word2vec, node2vec, graph2vec, X2vec: Towards a Theory of Vector Embeddings of Structured Data

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

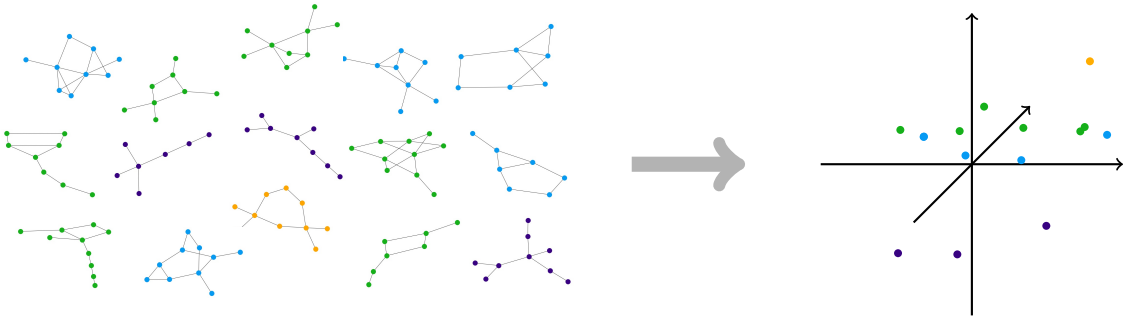


Figure 1: Embedding graphs into a vector space

ABSTRACT

Vector representations of graphs and relational structures, whether hand-crafted feature vectors or learned representations, enable us to apply standard data analysis and machine learning techniques to the structures. A wide range of methods for generating such embeddings have been studied in the machine learning and knowledge representation literature. However, vector embeddings have received relatively little attention from a theoretical point of view.

Starting with a survey of embedding techniques that have been used in practice, in this paper we propose two theoretical approaches that we see as central for understanding the foundations of vector embeddings. We draw connections between the various approaches and suggest directions for future research.

CCS CONCEPTS

• **Theory of computation** → **Database theory**; • **Computing methodologies** → **Knowledge representation and reasoning**; **Learning latent representations**; • **Mathematics of computing** → **Discrete mathematics**.

KEYWORDS

vector embedding; representation learning; graph kernel; graph neural net; homomorphism; Weisfeiler Leman

PODS'20, June 14–19, 2020, Portland, OR, USA

ACM ISBN 978-1-4503-7108-7/20/06…$15.00
https://doi.org/10.1145/3375395.3387641

ACM Reference Format:
Martin Grohe. 2020. word2vec, node2vec, graph2vec, X2vec: Towards a Theory of Vector Embeddings of Structured Data. In *Proceedings of the 39th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (PODS'20), June 14–19, 2020, Portland, OR, USA.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3375395.3387641

1 INTRODUCTION

Typical machine learning algorithms operating on structured data require representations of the often symbolic data as numerical vectors. Vector representations of the data range from handcrafted feature vectors via automatically constructed graph kernels to learned representations, either computed by dedicated embedding algorithms or implicitly computed by learning architectures like graph neural networks. The performance of machine learning methods crucially depends on the quality of the vector representations. Therefore, there is a wealth of research proposing a wide range of vector-embedding methods for various applications. Most of this research is empirical and often geared towards specific application areas. Given the importance of the topic, there is surprisingly little theoretical work on vector embeddings, especially when it comes to representing structural information that goes beyond metric information (that is, distances in a graph).

The goal of this paper is to give an overview over the various embedding techniques for structured data that are used in practice and to introduce theoretical ideas that can, and to some extent have been used to understand and analyse them. The research landscape on vector embeddings is unwieldy, with several communities working largely independently on related questions, motivated by different application areas such as social network analysis, knowledge graphs, chemoinformatics, computational biology, etc. Therefore, we need to be selective, focussing on common ideas and connections where we see them.

Vector embeddings can bridge the gap between the "discrete" world of relational data and the "differentiable" world of machine learning and for this reason have a great potential for database research. Yet relatively little work has been done on embeddings of relational data beyond the binary relations of knowledge graphs. Throughout the paper, I will try to point out potential directions for database related research questions on vector embeddings.

A *vector embedding* for a class $\mathcal{X}$ of objects is a mapping f from $\mathcal{X}$ into some vector space, called the *latent space*, which we usually assume to be a real vector space $\mathbb{R}^d$ of finite dimension d. The idea is to define a vector embedding in such a way that geometric relationships in the latent space reflect semantic relationships between the objects in $\mathcal{X}$. Most importantly, we want similar objects in $\mathcal{X}$ to be mapped to vectors close to one another with respect to some standard metric on the latent space (say, Euclidean). For example, in an embedding of words of a natural language we want words with similar meanings, like "shoe" and "boot", to be mapped to vectors that are close to each other. Sometimes, we want further-reaching correspondences between properties of and relations between objects in $\mathcal{X}$ and the geometry of their images in latent space. For example, in an embedding f of the entities of a knowledge base, among them `Paris`, `France`, `Santiago`, `Chile`, we may want $\boldsymbol{t} := f(\texttt{Paris}) - f(\texttt{France})$ to be (approximately) equal to $f(\texttt{Santiago}) - f(\texttt{Chile})$, so that the relation `is-capital-of` corresponds to the translation by the vector $\boldsymbol{t}$ in latent space.

A difficulty is that the semantic relationships and similarities between the objects in $\mathcal{X}$ can rarely be quantified precisely. They usually only have an intuitive meaning that, moreover, may be application dependent. However, this is not necessarily a problem, because we can learn vector representations in such a way that they yield good results when we use them to solve machine learning tasks (so-called *downstream tasks*). This way, we never have to make the semantic relationships explicit. As a simple example, we may use a nearest-neighbour based classification algorithm on the vectors our embedding gives us; if it performs well then the distance between vectors must be relevant for this classification task. This way, we can even use vector embeddings, trained to perform well on certain machine learning tasks, to define semantically meaningful distance measures on our original objects, that is, to define the distance $\text{dist}_f(X, Y)$ between objects $X, Y \in \mathcal{X}$ to be $\|f(X) - f(Y)\|$. We call dist_f the distance measure *induced* by the embedding f.

In this paper, the objects $X \in \mathcal{X}$ we want to embed either are graphs, possibly labelled or weighted, or more generally relational structures, or they are nodes of a (presumably large) graph or more generally elements or tuples appearing in a relational structure. When we embed entire graphs or structures, we speak of *graph embeddings* or *relational structure embeddings*; when we embed only nodes or elements we speak of *node embeddings*. These two types of embeddings are related, but there are clear differences. Most importantly, in node embeddings there are explicit relations such as adjacency and derived relations such as distance between the objects of $\mathcal{X}$ (the nodes of a graph), whereas in graph embeddings all relations between objects are implicit or "semantic", for example "having the same number of vertices" or"having the same girth" (see Figure 1).

The key theoretical questions we will ask about vector embeddings of objects in $\mathcal{X}$ are the following.

Expressivity: Which properties of objects $X \in \mathcal{X}$ are represented by the embedding? What is the meaning of the induced distance measure? Are there geometric properties of the latent space that represent meaningful relations on $\mathcal{X}$?

Complexity: What is the computational cost of computing the vector embedding? What are efficient embedding algorithms? How can we efficiently retrieve semantic information of the embedded data, for example, answer queries?

A third question that relates to both expressivity and complexity is what dimension to choose for the latent space. In general, we expect a trade-off between (high) expressivity and (low) dimension, but it may well be that there is an inherent dimension of the data set. It is an appealing idea (see, for example, [98]) to think of "natural" data sets appearing in practice as lying on a low dimensional manifold in high dimensional space. Then we can regard the dimension of this manifold as the inherent dimension of the data set.

Reasonably well-understood from a theoretical point of view are node embeddings of graphs that aim to preserve distances between nodes, that is, embeddings $f : V(G) \to \mathbb{R}^d$ of the vertex set $V(G)$ of some graph G such that $\text{dist}_G(x, y) \approx \|f(x) - f(y)\|$, where dist_G is the shortest-path distance in G. There is a substantial theory of such *metric embeddings* (see [64]). In many applications of node embeddings, metric embeddings are indeed what we need.

However, the metric is only one aspect of the information carried by a graph or relational structure, and arguably not the most important one from a database perspective. Moreover, if we consider graph embeddings rather than node embeddings, there is no metric to start with. In this paper, we are concerned with *structural* vector embeddings of graphs, relational structures, and their nodes. Two theoretical ideas that have been shown to help in understanding and even designing vector embeddings of structures are the *Weisfeiler-Leman algorithm* and various concepts in its context, and *homomorphism vectors*, which can be seen as a general framework for defining "structural" (as opposed to "metric") embeddings. We will see that these theoretical concepts have a rich theory that connects them to the embedding techniques used in practice in various ways.

The rest of the paper is organised as follows. Section 2 is a very brief survey of some of the embedding techniques that can be found in the machine learning and knowledge representation literature. In Section 3, we introduce the Weisfeiler-Leman algorithm. This algorithm, originally a graph isomorphism test, turns out to be an important link between the embedding techniques described in Section 2 and the theory of homomorphism vectors, which will be discussed in detail in Section 4. Finally, Section 5 is devoted to a discussion of similarity measures for graphs and structures.

2 EMBEDDING TECHNIQUES

In this section, we give a brief and selective overview of embedding techniques. More thorough recent surveys are [50] (on node embeddings), [104] (on graph neural networks), [102] (on knowledge graph embeddings), and [61] (on graph kernels).

2.1 From Metric Embeddings to Node Embeddings

Node embeddings can be traced back to the theory of embeddings of finite metric spaces and dimensionality reduction, which have been studied in geometry (e.g. [21, 55]) and algorithmic graph theory (e.g. [54, 64]). In statistics and data science, well-known traditional methods of metric embeddings and dimensionality reduction are multidimensional scaling [63], Isomap [98], and Laplacian eigenmap [11]. More recent related approaches are [1, 25, 85, 97]. The idea is always to embed the nodes of a graph in such a way that the distance (or similarity) between vectors approximates the distance (or similarity) between nodes. Sometimes, this can be viewed as a matrix factorisation. Suppose we have defined a similarity measure on the nodes of our graph $G = (V, E)$ that is represented by a similarity matrix $S \in \mathbb{R}^{V \times V}$. In the simplest version, we can just take S to be the adjacency matrix of the graph; in the literature this is sometimes referred to a *first-order proximity*. Another common choice is $S = (S_{vw})$ with $S_{vw} := \exp(-c \operatorname{dist}_G(v, w))$, where $c > 0$ is a parameter. We describe our embedding of V into $\mathbb{R}^d$ by a matrix $X \in \mathbb{R}^{V \times d}$ whose rows $\boldsymbol{x}_v \in \mathbb{R}^d$ are the images of the nodes. If we measure the similarity between vectors $\boldsymbol{x}, \boldsymbol{y}$ by their normalised inner product $\frac{\langle \boldsymbol{x}, \boldsymbol{y} \rangle}{\|\boldsymbol{x}\| \|\boldsymbol{y}\|}$ (this is known as the *cosine similarity*), then our objective is to find a matrix X with normalised rows that minimises $\|XX^\top - S\|_F$ with respect to the Frobenius norm $\|\cdot\|_F$ (or any other matrix norm, see Section 5). In the basic version with the Frobenius norm, the problem can be solved using the singular value decomposition of S. In a more general form, we compute a similarity matrix $\widehat{S} \in \mathbb{R}^{V \times V}$ whose (v, w)-entry quantifies the similarity between vectors $\boldsymbol{x}_v, \boldsymbol{x}_w$ and minimise the distance between S and $\widehat{S}$, for example using stochastic gradient descent. In [50], this approach to learning node embeddings is described as an *encoder-decoder* framework.

Learned word embeddings and in particular the WORD2VEC algorithm [74] introduced new ideas that had huge impact in natural language processing. These ideas also inspired new approaches to node embeddings like DEEPWALK [87] and NODE2VEC [48] based on taking short random walks in a graph and interpreting the sequence of nodes seen on such random walks as if they were words appearing together in a sentence. These approaches can still be described in the matrix-similarity (or encoder-decoder) framework: as the similarity between nodes v and w we take the probability that a fixed-length random walk starting in v ends in w. We can approximate this probability by sampling random walks. Note that, even in undirected graphs, this similarity measure is not necessarily symmetric.

From a deep learning perspective, the embedding methods described so far are all "shallow" in that they directly optimise the output vectors and there are no hidden layers; computing the vector $\boldsymbol{x}_v$ corresponding to a node v amounts to a table lookup. There are also deep learning methods for computing node embeddings (e.g. [26, 49, 101]). Before we discuss such approaches any further, let us introduce graph neural networks as a general deep learning framework for graphs that has received a lot of attention in recent years.

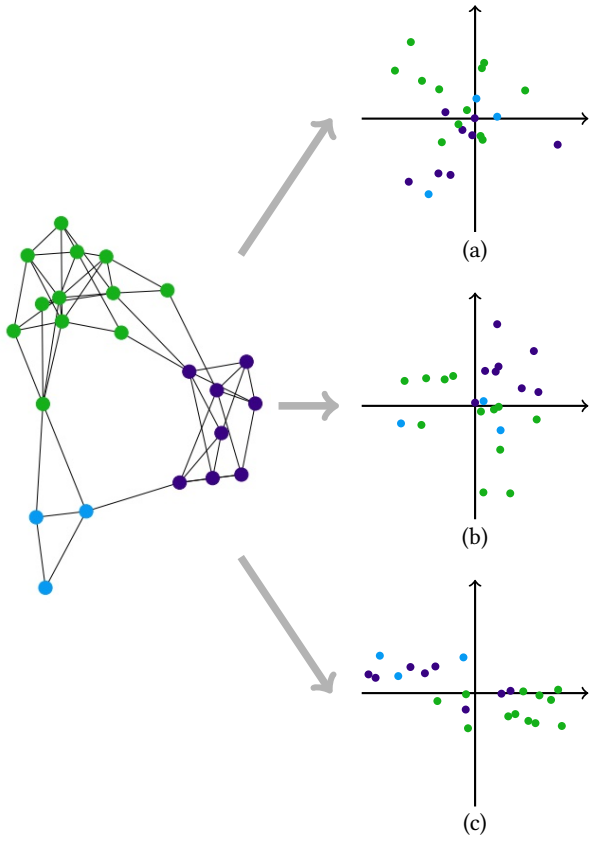


Figure 2: Three node embeddings of a graph using (a) singular value decomposition of adjacency matrix, (b) singular value decomposition of the similarity matrix with entries $S_{vw} = \exp(-2 \operatorname{dist}(v, w))$, (c) NODE2VEC [48]

2.2 Graph Neural Networks

When trying to apply deep learning methods to graphs or relational structures, we face two immediate difficulties: (1) we want the methods to scale across different graph sizes, but standard feed-forward neural networks have a fixed input size; (2) we want the methods to be isomorphism invariant and not depend on a specific representation of the input graph. Both points show that it is problematic to just feed the adjacency matrix (or any other standard representation of a graph) into a deep neural network in a generic "end-to-end" learning architecture.

Graph neural networks (GNNs) are a deep learning framework for graphs that avoids both of these difficulties, albeit at the price of limited expressiveness (see Section 3). Early forms of GNNs were introduced in [36, 90]; the version we present here is based on [49, 59, 88]. Intuitively, a GNN model can be thought of as a message passing network over the input graph $G = (V, E)$. Each node v has a state $\boldsymbol{x}_v \in \mathbb{R}^d$. Nodes can exchange messages along the edges of G and update their states. To specify the model, we need to specify two functions: an *aggregation* function that takes the current states of the neighbours of a node and aggregates them into a single vector, and an *update* function that takes the aggregate

value obtained from the neighbours and the current state of the node as inputs and computes the new state of the node. In a simple form, we may take the following functions:

$$\textsc{Aggregate}: \quad \boldsymbol{a}_v^{(t+1)} \leftarrow \sum_{w \in N(v)} W_{\text{AGG}} \cdot \boldsymbol{x}_w^{(t)}, \tag{2.1}$$

$$\textsc{Update}: \quad \boldsymbol{x}_v^{(t+1)} \leftarrow \sigma\left(W_{\text{UP}} \cdot \begin{pmatrix} \boldsymbol{x}_v^{(t)} \\ \boldsymbol{a}_v^{(t+1)} \end{pmatrix}\right), \tag{2.2}$$

where $W_{\text{AGG}} \in \mathbb{R}^{c \times d}$ and $W_{\text{UP}} \in \mathbb{R}^{d \times (c+d)}$ are learned parameter matrices and σ is a nonlinear "activation" function, for example the ReLU (rectified linear unit) function $\sigma(x) := \max\{0, x\}$ applied pointwise to a vector. It is important to note that the parameter matrices W_{AGG} and W_{UP} do not depend on the node v; they are shared across all nodes of a graph. This parameter sharing allows it to use the same GNN model for graphs of arbitrary sizes.

Of course, we can also use more complicated aggregation and update functions. We only want these functions to be differentiable to be able to use gradient descent optimisation methods in the training phase, and we want the aggregation function to be symmetric in its arguments $\boldsymbol{x}_w^{(t)}$ for $w \in N(v)$ to make sure that the GNN computes a function that is isomorphism invariant. For example, in [100] we use a linear aggregation function and an update function computed by an LSTM (long short-term memory, [51]), a specific recurrent neural network component that allows it to "remember" relevant information from the sequence $\boldsymbol{x}_v^{(0)}, \boldsymbol{x}_v^{(1)}, \ldots, \boldsymbol{x}_v^{(t)}$.

The computation of such a GNN model starts from a initial configuration $(\boldsymbol{x}_v^{(0)})_{v \in V}$ and proceeds through a fixed-number t of aggregation- and update-steps, resulting in a final configuration $(\boldsymbol{x}_v^{(t)})_{v \in V}$. Note that this configuration gives us a node embedding $v \mapsto \boldsymbol{x}_v^{(t)}$ of the input graph. We can also stack several such GNN layers, each with its own aggregation and activation function, on top of one another, using the final configuration of each (but the last) layer as the initial configuration of the following layer and the final configuration of the last layer as the node embedding. As initial states, we can take constant vectors like the all-ones vector for each node, or we can assign a random initial state to each node. We can also use the initial state to represent the node labels if the input graph is labelled.

To train a GNN for computing a node-embedding, in principle we can use any of the loss functions used by the embedding techniques described in Section 2.1. The reader may wonder what advantage the complicated GNN architecture has over just optimising the embedding matrix X (as the methods described in Section 2.1 do). The main advantage is that the GNN method is *inductive*, whereas the previously described methods are *transductive*. This means that a GNN represents a function that we can apply to arbitrary graphs, not just to the graph it was originally trained on. So if the graph changes over time and, for example, nodes are added, we do not have to re-train the embedding, but just embed the new nodes using the GNN model we already have, which is much more efficient. We can even apply the model to an entirely new graph and still hope it gives us a reasonable embedding. The most prominent example of an inductive node-embedding tool based on GNNs is GraphSage [49].

Let me close this section by remarking that GNNs are used for all kinds of machine learning tasks on graphs and not only to compute node embeddings. For example, a GNN based architecture for graph classification would plug the output of the GNN layer(s) into a standard feedforward network (possibly consisting only of a single softmax layer).

2.3 Knowledge Graph and Relational Structure Embeddings

Node embeddings of knowledge graphs have also been studied quite intensely in recent years, remarkably by a community that seems almost disjoint from that involved in the node embedding techniques described in Section 2.1. What makes knowledge graphs somewhat special is that they come with labelled edges (or, equivalently, many different binary relations) as well as labelled nodes. It is not completely straightforward to adapt the methods of Section 2.1 to edge- and vertex-labelled graphs. Another important difference is in the objective function: the methods of Section 2.1 mainly focus on the graph metric (even though approaches based on random walks like node2vec are flexible and also incorporate structural criteria). However, shortest-path distance is less relevant in knowledge graphs.

Rather than focussing on distances, knowledge graph embeddings focus on establishing a correspondence between the relations of the knowledge graph and geometric relationships in the latent space. A very influential algorithm, TransE [18] aims to associate a specific translation of the latent space with each relation. Recall the example of the introduction, where entities `Paris`, `France`, `Santiago`, `Chile` were supposed to be embedded in such a way that $\boldsymbol{x}_{\texttt{Paris}} - \boldsymbol{x}_{\texttt{France}} \approx \boldsymbol{x}_{\texttt{Santiago}} - \boldsymbol{x}_{\texttt{Chile}}$, so that the relation `is-capital-of` corresponds to the translation by $\boldsymbol{t} := \boldsymbol{x}_{\texttt{Paris}} - \boldsymbol{x}_{\texttt{France}}$.

Another way of mapping relations to geometric relationships is implemented in Rescal [83]. Here the idea is to associate a bilinear form β_R with each relation R in such a way that for all entities v, w it holds that $\beta_R(\boldsymbol{x}_v, \boldsymbol{x}_w) \approx 1$ if $(v, w) \in R$ and $\beta_R(\boldsymbol{x}_v, \boldsymbol{x}_w) \approx 0$ if $(v, w) \notin R$. We can represent such a bilinear form β_R by a matrix B_R such that $\beta_R(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{x}^\top B_R \boldsymbol{y}$. Then the objective is to minimise, simultaneously for all R, the term $\|X B_R X^\top - A_R\|$, where X is the embedding matrix with rows $\boldsymbol{x}_v$ and A_R is the adjacency matrix of the relation R. Note that this is a multi-relational version of the matrix-factorisation approach described in Section 2.1, with the additional twist that we also need to find the matrix B_R for each relation R.

Completing our remarks on knowledge graph embeddings, we mention that it is fairly straightforward to generalise the GNN based node embeddings to (vertex- and edge-)labelled graphs and hence to knowledge graphs [91].

While there is a large body of work on embedding knowledge graphs, that is, binary relational structures, not much is known about embedding relations of higher arities. Of course one approach to embedding relational structures of higher arities is to transform them into their binary incidence structures (see Section 4.2 for a definition) and then embed these using any of the methods available for binary structures. Currently, I am not aware of any empirical studies on the practical viability of this approach. An alternative

approach [16, 17] is based on the idea of treating the rows of a table, that is, tuples in a relation, like sentences in natural language and then use word embeddings to embed the entities.

2.4 Graph Kernels

There are machine learning methods that operate on vector representations of their input objects, but only use these vector representations implicitly and never actually access the vectors. All they need to access is the inner product between two vectors. For reasons that will become apparent soon, such methods are known as *kernel methods*. Among them are support vector machines for classification [29] and principal component analysis as well as k-means clustering for unsupervised learning [92] (see [93, Chapter 16] for background).

A *kernel functions* for a set $\mathcal{X}$ of objects is a binary function $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ that is symmetric, that is, $K(x, y) = K(y, x)$ for all $x, y \in \mathcal{X}$, and *positive semidefinite*, that is, for all $n \geq 1$ and all $x_1, \ldots, x_n \in \mathcal{X}$ the matrix M with entries $M_{ij} := K(x_i, x_j)$ is positive semidefinite. Recall that a symmetric matrix $M \in \mathbb{R}^{n \times n}$ is positive semidefinite if all its eigenvalues are nonnegative, or equivalently, if $\boldsymbol{x}^T M \boldsymbol{x} \geq 0$ for all $\boldsymbol{x} \in \mathbb{R}^n$. It can be shown that a symmetric function $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a kernel function if and only if there is a vector embedding $f : \mathcal{X} \to \mathbb{H}$ of $\mathcal{X}$ into some Hilbert space $\mathbb{H}$ such that K is the mapping induced by the inner product of $\mathbb{H}$, that is, $K(x, y) = \langle f(x), f(y) \rangle$ for all $x, y \in X$ (see [93, Lemma 16.2] for a proof). For our purposes, it suffices to know that a *Hilbert space* is a potentially infinite dimensional real vector space $\mathbb{H}$ with a a symmetric bilinear form $\langle \cdot, \cdot \rangle : \mathbb{H} \times \mathbb{H} \to \mathbb{R}$ satisfying $\langle \boldsymbol{x}, \boldsymbol{x} \rangle > 0$ for all $\boldsymbol{x} \in \mathbb{H} \setminus \{\boldsymbol{0}\}$. A difference between the embeddings underlying kernels and most other vector embeddings is that the kernel embeddings usually embed into higher dimensional spaces (even infinite dimensional Hilbert spaces). Dimension does not play a big role for kernels, because the embeddings are only used implicitly. Nevertheless, it can sometimes be more efficient to use the embeddings underlying kernels explicitly [62].

Kernel methods have dominated machine learning on graphs for a long time and, despite of the recent successes of GNNs, they are still competitive. Quoting [61], *"It remains a current challenge in research to develop neural techniques for graphs that are able to learn feature representations that are clearly superior to the fixed feature spaces used by graph kernels."*

The first dedicated graph kernels were the random walk graph kernels [37, 56] based on counting walks of all lengths. In some sense, they are similar to the random walk based node-embedding algorithms described in Section 2.1. Other graph kernels are based on counting shortest paths, trees and cycles, and small subgraph patterns [20, 52, 89, 95]. The important Weisfeiler-Leman kernels [94], which we will describe in Section 3, are based on aggregating local neighbourhood information. All these kernels are defined for graphs with discrete labels. The techniques, all essentially based on counting certain subgraphs, are not directly applicable to graphs with continuous labels such as edge weights. An adaptation to continuous labels based on hashing is proposed in [77].

Let me remark that there are also *node kernels* defined on the nodes of a graph [60, 82, 96]. They implicitly give a vector embedding of the nodes. However, compared to the node embedding techniques discussed before, they only play a minor role. For graph embeddings, we are in the opposite situation: kernels are the dominant technique. However, there are a few other approaches.

2.5 Graph Embeddings

GRAPH2VEC [80] is a transductive approach to embedding graphs inspired by WORD2VEC and some of the node embedding techniques discussed in Section 2.1. As before, "transductive" means that the embedding is computed for a fixed set of graphs in form of an embedding matrix (or look-up table) and thus it only yields an embedding for graphs known at training time. For typical machine learning applications it is unusual to operate on set of graphs that is fixed in advance, so inductive embedding approaches are clearly preferable.

GNNs can also be used to embed entire graphs, in the simplest form by just aggregating the embeddings of the nodes computed by the GNN. *Graph autoencoders* [58, 86] give a way to train such graph embeddings in an unsupervised manner. A more advanced GNN architecture for learning graph embedding, based on capsule neural networks, is proposed in [105].

3 THE WEISFEILER-LEMAN ALGORITHM

We slightly digress from our main theme and introduce the Weisfeiler-Leman algorithm, a very efficient combinatorial partitioning algorithm that was originally introduced as a fingerprinting technique for chemical molecules [75]. The algorithm plays an important role in the graph isomorphism literature, both in theory (for example, [7, 41]) and practice, where it appears as a subroutine in all competitive graph isomorphism tools (see [73]). As we will see, the algorithm has interesting connections with the embedding techniques discussed in the previous section.

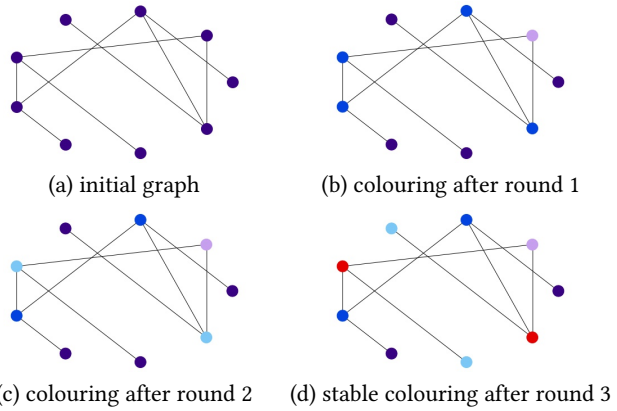
3.1 1-Dimensional Weisfeiler-Leman

The Weisfeiler-Leman algorithm has a parameter k, its *dimension*. We start by describing the 1-dimensional version 1-*WL*, which is also known as *colour refinement* or *naive vertex classification*. The algorithm computes a partition of the nodes of its input graph. It is convenient to think of the classes of the partition as colours of the nodes. A colouring (or partition) is *stable* if any two nodes v, w of the same colour c have the same number of neighbours of any colour d. The algorithm computes a stable colouring by iteratively refining an initial colouring as described in Algorithm 1. Figure 3 shows an example run of 1-WL. The algorithm is very efficient; it can be implemented to run in time $O((n + m) \log n)$, where n is the number of vertices and m the number of edges of the input graph [27]. Under reasonable assumptions on the algorithms used, this is best possible [12].

To use 1-WL as an isomorphism test, we note that the colouring computed by the algorithm is *isomorphism invariant*, which means that if we run the algorithm on two isomorphic graphs the resulting coloured graphs will still be isomorphic and in particular have the same numbers of nodes of each colour. Thus, if we run the algorithm on two graphs and find that they have distinct numbers of vertices of some colour, we have produced a certificate of non-isomorphism. If this is the case, we say that 1-WL *distinguishes*

1-WL
Input: Graph G
Initialisation: All nodes get the same colour.
Refinement Round: For all colours c in the current colouring and all nodes v, w of colour c, the nodes v and w get different colours in the new colouring if there is some colour d such that v and w have different numbers of neighbours of colour d.
The refinement is repeated until the colouring is stable, then the stable colouring is returned.

Algorithm 1: The 1-dimensional WL algorithm



(a) initial graph (b) colouring after round 1
(c) colouring after round 2 (d) stable colouring after round 3

Figure 3: A run of 1-WL

the two graphs. Unfortunately, 1-WL does not distinguish all non-isomorphic graphs. For example, it does not distinguish a cycle of length 6 from the disjoint union of two triangles. But, remarkably, 1-WL does distinguish *almost all* graphs, in a precise probabilistic sense [8].

3.2 Variants of 1-WL

The version of 1-WL we have formulated is designed for undirected graphs. For directed graphs it is better to consider in-neighbours and out-neighbours of nodes separately. 1-WL can easily be adapted to labelled graphs. If vertex labels are present, they can be incorporated in the initial colouring: two vertices get the same initial colour if and only if they have the same label(s). We can incorporate edge labels in the refinement rounds: two nodes v and w get different colours in the new colouring if there is some colour d and some edge label λ such that v and w have a different number of λ-neighbours of colour d.

However, if the edge labels are real numbers, which we interpret as edge *weights*, or more generally elements of an arbitrary commutative monoid, then we can also use the following weighted version of 1-WL due to [44]. Instead of refining by the number of edges into some colour, we refine by the sum of the edge weights into that colour. Thus the refinement round of Algorithm 1 is modified as follows: for all colours c in the current colouring and all nodes v, w of colour c, v and w get different colours in the new colouring

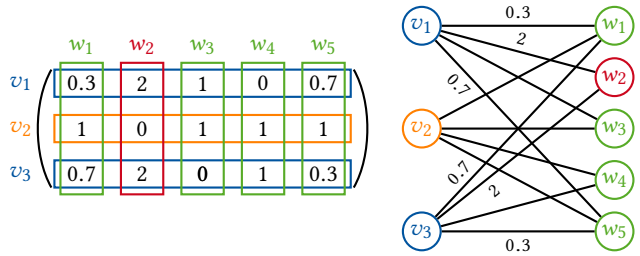


Figure 4: Stable colouring of a matrix and the corresponding weighted bipartite graph computed by matrix WL

if there is some colour d such that

$$\sum_{x \text{ of colour } d} \alpha(v, x) \neq \sum_{x \text{ of colour } d} \alpha(w, x), \tag{3.1}$$

where $\alpha(x, y)$ denotes the weight of the edge from x to y, and we set $\alpha(x, y) = 0$ if there is no edge from x to y. This idea also allows us to define 1-WL on matrices: with a matrix $A \in \mathbb{R}^{m \times n}$ we associate a weighted bipartite graph with vertex set $\{v_1, \ldots, v_m, w_1, \ldots, w_n\}$ and edge weights $\alpha(v_i, w_j) := A_{ij}$ and $\alpha(v_i, v_{i'}) = \alpha(w_j, w_{j'}) = 0$ and run weighted 1-WL on this weighted graph with initial colouring that distinguishes the v_i (rows) from the w_j (columns). An example is shown in Figure 4. This matrix-version of WL was applied in [44] to design a dimension reduction techniques that speeds up the solving of linear programs with many symmetries (or regularities).

3.3 Higher-Dimensional WL

For this paper, the 1-dimensional version of the Weisfeiler-Leman algorithm is the most relevant, but let us briefly describe the higher dimensional versions. In fact, it is the 2-dimensional version, also referred to as *classical WL*, that was introduced by Weisfeiler and Leman [103] in 1968 and gave the algorithm its name. The *k-dimensional Weisfeiler-Leman algorithm (k-WL)* is based on the same iterative-refinement idea as 1-WL. However, instead of vertices, k-WL colours k-tuples of vertices of a graph. Initially, each k-tuple is "coloured" by the isomorphism type of the subgraph it induces. Then in the refinement rounds, the colour information is propagated between "adjacent" tuples that only differ in one coordinate (details can be found in [24]). If implemented using similar ideas as for 1-WL, k-WL runs in time $O(n^{k+1} \log n)$ [53].

Higher-dimensional WL is much more powerful than 1-WL, but Cai, Fürer, and Immerman [24] proved that for every k there are non-isomorphic graphs G_k, H_k that are not distinguished by k-WL. These graphs, known as the *CFI graphs*, have size $O(k)$ and are 3-regular.

DeepWL, a WL-Version of unlimited dimension that can distinguish the CFI-graphs in polynomial time, was recently introduced in [47].

3.4 Logical and Algebraic Characterisations

The beauty of the WL algorithm lies in the fact that its expressiveness has several natural and completely unrelated characterisations.

Of these, we will see two in this section. Later, we will see two more characterisations in terms of GNNs and homomorphism numbers.

The logic C is the extension of first-order logic by counting quantifiers of the form $\exists^{\geq p} x$ ("there exists at least p elements x"). Every C-formula is equivalent to a formula of plain first-order logic. However, here we are interested in fragments of C obtained by restricting the number of variables of formulas, and the translation from C to first-order logic may increase the number of variables. For every $k \geq 1$, by C^k we denote the fragment of C consisting of all formulas with at most k (free or bound) variables. The finite variable logics C^k play an important role in finite model theory (see, for example, [40]). Cai, Fürer, and Immerman [24] have related these fragments to the WL algorithm.

Theorem 3.1 ([24]). *Two graphs are C^{k+1}-equivalent, that is, they satisfy the same sentences of the logic C^{k+1}, if and only if k-WL does not distinguish the graphs.*

Let us now turn to an algebraic characterisation of WL. Our starting point is the observation that two graphs G, H with vertex sets V, W and adjacency matrices $A \in \mathbb{R}^{V\times V}, B \in \mathbb{R}^{W\times W}$ are isomorphic if and only there is a permutation matrix $X \in \mathbb{R}^{V\times W}$ such that $X^\top AX = B$. Recall that a permutation matrix is a $\{0,1\}$-matrix that has exactly one 1-entry in each row and in each column. Since permutation matrices are orthogonal (i.e., they satisfy $X^\top = X^{-1}$), we can rewrite this as $AX = XB$, which has the advantage of being linear. This corresponds to the following linear equations in the variables X_{vw}, for $v \in V$ and $w \in W$:

$$\sum_{v'\in V} A_{vv'}X_{v'w} = \sum_{w'\in W} X_{vw'}B_{w'w} \qquad \text{for all } v \in V, w \in W. \quad (3.2)$$

We can add equations expressing that the row and column sums of the matrix X are 1, which implies that X is a permutation matrix if the X_{vw} are nonnegative integers.

$$\sum_{w'\in W} X_{vw'} = \sum_{v'\in V} X_{v'w} = 1 \qquad \text{for all } v \in V, w \in W. \quad (3.3)$$

Obviously, equations (3.2) and (3.3) have a nonnegative integer solution if and only if the graphs G and H are isomorphic. This does not help much from an algorithmic point of view, because it is NP-hard to decide if a system of linear equations and inequalities has an integer solution. But what about nonnegative rational solutions? We know that we can compute them in polynomial time. A nonnegative rational solution to (3.2) and (3.3), which can also be seen as a doubly stochastic matrix satisfying $AX = XB$, is called a *fractional isomorphism* between G and H. If such a fractional isomorphism exists, we say that G and H are *fractionally isomorphic*. Tinhofer [99] proved the following theorem.

Theorem 3.2 ([99]). *Graphs G and H are fractionally isomorphic if and only if 1-WL does not distinguish G and H.*

A corresponding theorem also holds for the weighted and the matrix version of 1-WL [44]. Moreover, Atserias and Maneva [5] proved a generalisation that relates k-WL to the level-k Sherali-Adams relaxation of the system of equations and thus yields an algebraic characterisation of k-WL indistinguishability (also see [45, 71] and [6, 13, 39, 84] for related algebraic aspects of WL).

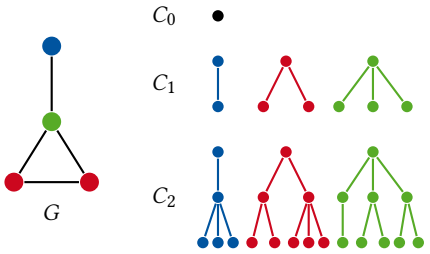


Figure 5: Viewing colours of WL as trees

Note that to decide whether two graphs G, H with adjacency matrices G, H are fractionally isomorphic, we can minimise the convex function $\|AX - XB\|_F$, where X ranges over the convex set of doubly stochastic matrices. To minimise this function, we can use standard gradient descent techniques for convex minimisation. It was shown in [57] that, surprisingly, the refinement rounds of 1-WL closely correspond to the iterations of the Frank-Wolfe convex minimisation algorithm.

3.5 Weisfeiler-Leman Graph Kernels

The WL algorithm collects local structure information and propagates it along the edges of a graph. We can define very effective graph kernels based on this local information. For every $i \geq 0$, let C_i be the set of colours that 1-WL assigns to the vertices of a graph in the i-th round. Figure 5 illustrates that we can identify the colours in C_i with rooted trees of height i. For every graph G and every colour $c \in C_i$, by $\mathsf{wl}(c, G)$ we denote the number of vertices that receive colour c in the ith round of 1-WL.

Example 3.3. For the graph G shown in Figure 5 we have

$$\mathsf{wl}\left(\,\raisebox{-0.3em}{  }\,, G\right) = 2, \qquad \mathsf{wl}\left(\,\raisebox{-0.3em}{  }\,, G\right) = 0.$$

For every $t \in \mathbb{N}$, the *t-round WL-kernel* is the mapping $K^{(t)}_{\mathrm{WL}}$ defined by

$$K^{(t)}_{\mathrm{WL}}(G, H) := \sum_{i=0}^{t} \sum_{c\in C_i} \mathsf{wl}(c, G) \cdot \mathsf{wl}(c, H)$$

for all graphs G, H. It is easy to see that this mapping is symmetric and positive-semidefinite and thus indeed a kernel mapping; the corresponding vector embedding maps each graph G to the vector

$$\left(\mathsf{wl}(c, G) \;\middle|\; c \in \bigcup_{i=0}^{t} C_i\right).$$

Note that formally, we are mapping G to an infinite dimensional vector space, because all the sets C_i for $i \geq 1$ are infinite. However, for a graph G of order n the vector as at most $kn+1$ nonzero entries. We can also define a version K_{WL} of the WL-kernel that does not depend on a fixed-number of rounds by letting

$$K_{\mathrm{WL}}(G, H) := \sum_{i\geq 0} \frac{1}{2^i} \sum_{c\in C_i} \mathsf{wl}(c, G) \cdot \mathsf{wl}(c, H).$$

The WL-kernel was introduced by Shervashidze et al. [94] under the name *Weisfeiler-Leman subtree kernel*. They also introduce variants such as a *Weisfeiler-Leman shortest path kernel*. A great advantage

the WL (subtree) kernel has over most of the graph kernels discussed in Section 2.4 is its efficiency, while performing at least as good as other kernels on downstream tasks. Shervashidze et al. [94] report that in practice, $t = 5$ is a good number of rounds for the t-round WL-kernel.

There are also graph kernels based on higher dimensional WL algorithm [76].

3.6 Weisfeiler-Leman and GNNs

Recall that a GNN computes a sequence $(\boldsymbol{x}_v^{(t)})_{v\in V}$, for $t \geq 0$, of vector embeddings of a graph $G = (V, E)$. In the most general form, it is recursively defined by

$$\boldsymbol{x}_v^{(t+1)} = f_{\mathrm{UP}}\Big(\boldsymbol{x}_v^{(t)}, f_{\mathrm{AGG}}\big(\boldsymbol{x}_w \mid w \in N(v)\big)\Big),$$

where the aggregation function f_{AGG} is symmetric in its arguments. It has been observed in several places [49, 78, 106] that this is very similar to the update process of 1-WL. Indeed, it is easy to see that if the initial embedding $\boldsymbol{x}_v^{(0)}$ is constant then for any two vertices v, w, if 1-WL assigns the same colour to v and w then $\boldsymbol{x}_v^{(t)} = \boldsymbol{x}_w^{(t)}$. This implies that two graphs that cannot be distinguished by 1-WL will give the same result for any GNN applied to them; that is, GNNs are at most as expressive as 1-WL. It is shown in [78] that a converse of this holds as well, even if the aggregation and update functions of the GNN are of a very simple form (like (2.1) and (2.2) in Section 2.2). Based on the connection between WL and logic, a more refined analysis of the expressiveness of GNNs was carried out in [10]. However, the limitations of the expressiveness only hold if the initial embedding $\boldsymbol{x}_v^{(0)}$ is constant (or at least constant on all 1-WL colour classes). We can increase the expressiveness of GNNs by assigning random initial vectors $\boldsymbol{x}_v^{(0)}$ to the vertices. The price we pay for this increased expressiveness is that the output of a run of the GNN model is no longer isomorphism invariant. However, the whole randomised process is still isomorphism invariant. More formally, the random variable that associates an output $(\boldsymbol{x}_v^{(0)})_{v\in V}$ with each graph G is isomorphism invariant.

A fully invariant way to increase the expressiveness of GNNs is to build "higher-dimensional" GNNs, inspired by the higher-dimensional WL algorithm. Instead of nodes of the graphs, they operate on constant sized tuples or sets of vertices. A flexible architecture for such higher-dimensional GNNs is proposed in [78].

4 COUNTING HOMOMORPHISMS

Most of the graph kernels and also some of the node embedding techniques are based on counting occurrences of substructures like walks, cycles, or trees. There are different ways of embedding substructures into a graph. For example, walks and paths are the same structures, but we allow repeated vertices in a walk. Formally, "walks" are homomorphic images of path graphs, whereas "paths" are embedded path graphs. It turns that homomorphisms and homomorphic images give us a very robust and flexible "basis" for counting all kinds of substructures [30].

A homomorphism from a graph F to a graph G is a mapping h from the nodes of F to the nodes of G such that for all edges uu' of F the image $h(u)h(u')$ is an edge of G. On labelled graphs, homomorphisms have to preserve vertex and edge labels, and on directed graphs they have to preserve the edge direction. Of course we can generalise homomorphisms to arbitrary relational structures, and we remind the reader of the close connection between homomorphisms and conjunctive queries. We denote the number of homomorphisms from F to G by $\mathsf{hom}(F, G)$.

Example 4.1. For the graph G shown in Figure 5 we have

$$\mathsf{hom}\left(\begin{array}{c}\bullet\\ \bullet\quad\bullet\end{array}, G\right) = 18, \qquad \mathsf{hom}\left(\begin{array}{c}\bullet\\ \bullet\ \bullet\ \bullet\ \bullet\end{array}, G\right) = 114.$$

To calculate these numbers, we observe that for the star S_k (tree of height 1 with k leaves) we have $\mathsf{hom}(S_k, G) = \sum_{v\in V(G)} \deg_G(v)^k$.

For every class $\mathcal{F}$ of graphs, the homomorphism counts $\mathsf{hom}(F, G)$ give a graph embedding $\mathsf{Hom}_{\mathcal{F}}$ defined by

$$\mathsf{Hom}_{\mathcal{F}}(G) := \big(\mathsf{hom}(F, G) \;\big|\; F \in \mathcal{F}\big)$$

for all graphs G. If $\mathcal{F}$ is infinite, the latent space $\mathbb{R}^{\mathcal{F}}$ of the embedding $\mathsf{Hom}_{\mathcal{F}}$ is an infinite dimensional vector space. By suitably scaling the infinite series involved, we can define an inner product on a subspace $\mathbb{H}_{\mathcal{F}}$ of $\mathbb{R}^{\mathcal{F}}$ that includes the range of $\mathsf{Hom}_{\mathcal{F}}$. This also gives us a graph kernel. One way of making this precise is as follows. For every k, we let $\mathcal{F}_k$ be the set of all $F \in \mathcal{F}$ of order $|F| := |V(F)| = k$. Then we let

$$K_{\mathcal{F}}(G, H) := \sum_{k=1}^{\infty} \frac{1}{|\mathcal{F}_k|} \sum_{F\in\mathcal{F}_k} \frac{1}{k^k} \mathsf{hom}(F, G) \cdot \mathsf{hom}(F, H). \tag{4.1}$$

There are various other ways of doing this, for example, rather than looking at the sum over all $F \in \mathcal{F}^k$ we may look at the maximum. In practice, one will simply cut off of the infinite series and only consider a finite subset of $\mathcal{F}$. A problem with using homomorphism vectors as graph embeddings is that the homomorphism numbers quickly get tremendously large. In practice, we take logarithms of theses numbers, possibly scaled by the size of the graphs from $\mathcal{F}$. So, a practically reasonable graph embedding based on homomorphism vectors would take a finite class $\mathcal{F}$ of graphs and map each G to the vector

$$\Big(\frac{1}{|F|} \log\big(\mathsf{hom}(F, G)\big) \;\Big|\; F \in \mathcal{F}\Big).$$

Initial experiments show that this graph embedding performs very well on downstream classification tasks even if we take $\mathcal{F}$ to be a small class (of size 20) of graphs consisting of binary trees and cycles. This is a good indication that homomorphism vectors extract relevant features from a graph. Note that the size of the class $\mathcal{F}$ is the dimension of the feature space.

Apart from these practical considerations, homomorphisms vectors have a beautiful theory that links them to various natural notions of similarity between structures, including indistinguishability by the Weisfeiler-Leman algorithm.

4.1 Homomorphism Indistinguishability

Two graphs G and H are *homomorphism-indistinguishable* over a class $\mathcal{F}$ of a graphs if $\mathsf{Hom}_{\mathcal{F}}(G) = \mathsf{Hom}_{\mathcal{F}}(H)$. Lovász proved that homomorphism indistinguishability over the class $\mathcal{G}$ of all graphs corresponds to isomorphism.

Theorem 4.2 ([65]). *For all graphs G and H,*

$$\mathsf{Hom}_{\mathcal{G}}(G) = \mathsf{Hom}_{\mathcal{G}}(H) \iff G \text{ and } H \text{ are isomorphic.}$$

Proof. The backward direction is trivial. For the forward direction, suppose that $\mathsf{Hom}_{\mathcal{G}}(G) = \mathsf{Hom}_{\mathcal{G}}(H)$, that is, for all graphs F it holds that $\mathsf{hom}(F, G) = \mathsf{hom}(F, H)$.

We can decompose every homomorphism $h : F \to F'$ as $h = f \circ g$ such that for some graph F'':

- $g : F \to F''$ is an *epimorphism*, that is, a homomorphism such that for every $v \in V(F'')$ there is a $u \in V(F)$ with $g(u) = v$ and for every $vv' \in E(F'')$ there is a $uu' \in E(F)$ with $g(u) = v$ and $g(u') = v'$;
- $f : F'' \to F'$ is an *embedding* (or *monomorphism*), that is, a homomorphism such that $f(u) \neq f(u')$ for all $u \neq u'$.

Note that the graph F'' is isomorphic to the image $h(F)$ and thus unique up to isomorphism. Moreover, there are precisely

$$\mathsf{aut}(F'') = \text{number of automorphisms of } F''$$

isomorphisms from F'' to $h(F)$. This means that there are $\mathsf{aut}(F'')$ pairs (f, g) such that $h = f \circ g$ and $g : F \to F''$ is an epimorphism and $f : F'' \to F'$ is an embedding. Thus we can write

$$\mathsf{hom}(F, F') = \sum_{F''} \frac{1}{\mathsf{aut}(F'')} \cdot \mathsf{epi}(F, F'') \cdot \mathsf{emb}(F'', F'), \tag{4.2}$$

where $\mathsf{epi}(F, F'')$ is the number of epimorphisms from F onto F'', $\mathsf{emb}(F'', F')$ is the number of embeddings of F'' into F', and the sum ranges over all isomorphism types of graphs F''. Furthermore, as $\mathsf{epi}(F, F'') = 0$ if $|F''| > |F|$, we can restrict the sum to F'' of order $|F''| \leq |F|$.

Let $F_1, \ldots, F_m$ be an enumeration of all graphs of order at most $n := \max\{|G|, |H|\}$ such that each graph of order at most n is isomorphic to exactly one graph in this list and that $i \leq j$ implies $|F_i| < |F_j|$ or $|F_i| = |F_j|$ and $\|F_i\| := |E(F_i)| \leq \|F_j\|$. Let HOM be the matrix with entries $HOM_{ij} := \mathsf{hom}(F_i, F_j)$, P the matrix with entries $P_{ij} := \mathsf{epi}(F_i, F_j)$, M the matrix with entries $M_{ij} := \mathsf{emb}(F_i, F_j)$, and D the diagonal matrix with entries $D_{ii} := \frac{1}{\mathsf{aut}(F_i)}$. Then (4.2) yields the following matrix equation;

$$HOM = P \cdot D \cdot M. \tag{4.3}$$

The crucial observation is that P is an lower triangular matrix, because $\mathsf{epi}(F_i, F_j) > 0$ implies $|F_i| \geq |F_j|$ and $\|F_i\| \geq \|F_j\|$. Moreover, P has positive diagonal entries, because $\mathsf{epi}(F_i, F_i) \geq 1$. Similarly, M is an upper triangular matrix with positive diagonal entries. Thus P and M are invertible. The diagonal matrix D is invertible as well, because $D_{ii} = \mathsf{aut}(F_i) \geq 1$. Thus the matrix HOM is invertible.

As $|G|, |H| \leq n$ there are graphs F_i, F_j such that G is isomorphic to F_i and H is isomorphic to F_j. Since $\mathsf{hom}(F_k, G) = \mathsf{hom}(F_k, H)$ for all k by the assumption of the lemma, the ith and jth column of HOM are identical. As HOM is invertible, this implies that $i = j$ and thus that G and H are isomorphic. □

The theorem can be seen as a the starting point for the theory of graph limits [19, 67, 68]. The graph embedding $\mathsf{Hom}_{\mathcal{G}}$ maps graphs into an infinite dimensional real vector space, which can be turned into a Hilbert space by defining a suitable inner product. This transformation enables us to analyse graphs with methods of linear algebra and functional analysis and, for example, to consider convergent sequences of graphs and their limits, called *graphons* (see [67]).

Figure 6: Co-spectral graphs

However, not only the "full" homomorphism vector $\mathsf{Hom}_{\mathcal{G}}(G)$ of a graph G, but also its projections $\mathsf{Hom}_{\mathcal{F}}(G)$ to natural classes $\mathcal{F}$ capture very interesting information about G. A first result worth mentioning is the following. This result is well-known, though usually phrased differently. Two graphs are *co-spectral* if their adjacency matrices have the same eigenvalues with the same multiplicities. Figure 6 shows two graph that are co-spectral, but not isomorphic.

Theorem 4.3 (Folklore). *For all graphs G and H,*

$$\mathsf{Hom}_{\mathcal{C}}(G) = \mathsf{Hom}_{\mathcal{C}}(H) \iff G \text{ and } H \text{ are co-spectral.}$$

Here $\mathcal{C}$ denotes the class of all cycles.

Proof sketch. Observe that for the cycle C_k of length k we have $\mathsf{hom}(C_k, G) := \mathsf{trace}(A^k)$, where A is the adjacency matrix of G. It is well-known that the trace of a symmetric real matrix is the sum of its eigenvalues and that the eigenvalues of A^k are the kth powers of the eigenvalues of A. This immediately implies the backward direction. By a simple linear-algebraic argument, it also implies the forward direction. □

Dvorák [33] proved that homomorphism counts of trees, and more generally, graphs of bounded tree width link homomorphism vectors to the Weisfeiler-Leman algorithm.

Theorem 4.4 ([33]). *For all graphs G and H and all $k \geq 1$,*

$$\mathsf{Hom}_{\mathcal{T}_k}(G) = \mathsf{Hom}_{\mathcal{T}_k}(H) \iff k\text{-WL does not distinguish } G \text{ and } H.$$

Here $\mathcal{T}_k$ denotes the class of all graphs of tree width at most k.

To prove the theorem, it suffices to consider connected graphs in $\mathcal{T}_k$, because for a graph F with connected components $F_1, \ldots, F_m$ we have $\mathsf{hom}(F, G) = \prod_{i=1}^{m} \mathsf{hom}(F_i, G)$. The connected graphs of tree width 1 are the trees. We sketch a proof of the theorem for trees in Section 4.4.

In combination with Theorem 3.2, Theorem 4.4 implies the following.

Corollary 4.5. *For all graphs G and H,*

$$\mathsf{Hom}_{\mathcal{T}}(G) = \mathsf{Hom}_{\mathcal{T}}(H) \iff G \text{ and } H \text{ are fractionally isomorphic, that is, equations (3.2) and (3.3) have a nonnegative rational solution.}$$

Here $\mathcal{T}$ denotes the class of all trees.

Remarkably, for paths we obtain a similar characterisation that involves the same equations, but drops the nonnegativity constraint.

Theorem 4.6 ([32]). *For all graphs G and H,*

$$\mathsf{Hom}_{\mathcal{P}}(G) = \mathsf{Hom}_{\mathcal{P}}(H) \iff \text{equations (3.2) and (3.3) have a rational solution.}$$

Here $\mathcal{P}$ denotes the class of all paths.

Figure 7: Two graphs that are homomorphism-indistinguishable over the class of paths

While the proof of Theorem 4.4 relies on techniques similar to the proof of Theorem 4.2, the proof of Theorem 4.3 is based on spectral techniques similar to the proof of Theorem 4.3.

Example 4.7. For the co-spectral graphs G, H shown in Figure 6 we have

$$\hom\left(\,\wedge\,, G\right) = 20, \qquad \hom\left(\,\wedge\,, H\right) = 16.$$

Thus $\mathsf{Hom}_{\mathcal{P}}(G) \neq \mathsf{Hom}_{\mathcal{P}}(H)$.

Example 4.8. Figure 7 shows graphs G, H with

$$\mathsf{Hom}_{\mathcal{P}}(G) = \mathsf{Hom}_{\mathcal{P}}(H).$$

Obviously, 1-WL distinguishes the two graphs. Thus $\mathsf{Hom}_{\mathcal{T}}(G) \neq \mathsf{Hom}_{\mathcal{T}}(H)$. It can also be checked that the graphs are not co-spectral. Hence $\mathsf{Hom}_{\mathcal{C}}(G) \neq \mathsf{Hom}_{\mathcal{C}}(H)$

Combined with Theorem 3.1, Theorem 4.4 implies the following correspondence between homomorphism counts of graphs of bounded tree width and the finite variable fragments of the counting logic C introduced in Section 3.4.

Corollary 4.9. *For all graphs G and H and all $k \geq 1$,*

$$\mathsf{Hom}_{\mathcal{T}_k}(G) = \mathsf{Hom}_{\mathcal{T}_k}(H) \iff G \text{ and } H \text{ are } \mathsf{C}^{k+1}\text{-equivalent.}$$

This is interesting, because it shows that the homomorphism vector $\mathsf{Hom}_{\mathcal{T}_k}(G)$ gives us all the information necessary to answer queries expressed in the logic C^{k+1}. Unfortunately, the result does not tell us how to answer C^{k+1}-queries algorithmically if we have access to the vector $\mathsf{Hom}_{\mathcal{T}_k}(G)$. To make the question precise, suppose we have oracle access that allows us to obtain, for every graph $F \in \mathcal{T}_k$, the entry $\hom(F, G)$ of the homomorphism vector. Is it possible to answer a C^{k+1}-query in polynomial time (either with respect to data complexity or combined complexity)?

Arguably, from a logical perspective it is even more natural to restrict the *quantifier rank* (maximum number of nested quantifiers) in a formula rather than the number of variables. Let C_k be the fragment of C consisting of all formulas of quantifier rank at most k. We obtain the following characterisation of C_k-equivalence in terms of homomorphism vectors over the class of graphs of *tree depth* at most k. Tree depth, introduced by Nešetřil and Ossona de Mendez [81], is another structural graph parameter that has received a lot of attention in recent years (e.g. [9, 23, 28, 34, 35]).

Theorem 4.10 ([42]). *For all graphs G and H and all $k \geq 1$,*

$$\mathsf{Hom}_{\mathcal{TD}_k}(G) = \mathsf{Hom}_{\mathcal{TD}_k}(H) \iff G \text{ and } H \text{ are } \mathsf{C}_k\text{-equivalent.}$$

Here $\mathcal{TD}_k$ denotes the class of all graphs of tree depth at most k.

Another very remarkable result, due to Mančinska and Roberson [72], states that two graphs are homomorphism-indistinguishable over the class of all planar graphs if and only if they are *quantum isomorphic*. Quantum isomorphism, introduced in [4], is a complicated notion that is based on similar systems of equations as (3.2) and (3.3) and their generalisation characterising higher-dimensional Weisfeiler-Leman indistinguishability, but with non-commutative variables ranging over the elements of a C^*-algebra.

4.2 Beyond Undirected Graphs

So far, we have only considered homomorphism indistinguishability on undirected graphs. A few results are known for directed graphs. In particular, Theorem 4.2 directly extends to directed graphs. Actually, we have the following stronger result, also due to Lovász [66] (also see [14]).

Theorem 4.11 ([66]). *For all directed graphs G and H,*

$$\mathsf{Hom}_{\mathcal{DA}}(G) = \mathsf{Hom}_{\mathcal{DA}}(G) \iff G \text{ and } H \text{ are isomorphic.}$$

Here $\mathcal{DA}$ denote the class of all directed acyclic graphs.

It is straightforward to extend Theorem 4.2, Theorem 4.4 (for the natural generalisation of the WL algorithms to relational structures), and Theorem 4.10 to arbitrary relational structures. This is very useful for binary relational structures such as knowledge graphs. But for relations of higher arity one may consider another version based on the incidence graph of a structure.

Let $\sigma = \{R_1, \dots, R_m\}$ be a relational vocabulary, where R_i is a k_i-ary relation symbol. Let k be the maximum of the k_i. We let $\sigma_I := \{E_1, \dots, E_k, P_1, \dots, P_m\}$, where the E_i are binary and the P_j are unary relation symbols. With every σ-structure $A = (V(A), R_1(A), \dots, R_m(A))$ we associates a σ_I-structure A_I called the *incidence structure* of A as follows:

- the universe of A_I is

$$V(A_I) := V(A) \cup \bigcup_{i=1}^{m} \big\{(R_i, v_1, \dots, v_{k_i} \mid (v_1, \dots, v_{k_i}) \in R_i(A)\big\};$$

- for $1 \leq j \leq k$, the relation $E_j(A_I)$ consists of all pairs

$$\big(v_j, (R_i, v_1, \dots, v_{k_i})\big)$$

for $(R_i, v_1, \dots, v_{k_i}) \in V(A_I)$ with $k_i \geq j$;

- for $1 \leq i \leq m$, the relation P_i consist of all $(R_i, v_1, \dots, v_{k_i}) \in V(A_I)$.

With this encoding of general structures as binary incidence structures we obtain the following corollary.

Corollary 4.12. *For all σ-structures A and B, the following are equivalent.*

(1) $\mathsf{Hom}_{\mathcal{T}(\sigma_I)}(A_I) = \mathsf{Hom}_{\mathcal{T}(\sigma_I)}(B_I)$, where $\mathcal{T}(\sigma_I)$ denotes the class of all σ_I-structures whose underlying (Gaifman) graph is a tree;
(2) A_I and B_I are not distinguished by 1-WL;
(3) A_I and B_I are C^2-equivalent.

Böker [14] gave a generalisation of Theorem 4.4 to hypergraphs that is also based on incidence graphs.

In a different direction, we can generalise the results to weighted graphs. Let us consider undirected graphs with real-valued edge weights. We can also view them as symmetric matrices over the

reals. Recall that we denote the weight of an edge uv by $\alpha(u,v)$ and that weighted 1-WL refines by sums of edge weights (instead of numbers of edges). Let F be an unweighted graph and G a weighted graph. For every mapping $h : V(F) \to V(G)$ we let

$$\mathrm{wt}(h) := \prod_{uu' \in E(F)} \alpha(h(u), h(u')).$$

As $\alpha(v,v') = 0$ if and only if $vv' \notin E(G)$, we have $\mathrm{wt}(h) \neq 0$ if and only if h is a homomorphism from F to G; the weight of this homomorphism is the product of the weights of the edges in its image. We let

$$\mathsf{hom}(F,G) := \sum_{h:V(F)\to V(G)} \mathrm{wt}(h).$$

In statistical physics, such sum-product functions are known as *partition functions*. For a class $\mathcal{F}$ of graphs, we let

$$\mathsf{Hom}_{\mathcal{F}}(G) := \big(\mathsf{hom}(F,G) \;\big|\; F \in \mathcal{F}\big).$$

Theorem 4.13 ([22]). *For all weighted graphs G and H, the following are equivalent.*

(1) $\mathsf{Hom}_{\mathcal{T}}(G) = \mathsf{Hom}_{\mathcal{T}}(H)$ *(recall that $\mathcal{T}$ denotes the class of all trees);*
(2) G and H are not distinguished by weighted 1-WL;
(3) equations (3.2) and (3.3) have a nonnegative rational solution.

4.3 Complexity

In general, counting the number of homomorphisms from a graph F to a graph G is a #P-hard problem. Dalmau and Jonsson [31] proved that, under the reasonable complexity theoretic assumption $\#\mathsf{W}[1] \neq \mathsf{FPT}$ from parameterised complexity theory, for all classes $\mathcal{F}$ of graphs, computing $\mathsf{hom}(F,G)$ given a graph $F \in \mathcal{F}$ and an arbitrary graph G, is in polynomial time if and only if $\mathcal{F}$ has bounded tree width. This makes Theorem 4.4 even more interesting, because the entries of a homomorphism vector $\mathsf{Hom}_{\mathcal{F}}(G)$ are computable in polynomial time precisely for bounded tree width classes.

However, the computational problem we are facing is not to compute individual entries of a homomorphism vectors, but to decide if two graphs have the same homomorphism vector, that is, if they are homomorphism indistinguishable. The characterisation theorems of Section 4.1 imply that homomorphism indistinguishability is polynomial-time decidable over the classes $\mathcal{P}$ of paths, $\mathcal{C}$ of cycles, $\mathcal{T}$ of trees, $\mathcal{T}_k$ of graphs of tree width at most k, $\mathcal{TD}_k$ of tree depth at most k. Moreover, homomorphism indistinguishability over the class of $\mathcal{G}$ of all graphs is decidable in quasi-polynomial time by Babai's [7] celebrated result that graph isomorphism is decidable in quasi-polynomial time. It was proved in [15] that homomorphism indistinguishability over the class of complete graphs is complete for the complexity class $\mathsf{C}_=\mathsf{P}$, which implies that it is co-NP hard, and that there is a polynomial time decidable class $\mathcal{F}$ of graphs of bounded tree width such that homomorphism indistinguishability over $\mathcal{F}$ is undecidable. Quite surprisingly, the fact that quantum isomorphism is undecidable [4] implies that homomorphism distinguishability over the class of planar graphs is undecidable.

4.4 Homomorphism Node Embeddings

So far, we have defined graph and structure embeddings based on homomorphism vectors. But we can also use homomorphism vectors to define node embeddings. A *rooted graph* is a pair (G,v) where G is a graph and $v \in V(G)$. For two rooted graphs (F,u) and (G,v), by $\mathsf{hom}(F,G;u \mapsto v)$ we denote the number of homomorphism h from F to G with $h(u) = v$. For a class $\mathcal{F}^*$ of rooted graphs and a rooted graph (G,v), we let

$$\mathsf{Hom}_{\mathcal{F}^*}(G,v) := \big(\mathsf{hom}(F,G;u \mapsto v) \;\big|\; (F,u) \in \mathcal{F}^*\big).$$

If we keep the graph G fixed, this gives us an embedding of the nodes of G into an infinite dimensional vector space. Note that in the terminology of Section 2.1, this embedding is "inductive" and not "transductive", because it is not tied to a fixed graph. (Nevertheless, the term "inductive" is not fitting very well here, because the embedding is not learned.) In the same way we defined graph kernels based on homomorphism vectors of graphs, we can now define node kernels.

It is straightforward to generalise Theorem 4.2 to rooted graphs, showing that for all rooted graphs (G,v) and (H,w) it holds that

$$\mathsf{Hom}_{\mathcal{G}^*}(G,v) = \mathsf{Hom}_{\mathcal{G}^*}(H,w) \iff \text{there is an isopmorphism } f \text{ from } G \text{ to } H \text{ with } f(v) = w.$$

Here $\mathcal{G}^*$ denote the class of all rooted graphs. Maybe the easiest way to prove this is by a reduction to node-labelled graphs.

Another key result of Section 4.1 that can be adapted to the node setting is Theorem 4.4. We only state the version for trees.

Theorem 4.14. *For all graphs G, H and all $v \in V(G), w \in V(H)$, the following are equivalent.*

(1) $\mathsf{Hom}_{\mathcal{T}^*}(G,v) = \mathsf{Hom}_{\mathcal{T}^*}(H,w)$ *for the class $\mathcal{T}^*$ of all rooted trees;*
(2) 1-WL assigns the same colour to v and w.

This result is implicit in the proof of Theorem 4.4 (see [32, 33]). In fact, it can be viewed as the graph theoretic core of the proof. We sketch the proof here and also show how to derive Theorem 4.4 (for trees) from it.

Proof sketch. Recall from Section 3.5 that we can view the colours assigned by 1-WL as rooted trees (see Figure 5). For the kth round of WL, this is a tree of height k, and for the stable colouring we can view it as an infinite tree. Suppose now that the colour of a vertex v of G is T. The crucial observation is that for every rooted tree (S,r) the number $\mathsf{hom}(S,G;r \mapsto v)$ is precisely the number of mappings $h : V(S) \to V(T)$ that map the root r of S to the root of T and, for each node $s \in V(S)$, map the children of s to the children of $h(s)$ in T. Let us call such mappings *rooted tree homomorphisms*.

Implication (2) $\implies$ (1) follows directly from this observation. Implication (1) $\implies$ (2) follows as well, because by an argument similar to that used in the proof of Theorem 4.2 it can be shown that for distinct rooted trees T, T' there is a rooted tree that has distinct numbers of rooted tree homomorphisms to T, T'. □

Corollary 4.15. *For all graphs G, H and all $v \in V(G), w \in V(H)$, the following are equivalent.*

(1) $\mathsf{Hom}_{\mathcal{T}^*}(G,v) = \mathsf{Hom}_{\mathcal{T}^*}(H,w)$*;*
(2) for all formulas $\varphi(x)$ of the logic C^2,

$$G \models \varphi(v) \iff H \models \varphi(w).$$

Note that the node embeddings based on homomorphism vectors are quite different from the node embeddings described in

Section 2.1. They are solely based on structural properties and ignore the distance information. Results like Corollary 4.15 show that the structural information captured by the homomorphism-based embeddings in principle enables us to answer queries directly on the embedding, which may be more useful than distance information in database applications.

We close this section by sketching how Theorem 4.4 (for trees) follows from Theorem 4.14.

Proof sketch of Theorem 4.4 (for trees). Let G, H be graphs. We need to prove

$$\mathsf{Hom}_{\mathcal{T}}(G) = \mathsf{Hom}_{\mathcal{T}}(H) \iff \text{1-WL does not distinguish } G \text{ and } H. \tag{4.4}$$

Without loss of generality we assume that $V(G) \cap V(H) = \emptyset$. For $x, y \in V(G) \cup V(H)$, we write $x \sim y$ if 1-WL assigns the same colour to x and y. By Theorem 4.14, for $X, Y \in \{G, H\}$ and $x \in V(X)$, $y \in V(Y)$ we have $x \sim y$ if and only if $\mathsf{Hom}_{\mathcal{T}^*}(X, x) = \mathsf{Hom}_{\mathcal{T}^*}(X, y)$. Let $R_1, \ldots, R_n$ be the $\sim$-equivalence classes, and for every j, let $P_j := R_j \cap V(G)$ and $Q_j := R_j \cap V(H)$. Furthermore, let $p_j := |P_j|$ and $q_j := |Q_j|$.

We first prove the backward direction of (4.4). Assume that 1-WL does not distinguish G and H. Then $p_j = q_j$ for all $j \in [n]$. Let T be a tree, and let $t \in V(T)$. Let $h_j := \mathsf{hom}(T, X; t \mapsto x)$ for $x \in R_j$ and $X \in \{G, H\}$ with $x \in V(X)$. Then

$$\begin{aligned}\mathsf{hom}(T, G) &= \sum_{v \in V(G)} \mathsf{hom}(T, G; t \mapsto v) \\ &= \sum_{j=1}^{n} p_j h_j = \sum_{j=1}^{n} q_j h_j \\ &= \sum_{w \in V(H)} \mathsf{hom}(T, H; t \mapsto w) = \mathsf{hom}(T, H).\end{aligned}$$

Since T was arbitrary, this proves $\mathsf{Hom}_{\mathcal{T}}(G) = \mathsf{Hom}_{\mathcal{T}}(H)$.

The proof of the forward direction of (4.4) is more complicated. Assume $\mathsf{Hom}_{\mathcal{T}}(G) = \mathsf{Hom}_{\mathcal{T}}(H)$. There is a finite collection of $m \leq \binom{n}{2}$ rooted trees $(T_1, r_1), \ldots, (T_m, r_m)$ such that for all $X, Y \in \{G, H\}$ and $x \in V(X)$, $y \in V(Y)$ we have $x \sim y$ if and only if for all $i \in [m]$,

$$\mathsf{hom}(T_i, X; r_i \mapsto x) = \mathsf{hom}(T_i, Y; r_i \mapsto y).$$

Let $a_{ij} := \mathsf{hom}(T_i, X; r_i \mapsto x)$ for $x \in R_j$ and $X \in \{G, H\}$ with $x \in V(X)$. Then for all i we have

$$\sum_{j=1}^{n} a_{ij} p_j = \mathsf{hom}(T_i, G) = \mathsf{hom}(T_i, H) = \sum_{j=1}^{n} a_{ij} q_j$$

Unfortunately, the matrix $A = (a_{ij})_{i \in [m], j \in [n]}$ is not necessarily invertible, so we cannot directly conclude that $p_j = q_j$ for all j. All we know is that for any two columns of the matrix there is a row such that the two columns have distinct values in that row. It turns out that this is sufficient. For every vector $\boldsymbol{d} = (d_1, \ldots, d_m)$ of nonnegative integers, let $(T^{(\boldsymbol{d})}, r^{(\boldsymbol{d})})$ be the rooted tree obtained by taking the disjoint union of d_i copies of T_i for all i and then identifying the roots of all these trees. It is easy to see that

$$\mathsf{hom}(T^{(\boldsymbol{d})}, X; r^{(\boldsymbol{d})} \mapsto x) = \prod_{i=1}^{m} \mathsf{hom}(T_i, X; r_i \mapsto x).$$

Thus, letting $a_j^{(\boldsymbol{d})} := \prod_{i=1}^{m} a_{ij}^{d_i}$, we have

$$\sum_{j=1}^{n} a_j^{(\boldsymbol{d})} p_j = \mathsf{hom}(T^{(d)}, G) = \mathsf{hom}(T^{(d)}, H) = \sum_{j=1}^{n} a_j^{(d)} q_j.$$

Using these additional equations, it can be shown that $p_j = q_j$ for all j (see [43, Lemma 4.2]). Thus WL does not distinguish G and H. □

4.5 Homomorphisms and GNNs

We have a correspondence between homomorphism vectors and and the Weisfeiler-Leman algorithm (Theorems 4.4 and 4.14) and between the the WL algorithm and GNNs (see Section 3.6). This also establishes a correspondence between homomorphism vectors and GNNs. More directly, the correspondence between GNNs and homomorphism counts is also studied in [69].

5 SIMILARITY

The results described in the previous section can be interpreted as results on the expressiveness of homomorphism-based embeddings of structures and their nodes. However, all these results only show what it means that two objects are mapped to the same homomorphism vector. More interesting is the similarity measure the vector embeddings induce via some an inner product/norm on the latent space (see (4.1)). We can speculate that, given the nice results regarding equality of vectors, the similarity measure will have similarly nice properties. Let me propose the following, admittedly vague, hypothesis.

> *For suitable classes $\mathcal{F}$, the homomorphism embedding $\mathsf{Hom}_{\mathcal{F}}$ combined with a suitable inner product on the latent space induces a natural similarity measure on graphs or relational structures.*

From a practical perspective, we could support this hypothesis by showing that the vector embeddings give good results when combined with similarity based downstream tasks. As mentioned earlier, initial experiments show that homomorphism vectors in combination with support vector machines perform well on standard graph classification benchmarks. But a more thorough experimental study will be required to have conclusive results.

From a theoretical perspective, we can compare the homomorphism-based similarity measures with other similarity measures for graphs and discrete structures. If we can prove that they coincide or are close to each other, then this would support our hypothesis.

5.1 Similarity from Matrix Norms

A standard way of defining similarity measures on graphs is based on comparing their adjacency matrices. Let us briefly review a few matrix norms. Recall the standard ℓ_p-vector norm $\|\boldsymbol{x}\|_p := \left(\sum_i |x_i|^p\right)^{1/p}$. [1] The two best-known matrix norms are the *Frobenius norm* $\|M\|_F := \sqrt{\sum_{i,j} M_{ij}^2}$ and the *spectral norm* $\|M\|_{\langle 2 \rangle} :=$

[1] Note that $\|\boldsymbol{x}\|_2$ is just the Euclidean norm, which we denoted by $\|\boldsymbol{x}\|$ earlier in this paper.

$\sup_{\boldsymbol{x}\in\mathbb{R}^n, \|\boldsymbol{x}\|_2=1} \|M\boldsymbol{x}\|_2$. More generally, for every $p > 0$ we define

$$\|M\|_p := \left(\sum_{i,j} |M_{ij}|^p\right)^{1/p}$$

(so $\|M\|_F = \|M\|_2$) and

$$\|M\|_{\langle p\rangle} := \sup_{\boldsymbol{x}\in\mathbb{R}^n, \|\boldsymbol{x}\|_p=1} \|M\boldsymbol{x}\|_p, .$$

Thus both $\|M\|_p$ and $\|M\|_{\langle p\rangle}$ are derived from the ℓ_p-vector norm. For $\|M\|_p$, the matrix is simply flattened into a vector, and for $\|M\|_{\langle p\rangle}$ the matrix is viewed as a linear operator. Another matrix norm that is interesting here is the *cut norm* defined by

$$\|M\|_\square := \max_{S,T} \left| \sum_{i\in S, j\in T} M_{ij} \right|,$$

where S, T range over all subsets of the index set of the matrix. Observe that for $M \in \mathbb{R}^{n\times n}$ we have

$$\|M\|_\square \le \|M\|_1 \le n\|M\|_F,$$

where the second inequality follows from the Cauchy-Schwarz inequality. If we compare matrices of different size, it can be reasonable to scale the norms by a factor depending on n.

For technical reasons, we only consider matrix norms $\|\cdot\|$ that are invariant under permutations of the rows and columns, that is,

$$\|M\| = \|MP\| = \|QM\| \quad \text{for all permutation matrices } P, Q. \tag{5.1}$$

It is easy to see that the norms discussed above have this property.

Now let G, H be graphs with vertex sets V, W and adjacency matrices $A \in \mathbb{R}^{V\times V}, B \in \mathbb{R}^{W\times W}$. For convenience, let us assume that $|G| = |H| =: n$. Then both A, B are $n \times n$-matrices, and we can compare them using a matrix norm. However, it does not make much sense to just consider $\|A - B\|$, because graphs do not have a unique adjacency matrix, and even if G and H are isomorphic, $\|A - B\|$ may be large. Therefore, we align the two matrices in an optimal way by permuting the rows and columns of A. For a matrix norm $\|\cdot\|$, we define a graph distance measure $\mathrm{dist}_{\|\cdot\|}$

$$\mathrm{dist}_{\|\cdot\|}(G, H) := \min_{\substack{P\in\{0,1\}^{V\times W}\\ \text{permutation matrix}}} \|P^\top AP - B\|.$$

It follows from (5.1) that $\mathrm{dist}_{\|\cdot\|}$ is well-defined, that is, does not depend on the choice of the particular adjacency matrices A, B. It also follows from (5.1) and that fact that $P^{-1} = P^\top$ for permutation matrices that

$$\mathrm{dist}_{\|\cdot\|}(G, H) = \min_{\substack{P\in\{0,1\}^{V\times W}\\ \text{permutation matrix}}} \|AP - PB\|, \tag{5.2}$$

which is often easier to work with because the expression $AP - PB$ is linear in the "variables" P_{ij}. To simplify the notation, we let $\mathrm{dist}_p := \mathrm{dist}_{\|\cdot\|_p}$ and $\mathrm{dist}_{\langle p\rangle} := \mathrm{dist}_{\|\cdot\|_{\langle p\rangle}}$ for all p, and we let $\mathrm{dist}_\square := \mathrm{dist}_{\|\cdot\|_\square}$.

The distances defined from the ℓ_1-norm have natural interpretations as edit distances. $\mathrm{dist}_1(G, H)$ is twice the number of edges that need to be flipped to turn G into a graph isomorphic to H, and $\mathrm{dist}_{\langle 1\rangle}(G, H)$ is the maximum number of edges incident with a single vertex we need to flip to turn G into a graph isomorphic to H. Formally,

$$\mathrm{dist}_1(G, H) = 2 \min_{\substack{f:V\to W\\ \text{bijection}}} \left|\{f(v)f(v') \mid vv' \in E(G)\} \triangle E(H)\right|. \tag{5.3}$$

$$\mathrm{dist}_{\langle 1\rangle}(G, H) = \min_{\substack{f:V\to W\\ \text{bijection}}} \max_{v\in V} \left|\{f(v') \mid v' \in N_G(v)\} \triangle \{w' \mid w \in N_H(f(v))\}\right|. \tag{5.4}$$

Here $\triangle$ denotes the symmetric difference. Equation (5.3) is obvious; the factor '2' comes from the fact that we regard (undirected) edges as 2-element sets and not as ordered pairs. To prove (5.4), we observe that for every matrix $M \in \mathbb{R}^{n\times n}$ it holds that $\|M\|_{\langle 1\rangle} = \max_{j\in[n]} \sum_{i=1}^n |M_{ij}|$.

Despite these intuitive interpretations, it is debatable how much "semantic relevance" these distance measures have. How similar are two graphs that can be transformed into each other by flipping, say, 5% of the edges? Again, the answer to this question may depend on the application context.

A big disadvantage the graph distance measures based on matrix norms have is that computationally they are highly intractable (see, for example, [3] and the references therein). It is even NP-hard to compute the distance between two trees (see [46] for Frobenius distance and [38] for the distances based on operator norms), and distances are hard to approximate. The problem of computing these distances is related to the maximisation version of the quadratic assignment problem (see [70, 79]), a notoriously hard combinatorial optimisation problem. Better behaved is the cut-distance $\mathrm{dist}_\square$; at least it can be approximated within a factor of 2 [2].

The main source of hardness is the minimisation over the unwieldy set of all permutations (or permutation matrices). To alleviate this hardness, we can relax the integrality constraints and minimise over the convex set of all doubly stochastic matrices instead. That is, we define a relaxed distance measure

$$\widetilde{\mathrm{dist}}_{\|\cdot\|}(G, H) = \min_{\substack{X\in\{0,1\}^{V\times W}\\ \text{doubly stochastic}}} \|AX - XB\|. \tag{5.5}$$

Note that $\widetilde{\mathrm{dist}}_{\|\cdot\|}$ is only a pseudo-metric: the distance between non-isomorphic graphs may be 0. Indeed, it follows from Theorem 3.2 that $\widetilde{\mathrm{dist}}_{\|\cdot\|}(G, H) = 0$ if and only if G and H are fractionally isomorphic. The advantage of these "relaxed" distances is that for many norms $\|\cdot\|$, computing $\widetilde{\mathrm{dist}}_{\|\cdot\|}$ is a convex minimisation problem that can be solved efficiently.

So far, we have only discussed distance measures for graphs of the same order. To extend these distance measures to arbitrary graphs, we can replace vertices by sets of identical vertices in both graphs to obtain two graphs whose order is the least common multiple of the orders of the two initial graphs (see [67, Section 8.1] for details).

Note that these matrix based similarity measures are only defined for (possibly weighted) graphs. In particular for the operator norms, it is not clear how to generalise them to relational structures, and if such a generalisation would even be meaningful.

5.2 Comparing Homomorphism Distances and Matrix Distances

It would be very nice if we could establish a connection between graph distance measures based on homomorphism vectors and those based on matrix norms. At least one important result in this direction exists: Lovász [67] proves an equivalence between the cut-distance of graphs and a distance measure derived from a suitably scaled homomorphism vector $\mathsf{Hom}_{\mathcal{G}}$.

It is tempting to ask if a similar correspondence can be established between $\widetilde{\mathrm{dist}}_{\|\cdot\|}$ and $\mathsf{Hom}_{\mathcal{F}}$. There are many related question that deserve further attention.

6 CONCLUDING REMARKS

In this paper, we gave an overview of embeddings techniques for graphs and relational structures. Then we discussed two related theoretical approaches, the Weisfeiler-Leman algorithm with its various ramifications and homomorphism vectors. We saw that they have a rich and beautiful theory that leads to new, generic families of vector embeddings and helps us to get a better understanding of some of the techniques used in practice, for example graph neural networks.

Yet we have also seen that we are only at the beginning and many questions remain open, in particular when it comes to similarity measures defined on graphs and relational structures.

From a database perspective, it will be important to generalise the embedding techniques to relations of higher arities, which is not as trivial as it may seem (and where surprisingly little has been done so far). A central question is then how to query the embedded data. Which queries can we answer at all when we only see the vectors in latent space? How do imprecisions and variations due to randomness affect the outcome of such query answers? Probably, we can only answer queries approximately, but what exactly is the semantics of such approximations? These are just a few questions that need to be answered, and I believe they offer very exciting research opportunities for both theoreticians and practitioners.

Acknowledgements

This paper was written in strange times during COVID-19 lockdown. I appreciate it that some of my colleagues nevertheless took the time to answer various questions I had on the topics covered here and to give valuable feedback on an earlier version of this paper. In particular, I would like to thank Pablo Barceló, Neta Friedman, Benny Kimelfeld, Christopher Morris, Petra Mutzel, Martin Ritzert, and Yufei Tao.

REFERENCES

[1] A. Ahmed, N. Shervashidze, S. Narayanamurthy, V. Josifovski, and A.J. Smola. 2013. Distributed large-scale natural graph factorization. In *Proceedings of the 22nd International World Wide Web Conference*. 37–48.

[2] N. Alon and A. Naor. 2006. Approximating the Cut-Norm via Grothendieck's Inequality. *SIAM J. Comput.* 35 (2006), 787–803.

[3] V. Arvind, J. Köbler, S. Kuhnert, and Y. Vasudev. 2012. Approximate Graph Isomorphism.. In *Proceedings of the 37th International Symposium on Mathematical Foundations of Computer Science (Lecture Notes in Computer Science)*, B. Rovan, V. Sassone, and P. Widmayer (Eds.), Vol. 7464. Springer Verlag, 100–111.

[4] A. Atserias, Laura Mančinska, D.E. Roberson, R. Šámal, S. Severini, and A. Varvitsiotis. 2019. Quantum and non-signalling graph isomorphisms. *Journal of Combinatorial Theory, Series B* 136 (2019), 289–328.

[5] A. Atserias and E. Maneva. 2013. Sherali–Adams Relaxations and Indistinguishability in Counting Logics. *SIAM J. Comput.* 42, 1 (2013), 112–137.

[6] A. Atserias and J. Ochremiak. 2018. Definable Ellipsoid Method, Sums-of-Squares Proofs, and the Isomorphism Problem. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*. 66–75.

[7] L. Babai. 2016. Graph Isomorphism in Quasipolynomial Time. In *Proceedings of the 48th Annual ACM Symposium on Theory of Computing (STOC '16)*. 684–697.

[8] L. Babai, P. Erdös, and S. Selkow. 1980. Random graph isomorphism. *SIAM J. Comput.* 9 (1980), 628–635.

[9] M. Bannach and T. Tantau. 2016. Parallel Multivariate Meta-Theorems. In *Proceedings of the 11th International Symposium on Parameterized and Exact Computation (LIPIcs)*, J. Guo and D. Hermelin (Eds.), Vol. 63. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 4:1–4:17.

[10] P. Barceló, E.V. Kostylev, M. Monet, J. Pérez, J. Reutter, and J.P. Silva. 2020. The Logical Expressiveness of Graph Neural Networks. In *Proceedings of the 8th International Conference on Learning Representations*. https://openreview.net/forum?id=r1lZ7AEKvB

[11] M. Belkin and P. Niyogi. 2003. Laplacian Eigenmaps for Dimensionality Reduction and Data Representation. *Neural Computation* 15, 6 (2003), 1373–1396.

[12] C. Berkholz, P. Bonsma, and M. Grohe. 2017. Tight Lower and Upper Bounds for the Complexity of Canonical Colour Refinement. *Theory of Computing Systems* 60, 4 (2017), 581–614.

[13] C. Berkholz and M. Grohe. 2015. Limitations of Algebraic Approaches to Graph Isomorphism Testing. In *Proceedings of the 42nd International Colloquium on Automata, Languages and Programming, Part I (Lecture Notes in Computer Science)*, M.M. Halldórsson, K. Iwama, N. Kobayashi, and B. Speckmann (Eds.), Vol. 9134. Springer Verlag, 155–166.

[14] J. Böker. 2019. Color Refinement, Homomorphisms, and Hypergraphs. In *Proceedings of the 45th International Workshop on Graph-Theoretic Concepts in Computer Science (Lecture Notes in Computer Science)*, I. Sau and D.M. Thilikos (Eds.), Vol. 11789. Springer, 338–350.

[15] J. Böker, Y. Chen, M. Grohe, and G. Rattan. 2019. The Complexity of Homomorphism Indistinguishability. In *Proceedings of the 44th International Symposium on Mathematical Foundations of Computer Science (Leibniz International Proceedings in Informatics (LIPIcs))*, P. Rossmanith, P. Heggernes, and J.-P. Katoen (Eds.), Vol. 138. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 54:1–54:13.

[16] R. Bordawekar and O. Shmueli. 2017. Using word embedding to enable semantic queries in relational databases. In *Proceedings of the Data Management for End to End Learning Work Workshop, SIGMOD'17*.

[17] R. Bordawekar and O. Shmueli. 2019. Exploiting Latent Information in Relational Databases via Word Embedding and Application to Degrees of Disclosure. In *Proceedings of the 9th Biennial Conference on Innovative Data Systems Research*.

[18] A. Bordes, N. Usunier, A. Garcia-Duran, J. Weston, and O. Yakhnenko. 2013. Translating embeddings for modeling multi-relational data. In *Advances in neural information processing systems*. 2787–2795.

[19] C. Borgs, J. Chayes, L. Lovász, V. Sós, B. Szegedy, and K. Vesztergombi. 2006. Graph limits and parameter testing. In *Proceedings of the 38th Annual ACM Symposium on Theory of Computing*. 261–270.

[20] K.M. Borgwardt and H.-P. Kriegel. 2005. Shortest-path kernels on graphs. In *Proceedings of the 5th IEEE International Conference on Data Mining*. 74–81.

[21] J. Bourgain. 1985. On Lipschitz embeddings of finite metric spaces in Hilbert spaces. *Israel Journal of Mathematics* 52, 1-2 (1985), 46–52.

[22] A. Bulatov, M. Grohe, and G. Rattan. [n.d.]. In preparation.

[23] J. Bulian and A. Dawar. 2014. Graph isomorphism parameterized by elimination distance to bounded degree. In *Proceedings of the 9th International Symposium on Parameterized and Exact Computation (Lecture Notes in Computer Science)*, M. Cygan and P. Heggernes (Eds.), Vol. 8894. Springer Verlag, 135–146.

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

[25] S. Cao, W. Lu, and Q. Xu. 2015. GraRep: Learning graph representations with global structural information. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*. 891–900.

[26] S. Cao, W. Lu, and Q. Xu. 2016. Deep neural networks for learning graph representations. In *Proceedings of the 30th AAAI Conference on Artificial Intelligence*. 1145–1152.

[27] A. Cardon and M. Crochemore. 1982. Partitioning a graph in $O(|A| \log_2 |V|)$. *Theoretical Computer Science* 19, 1 (1982), 85 – 98.

[28] Y. Chen and J. Flum. 2018. Tree-depth, Quantifier Elimination, and Quantifier Rank. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*. 225–234.

[29] C. Cortes and V. Vapnik. 1995. Support-vector networks. *Machine Learning* 20, 3 (1995), 273–297.

[30] R. Curticapean, H. Dell, and D. Marx. 2017. Homomorphisms are a good basis for counting small subgraphs. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing (STOC '17)*. 210–223.

[31] V. Dalmau and P. Jonsson. 2004. The complexity of counting homomorphisms seen from the other side. *Theoretical Computer Science* 329, 1-3 (2004), 315–323.

[32] H. Dell, M. Grohe, and G. Rattan. 2018. Lovász Meets Weisfeiler and Leman. In *Proceedings of the 45th International Colloquium on Automata, Languages and Programming (Track A) (LIPIcs)*, I. Chatzigiannakis, C. Kaklamanis, D. Marx, and D. Sannella (Eds.), Vol. 107. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 40:1–40:14.

[33] Z. Dvorák. 2010. On recognizing graphs by numbers of homomorphisms. *Journal of Graph Theory* 64, 4 (2010), 330–342.

[34] M. Elberfeld, M. Grohe, and T. Tantau. 2016. Where First-Order and Monadic Second-Order Logic Coincide. *ACM Transaction on Computational Logic* 17, 4 (2016). Article No. 25.

[35] M. Elberfeld, A. Jakoby, and T. Tantau. 2012. Algorithmic Meta Theorems for Circuit Classes of Constant and Logarithmic Depth. In *Proceedings of the 29th International Symposium on Theoretical Aspects of Computer Science (LIPIcs)*, C. Dürr and T. Wilke (Eds.), Vol. 14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 66–77.

[36] C. Gallicchio and A. Micheli. 2010. Graph echo state networks. In *Proceedings of the IEEE International Joint Conference on Neural Networks*.

[37] T. Gärtner, P. Flach, and S. Wrobel. 2003. On graph kernels: Hardness results and efficient alternatives. In *Learning theory and kernel machines*. Springer Verlag, 129–143.

[38] T. Gervens. 2018. Spectral Graph Similarity. Master Thesis at RWTH Aachen.

[39] E. Grädel, M. Grohe, B. Pago, and W. Pakusa. 2019. A Finite-Model-Theoretic View on Propositional Proof Complexity. *Logical Methods in Computer Science* 15, 1 (2019), 4:1–4:53.

[40] E. Grädel, P.G. Kolaitis, L. Libkin, M. Marx, J. Spencer, M.Y. Vardi, Y. Venema, and S. Weinstein. 2007. *Finite Model Theory and Its Applications*. Springer Verlag.

[41] M. Grohe. 2017. *Descriptive Complexity, Canonisation, and Definable Graph Structure Theory*. Lecture Notes in Logic, Vol. 47. Cambridge University Press.

[42] M. Grohe. 2020. Counting Bounded Tree Depth Homomorphisms. *ArXiv* arXiv:2003.08164 [cs.LO] (2020).

[43] M. Grohe. 2020. Counting Bounded Tree Depth Homomorphisms. In *Submitted*.

[44] M. Grohe, K. Kersting, M. Mladenov, and E. Selman. 2014. Dimension Reduction via Colour Refinement. In *Proceedings of the 22nd Annual European Symposium on Algorithms (Lecture Notes in Computer Science)*, A. Schulz and D. Wagner (Eds.), Vol. 8737. Springer-Verlag, 505–516.

[45] M. Grohe and M. Otto. 2015. Pebble Games and Linear Equations. *Journal of Symbolic Logic* 80, 3 (2015), 797–844.

[46] M. Grohe, G. Rattan, and G. Woeginger. 2018. Graph Similarity and Approximate Isomorphism. In *Proceedings of the 43rd International Symposium on Mathematical Foundations of Computer Science (LIPIcs)*, I. Potapov, P.G. Spirakis, and J. Worrell (Eds.), Vol. 117. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 20:1–20:16.

[47] M. Grohe, P. Schweitzer, and D. Wiebking. 2020. Deep Weisfeiler Leman. *ArXiv* arXiv:2003.10935 [cs.LO] (2020).

[48] A. Grover and J. Leskovec. 2016. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, B. Krishnapuram, M. Shah, A.J. Smola, C.C. Aggarwal, D. Shen, and R. Rastogi: (Eds.). 855–864.

[49] W. Hamilton, R. Ying, and J. Leskovec. 2017. Inductive Representation Learning on Large Graphs. In *Proceedings of the 30th Annual Conference on Neural Information Processing Systems*. 1024–1034.

[50] W.L. Hamilton, R. Ying, and J. Leskovec. 2017. Representation learning on graphs: methods and applications. *ArXiv* arXiv:1709.05584 [cs.SI] (2017).

[51] S. Hochreiter and J. Schmidhuber. 1997. Long short-term memory. *Neural Computation* 9, 8 (1997), 1735–1780.

[52] T. Horváth, T. Gärtner, and S. Wrobel. 2004. Cyclic pattern kernels for predictive graph mining. In *Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 158–167.

[53] N. Immerman and E. Lander. 1990. Describing graphs: A first-order approach to graph canonization. In *Complexity theory retrospective*, A. Selman (Ed.). Springer-Verlag, 59–81.

[54] P. Indyk. 2001. Algorithmic applications of low-distortion geometric embeddings. In *Proceedings of the 42nd IEEE Symposium on Foundations of Computer Science*. 10–33.

[55] W. Johnson and J. Lindenstrauss. 1984. Extensions of Lipschitz mappings into a Hilbert space. *Contemp. Math.* 26 (1984), 189–206.

[56] H. Kashima, K. Tsuda, and A. Inokuchi. 2003. Marginalized kernels between labeled graphs. In *Proceedings of the 20th International Conference on Machine Learning*. 321–328.

[57] K. Kersting, M. Mladenov, R. Garnet, and M. Grohe. 2014. Power Iterated Color Refinement. In *Proceedings of the 28th AAAI Conference on Artificial Intelligence*, P. Stone C.E. Brodley (Ed.). 1904–1910.

[58] T.N. Kipf and M. Welling. 2016. Variational Graph Auto-Encoders. *ArXiv* arXiv:1611.07308 [stat.ML] (2016).

[59] T. N. Kipf and M. Welling. 2017. Semi-supervised classification with graph convolutional networks. In *Proceedings of the 5th International Conference on Learning Representations*.

[60] R. Kondor and J.D. Lafferty. 2002. Diffusion Kernels on Graphs and Other Discrete Input Spaces. In *Proceedings of the 19th International Conference on Machine Learning*. 315–322.

[61] N.M. Kriege, F.D Johansson, and C. Morris. 2019. A survey on graph kernels. *ArXiv* arXiv:1903.11835 [cs.LG] (2019).

[62] N. Kriege, M. Neumann, C. Morris, K. Kersting, and P. Mutzel. 2019. A unifying view of explicit and implicit feature maps of graph kernels. *Data Mining and Knowledge Discovery* 33, 6 (2019), 1505–1547. https://doi.org/10.1007/s10618-019-00652-0

[63] J.B. Kruskal. 1964. Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. *Psychometrika* 29, 1 (1964), 1–27.

[64] N. Linial, E. London, and Y. Rabinovich. 1995. The geometry of graphs and some of its algorithmic applications. *Combinatorica* 15 (1995), 212–245.

[65] L. Lovász. 1967. Operations with Structures. *Acta Mathematica Hungarica* 18 (1967), 321–328.

[66] L. Lovász. 1971. On the cancellation law among finite relational structures. *Periodica Mathematica Hungarica* 1, 2 (1971), 145–156.

[67] L. Lovász. 2012. *Large Networks and Graph Limits*. American Mathematical Society.

[68] L. Lovász and B. Szegedy. 2006. Limits of dense graph sequences. *Journal of Combinatorial Theory, Series B* 96, 6 (2006), 933–957.

[69] T. Maehara and H. NT. 2019. A Simple Proof of the Universality of Invariant/Equivariant Graph Neural Networks. *ArXiv* arXiv:1910.03802 [cs.LG] (2019).

[70] K. Makarychev, R. Manokaran, and M. Sviridenko. 2014. Maximum quadratic assignment problem: Reduction from maximum label cover and lp-based approximation algorithm. *ACM Transactions on Algorithms* 10, 4 (2014), 18.

[71] P. Malkin. 2014. Sherali–Adams relaxations of graph isomorphism polytopes. *Discrete Optimization* 12 (2014), 73–97.

[72] L. Mančinska and D.E. Roberson. 2019. Quantum isomorphism is equivalent to equality of homomorphism counts from planar graphs. *ArXiv* arXiv:1910.06958v2 [quant-ph] (2019).

[73] B.D. McKay and A. Piperno. 2014. Practical graph isomorphism, II. *Journal of Symbolic Compututation* 60 (2014), 94–112.

[74] T. Mikolov, I. Sutskever, K. Chen, G.S. Corrado, and J. Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Proceedings of the 27th Annual Conference on Neural Information Processing Systems*. 3111–3119.

[75] H.L. Morgan. 1965. The generation of a unique machine description for chemical structures—a technique developed at chemical abstracts service. *Journal of Chemical Documentation* 5, 2 (1965), 107–113.

[76] C. Morris, K. Kersting, and P. Mutzel. 2017. Globalized Weisfeiler-Lehman Graph Kernels: Global-Local Feature Maps of Graphs. In *Proceedings of the 2017 IEEE International Conference on Data Mining*. 327–336.

[77] C. Morris, N.M. Kriege, K. Kersting, and P. Mutzel. 2016. Faster kernel for graphs with continuous attributes via hashing. In *Proceedings of the 16th IEEE International Conference on Data Mining*. 1095–1100.

[78] C. Morris, M. Ritzert, M. Fey, W. Hamilton, J.E. Lenssen, G. Rattan, and M. Grohe. 2019. Weisfeiler and Leman Go Neural: Higher-order Graph Neural Networks. In *Proceedings of the 33rd AAAI Conference on Artificial Intelligence*, Vol. 4602-4609. AAAI Press.

[79] V. Nagarajan and M. Sviridenko. 2009. On the maximum quadratic assignment problem. In *Proceedings of the twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*. 516–524.

[80] A. Narayanan, M. Chandramohan, R. Venkatesan, L. Chen, Y. Liu, and S. Jaiswal. 2017. graph2vec: Learning Distributed Representations of Graphs. *ArXiv (CoRR)* arXiv:1707.05005 [cs.AI] (2017).

[81] J. Nešetřil and P. Ossona de Mendez. 2006. Linear time low tree-width partitions and algorithmic consequences. In *Proceedings of the 38th ACM Symposium on Theory of Computing*. 391–400.

[82] M. Neumann, R. Garnett, and K. Kersting. 2013. Coinciding walk kernels: Parallel absorbing random walks for learning with graphs and few labels. In *Proceedings of the 5th Asian Conference on Machine Learning*. 357–372.

[83] M. Nickel, V. Tresp, and H.-P. Kriegel. 2011. A three-way model for collective learning on multi-relational data. In *Proceedings of the 28th International Conference on Machine Learning*. 809–816.

[84] R. O'Donnell, J. Wright, C. Wu, and Y. Zhou. 2014. Hardness of Robust Graph Isomorphism, Lasserre Gaps, and Asymmetry of Random Graphs. In *Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms*. 1659–1677.

[85] M. Ou, P. Cui, J. Pei, Z. Zhang, and W. Zhu. 2016. Asymmetric transitivity preserving graph embedding. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 1105–1114.

[86] S. Pan, R. Hu, G. Long, J. Jiang, L. Yao, and C. Zhang. 2018. Adversarially Regularized Graph Autoencoder for Graph Embedding. *ArXiv (CoRR)* arXiv:1802.04407 [cs.LG] (2018). http://arxiv.org/abs/1802.04407

[87] B. Perozzi, R. Al-Rfou, and S. Skiena. 2014. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 710–710.

[88] T. Pham, T. Tran, D. Phung, and S. Venkatesh. 2017. Column networks for collective classification. In *Proceedings of the 31st AAAI Conference on Artificial Intelligence*. 2485–2491.

[89] J. Ramon and T. Gärtner. 2003. Expressivity versus efficiency of graph kernels. In *Proceedings of the 1st International Workshop on Mining Graphs, Trees and Sequences*. 65–74.

[90] F. Scarselli, M. Gori, A.C. Tsoi, M. Hagenbuchner, and G. Monfardini. 2009. The graph neural network model. *IEEE Transactions on Neural Networks* 20, 1 (2009), 61–80.

[91] M . Schlichtkrull, T.N. Kipf, P. Bloem, R. Van Den Berg, I. Titov, and M. Welling. 2018. Modeling relational data with graph convolutional networks. In *Proceedings of the European Semantic Web Conference (Lecture Notes in Computer Science)*, A. Gangemi, R. Navigli, M.-E. Vidal, P. Hitzler, R. Troncy, L. Hollink, A. Tordai, and M. Alam (Eds.), Vol. 10843. Springer Verlag, 593–607.

[92] B. Schölkopf, A. Smola, and K.-R. Müller. 1997. Kernel principal component analysis. In *Proceedings of the International Conference on Artificial Neural Networks (Lecture Notes in Computer Science)*, W. Gerstner, A. Germond, M. Hasler, and J.D. Nicoud (Eds.), Vol. 1327. Springer Verlag, 583–588.

[93] S. Shalev-Shwartz and S. Ben-David. 2014. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press.

[94] N. Shervashidze, P. Schweitzer, E.J. van Leeuwen, K. Mehlhorn, and K.M. Borgwardt. 2011. Weisfeiler-Lehman Graph Kernels. *Journal of Machine Learning Research* 12 (2011), 2539–2561.

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

[96] A.J. Smola and R. Kondor. 2003. Kernels and Regularization on Graphs. In *Proceedings of the 16th Annual Conference on Computational Learning Theory (Lecture Notes in Computer Science)*, B. Schölkopf and M.K. Warmuth (Eds.), Vol. 2777. Springer Verlag, 144–158.

[97] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei. 2015. LINE: Large-scale information network embedding. In *Proceedings of the 24th International World Wide Web Conference*. 1067–1077.

[98] J. Tenenbaum, V. De Silva, and J. Langford. 2000. A global geometric framework for nonlinear dimensionality reduction. *Science* 290 (2000), 2319–2323.

[99] G. Tinhofer. 1991. A note on compact graphs. *Discrete Applied Mathematics* 30 (1991), 253–264.

[100] J. Tönshoff, M. Ritzert, H. Wolf, and M. Grohe. 2019. Graph Neural Networks for Maximum Constraint Satisfaction. *ArXiv (CoRR)* arXiv:1909.08387 [cs.AI] (2019).

[101] D. Wang, P. Cui, and W. Zhu. 2016. Structural deep network embedding. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 1225–1234.

[102] Q. Wang, Z. Mao, B. Wang, and L. Guo. 2017. Knowledge Graph Embedding: A Survey of Approaches and Applications. *IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING* 29, 12 (2017), 2724–2743.

[103] B. Weisfeiler and A. Leman. 1968. The reduction of a graph to canonical form and the algebgra which appears therein. *NTI, Series 2* (1968). English transalation by G. Ryabov avalable at https://www.iti.zcu.cz/wl2018/pdf/wl_paper_translation.pdf.

[104] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P.S. Yu. 2019. A comprehensive survey on Graph Neural Networks. *ArXiv* arXiv:1901.00596 [cs.Lg] (2019).

[105] Z. Xinyi and L. Chen. 2019. Capsule Graph Neural Network. In *Proceedings of the 7th International Conference on Learning Representations*. OpenReviw.net. https://openreview.net/forum?id=Byl8BnRcYm

[106] K. Xu, W. Hu, J. Leskovec, and S. Jegelka. 2019. How powerful are graph neural networks?. In *Proceedings of the 7th International Conference on Learning Representations*.