepWalk [51]. This section describes in detail how self-supervised IGEL embeddings are learned. We also provide a qualitative analysis of IGEL in the self-supervised setting using networks that are amenable for visualization. We focus on analyzing how α influences the learned representations.

IGEL *& Self-Supervised Node Representations*

IGEL can be easily incorporated in standard node representation methods like DeepWalk [51] or node2vec [52]. Due to its relative simplicity, integrating IGEL only requires replacing the input to the embedding method so that IGEL encodings are used rather than node identities. We provide an overview of the process of generating embeddings through DeepWalk, which involves (a) sampling random walks to capture the relationships between nodes and (b) training a negative-sampling based embedding model in the style of word2vec [62] on the random walks to embed random walk information in a compact latent space.

Distributional Sampling through Random Walks— First, we sample π random walks from each vertex in the graph. Walks are generated by selecting the next vertex uniformly from the neighbors of the current vertex. Figure A3 illustrates a possible random walk of length 9 in the graph of Figure 1. By randomly sampling walks, we obtain traversed node sequences to use as inputs for the following negative sampling optimization objective.

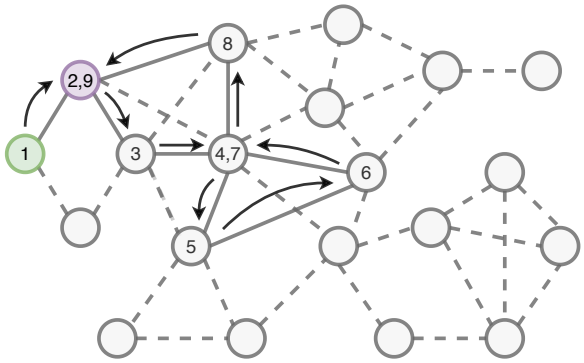


Figure A3. Example of random walk starting on the green node and finishing on the purple node. Nodes contain the time-step when they were visited. The context of a given vertex will be any nodes whose time-steps are within the context window of that node.

Negative Sampling Optimization Objective— Given a random walk ω, defined as a sequence of nodes of length s, we define the context $\mathcal{C}(v, \omega)$ associated with the occurrence of vertex v in ω as the the sub-sequence in ω containing the nodes that appear close to v, including repetitions. Closeness is determined by p, the size of the positive context window, i.e., the context contains all nodes that appear at most p steps before/after the node within ω. In DeepWalk, a skip-gram negative sampling objective learns to represent vertices appearing in similar contexts within random walks.

Given a node $v_c \in V$ in random walk ω, our task is to learn embeddings that assign high probability for nodes $v_o \in V$ appearing in the context $\mathcal{C}(v_c, \omega)$ and lower probability for nodes not appearing in the context. Let $\sigma(\cdot)$ denote the logistic function. As we focus on the learned representation capturing these symmetric relationships, the probability of v_o being in $\mathcal{C}(v_c, \omega)$ is given by

$$p(v_o \in \mathcal{C}(v_c, \omega)) = \sigma(\mathbf{e}_{v_c}^\top \cdot \mathbf{e}_{v_o}),$$

Table A2 summarizes the hyper-parameters of DeepWalk. Our global objective function is the following negative sampling log-likelihood. For each of the π random walks and each vertex v_c at the center of a context in the random walk, we sum the term corresponding to the positive cases for vertices found in the context and the expectation over z negative, randomly sampled vertices.

Table A2. DeepWalk hyper-parameters.

Param.	Description
$\pi \in \mathbb{N}$	Random walks per node
$s \in \mathbb{N}$	Steps per random walk
$z \in \mathbb{N}$	# of neg. samples
$p \in \mathbb{N}$	Context window size

Let $P_n(V)$ be a noise distribution from which the z negative samples are drawn; our task is to maximize (A4) through gradient ascent:

$$l_{\mathrm{u}}(\mathbf{W}) = \sum_{j=1}^{w} \sum_{\substack{v_c \in \omega_j, \\ n_o \in \mathcal{C}(v_c, \omega_j)}} \left[\log \sigma(\mathbf{e}_{v_c}^\top \cdot \mathbf{e}_{n_o}) + \sum_{i=1}^{z} \mathbb{E}_{v_i \sim P_n(V)} \left[\log \sigma(-\mathbf{e}_{v_c}^\top \cdot \mathbf{e}_{v_i}) \right] \right]. \tag{A4}$$

Defining the IGEL*-DeepWalk embedding function*— In DeepWalk, for every vertex $v \in V$, there is a corresponding t-dimensional embedding vector $e_v \in \mathbb{R}^t$. As such, one can represent the embedding function as a product between a one-hot encoded vector corresponding to the index of the vertex $\texttt{one-hot}_v \in \mathbb{B}^n$ where and an embedding matrix $\mathbf{E}_V \in \mathbb{R}^{n \times t}$:

$$\mathbf{e}_v = \texttt{one-hot}_v \cdot \mathbf{E}_V$$

Introducing IGEL requires modifying the shape of E_V to account for the shape of t_{vec}-dimensional $\mathrm{IGEL}^{\alpha}_{\texttt{vec}}(v)$ encoding vectors dependent on α and $d_{\texttt{max}}$, so that IGEL embeddings are computed as a weighted sum of embeddings corresponding to each (path length, degree) pair. Let $\mathbf{E}^{\alpha}_{\mathrm{IGEL}} \in \mathbb{R}^{t_{vec} \times t_{emb}}$ define a structural embedding matrix with one embedding per (path length, degree) pair; a linear IGEL can be defined as:

$$\mathrm{IGEL}^{\alpha}_{\texttt{emb}}(v) = \mathrm{IGEL}^{\alpha}_{\texttt{vec}}(v) \cdot \mathbf{E}^{\alpha}_{\mathrm{IGEL}}$$

Since the definition of $\mathrm{IGEL}^{\alpha}_{\texttt{emb}}(v)$ is differentiable, it can be used as a drop-in replacement of $\mathbf{e}_v$ in Equation (A4).

Appendix I. Comparison of Existing Methods

In this section, we provide a summarized overview of the existing work presented in Section 2, including an overview of the main methods that exemplify *distance-aware*, *subgraph*, *structural*, and *spectral* approaches most related to our theoretical and practical work. Table A3 summarizes the main model architectures, the extensions they introduce within the message passing mechanism, and the limitations associated with each approach.

In contrast with existing methods, IGEL is capable of boosting the expressivity of underlying MP-GNN architectures as additional features concatenated to node/edge attributes—that is, without requiring any modification to the design of the downstream model.

Table A3. Comparison of state-of-the-art methods most closely related to IGEL. We provide an overview of the four families of approaches that extend the expressivity of GNNs beyond 1-WL, summarizing key model architectures, the message passing extensions that they introduce in the learning process, and the common limitations for each approach.

Approach	Model Architectures	Message Passing Extensions	Limitations
Distance-Aware	PGNNs [15] DE-GNNs [38] SPNNs [40]	Introduce distance to target nodes, ego network roots as an encoded variables, or as the explicit distance as part of the message passing mechanism.	Considers a fixed set of target nodes or requires architecture modifications to introduce permutation equivariant distance signals.
subgraph	k-hop GNNs [16] Nested-GNNs [20] ESAN [22] GNN-AK [21] SUN [23]	Propagate information through subgraphs around k-hops, either applying GNNs on the subgraphs, pooling subgraphs, and introducing distance/context signals within the subgraph.	Requires learning over subgraphs with an increased time and memory cost and architecture modifications to propagate messages across subgraphs (incl. distance, context, and node attribute information).
Structural	SMP [18] GSNs [19]	Explicitly introduce structural information in the message passing mechanism, e.g., counts of cycles or stars where a node appears.	Requires identifying and introducing structural features (e.g., node roles in the network) during message passing through architecture modifications.
Spectral	GNN-ML3 [26]	Introduce features from the spectral domain of the graph in the message passing mechanism.	Requires cubic-time eigenvalue decomposition to construct spectral features and architecture modifications to introduce them during message passing.

Table A4. Overview of the graphs used in the experiments. We show the average number of vertices (Avg. n), edges (Avg. m), number of graphs, target task, output shape, and splits (when applicable).

	Avg. n	Avg. m	Num. Graphs	Task	Output Shape	Splits (Train/Valid/Test)
Enzymes	32.63	62.14	600	Multi-class Graph Class.	6 (multi-class probabilities)	9-fold/1 fold (Graphs, Train/Eval)
Mutag	17.93	39.58	188	Binary Graph Class.	2 (binary class probabilities)	9-fold/1 fold (Graphs, Train/Eval)
Proteins	39.06	72.82	1113	Binary Graph Class.	2 (binary class probabilities)	9-fold/1 fold (Graphs, Train/Eval)
PTC	25.55	51.92	344	Binary Graph Class.	2 (binary class probabilities)	9-fold/1 fold (Graphs, Train/Eval)
Graph8c	8.0	28.82	11,117	Non-isomorphism Detection	N/A	N/A
EXP Classify	44.44	111.21	600	Binary Class. (pairwise graph distinguishability)	1 (non-isomorphic graph pair probability)	Graph pairs 400/100/100
SR25	25	300	15	Non-isomorphism Detection	N/A	N/A
RandomGraph	18.8	62.67	5000	Regression (Graphlet Counting)	1 (graphlet counts)	Graphs 1500/1000/2500
ZINC-12K	23.1	49.8	12,000	Molecular prop. regression	1	Graphs 10,000/1000/1000
PATTERN	118.9	6079.8	14,000	Recognize subgraphs	2	Graphs 10,000/2000/2000
ArXiv ASTRO-PH	18,722	198,110	1	Binary Class. (Link Prediction)	1 (edge probability)	Randomly sampled edges 50% train/50% test
Facebook	4039	88,234	1	Binary Class. (Link Prediction)	1 (edge probability)	Randomly sampled edges 50% train/50% test
PPI	2373	68,342.4	24	Multi-label Vertex Class.	121 (binary class probabilities)	Graphs 20/2/2

References

1. Bronstein, M.M.; Bruna, J.; Cohen, T.; Veličković, P. Geometric Deep Learning: Grids, Groups, Graphs, Geodesics, and Gauges. *arXiv* **2021**, arXiv:2104.13478.
2. 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 International Conference on Knowledge Discovery & Data Mining, London, UK, 19–23 August 2018; pp. 974–983.
3. Duvenaud, D.K.; Maclaurin, D.; Iparraguirre, J.; Bombarell, R.; Hirzel, T.; Aspuru-Guzik, A.; Adams, R.P. Convolutional Networks on Graphs for Learning Molecular Fingerprints. In Proceedings of the Advances in Neural Information Processing Systems, Montreal, QC, Canada, 7–12 December 2015; Volume 28.
4. Gilmer, J.; Schoenholz, S.S.; Riley, P.F.; Vinyals, O.; Dahl, G.E. Neural Message Passing for Quantum Chemistry. *Proc. 34th Int. Conf. Mach. Learn.* **2017**, *70*, 1263–1272.
5. Samanta, B.; De, A.; Jana, G.; Gómez, V.; Chattaraj, P.; Ganguly, N.; Gomez-Rodriguez, M. NEVAE: A Deep Generative Model for Molecular Graphs. *J. Mach. Learn. Res.* **2020**, *21*, 1–33. [CrossRef]
6. Battaglia, P.; Pascanu, R.; Lai, M.; Jimenez Rezende, D.; Kavukcuoglu, K. Interaction Networks for Learning about Objects, Relations and Physics. In Proceedings of the Advances in Neural Information Processing Systems, Barcelona, Spain, 5–10 December 2016; Volume 29.
7. Weisfeiler, B.; Leman, A. The Reduction of a Graph to Canonical Form and the Algebra which Appears Therein. *Nauchno-Tech. Informatsia* **1968**, *2*, 12–16.
8. 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.
9. 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. *Proc. AAAI Conf. Artif. Intell.* **2019**, *33*, 4602–4609.
10. Grohe, M. *Descriptive Complexity, Canonisation, and Definable Graph Structure Theory*; Lecture Notes in Logic; Cambridge University Press: Cambridge, UK, 2017.
11. Brijder, R.; Geerts, F.; Bussche, J.V.D.; Weerwag, T. On the Expressive Power of Query Languages for Matrices. *ACM Trans. Database Syst.* **2019**, *44*, 1–31. [CrossRef]
12. Barceló, P.; Kostylev, E.V.; Monet, M.; Pérez, J.; Reutter, J.; Silva, J.P. The Logical Expressiveness of Graph Neural Networks. In Proceedings of the International Conference on Learning Representations, Virtual Conference, 26 April–1 May 2020.
13. Geerts, F. On the Expressive Power of Linear Algebra on Graphs. *Theory Comput. Syst.* **2021**, *65*, 1–61. [CrossRef]
14. Morris, C.; Lipman, Y.; Maron, H.; Rieck, B.; Kriege, N.M.; Grohe, M.; Fey, M.; Borgwardt, K. Weisfeiler and Leman go machine learning: The story so far. *arXiv* **2021**, arXiv:2112.09992.
15. You, J.; Ying, R.; Leskovec, J. Position-aware Graph Neural Networks. In Proceedings of the 36th International Conference on Machine Learning, Long Beach, CA, USA, 9–15 June 2019; Volume 97, pp. 7134–7143.
16. Nikolentzos, G.; Dasoulas, G.; Vazirgiannis, M. k-hop graph neural networks. *Neural Netw.* **2020**, *130*, 195–205. [CrossRef] [PubMed]
17. Bodnar, C.; Frasca, F.; Otter, N.; Wang, Y.; Liò, P.; Montufar, G.F.; Bronstein, M. Weisfeiler and Lehman Go Cellular: CW Networks. *Adv. Neural Inf. Process. Syst.* **2021**, *34*, 2625–2640.
18. Vignac, C.; Loukas, A.; Frossard, P. Building powerful and equivariant graph neural networks with structural message-passing. *Adv. Neural Inf. Process. Syst.* **2020**, *33*, 14143–14155.
19. Bouritsas, G.; Frasca, F.; Zafeiriou, S.; Bronstein, M.M. Improving Graph Neural Network Expressivity via Subgraph Isomorphism Counting. *IEEE Trans. Pattern Anal. Mach. Intell.* **2022**, *45*, 657–668. [CrossRef]
20. Zhang, M.; Li, P. Nested graph neural networks. *Adv. Neural Inf. Process. Syst.* **2021**, *34*, 15734–15747.
21. Zhao, L.; Jin, W.; Akoglu, L.; Shah, N. From Stars to Subgraphs: Uplifting Any GNN with Local Structure Awareness. In Proceedings of the International Conference on Learning Representations, Virtual Conference, 25–29 April 2022
22. Bevilacqua, B.; Frasca, F.; Lim, D.; Srinivasan, B.; Cai, C.; Balamurugan, G.; Bronstein, M.M.; Maron, H. Equivariant Subgraph Aggregation Networks. In Proceedings of the International Conference on Learning Representations, Virtual Conference, 25–29 April 2022 .
23. Frasca, F.; Bevilacqua, B.; Bronstein, M.M.; Maron, H. Understanding and Extending Subgraph GNNs by Rethinking Their Symmetries. In Proceedings of the Advances in Neural Information Processing Systems, Virtual conference and New Orleans, LA, USA, 28 November–9 December 2022 .
24. Michel, G.; Nikolentzos, G.; Lutzeyer, J.; Vazirgiannis, M. Path Neural Networks: Expressive and Accurate Graph Neural Networks. In Proceedings of the 40th International Conference on Machine Learning (ICML), Honolulu, HI, USA, 23–29 July, 2023 .
25. Maron, H.; Ben-Hamu, H.; Serviansky, H.; Lipman, Y. Provably Powerful Graph Networks. In Proceedings of the Advances in Neural Information Processing Systems, Vancouver, BC, Canada, 8–14 December 2019; Volume 32.
26. Balcilar, M.; Héroux, P.; Gaüzère, B.; Vasseur, P.; Adam, S.; Honeine, P. Breaking the Limits of Message Passing Graph Neural Networks. In Proceedings of the 38th International Conference on Machine Learning (ICML), Virtual conference, 18–24 July 2021 .

27. Hu, W.; Fey, M.; Zitnik, M.; Dong, Y.; Ren, H.; Liu, B.; Catasta, M.; Leskovec, J. Open Graph Benchmark: Datasets for Machine Learning on Graphs. In Proceedings of the Advances in Neural Information Processing Systems 33 (NeurIPS 2020), Virtual Conference, 6–12 December 2020; Volume 33.
28. Morris, C.; Kriege, N.M.; Bause, F.; Kersting, K.; Mutzel, P.; Neumann, M. TUDataset: A collection of benchmark datasets for learning with graphs, ICML 2020 Workshop on Graph Representation Learning and Beyond (GRL+ 2020), Virtual Conference, 12–18 July 2020.
29. You, J.; Ying, Z.; Leskovec, J. Design Space for Graph Neural Networks. *Adv. Neural Inf. Process. Syst.* **2020**, *33*, 17009–17021.
30. Defferrard, M.; Bresson, X.; Vandergheynst, P. Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering. In Proceedings of the Advances in Neural Information Processing Systems, Barcelona, Spain, 5–10 December 2016; Volume 29.
31. Kipf, T.N.; Welling, M. Semi-Supervised Classification with Graph Convolutional Networks. In Proceedings of the 5th International Conference on Learning Representations ICLR, Toulon, France, 24–26 April 2017 .
32. Hamilton, W.; Ying, Z.; Leskovec, J. Inductive Representation Learning on Large Graphs. In Proceedings of the Advances in Neural Information Processing Systems 30, Long Beach, CA, USA, 4–9 December 2017.
33. 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.
34. Gao, H.; Wang, Z.; Ji, S. Large-Scale Learnable Graph Convolutional Networks. In Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, London, UK, 19–23 August 2018; pp. 1416–1424.
35. Babai, L.; Kucera, L. Canonical labelling of graphs in linear average time. In Proceedings of the 20th Annual Symposium on Foundations of Computer Science (sfcs 1979), San Juan, PR, USA, 29–31 October 1979; pp. 39–46. [CrossRef]
36. Huang, N.T.; Villar, S. A Short Tutorial on The Weisfeiler–Lehman Test And Its Variants. In Proceedings of the ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Toronto, ON, Canada, 6–11 June 2021; pp. 8533–8537.
37. Veličković, P. Message passing all the way up. In Proceedings of the Workshop on Geometrical and Topological Representation Learning, International Conference on Learning Representations, Virtual Conference, 25–29 April 2022.
38. Li, P.; Wang, Y.; Wang, H.; Leskovec, J. Distance Encoding: Design Provably More Powerful Neural Networks for Graph Representation Learning. *Adv. Neural Inf. Process. Syst.* **2020**, *33*, 4465–4478.
39. You, J.; Gomes-Selman, J.M.; Ying, R.; Leskovec, J. Identity-aware graph neural networks. *Proc. AAAI Conf. Artif. Intell.* **2021**, *35*, 10737–10745.
40. Abboud, R.; Dimitrov, R.; Ceylan, İ.İ. Shortest Path Networks for Graph Property Prediction. In Proceedings of the First Learning on Graphs Conference (LoG), Virtual Conference, 9–12 December 2022 .
41. Maron, H.; Ben-Hamu, H.; Shamir, N.; Lipman, Y. Invariant and Equivariant Graph Networks. In Proceedings of the International Conference on Learning Representations, New Orleans, LA, USA, 6–9 May 2019 .
42. Zhang, B.; Luo, S.; Wang, L.; He, D. Rethinking the Expressive Power of GNNs via Graph Biconnectivity. In Proceedings of the International Conference on Learning Representations, Kigali, Rwanda, 1–5 May 2023 .
43. Papp, P.A.; Wattenhofer, R. A Theoretical Comparison of Graph Neural Network Extensions. *Proc. Mach. Learn. Res.* **2022**, *162*, 17323–17345.
44. Albert, R.; Barabási, A.L. Statistical mechanics of complex networks. *Rev. Mod. Phys.* **2002**, *74*, 47–97. [CrossRef]
45. Shervashidze, N.; Schweitzer, P.; Jan, E.; Leeuwen, V.; Mehlhorn, K.; Borgwardt, K. Weisfeiler–Lehman Graph Kernels. *J. Mach. Learn. Res.* **2010**, *1*, 1–48.
46. Alvarez-Gonzalez, N.; Kaltenbrunner, A.; Gómez, V. Beyond 1-WL with Local Ego-Network Encodings. In Proceedings of the the non-archival Extended Abstract track in the First Learning on Graphs Conference (LoG), Virtual Conference, 9–12 December 2022.
47. Van Dam, E.R.; Haemers, W.H. Which Graphs Are Determined by Their Spectrum? *Linear Algebra Its Appl.* **2003**, *373*, 241–272. [CrossRef]
48. Brouwer, Andries E. and Van Maldeghem, Hendrik. *Strongly Regular Graphs*; Cambridge University Press: Cambridge, UK, 2022; Volume 182, p. 462.
49. Arvind, V.; Fuhlbrück, F.; Köbler, J.; Verbitsky, O. On Weisfeiler-Leman invariance: Subgraph counts and related graph properties. *J. Comput. Syst. Sci.* **2020**, *113*, 42–59. [CrossRef]
50. Dwivedi, V.P.; Joshi, C.K.; Laurent, T.; Bengio, Y.; Bresson, X. Benchmarking Graph Neural Networks. *arXiv* **2020**, arXiv:2003.00982.
51. Perozzi, B.; Al-Rfou, R.; Skiena, S. DeepWalk: Online Learning of Social Representations. In Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, New York, NY, USA, 24–27 August 2014; pp. 701–710.
52. Grover, A.; Leskovec, J. Node2Vec: Scalable Feature Learning for Networks. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, 13–17 August 2016; pp. 855–864.
53. Abboud, R.; Ceylan, I.I.; Grohe, M.; Lukasiewicz, T. The Surprising Power of Graph Neural Networks with Random Node Initialization. In Proceedings of the International Joint Conferences on Artificial Intelligence Organization, Virtual Conference, 19–26 August 2021, pp. 2112–18.
54. Chen, Z.; Chen, L.; Villar, S.; Bruna, J. Can Graph Neural Networks Count Substructures? *Adv. Neural Inf. Process. Syst.* **2020**, *33*, 10383–10395.
55. Hu, W.; Liu, B.; Gomes, J.; Zitnik, M.; Liang, P.; Pande, V.; Leskovec, J. Strategies for Pre-training Graph Neural Networks. In Proceedings of the International Conference on Learning Representations, Virtual Conference, 26 April–1 May 2020 .

56. Leskovec, J.; Krevl, A. SNAP Datasets: Stanford Large Network Dataset Collection, 2014. Available online: https://snap.stanford.edu/data/index.html (accessed on 16 September 2023) .
57. Niepert, M.; Ahmed, M.; Kutzkov, K. Learning Convolutional Neural Networks for Graphs. In Proceedings of the 33rd International Conference on Machine Learning, New York, NY, USA, 19–24 June 2016; Volume 48, pp. 2014–2023.
58. Shervashidze, N.; Borgwardt, K. Fast subtree kernels on graphs. In Proceedings of the Advances in Neural Information Processing Systems, Vancouver, BC, Canada, 7–12 December 2009; Volume 22.
59. Rieck, B.; Bock, C.; Borgwardt, K. A Persistent Weisfeiler–Lehman Procedure for Graph Classification. *Proc. Mach. Learn. Res.* **2019**, *97*, 5448–5458.
60. Togninalli, M.; Ghisu, E.; Llinares-López, F.; Rieck, B.; Borgwardt, K. Wasserstein Weisfeiler–Lehman Graph Kernels. *Adv. Neural Inf. Process. Syst.* **2019**, *32*, 6436–6446.
61. Kriege, N.M.; Morris, C.; Rey, A.; Sohler, C. A Property Testing Framework for the Theoretical Expressivity of Graph Kernels. *Int. Jt. Conf. Artif. Intell. Organ.* **2018**, *7*, 2348–2354.
62. Mikolov, T.; Sutskever, I.; Chen, K.; Corrado, G.; Dean, J. Distributed Representations of Words and Phrases and Their Compositionality. In Proceedings of the 26th International Conference on Neural Information Processing Systems, Lake Tahoe, NV, USA, 5–10 December 2013; Volume 2, pp. 3111–3119.

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.