algorithms

MDPI

Article

Hierarchical and Unsupervised Graph Representation Learning with Loukas's Coarsening

Louis Béthune [1,*,†], Yacouba Kaloga [1,†], Pierre Borgnat [1], Aurélien Garivier [2] and Amaury Habrard [3]

1 ENS de Lyon, UCB Lyon 1, CNRS, Laboratoire de Physique, UMR 5672, 69342 Lyon, France; yacouba.kaloga@ens-lyon.fr (Y.K.); pierre.borgnat@ens-lyon.fr (P.B.)
2 ENS de Lyon, UCB Lyon 1, CNRS, UMPA UMR 5669 and LIP UMR 5668, 69342 Lyon, France; aurelien.garivier@ens-lyon.fr
3 Laboratoire Hubert Curien, UJM-Saint-Etienne, UMR 5516, 42100 Saint-Étienne, France; amaury.habrard@univ-st-etienne.fr
* Correspondence: louis.bethune@ens-lyon.fr
† These authors contributed equally to this work.

Received: 6 July 2020; Accepted: 17 August 2020; Published: 21 August 2020

Abstract: We propose a novel algorithm for unsupervised graph representation learning with attributed graphs. It combines three advantages addressing some current limitations of the literature: (i) The model is inductive: it can embed new graphs without re-training in the presence of new data; (ii) The method takes into account both micro-structures and macro-structures by looking at the attributed graphs at different scales; (iii) The model is end-to-end differentiable: it is a building block that can be plugged into deep learning pipelines and allows for back-propagation. We show that combining a coarsening method having strong theoretical guarantees with mutual information maximization suffices to produce high quality embeddings. We evaluate them on classification tasks with common benchmarks of the literature. We show that our algorithm is competitive with state of the art among unsupervised graph representation learning methods.

Keywords: graph representation learning; Graph2Vec; graph convolutional networks; graph coarsening; unsupervised learning; mutual information maximization

1. Introduction

Graphs are a canonical way of representing objects and relationships among them. They have proved remarkably well suited in many fields such as chemistry, biology, social sciences or computer science in general. The connectivity information (edges) is often completed by discrete labels or continuous attributes on nodes, resulting in so-called attributed graphs. Many real-life problems involving high dimensional objects and their links can be modeled using attributed graphs.

Machine learning offers several ways to solve problems such as classification, clustering or inference, provided that a sufficient amount of training examples is available. Yet, the most classical frameworks are devoted to data living in regular spaces (e.g., vector spaces), and they are not suitable to deal with attributed graphs. One way to overcome this issue is to represent or encode the attributed graphs in such a way that usual machine learning approaches are efficient. A recent take on that is known as graph representation learning [1]: the graphs are embedded in a fixed dimensional latent space such that similar graphs share similar embeddings.

Three properties are desirable in order for a method of attributed graph representation learning to be widely applicable and expressive enough. We expect a method to be: I. unsupervised because labels are expensive, and not always available; II. inductive so that computing the embedding of an

unseen graph (not belonging to the training set) can be done on the fly (in contrast to transductive methods); III. hierarchical so as to take into account properties on both local and global scales; indeed, structured information in graphs can reside at various scales, from small neighborhoods to the entire graph.

In order to obtain these three desirable properties for attributed graphs representation learning, the present work introduces a new Hierarchical Graph2Vec (HG2V) model. Like Graph2Vec [2] with which it shares some similarities, it is based on the maximization of some mutual information. Thanks to a proper use of coarsening, as proposed by Loukas [3], it is hierarchical and incorporates information at all scales, from micro-structures like node neighborhoods up to macro-structures (communities, bridges), by considering a pyramid of attributed graphs of decreasing size.

The article is organised as follows. Section 2 presents some related work in the literature. In Section 3, we introduce the notation and the required background. Section 4 is dedicated to the detailed presentation of our main contribution: the Hierarchical Graph2Vec method. In Section 5, an experimental study is reported that demonstrates the effectiveness of the framework for various tasks.

2. Related Work

Graph Representation learning is related to a large spectrum of works, from kernel algorithms to graph neural networks.

Kernels methods. Graph kernels have become a well established and a widely used technique for learning graph representations [4,5]. They use handcrafted similarity measures between every pair of graphs. Some are restricted to discrete labels [6,7], while others can handle continuous attributes [8–10]. The main drawback of kernel methods is the computational burden of building and storing the kernel matrix, which has quadratic complexity unless using approximation techniques.

Infomax Principle hypothesizes that good representations maximize mutual information between the input data and its embedding. Deep Graph Infomax [11] and GraphSAGE [12] rely on negative sampling to build an estimator of Mutual Information (MI). It is used to produce node embeddings for solving a classification task. InfoGraph [13] uses the same estimator to produce embeddings for entire attributed graphs.

Graph2Vec [2] uses the same MI estimator in combination with Wesfeiler–Lehman procedure (see Section 3) to produce graph embeddings. It was originally inspired by languages models (especially Word2Vec) and considers node embedding as the vocabulary used to "write" a graph.

Graph coarsening. The aim of graph coarsening is to produce a sequence of graphs of decreasing sizes; it can be done by node pooling, as with Loukas's algorithm [3], or by node decimation, like for example Kron reduction [14,15]. Coarsening can be combined with edge sparsification [15,16], so as to reduce the density of coarsened graphs. In another context, DiffPool [17] performs graph coarsening using clustering but it learns the pooling function specific to each task in a supervised manner. MinCutPool [18] also relies on clustering by optimizing a relaxed version of minCut objective, and provides high quality partitioning of the graph.

Graph Neural Networks (GNNs). Developed after the renewed interest in Neural Networks (NNs), they are known to offer interesting graph embedding methods [1], in addition to solving several graph-related task, see [19] (and references therein) for a survey, and specifically the popular Chebyshev GNN [20], GCN [21] and GAT [22]. Stacking them to increase the receptive field may raise scalability issues [23].

Still, it has been shown that some easy problems on graphs (diameter approximation, shortest paths, cycle detection, s-t cut) cannot be solved by a GNN of insufficient depth [24]. By combining GNNs with coarsening, e.g., [3], and node pooling, e.g., [14,17,25], those impossibility results no longer apply. This combination thus helps in designing a method encompassing all structural properties of graphs.

3. Definitions and Background

The proposed method is inspired by Graph2Vec [2], which was itself built upon negative sampling and the Weisfeiler–Lehman (WL) algorithm [26]. The present section recalls the fundamental ideas of those algorithms. The Weisfeiler–Lehman method produces efficient descriptors of the topology of the graph, which allows to create embeddings of good quality through the maximization of Mutual Information.

Definition 1. *An attributed graph is a tuple* (V, A, Z)*, where* V *is the set of nodes,* $A \in \mathbb{R}^{|V|\times|V|}$ *is a weighted adjacency matrix, and* $Z : V \to \mathbb{R}^n$ *is the function that maps each node* $u \in V$ *to its attribute vector* $Z(u)$*. Let* $\mathbb{G}$ *be the space of attributed graphs.*

Definition 2. *A graph embedding is a function* $\mathcal{E} : \mathbb{G} \to \mathbb{R}^d$ *that maps each attributed graph to a vector in the latent space* $\mathbb{R}^d$ *for some non-negative integer* d*.*

3.1. Weisfeiler–Lehman Procedure (WL)

The seminal paper [26] proposes an algorithm initially created in an attempt to solve the graph isomorphism problem (whether or not the graph isomorphism problem belongs to P is still an open problem). It maps the original graph, with discrete node labels, onto a sequence of labelled graphs, by repeatedly applying the same deterministic operator, as sketched in Figure 1. The sequence of node representations generated at each iteration can be used to distinguish between two graphs (although different graphs producing the same sequence exist). The method can be used to build efficient kernels for graph learning [7]. In the following, we will use the **WL**-Optimal Assignment kernel [27] as state-of-the-art. The procedure to generate the labels is the following:

$$x^0(u) = Z(u) \tag{1}$$

$$x^{l+1}(u) = \text{hash}\Big(\big\{x^l(v) | v \in \mathcal{N}(u) \cup \{u\}\big\}\Big) \tag{2}$$

where $\mathcal{N}(u)$ is the set of neighbours of u.

The hashing function has to be injective in order to distinguish between different rooted subtrees. The notation $\{\}$ emphasizes the fact that the output only depends on the unordered set of labels ("bag of labels"). The procedure is iterated on l up to a (user defined) depth L: the deeper, the wider the neighborhood used to distinguish between graphs.

By definition, the label $x^l(u)$ of a node u at the l-th level depends only on the labels of the nodes at distance at most l from u. The output is invariant under node permutation, and hence is the same for isomorphic graphs. If graph g_i contains N_i nodes, then it produces at most N_i new labels per level, for a maximal number of $N_i L$ new labels at the end.

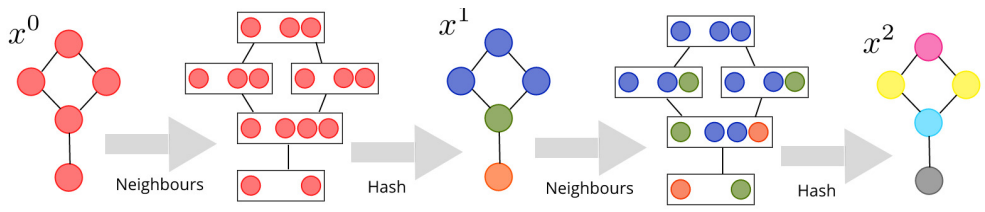

Figure 1. Two iterations of the Weisfeiler–Lehman (**WL**) algorithm, starting with uniform labels.

3.2. Negative Sampling and Mutual Information

Many problems in machine learning amount to learn a distribution $P_{Y|X}$ given training examples $(x, y) \sim P_{X,Y}$. It usually requires the usage of a *Softmax* function, implying the expensive computation of a normalization constant: this is unfeasible when the domain of Y is too big (for example in the context of natural language processing, given the size of vocabulary). Hence, Word2Vec [28] replaces

it by a sequence of binary classification problems, where a discriminator T learns to distinguish between real and fake pairs. Instead of modeling the conditional distribution $P_{Y|X}$, the binary classifier learns to distinguish between examples coming from the joint distribution $P_{X,Y}$ and coming from the product distribution $P_X \otimes P_Y$. Negative Sampling can be used to build the Jensen–Shannon estimator of Mutual Information (MI) between X and Y (see [29] for the first usage of this estimator, see [30] for its construction and [31] for other insights):

$$\widehat{\mathcal{I}}^{(\mathrm{JSD})}(X,Y) = -\mathbb{E}_{(x,y)\sim P_{XY}} \log \sigma(T_\theta(x,y)) - \mathbb{E}_{(x,y)\sim P_X \otimes P_Y} \log \sigma(-T_\theta(x,y)) \tag{3}$$

where $T_\theta : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$ is the discriminator, i.e., a function parameterized by θ (whose exact form is specified by the user, it can be a neural network), and typically $\sigma(x) = 1/(1+e^{-x})$. Maximizing this estimator of MI is equivalent to minimizing cross entropy between the prediction $\sigma(T(x,y))$ and the labels of the binary classification, with $P_{X,Y}$ (resp. $P_X \otimes P_Y$) is the distribution of class 1 (resp. 2).

3.3. Graph2Vec

Those two ideas are combined by [2] to produce its graph embedding. The joint probability $P_{XY} = P_{X|Y}P_Y$ is constructed by sampling a graph g from the dataset, and then by sampling a label x from the sequence generated by **WL** from this graph. Minimizing the cross entropy with respect to θ leads to the following expression for the loss:

$$\mathcal{L} = \mathbb{E}_{(x,g)\sim P_{XY}} \log \sigma(\theta_g \cdot \theta_x) + \mathbb{E}_{(x,g)\sim P_X \otimes P_Y} \log \sigma(-\theta_g \cdot \theta_x) \tag{4}$$

The discriminator function $T_\theta(x,y) = \theta_y \cdot \theta_x$ is taken as a dot product between a graph embedding $\theta_g \in \mathbb{R}^d$ and a **WL** label embedding $\theta_x \in \mathbb{R}^d$, which are vector of parameters randomly initialized and optimized with SGD. There is one such vector for each graph g and each label x produced by **WL** . The resulting graph embedding is $\mathcal{E}(g) = \theta_g$ while θ_x can be discarded. Optimizing this objective ensures to maximize the mutual information between the **WL** labels and the graph embeddings, which is a way to compress information about the distribution of **WL** labels into the embedding.

4. Contribution: Hierarchical Graph2Vec (HG2V)

Our motivations for this work can be summarized as follows:

- we first show that **WL** fails to capture global scale information, which is hurtful for many tasks;
- we then show that such flaw can be corrected by the use of graph coarsening. In particular, Loukas's coarsening exhibits good properties in this regard;
- we finally show that the advantage of GNN over **WL** is to be continuous functions in node features. They are robust to small perturbations.

Based on those observations, we propose a new algorithm building on graph coarsening and mutual information maximization, which we term Hierarchical Graph2Vec (HG2V). It has the following properties:

- The training is unsupervised. No label is required. The representation can be used for different tasks.
- The model is inductive, trained once for all with the graphs of the dataset in linear time. The training dataset is used as a prior to embed new graphs, whatever their underlying distribution.
- It handles continuous nodes attributes by replacing the hash function in **WL** procedure by a Convolutional Graph NN. It can be combined with other learning layers, serving as pre-processing step for feature extraction.
- The model is end-to-end differentiable. Its input and its output can be connected to other deep neural networks to be used as building block in a full pipeline. The signal of the loss can be back-propagated through the model to train feature extractors, or to retrain the model in transfer

learning. For example, if the node features are rich and complex (images, audio), a CNN can be connected to the input to improve the quality of representation.

- The structures of the graph at all scales are summarized using Loukas coarsening. The embedding combines **local view and global view** of the graph.

The resulting algorithm shares a similar spirit with Graph2Vec (MI maximization between node and graph descriptors), but it corrects some of its above-mentioned flaws. A high level overview of the method is provided in Algorithm 1. Table 1 summarizes key properties of the method, against the other ones found in literature.

In the following, we introduce in Section 4.1 the Loukas's coarsening method of [3], and detail how we use it in Section 4.2. Then, Section 4.3 deals with the continuity property achieved by GNN, while Section 4.4 explains how to train our proposed model **HG2V**.

Algorithm 1: High-level version of the Hierarchical Graph2Vec (HG2V) algorithm

Result: Graph embedding $\mathcal{E}(g)$ for each graph g
Input: A training set of attributed graphs g, subset of $\mathbb{G}$ and the number of stages L ; GNNs F_θ^l and H_θ^l with randomly initialized θ, $1 \leq l \leq L$
foreach *batch of attributed graphs* **do**
  foreach *graph g in the batch* **do**
    See Section 4.1: run Loukas's algorithm on g to produce a sequence of coarsened graphs g^l, $1 \leq l \leq L$;
  end
  foreach *level* $1 \leq l \leq L$ **do**
    foreach *graph g^l in the batch* **do**
      foreach *node u in g^l* **do**
        See Section 4.2: Generate local neighborhood embedding $x^l(u)$ using H_θ^l;
        Let $\mathcal{P}(u) \in g^{l+1}$ the image of u after pooling a node of the coarsened graph;
        Generate node embedding $g^{l+1}(\mathcal{P}(u))$ using F_θ^l;
        Create positive example $(x^l(u), g^{l+1}(\mathcal{P}(u))$;
      end
    end
    foreach *pair of graphs* (g, g') **do**
      foreach *pair of nodes* $(u, v) \in (g, \tilde{g})$ **do**
        Create negative examples $(x^l(u), \tilde{g}^{l+1}(\mathcal{P}(v))$;
      end
    end
    Minimize the cross entropy in Equation (3) between positive and negative examples with discriminator $T(x, y) = x \cdot y$ and using Section 4.4.
  end
end

Table 1. Key properties of methods (related or proposed) for graph embedding. N is the number of graphs. Symbol ✗ for complexity (inference) means the method is transductive (and not inductive) and one needs to use the same time as for training. Symbol ✓ for supervised means labels are required to learn a representation (by back-propagating classification loss).

Method	Continuous Attributes	Complexity (Training)	Complexity (Inference)	End-to-End Differentiable	Supervised
Kernel methods, e.g., **WL**-OA [27], **WWL** [4]	✓	$\mathcal{O}(N^2)$*	$\mathcal{O}(N)$*	✗	✗
Graph2Vec [2]	✗	$\mathcal{O}(N)$	✗	✗	✗
GIN [32], DiffPool [17], MinCutPool [18]	✓	$\mathcal{O}(N)$	$\mathcal{O}(1)$	✓	✓
HG2V (Section 4), Infograph [13]	✓	$\mathcal{O}(N)$	$\mathcal{O}(1)$	✓	✗

* Can be improved with Nystrom approximation or Random Fourier Features.

4.1. Loukas's Coarsening

In this section we detail the main drawback of **WL** procedure, and the benefit of graph coarsening to overcome this issue. For simplification, we will put aside the node attributes for a moment, and only focus on the graph structure. Even in this simplified setting, **WL** appears to be sensitive to structural noise.

4.1.1. Wesfeiler-Lehman Sensibility to Structural Noise

The ability of **WL** to discriminate all graph patterns comes with the incapacity to recognize as similar a graph and its noisy counterpart. Each edge added or removed can strongly perturb the histogram of labels produced by **WL**. Said otherwise, **WL** is not a good solution to inexact graph matching problem.

We perform experiments to evaluate the effect of adding or removing edges on different graphs. We randomly generate 100 graphs of 500 nodes each, that belong to four categories (cycle, tree, wheel and ladder), using the routines of NetworkX [33] library. For each generated graph g, we randomly remove from 1 to 10 edges, sampled with independent and uniform probability, to create the graph g'. One may hope that such little modification over this huge edge set would not perturb excessively the labels of **WL** procedure.

To evaluate the consequences of edge removal we propose to use as similarity score the intersection over union of histogram of labels of g and g' at each stage $1 \leq l \leq 5$:

$$\mathcal{S}^l(g, g') = 100 \times \frac{|\text{histo}(\mathbf{WL}^l(g)) \cap \text{histo}(\mathbf{WL}^l(g'))|}{|\text{histo}(\mathbf{WL}^l(g)) \cup \text{histo}(\mathbf{WL}^l(g'))|} \tag{5}$$

The average similarity score $\tilde{\mathcal{S}}^l(g, g')$ over the 100 graphs is reported in Figure 2.

The similarity decreases monotonically with the number of edges removed, even when restricting the procedure to the first stage (neighborhood of width 1). On higher stages (wider neighborhood) the effect is even worse. On graphs with small diameter (such as *wheel* graph or 3-regular tree) a significant drop in similarity can be noticed. On ladder graph and cycle, sparse graphs with huge diameter, the effect of edge removal remains significant.

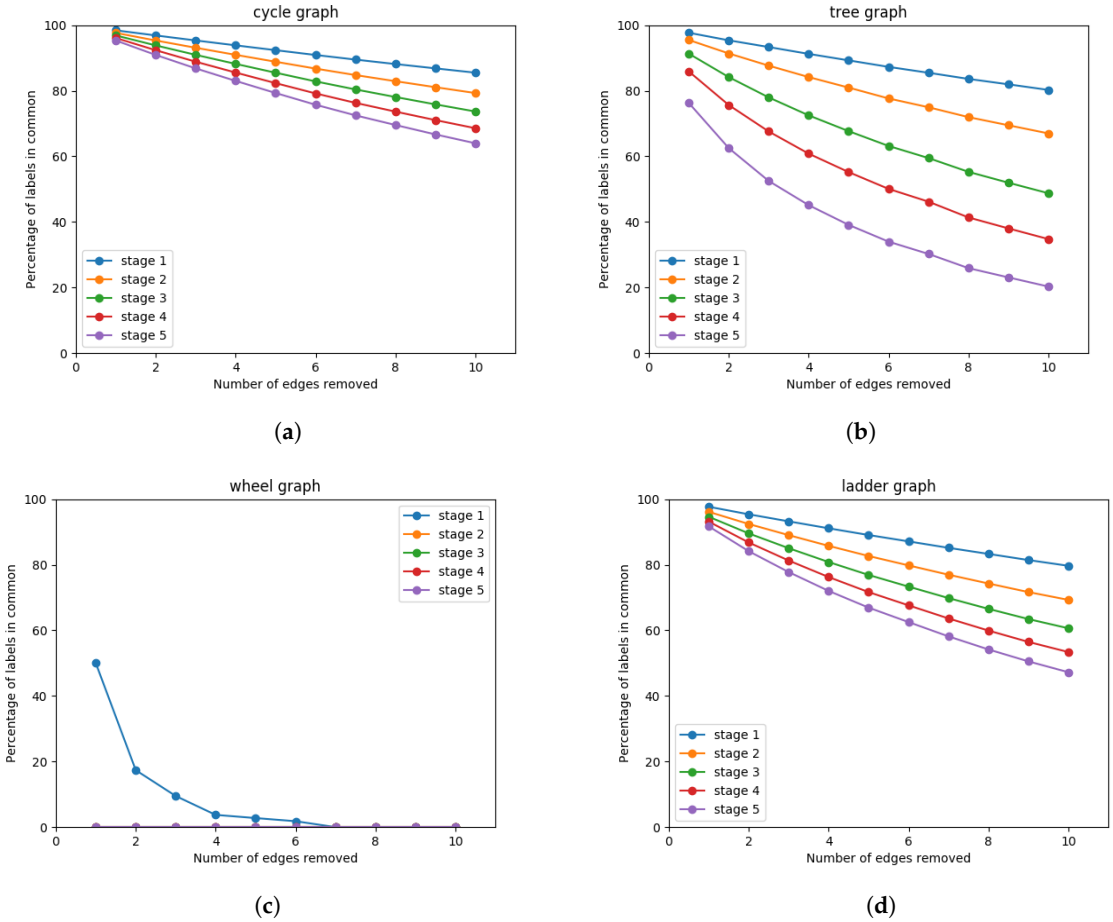

Figure 2. Similarity score as a function of edges removed for different stages of **WL** iterations. The similarity score reaches 100% for identical sets of labels, and 0% for disjoint sets of labels. (**a**) Cycle: the 2-regular graph with one giant connected component. (**b**) Tree: three children per node, except the ones at the last level. (**c**) Wheel: like cycle graph, with an additional node connected to all the others. (**d**) Ladder: two paths of 250 nodes each, where each pair of nodes is joined by an edge.

4.1.2. Robustness to Structural Noise with Loukas's Coarsening

WL procedure characterizes a graph as a sequence of rooted subtrees with increasing width. While this description is suitable for small patterns and inexact graph matching at local scale, it is very sensitive to structural noise. Adding or removing few edges (without hurting the global shape) changes the labels completely for subtrees of higher width. So as to characterize global features (e.g., communities, bridges, sparsest cuts...), we rely on a sequence of coarsened graphs based on the Loukas's procedure, that replaces the different iterations of neighborhoods of **WL**.

Definition 3 (Graph Coarsening)**.** *Graph coarsening is an operation mapping a graph* (V_1, E_1) *to a new graph* (V_2, E_2) *verifying* $|V_2| < |V_1|$*, using a surjective graph homomorphism* $\mathcal{P} : V_1 \to V_2$ *called* ***pooling function****:*

$$\begin{aligned} (u,v) \in E_1 &\implies (\mathcal{P}(u), \mathcal{P}(v)) \in E_2 \text{ or } \mathcal{P}(u) = \mathcal{P}(v) \\ (\mathcal{P}(u), \mathcal{P}(v)) \notin E_2 &\implies (u,v) \notin E_1 \end{aligned} \tag{6}$$

Loukas's method [3] is a graph coarsening operation. This spectral reduction of attributed graphs offers the guarantee, in the spectral domain, that a coarsened graph approximates well its larger version. The idea is that the graph spectrum (eigenspaces and eigenvalues of their Laplacian) describes global and local structures of graphs. Structural noise may have very little consequences on the spectrum, depending of the nature of the intervention. Hence Loukas's coarsening, which preserves

components associated with a low frequency in the graph spectrum, will produce a smaller graph with the same global shape as the input one, as demonstrated in Figure 3.

In a nutshell, the method computes a sequence of projection matrices $(P_l)_{1 \leq l \leq L}$ such that the Laplacian of the coarsened graph can be written as $L_{l+1} = P_l^{\mp} L_l P_l$ (where $\mp$ denotes the transposed pseudo-inverse). Since the algorithm is quite evolved and its explanation beyond the scope of our work, we refer to the original paper [3] for an explicit description of Loukas's method and an extensive list of properties and guarantees.

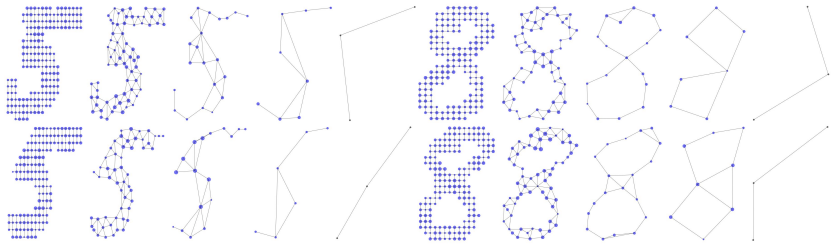

Figure 3. Coarsening of four graphs built from MNIST digits using Loukas's algorithm. The bigger the node, the wider the neighborhood pooled. Similar digits share similar shapes.

The interest of this coarsening method is that, if two graphs g^{l-1} and h^{l-1} are close enough, the coarsened graphs g^l is itself a satisfying coarsening for h^{l-1}. By symmetry, the same result follows for h^l and g^{l-1}. Hence, one may hope that g^l and h^l share similar patterns (Figure 4), and it will be advantageous for the **WL** procedure.

Such intuition can be confirmed experimentally. On four datasets (see Section 5 for their description), we sample two graphs g^0 and h^0 with independent and uniform probability.

We measure their similarity using the Wasserstein distance between their spectra. Such choice is motivated by the fact that two graphs with different number of nodes have spectrum of different sizes. Wasserstein provides an elegant way to measure distance between sets of different size, in addition to be fast and easy to compute.

Definition 4 (Wasserstein Distance between Graph Spectra)**.** *Let* $\boldsymbol{\lambda} = \{\lambda_1, \lambda_2, \ldots, \lambda_n\}$ *and* $\boldsymbol{\mu} = \{\mu_1, \mu_2, \ldots, \mu_m\}$ *the spectra of two graphs. The Wasserstein distance is defined as:*

$$d_{\mathcal{W}}(\boldsymbol{\lambda}, \boldsymbol{\mu}) = \min_{\pi \in \Pi} \sum_{i,j} \pi(i,j) |\lambda_i - \mu_j| \tag{7}$$

where Π *is the collection of measures over* $\boldsymbol{\lambda} \times \boldsymbol{\mu}$ *whose marginals are discrete uniform distributions over* $\boldsymbol{\lambda}$ *and* $\boldsymbol{\mu}$ *respectively.*

Since Loukas's coarsening preserves spectrum, we expect the distance between g^0 and h^0 to be correlated with the distance between their coarsened counterpart g^1 and h^1. Each dot in Figure 4 corresponds to an experiment, which are repeated 1000 times. Interestingly, this correlation strongly depends of the underlying dataset.

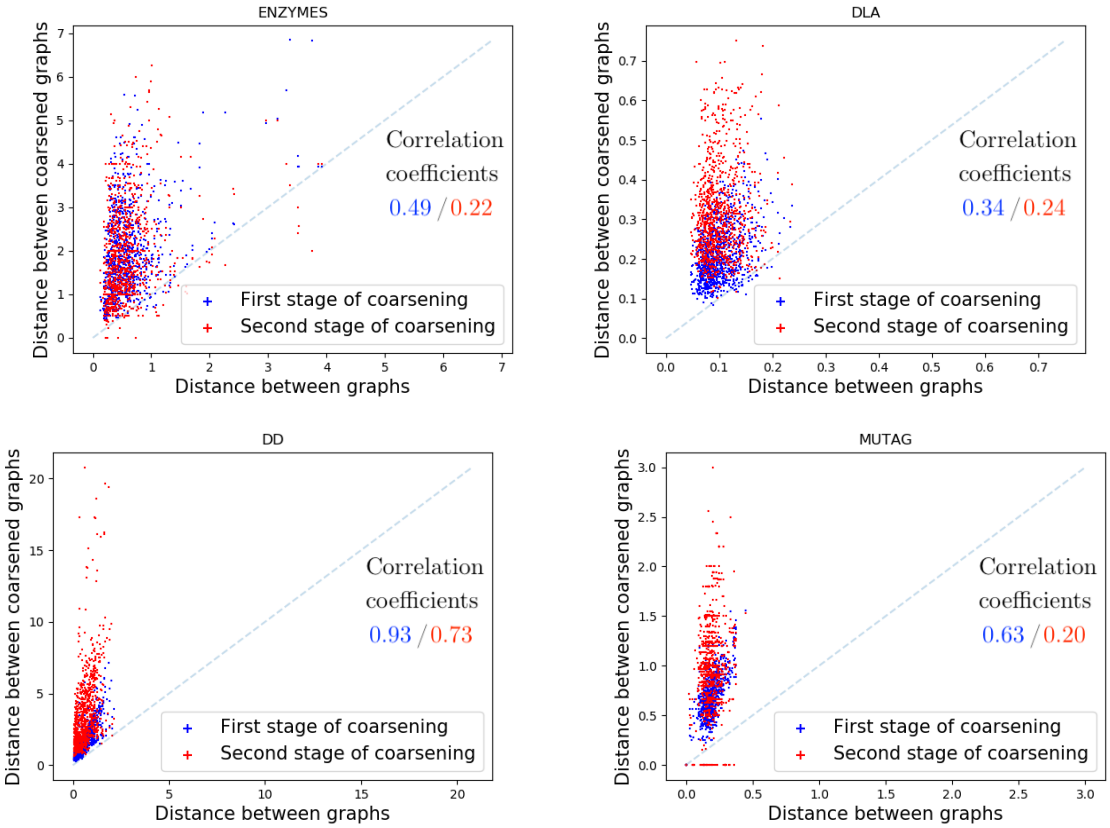

Figure 4. Wasserstein distance between spectra of graphs g^0 and h^0 sampled from different datasets (see Section 5 for their description) compared to the same distance between their coarsened graphs g^1 and h^1 and their two times coarsened graphs g^2 and h^2 . In blue (resp. red) are given the correlation coefficients of the distance between g^0 & h^0 and g^1 & h^1 (resp. g^2 & h^2). These coefficients have been computed after averaging 10 runs where we sampled 1000 graphs couples (g^0, h^0) for each dataset. As expected the more we coarsened graph the more the correlation coefficient decreases because coarsening always lost some structural informations.

4.2. Hierarchy of Neighborhoods

Taking advantage of the previous observation, we propose to build a hierarchy of coarsened graphs g_i^l using Loukas's method. It induces a hierarchy of nested neighborhoods $u, \mathcal{P}(u), \mathcal{P}(\mathcal{P}(u)), \ldots, \mathcal{P}^L(u)$ by pooling the nodes at each level.

We learn the node embedding $g^l(u)$ (of node u) at each level. This node embedding is used to produce a local neighborhood embedding $x^l(u)$ using function H_θ^l, and to produce the node embedding of the next level $g^{l+1}(\mathcal{P}(u))$ using function F_θ^l. Formally, the recurrent equations defining the successive embeddings are:

$$g^0(u) = Z(u) \tag{8}$$

$$x^l(u) = H_\theta^l(\{g^l(v) | v \in \mathcal{N}(u) \cup \{u\}\}) \tag{9}$$

$$g^{l+1}(\mathcal{P}(u)) = F_\theta^l(\{g^l(v) | v \in \mathcal{N}(u) \cup \{u\}\}) \tag{10}$$

The procedure is illustrated in Figure 5. In practice, functions H_θ^l and F_θ^l are graph neural networks parametrized by θ, and whose exact form will be specified in next Section.

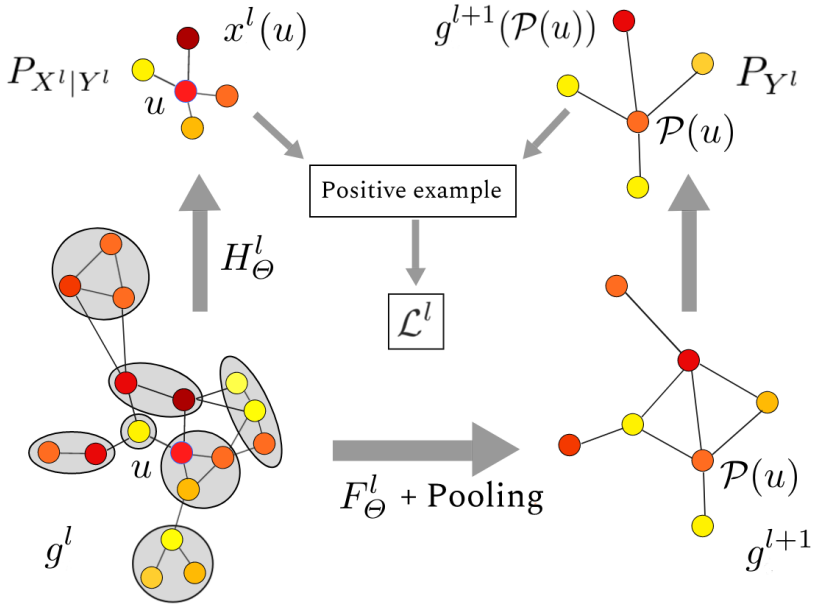

Figure 5. Single level of the pyramid. Local information $x^l(u)$ centered around node u is extracted from graph g^l. The graph is coarsened to form a new graph g^{l+1}. There, $g^{l+1}(\mathcal{P}(u))$ captures information at a larger scale, centered on node $\mathcal{P}(u)$. The pair $(x^l(u), g^{l+1}(\mathcal{P}(u)))$ is used as positive example in the negative sampling algorithm, and it helps to maximize mutual information between global and local view.

4.3. Handling Continuous Node Attributes with Truncated Krylov

The **WL** algorithm uses a discrete hash function in Equation (2), with the issue that nodes sharing similar but not quite identical neighborhoods are considered different. If the differences are caused by noise in computations or measures, they should not result in much differences in the labels. For that, we relax the injectivity property of **WL** by replacing it by a function with a continuity property.

It is known that Graph Neural Networks (GNN) have a discriminative power at least equivalent to **WL** [32,34,35]. We require the opposite, and we emphasize the importance of not having a too strong discriminator. We use the extension of the Gromov–Wasserstein distance to attributed graphs, that requires the mapping to preserve both edge weights and node attributes. The resulting distance is a special case of the Fused Gromov–Wasserstein distance [36].

Definition 5 (Fused Gromov–Wasserstein distance.)**.** *Let $g_1 = (V_1, A_1, Z_1)$ and $g_2 = (V_2, A_2, Z_2)$ be two attributed graphs in $\mathbb{G}$.*

$$d_{\mathbb{G}}(g_1, g_2) = \min_{\pi \in \Pi} \sum_{u,u',v,v'} \pi(u,v)\pi(u',v')\big(|A_1(u,u') - A_2(v,v')| + |Z_1(u) - Z_2(v)| + |Z_1(u') - Z_2(v')|\big) \quad (11)$$

where Π is the collection of measures over $V_1 \times V_2$ whose marginals are discrete uniform distributions over V_1 and V_2.

This definition allows us to state the continuity of GNN under this metric.

Lemma 1 (GNN continuity.)**.** *GNNs with continuous activation function are continuous on the topological space induced by the metric $d_{\mathbb{G}}$.*

Proof in Appendix A.

GNNs are usually parameterized functions learnable with stochastic gradient descent, and hence fulfill this continuity constraint. Moreover, some attributes may be irrelevant, and should be discarded,

or sometimes the graph is only characterized by some function (for example a linear combination) of the attributes. Such a function can be inferred directly from the dataset itself, simply by looking by the co-occurrences of some similar sub-graphs. Hence, the ability of GNN to learn features relevant for some specific task is an other advantage over **WL**.

GCN [21] is as a baseline among this family of networks (see Section 2 for a brief survey). Unfortunately, [23] have shown that they behave badly when stacked. They propose Truncated Krylov GNN in replacement of GCN, correcting its flaws, with better theoretical guarantees and empirical evidence of their superiority over GCN. We chose it, encouraged by the promising results of early experiments. It amounts to consider, for a given node, a receptive field extended to the neighborhood of nodes at most at given distance a. The resulting layer uses the normalized adjacency matrix $\tilde{A} = D^{-1/2}(A + I)D^{-1/2}$ (like in GCN) of the graph g^l:

$$\begin{aligned} g^{l+1} &= \text{Pool}(\tanh([g^l, \tilde{A}g^l, \tilde{A}^2g^l, \ldots, \tilde{A}^a g^l]\theta_1^l))\theta_2^l + \theta_3^l \\ x^l &= \tanh([g^l, \tilde{A}g^l, \tilde{A}^2g^l, \ldots, \tilde{A}^a g^l]\theta_4^l). \end{aligned} \tag{12}$$

The vector θ is made of the trainable parameters, and $[\ldots]$ denotes the concatenation operation. We abused the notations, using x and g as real-valued vectors indexed by node set V. In Equation (12), one implements a specific graph filter of order a, close to the ones considered by Chebyshev polynomial approximation in [20].

Pool denotes the merging of nodes. Their features are summed because it preserves information on multiset, as argued in [32].

Note that F_θ^l and H_θ^l span the same neighborhood (up to distance a). However, the usage of coarsening constraints F_θ^l to summarize the information content in the group of pooled nodes. It is this last point that makes g^{l+1} a global descriptor. The affine transformation $X \mapsto X\theta_2^l + \theta_3^l$ in (12) is used to improve the expressiveness of the model thanks to a bias term. Indeed, g^l must both summarize the current level l an be used as input for the next one, while x^l only aims to summarize the neighborhood.

4.4. Hierarchical Negative Sampling

Following the principles of Graph2Vec we aim to maximize mutual information between node embeddings and local neighborhood embeddings at each level. We achieve this by training a classifier to distinguish between positive and negative examples of such pairs of embeddings. Positive pairs are defined as node v in level l and its "super node" $\mathcal{P}(v)$ in level $l+1$.

More precisely, consider the l-th level of the hierarchical pyramid. The probability P_{Y^l} is built by sampling a graph g from the dataset, then by sampling a node u with uniform probability from this graph. $P_{X^l|Y^l}$ is obtained by sampling a node from $\mathcal{P}^{-1}(u)$. It gives a pair $(x^l(\mathcal{P}^{-1}(u)), g^{l+1}(u))$ sampled according to $P_{X^lY^l} = P_{Y^l}P_{X^l|Y^l}$. The negative pairs are built as usual from the independent probability $P_{X^l} \otimes P_{Y^l}$. The corresponding loss function takes the form:

$$\mathcal{L}^l = \mathbb{E}_{(x_j^l(u), g^l(\mathcal{P}(u)))\sim P_{X^lY^l}} \log\sigma(x^l(u)\cdot g^l(\mathcal{P}(u))) + \mathbb{E}_{(x^l(u), g^l(v))\sim P_{X^l}\otimes P_{Y^l}} \log\sigma(-x^l(u)\cdot g^l(v)) \tag{13}$$

The overall method is described in Algorithm 1.

A descriptor $\mathcal{E}_l(g)$ for each level l of the graph g is computed by global pooling over node embeddings. The final graph embedding is obtained by concatenating those descriptors. The sum aggregator is always preferred over mean aggregator, because of its better ability to distinguish between multisets instead of distributions, as argued in [32]. Hence, the learned representation is:

$$\mathcal{E}_l(g) = \left[\sum_{u\in V} g^l(u), \max_{u\in V} g^l(u)\right] \tag{14}$$

$$\mathcal{E}(g) = \text{CONCATENATE}\,(\mathcal{E}_1(g), \mathcal{E}_2(g), ..., \mathcal{E}_L(g)) \tag{15}$$

In the Loukas's method, the number of stages is not pre-determined in advance: only the final coarsened graph size can be chosen. When the number of stages produced by the Loukas method is not aligned with the depth of the neural network, the pooling is either cropped (too much stages) or completed with identity poolings (not enough stages). The resulting vector can be used for visualization purposes (PCA, t-SNE) or directly plug into a classifier with labels on graphs.

Complexity

Time. The coarsening of each graph can be pre-computed once for all, in linear time in the number of nodes and edges [3]. Hence, the main bottleneck is the maximization of mutual information. Let $|V|$ the maximum number of nodes in a graph. Let B be the batch size. Let L be the number of stages. Let a the order of Truncated Krylov. Let d the dimension of node embedding. The complexity of the algorithm per batch is $\mathcal{O}(BLa|V|^3 + BLa|V|^2d + BLa|V|d^2 + B^2Ld)$. The first term is due to exponentiation of adjacency matrix, the second one to diffusion along edges, the third one to forward pass in network layers, and the last one to the Cartesian product to create positive and negative examples. The average complexity per graph is hence: $\mathcal{O}(La|V|(|V| + d)^2 + BLd)$. The most sensitive factor is the number of nodes, followed by the the number of features and the batch size. The magnitude of those matrices allows to handle graphs with hundred of nodes efficiently on modern GPUs, with embedding as big as 512 and up to 8 graphs per batch and 5 stages of coarsening. In practice, the bottleneck turn out to be the pre-computation of coarsening, which does not benefit the GPU speed.

Space. Note that for datasets with small graphs we have $\dim(\mathcal{E}_i) = 2Ld > \dim(X_i) + \dim(A_i) = n|V| + |V|^2$. However, when the number of nodes exceeds 50, the embedding size is always smaller than the adjacency matrix. Hence, this method is more suitable for big graphs with hundreds of nodes.

5. Evaluation

The code for the proposed method and to reproduce the experiments can be found on: https://github.com/Algue-Rythme/GAT-Skip-Gram.

An experimental evaluation is conducted on various attributed graphs datasets, and the quality of the embeddings is assessed on supervised classification tasks in Section 5.2. Our method is inductive: the model can be trained over a dataset and be used to embed graphs coming from another dataset. This property is analysed in Section 5.3. HG2V differs from Graph2Vec by the usage of GNN and Loukas coarsening. The influence of those two elements is analysed with ablatives studies in Section 5.4.

5.1. Datasets

Standard datasets.

We use standard datasets from literature: PROTEINS, ENZYMES, D&D, NCI1, NCI109, MUTAG, IMDB (binary and multi) and PTC_FR downloaded from [37]. We also use the challenging REDDIT (binary and 5K versions).

Synthetic datasets.

Additionally, we introduce a novel dataset for the community: Diffusion Limited Aggregation (DLA). The attributed graphs are created by a random process that makes the graphs scale free, DLA being known to be fractal objects. The graphs have then an interesting property that justifies the creation of a new benchmark. In addition it provides natural features for the nodes: the coordinates in space (and this is why we prefer it to the well known and usual scale-free network model that is the Barabasi-Albert model). We refer to Appendix C.1 for more details. The code can be found on https://github.com/Algue-Rythme/DiffusionLimitedAgregation

Image datasets. We convert MNIST and USPS popular datasets of the computer vision community into graphs, by removing blank pixels, and adding luminosity and (x, y) coordinates as node features. To solve the task, the method should be able to recognize the shape of the graph, which is a global

property. Due to the size of these datasets, we restrain ourselves to a subset of $10,000$ images randomly sampled (instead of the $70,000$ available).

Frankenstein dataset was created in [38] by replacing nodes labels of BURSI dataset with MNIST images.

Pre-processing

All the continuous node attributes are centered and normalized. The discrete node labels use one hot encoding. When there is no node feature, we use the degree of the node instead.

5.2. Supervised Classification

In the first task, the model is trained over all the attributed graphs in the dataset. The quality of these embeddings is assessed on a supervised classification task. The classifier used is C-SVM with RBF kernel from scikit-learn library.

5.2.1. Training Procedure

The embeddings are trained over 10 epochs. At each step, eight graphs are picked randomly from the dataset, and all the vocabulary from these graphs is used to produce positive and negative pairs (by Cartesian product). Hence, the number of negative examples in each batch is higher than the number of positive examples. Consequently we normalize the loss of negative samples to reduce the unbalance. The optimizer is Adam [39] and the learning rate follows a geometric decay, being divided by approximately 1000 over the 10 epochs.

5.2.2. Model Selection

The relevant hyper-parameters of HG2V are the number of features $d \in \{16, 128, 256\}$ at each stage, the receptive field of Truncated Krylov $a \in \{2, 4\}$, and the maximum depth of Loukas's coarsening $L \in \{3, 5\}$. They are selected using a grid search. Five random split of the dataset are generated: *TrainVal* (80% of the data) and *Test* (20% of the data). HG2V is trained over *TrainVal*. Then, a C-SVM is trained over *TrainVal*, using 5-cross validation grid search for the selection of its hyper-parameters (C, Gamma). The average validation score of the best C-SVM classifier is used to select the hyper-parameters (d^*, a^*, L^*) of HG2V. The average test score of the best C-SVM classifier (accuracy over *Test* split) is reported in Table 2. Note that HG2V could be trained on the test set *without* using labels, due to its unsupervised nature. We decided not doing it, to ensure fair comparison with the other methods.

Table 2. Accuracy on classification tasks. HG2V is trained over both *TrainVal+Test* splits, without using labels due to its unsupervised nature. Model selection of C-SVM and hyper-parameters of HG2V have been done with 5-cross validation over *TrainVal* split. We report on the accuracy over the *Test* split, averaged over 10 runs, and with standard deviation. Unavailable result marked as ✗.

DATASET	#graphs	#nodes	HG2V (Ours)	Graph2Vec	Infograph	DiffPool (Supervised)	GIN (Supervised)	MinCutPool (Supervised)	WL-OA (Kernel)	WWL (Kernel)
IMDB-m	1500	13	47.9 ± 1.0	**50.4 ± 0.9**	49.6 ± 0.5	45.6 ± 3.4	48.5 ± 3.3	✗	✗	✗
PTC_FR	351	15	**67.5 ± 0.5**	60.2 ± 6.9	✗	✗	✗	✗	63.6 ± 1.5	✗
FRANK.	4337	17	**65.3 ± 0.7**	60.4 ± 1.3	✗	✗	✗	✗	✗	✗
MUTAG	188	18	81.8 ± 1.8	83.1 ± 9.2	**89.0 ± 1.1**	✗	✗	✗	84.5 ± 1.7	87.3 ± 1.5
IMDB-b	1000	20	71.3 ± 0.8	63.1 ± 0.1	73.0 ± 0.9	68.4 ± 3.3	71.2 ± 3.9	✗	✗	**74.4 ± 0.8**
NCI1	4110	30	76.3 ± 0.8	73.2 ± 1.8	✗	76.9 ± 1.9	80.0 ± 1.4	✗	**86.1 ± 0.2**	85.8 ± 0.2
NCI109	4127	30	75.6 ± 0.7	74.3 ± 1.5	✗	✗	✗	✗	**86.3 ± 0.2**	✗
ENZYMES	600	33	66.0 ± 2.5	51.8 ± 1.8	✗	59.5 ± 5.6	59.6 ± 4.5	✗	59.9 ± 1.1	**73.3 ± 0.9**
PROTEINS	1113	39	75.7 ± 0.7	73.3 ± 2.0	✗	73.7 ± 3.5	73.3 ± 4.0	76.5 ± 2.6	76.4 ± 0.4	**77.9 ± 0.8**
MNIST	10000	151	**96.1 ± 0.2**	56.3 ± 0.7	✗	✗	✗	✗	✗	✗
D&D	1178	284	79.2 ± 0.8	58.6 ± 0.1	✗	75.0 ± 3.5	75.3 ± 2.9	**80.8 ± 2.3**	79.2 ± 0.4	79.7 ± 0.5
REDDIT-b	2000	430	91.2 ± 0.6	75.7 ± 1.0	82.5 ± 1.4	87.8 ± 2.5	89.9 ± 1.9	**91.4 ± 1.5**	89.3	✗
DLA	1000	501	**99.9 ± 0.1**	77.2 ± 2.5	✗	✗	✗	✗	✗	✗
REDDIT-5K	4999	509	55.5 ± 0.7	47.9 ± 0.3	53.5 ± 1.0	53.8 ± 1.4	**56.1 ± 1.7**	✗	✗	✗

5.2.3. Baselines

We compare our work to various baselines of the literature:

Kernel Methods: All the results reported are extracted from the corresponding papers [27,40], giving an idea of the best possible performance achievable by **WL**-Optimal Assignment [27] and the Wasserstein Wesfeiler–Lehman [4] graph kernels. It almost always outperform inductive methods based on neural networks. However, like many kernel-based method, they have quadratic time complexity in the number of graphs, which is prohibitive for dealing with large datasets.

DiffPool, GIN: We report the results of the rigorous benchmarks of [41], including the popular DiffPool [17] and GIN [32]. Those algorithms are end-to-end differentiable, but they are **supervised**. DiffPool also relies on graph coarsening, but their pooling function is learned, while Loukas coarsening is task agnostic.

MinCutPool: We report the results of the original paper [18]. Note that they used not only node degree, but also clustering coefficient as node features. Consequently they benefit from additional information compared to our setting.

Infograph We also report the results of Infograph [13]. It is the closest method to our work: it is unsupervised, end-to-end differentiable, and also relies on mutual information maximization, but it does not benefit of coarsening. Infograph is currently the state of the art in unsupervised graph representation learning.

5.2.4. Results

We note that get substantial improvements over Graph2Vec baseline for many datasets, more specifically when the graph are big and carry high dimensional features.

For FRANKENSTEIN, if we connect a randomly initialized two-layer CNN to the input of the model for better feature extraction, the results are improved and reach $66.5 \pm 0.4\%$ which is a noticeable improvement. Thanks to the end-to-end differentiability of the model, the CNN is trained with backpropagation, benefiting from the unsupervised loss signal.

On the notably difficult REDDIT-B and REDDIT-5K we reach high results, comparable to SOTA. The coarsening operation is beneficial to these datasets, considering the size of the graphs. On datasets with smaller graphs, the results are less significant.

5.2.5. Computation Time

Training on only 1 epoch already provides a strong baseline for molecule datasets, and lasts less than 1 min, using GTX 1080 GPU. The most challenging dataset was REDDIT-MULTI-5K, trained on V100 GPU, with 5000 graphs, an average of 508 nodes and 595 edges per graph. The pre-computation of Loukas's coarsening required 40 min (that can be improved with parallelization, not implemented). After this step, the runtime never exceed 190 s per epoch, for a total training time of 70 min.

5.3. Inductive Learning

The method is inductive: it allows us to train HG2V on a dataset and test on another dataset. The training set is used to extract relevant features, that are expected to be seen again during inference. In Section 5.2 we showed that when the domain of training set and inference set are the same (different splits of the same dataset) it provides good results. In this section, we prove that even when the training set and the inference set are disjoint, the model is still able to produce good representations.

We emphasize that this property is specific to unsupervised inductive methods. Hence, it is not possible to perform such experiment on Graph2Vec (transductive), or on models trained with supervised loss (e.g., GIN, DiffPool). To the best of our knowledge, no such domain adaption experiments have been done in graph classification tasks. Consequently, we miss comparisons to other methods. A more in-depth analysis of this property is left as future work. Our preliminary results with this regard are summarized in Table 3, and we hope that it will encourage community to perform similar experiments.

Table 3. Accuracy on classification tasks by training on some input distribution and performing inference on an other. The hyper-parameters selected are identical to the ones of Table 2.

Training Set	Inference Set	Accuracy (Inference)	Delta with baseline (see Table 2)
MNIST	USPS	94.86	✗
USPS	MNIST	93.68	−2.40
REDDIT-b	REDDIT-5K	55.00	−0.48
REDDIT-5K	REDDIT-b	91.00	−0.15
REDDIT-b	IMDB-b	69.00	−2.25
REDDIT-5K	IMDB-b	69.50	−1.75
MNIST	FASHION MNIST	83.35	✗

The best hyper-parameters found in the previous section are kept as is without further hyper-parameter tuning. The goal was not to reach the best possible result, but ensures that we can reuse trained weights to produce quickly good embeddings.

USPS is a dataset of handwritten digits similar to MNIST, hence we expect that a model trained on it can also embed MNIST graphs.

FASHION MNIST introduced in [42] is built similarly to MNIST: a set of 28×28 grayscale images. We only work on a subset of $10,000$ training examples.

Results

We see that the model can easily transfer from one domain to another. MNIST seems to be a better prior than USPS, which is a behavior previously observed in transfer learning. On the other datasets, the accuracy drops compared to the baseline: it is expected, since the new graphs are out of

distribution of the training set. Nevertheless, even in this unfavorable setting, the accuracy remains within a comparable range to the baseline.

We conclude that the inductivity property is not only theoretical, but also a property that can be checked in practice, without specific hyper-parameter tuning or costly manual adaptation.

5.4. Ablative Studies

We perform ablative studies by removing separately Loukas coarsening and GNN. If we remove both, we fall back to Graph2Vec. The dimension of the embeddings is chosen equal to 1024.

Graph2Vec+GNN

We remove Loukas coarsening. The only difference with Graph2Vec is the replacement of **WL** iterations with forward pass through a GNN. All the attributes available are used.

Graph2Vec+Loukas

We remove GNN. The resulting algorithm is Graph2Vec applied on the sequence of coarsened graphs. On the coarsened graphs, new labels are generated by concatenating and hashing the labels of the nodes pooled (like a **WL** iteration would do). The sequence of (unconnected) graphs is fed into Graph2Vec. Continuous attributes are ignored because **WL** can not handle them.

The results are summarized in Table 4.

Table 4. Ablative studies. The accuracy on test set is in the column "Accuracy". The column "Delta" corresponds to the difference in average accuracy with Graph2Vec. **OOM**—Out of Memory Error.

		HG2V		**Graph2Vec + GNN**		**Graph2Vec + Loukas**		**Graph2Vec**
DATASET	**#nodes**	**Accuracy**	**Delta**	**Accuracy**	**Delta**	**Accuracy**	**Delta**	
IMDB-b	20	70.85	+7.75	70.70	+7.60	57.5	−5.60	63.10
NCI1	30	77.97	+4.75	75.40	+2.18	65.45	−7.77	73.22
MNIST	151	95.83	+39.56	91.05	+34.78	72.5	+16.23	56.27
D&D	284	78.01	+19.37	79.26	+13.16	66.10	+7.45	58.64
REDDIT-B	430	91.95	+16.23	**OOM**	✗	82.50	+6.78	75.72

Results

On datasets with small graphs (less than 30 nodes in average) the use of coarsening is hurtful, resulting in loss compared to Graph2Vec. As soon the graphs get big, coarsening leads to huge improvements in accuracy. We also notice that the usage of GNN and its ability to handle continuous attributes, and to be trained to extract co-occurring features, leads to significant improvements on all datasets.

5.5. Latent Space

Figure 6 illustrates some graphs (leftmost column) with their six closest neighbors in the learned latent space (right columns 1 to 6), from the closest to the farthest, taken from MNIST and IMDB. We observe that isomorphic graphs share similar embeddings, and even when the graphs are not quite isomorphic they exhibit similar structures.

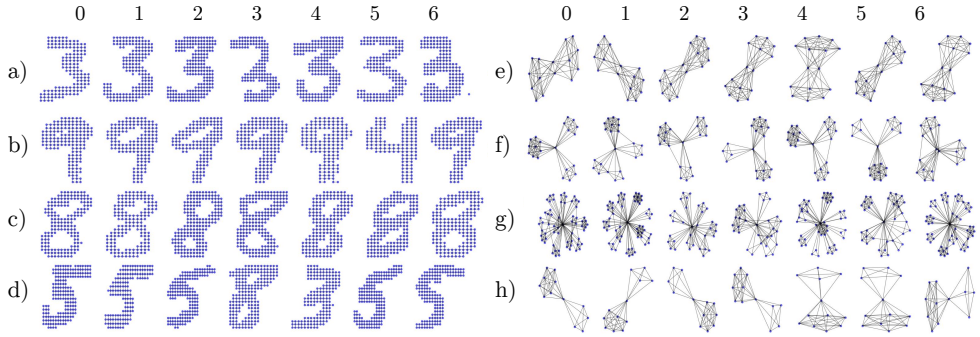

Figure 6. Six nearest neighbors of graphs in the learned latent space for four graphs from IMDB-b and MNIST. Column 0 corresponds to a randomly chosen graph, then the six nearest neighbors are drawn in increasing distance order from left to right (1 to 6).

6. Conclusions

We proposed a new method for Graph Representation Learning. It is fully unsupervised, and it learns to extract features by Mutual Information maximization. The usage of Loukas's coarsening allows us to tackle all scales simultaneously, thanks to its capacity to preserve the graph spectrum. The method is inductive, and can be trained on only a fraction of the data before being used in transfer learning settings. Despite being unsupervised, the method produces high quality embeddings leading to competitive results on classification tasks against supervised methods.

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

Funding: We thank ACADEMICS ("Machine Learning & Data Science for Complex and Dynamical Models") project for their funding, an Scientific Breakthrough Project given by the IDEXLYON project of the Université de Lyon, as part of the "Programme Investissements d'Avenir" ANR-16-IDEX-0005.

Acknowledgments: We thank the Centre Blaise Pascal (ENS de Lyon) for the computational resources, the GPUs, and their technical support, especially Emmanuel Quémener.

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

Appendix A. Proofs

Proof of Lemma 1.

Proof. We use Message Passing Neural Networks (MPNN) framework introduced in [43]. In order to demonstrate our theorem, we use the same notations as their paper. Let $g = (V, A, Z) \in \mathbb{G}$. Let $u \in V$ a node, and t the index of a layer. Then:

$$h^0(u) = Z(u) \tag{A1}$$

$$m^{t+1}(u) = \sum_{v \in \mathcal{N}(u)} M_t(h^t(u), h^t(v), e_{uv}) \tag{A2}$$

$$h^{t+1}(u) = U_t(h^t(u), m^{t+1}(u)) \tag{A3}$$

Every GNN layer fall under MPNN framework. When the activation function used is continuous, the functions M_t and U_t are also continuous, because they involve matrix products (which are continuous operations). Moreover, the sum over $\mathcal{N}(u)$ is finite because we consider finite graphs. Consequently, $h^{t+1}(u)$ depend continuously of $\{h^t(v), v \in \mathcal{N}(u)\}$ through $S_t = U_t \circ M_t$. By composition, the result holds for any number of layers T. Let F the resulting function on graph domain $\mathbb{G}$, i.e., if $(V', A', Z') = g' = F(g)$ then $A = A'$ and $V = V'$ (GNNs do not modify topology), and for every node $u \in V$ we have $Z'(u) = h^T(u)$.

Let $(g_n)_{n\in\mathbb{N}} \in \mathbb{G}^{\mathbb{N}}$ a sequence of attributed graphs such that $\lim_{n\to\infty} g_n = g$ in the topological space induced by the metric $d_{\mathbb{G}}$. Let $(\pi_n : V_n \times V)_{n\in\mathbb{N}}$ the corresponding set of transportation plans between g_n and g. Let $g'_n = (V'_n, A'_n, Z'_n) = F(g_n)$ and $g' = (V', A', Z') = F(g)$. Then:

$$\begin{aligned} d_{\mathbb{G}}(g'_n, g') &\leq \sum_{u,u',v,v'} \pi_n(u,v)\pi_n(u',v') \left(|A'_n(u,u') - A'(v,v')| + |Z'_n(u) - Z'(v)| + |Z'_n(u') - Z'(v')|\right) \\ &= \sum_{u,u',v,v'} \pi_n(u,v)\pi_n(u',v') \left(|A_n(u,u') - A(v,v')| + |Z'_n(u) - Z'(v)| + |Z'_n(u') - Z'(v')|\right) \end{aligned} \tag{A4}$$

Now, because $\lim_{n\to\infty} g_n = g$ we have necessarily:

$$\lim_{n\to\infty} \pi_n(u,v)\pi_n(u',v')|Z_n(u) - Z(u)| = 0 \tag{A5}$$

$$\lim_{n\to\infty} \pi_n(u,v)\pi_n(u',v')|A_n(u,u') - A(v,v')| = 0 \tag{A6}$$

The continuity of S_t allows to further conclude that:

$$\lim_{n\to\infty} \pi_n(u,v)\pi_n(u',v')|Z'_n(u) - Z'(u)| = 0 \tag{A7}$$

Finally, the right hand size of (A4) must have limit 0, hence:

$$\lim_{n\to\infty} g'_n = g' \tag{A8}$$

We just proved that $\lim_{n\to\infty} g_n = g$ implies $\lim_{n\to\infty} g'_n = g'$, which is precisely the definition of F being continuous w.r.t the topology induced by $d_{\mathbb{G}}$. □

Appendix B. Additional Visualizations of the Embeddings

We present other randomly sampled graphs and their six closest neighbors from MNIST-Graph, IMDB and PTC Datasets in, respectively, Figures A1–A3.

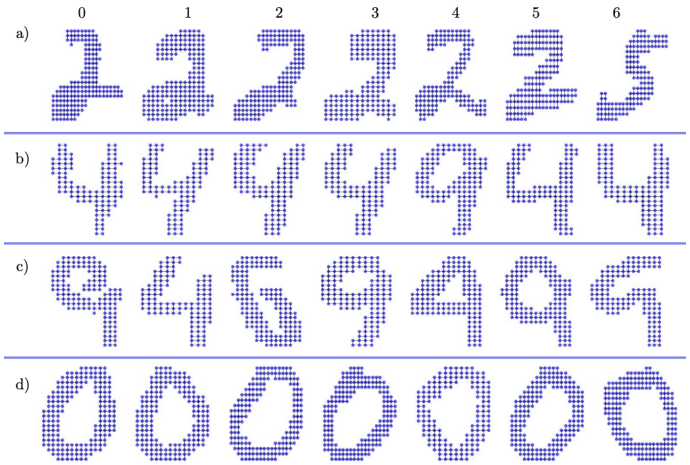

Figure A1. Six nearest neighbors of graphs in latent space from MNIST for four graphs. Column 0 correspond to the randomly chosen graph then the six nearest neighbors are draw in increasing distance order from left to right (from 1 to 6).

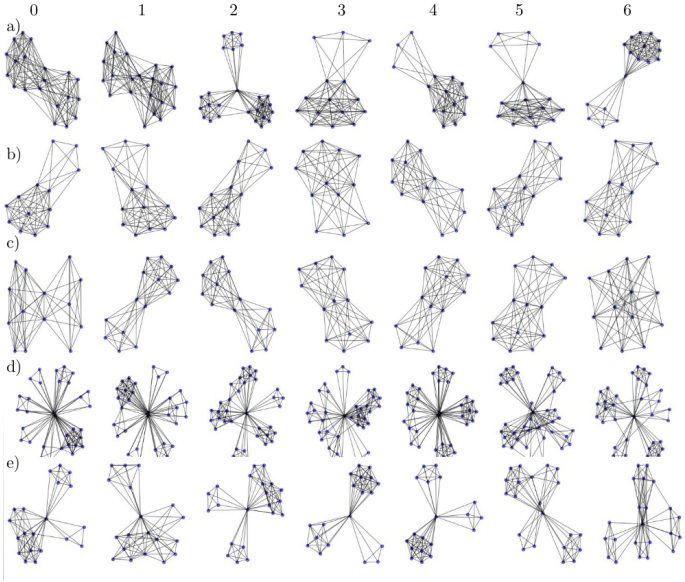

Figure A2. Six nearest neighbors of graphs in latent space from IMDB for ten graphs. Column 0 correspond to the randomly chosen graph then the six nearest neighbors are draw in increasing distance order from left to right (from 1 to 6).

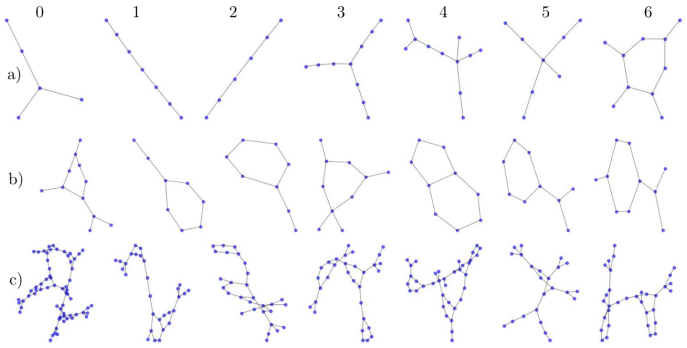

Figure A3. Six nearest neighbors of graphs in latent space from PTC for six graphs. Column 0 correspond to the randomly chosen graph then the six nearest neighbors are draw in increasing distance order from left to right (from 1 to 6).

Appendix C. Details about the Datasets

In this section we give additional details on some datasets we used.

Appendix C.1. DLA

The DLA dataset has been artificially generated with Diffusion Limited Aggregation [44], a random process that creates cluster of particles following a Brownian motion. Particles are added one by one, and when two particles touch, they can aggregate with some probability of stickiness p. The resulting structure is a tree, each particle being a node and each link corresponding to a bond. The resulting graphs have scale free properties [45]. The degree distribution of the nodes and their position in space will depend on p.

We generated a total of 1000 graphs with 500 nodes each. This dataset is splited into two classes, one with stickiness $p = 1$ and the other with stickiness $p = 0.05$. The attributes are the x and y coordinates of the particles following a 2D Brownian motion for simplicity.

It has been observed that Graph2Vec is unable to reach a good accuracy by relying solely on node degree distribution, while Hierarchical Graph2Vec is able to use the features and reach near perfect

accuracy. The code that generated this dataset can be found here: https://github.com/Algue-Rythme/DiffusionLimitedAgregation.

Appendix C.2. MNIST and USPS

We produce graphs from MNIST (resp. USPS) handwritten digits dataset . The graphs are created by removing all the pixel with luminosity equals to 0 (resp. less than 0.3), by mapping each remaining pixel to a node, and then by adding the x and y coordinate of each node to the vector of attributes. Due to the size of MNIST ($70,000$ images in total) we kept only the test split ($10,000$ images) to train the embeddings, and kept the whole dataset for USPS (9298 images). However these datasets and their graphs remain way larger than the other standard benchmarks.

References

1. Hamilton, W.L.; Ying, R.; Leskovec, J. Representation learning on graphs: Methods and applications. *arXiv* **2017**, arXiv:1709.05584.
2. Narayanan, A.; Chandramohan, M.; Venkatesan, R.; Chen, L.; Liu, Y.; Jaiswal, S. graph2vec: Learning distributed representations of graphs. *arXiv* **2017**, arXiv:1707.05005.
3. Loukas, A. Graph reduction with spectral and cut guarantees. *J. Mach. Learn. Res.* **2019**, *20*, 1–42.
4. Togninalli, M.; Ghisu, E.; Llinares-López, F.; Rieck, B.; Borgwardt, K. Wasserstein weisfeiler-lehman graph kernels. In Proceedings of the Annual Conference on Neural Information Processing Systems 2019, Vancouver, BC, Canada, 8–14 December 2019; pp. 6439–6449.
5. Vishwanathan, S.V.N.; Schraudolph, N.N.; Kondor, R.; Borgwardt, K.M. Graph kernels. *J. Mach. Learn. Res.* **2010**, *11*, 1201–1242.
6. Shervashidze, N.; Borgwardt, K.M. Fast subtree kernels on graphs. In Proceedings of the 23rd Annual Conference on Neural Information Processing Systems 2009, Vancouver, BC, Canada, 7–10 December 2009; pp. 1660–1668.
7. Shervashidze, N.; Schweitzer, P.; Leeuwen, E.J.V.; Mehlhorn, K.; Borgwardt, K.M. Weisfeiler-lehman graph kernels. *J. Mach. Learn. Res.* **2011**, *12*, 2539–2561.
8. Feragen, A.; Kasenburg, N.; Petersen, J.; De Bruijne, M.; Borgwardt, K. Scalable kernels for graphs with continuous attributes. In Proceedings of the 27th Annual Conference on Neural Information Processing Systems 2013, Lake Tahoe, NV, USA, 5–8 December 2013; pp. 216–224.
9. Kriege, N.; Mutzel, P. Subgraph matching kernels for attributed graphs. *arXiv* **2012**, arXiv:1206.6483.
10. Morris, C.; Kriege, N.M.; Kersting, K.; Mutzel, P. Faster kernels for graphs with continuous attributes via hashing. In Proceedings of the 2016 IEEE 16th International Conference on Data Mining (ICDM), Barcelona, Spain, 12–15 December 2016.
11. Veličković, P.; Fedus, W.; Hamilton, W.L.; Liò, P.; Bengio, Y.; Hjelm, R.D. Deep graph infomax. In Proceedings of the ICLR, New Orleans, LA, USA, 6–9 May 2019.
12. Hamilton, W.; Ying, Z.; Leskovec, J. Inductive representation learning on large graphs. In Proceedings of the Annual Conference on Neural Information Processing Systems 2017, Long Beach, CA, USA, 4–9 December 2017; pp. 1024–1034.
13. Sun, F.Y.; Hoffman, J.; Verma, V.; Tang, J. Infograph: Unsupervised and semi-supervised graph-level representation learning via mutual information maximization. In Proceedings of the ICLR, Addis Ababa, Ethiopia, 26–30 April 2020.
14. Bianchi, F.M.; Grattarola, D.; Livi, L.; Alippi, C. Hierarchical representation learning in graph neural networks with node decimation pooling. *arXiv* **2019**, arXiv:1910.11436.
15. Dorfler, F.; Bullo, F. Kron reduction of graphs with applications to electrical networks. *IEEE Trans. Circuits Syst. I Regul. Pap.* **2013**, *60*, 150–163. [CrossRef]
16. Bravo Hermsdorff, G.; Gunderson, L. A unifying framework for spectrum-preserving graph sparsification and coarsening. In Proceedings of the Annual Conference on Neural Information Processing Systems 2019, Vancouver, BC, Canada, 8–14 December 2019; pp. 7736–7747.

17. Ying, Z.; You, J.; Morris, C.; Ren, X.; Hamilton, W.; Leskovec, J. Hierarchical graph representation learning with differentiable pooling. In Proceedings of the Annual Conference on Neural Information Processing Systems 2018, Montreal, QC, Canada, 3–8 December 2018; pp. 4800–4810.
18. Bianchi; Maria, F.; Grattarola, D.; Alippi, C. Spectral clustering with graph neural networks for graph pooling. In Proceedings of the 37th International Conference on Machine Learning, Online Event, 12–18 July 2020.
19. Wu, Z.; Pan, S.; Chen, F.; Long, G.; Zhang, C.; Philip, S.Y. A comprehensive survey on graph neural networks. *IEEE Trans. Neural Networks Learn. Syst.* **2020**, 1–21
20. Defferrard, M.; Bresson, X.; V.; Vandergheynst, P. Convolutional neural networks on graphs with fast localized spectral filtering. In Proceedings of the Annual Conference on Neural Information Processing Systems 2016, Barcelona, Spain, 5–10 December 2016; pp. 3844–3852.
21. Kipf, T.N.; Welling, M. Semi-supervised classification with graph convolutional networks. In Proceedings of the ICLR, Toulon, France, 24–26 April 2017.
22. Veličković, P.; Cucurull, G.; Casanova, A.; Romero, A.; Liò, P.; Bengio, Y. Graph attention networks. In Proceedings of the ICLR, Vancouver, BC, Canada, 30 April–3 May 2018.
23. Luan, S.; Zhao, M.; Chang, X.W.; Precup, D. Break the ceiling: Stronger multi-scale deep graph convolutional networks. In Proceedings of the Annual Conference on Neural Information Processing Systems 2019, Vancouver, BC, Canada, 8–14 December 2019; pp. 10945–10955.
24. Loukas, A. What graph neural networks cannot learn: Depth vs. width. In Proceedings of the ICLR, Addis Ababa, Ethiopia, 26–30 April 2020.
25. Gama, F.; Marques, A.G.; Leus, G.; Ribeiro, A. Convolutional neural network architectures for signals supported on graphs. *IEEE Trans. Signal Process.* **2019**, *67*, 1034–1049. [CrossRef]
26. Weisfeiler, B.; Lehman, A.A. A reduction of a graph to a canonical form and an algebra arising during this reduction. *Nauchno-Tech. Informatsia* **1968**, *2*, 12–16.
27. Kriege, N.M.; Giscard, P.L.; Wilson, R. On valid optimal assignment kernels and applications to graph classification. In Proceedings of the Annual Conference on Neural Information Processing Systems 2016, Barcelona, Spain, 5–10 December 2016; pp. 1623–1631.
28. Mikolov, T.; Sutskever, I.; Chen, K.; Corrado, G.S.; Dean, J. Distributed representations of words and phrases and their compositionality. In Proceedings of the 27th Annual Conference on Neural Information Processing Systems 2013, Lake Tahoe, NV, USA, 5–8 December 2013; pp. 3111–3119.
29. Hjelm, R.D.; Fedorov, A.; Lavoie-Marchildon, S.; Grewal, K.; Bachman, P.; Trischler, A.; Bengio, Y. Learning deep representations by mutual information estimation and maximization. In Proceedings of the ICLR, New Orleans, LA, USA, 6–9 May 2019.
30. Nowozin, S.; Cseke, B.; Tomioka, R. f-gan: Training generative neural samplers using variational divergence minimization. In Proceedings of the Annual Conference on Neural Information Processing Systems 2016, Barcelona, Spain, 5–10 December 2016; pp. 271–279.
31. Melamud, O.; Goldberger, J. Information-theory interpretation of the skip-gram negative-sampling objective function. In Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Short Papers), Vancouver, BC, Canada, 30 July–4 August 2017; Volume 2, pp. 167–171
32. Xu, K.; Hu, W.; Leskovec, J.; Jegelka, S. How powerful are graph neural networks? In Proceedings of the ICLR, New Orleans, LA, USA, 6–9 May 2019.
33. Hagberg, A.; Swart, P.; S Chult, D. Exploring network structure, dynamics, and function using NetworkX. In Proceedings of the 7th Python in Science Conference (SciPy2008), Pasadena, CA, USA, 19–24 August 2008; Varoquaux, G., Vaught, T., Millman, J., Eds.; 2008; pp. 11–15.
34. Maron, H.; Ben-Hamu, H.; Serviansky, H.; Lipman, Y. Provably powerful graph networks. In Proceedings of the Annual Conference on Neural Information Processing Systems 2019, Vancouver, BC, Canada, 8–14 December 2019; pp. 2156–2167.
35. Morris, C.; Ritzert, M.; Fey, M.; Hamilton, W.L.; Lenssen, J.E.; Rattan, G.; Grohe, M. Weisfeiler and leman go neural: Higher-order graph neural networks. In Proceedings of the AAAI Conference on Artificial Intelligence, Honolulu, HI, USA, 27 January–1 February 2019; Volume 33, pp. 4602–4609. [CrossRef]
36. Vayer, T.; Chapel, L.; Flamary, R.; Tavenard, R.; Courty, N. Optimal Transport for structured data with application on graphs. In Proceedings of the ICML 2019—36th International Conference on Machine Learning, Long Beach, CA, USA, 9–15 June 2019.

37. Kersting, K.; Kriege, N.M.; Morris, C.; Mutzel, P.; Neumann, M. Benchmark Data Sets for Graph Kernels. 2016. Available online: https://chrsmrrs.github.io/datasets/docs/datasets/ (accessed on 20 August 2020).
38. Orsini, F.; Frasconi, P.; DeRaedt, L. Graph invariant kernels. In Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, Buenos Aires, Argentina, 25–31 July 2015; pp. 3756–3762.
39. Kingma, D.P.; Ba, J. Adam: A method for stochastic optimization. *arXiv* **2014**, arXiv:1412.6980.
40. Nikolentzos, G.; Siglidis, G.; Vazirgiannis, M. Graph kernels: A survey. *arXiv* **2019**, arXiv:1904.12218.
41. Errica, F.; Podda, M.; Bacciu, D.; Micheli, A. A Fair Comparison of Graph Neural Networks for Graph Classification. *arXiv* **2019**, arXiv:1912.09893.
42. Xiao, H.; Rasul, K.; Vollgraf, R. Fashion-MNIST: A Novel Image Dataset for Benchmarking Machine Learning Algorithms. *arXiv* **2017**, arXiv:1708.07747.
43. Gilmer, J.; Schoenholz, S.S.; Riley, P.F.; Vinyals, O.; Dahl, G.E. Neural Message Passing for Quantum Chemistry. *arXiv* **2017**, arXiv:1704.01212.
44. Witten, T.A., Jr.; Sander, L.M. Diffusion-limited aggregation, a kinetic critical phenomenon. *Phys. Rev. Lett.* **1981**, *47*, 1400. [CrossRef]
45. Witten, T.A.; Sander, L.M. Diffusion-limited aggregation. *Phys. Rev. B* **1983**, *27*, 5686. [CrossRef]