Graph Alignment Kernels using Weisfeiler and Leman Hierarchies

Giannis Nikolentzos
LIX, École Polytechnique
Institut Polytechnique de Paris, France

Michalis Vazirgiannis
LIX, École Polytechnique
Institut Polytechnique de Paris, France

Abstract

Graph kernels have become a standard approach for tackling the graph similarity and learning tasks at the same time. Most graph kernels proposed so far are instances of the $\mathcal{R}$-convolution framework. These kernels decompose graphs into their substructures and sum over all pairs of these substructures. However, considerably less attention has been paid to other types of kernels. In this paper, we propose a new kernel between graphs which reorders the adjacency matrix of each graph based on soft permutation matrices, and then compares those aligned adjacency matrices to each other using a linear kernel. To compute the permutation matrices, the kernel finds corresponding vertices in different graphs. Two vertices match with each other if the Weisfeiler-Leman test of isomorphism assigns the same label to both of them. The proposed kernel is evaluated on several graph classification and graph regression datasets. Our results indicate that the kernel is competitive with traditional and state-of-the-art methods. The code is available at `https://github.com/giannisnik/gawl`.

1 Introduction

Graphs have attracted a lot of attention in the past years since they arise naturally in several disciplines ranging from social networks, to chemo- and bio-informatics. This abundance of graph-structured data has created the need for machine learning algorithms that can operate on graphs. To date, there exist two major families of such algorithms, namely *graph kernels* and *graph neural networks*.

Kernel methods can be applied to any type of data, provided that there exists a kernel that can compare any two data objects to each other. This led to the development of kernels for different kinds of structured data including graphs. Over the past years 20 years, graph kernels have received a lot of research interest and have achieved state of the art predictive performance in several classification problems (Kriege et al., 2020; Borgwardt et al., 2020; Nikolentzos et al., 2021). Most graph kernels are instances of the $\mathcal{R}$-convolution framework (Haussler, 1999). These kernels decompose graphs into substructures (e. g., paths, subgraphs, walks) which they compare to each other to compute local similarities that are then aggregated. $\mathcal{R}$-convolution kernels have been applied with success to several problems, however, they aggregate similarities not only between similar substructures but between all substructures. Furthermore, these kernels are also prone to the diagonal dominance problem (Yanardag and Vishwanathan, 2015). On the other hand, assignment kernels compute a matching between substructures of two objects such that the overall similarity of the two objects is maximized. Such a matching can reveal structural correspondences between the two objects. Unfortunately, not all assignment functions are positive semidefinite (i. e., valid kernels). This issue emerged early on in the development of the field of graph kernels, when an optimal assignment kernel that was proposed to compute a correspondence between the atoms of molecules (Fröhlich et al., 2005) was later proven not to always be positive semidefinite (Vert, 2008). This slowed down the development of assignment kernels. Indeed, it was not until recently that some kernels of this sort started to emerge (Nikolentzos et al., 2017). Interestingly, it was shown that there exists a class of base kernels used to compare substructures that guarantees positive semidefinite optimal assignment kernels (Kriege et al., 2016). Recently, ideas from optimal transport theory are combined with graph representations derived from graph kernels (Togninalli et al., 2019).

In this paper, we propose a new kernel between graphs, so-called *Weisfeiler-Leman graph alignment kernel* (GAWL). The kernel aligns a corpus of graphs, i. e., it finds corresponding vertices in different graphs, and it produces a permutation matrix for each graph based on the alignment. Then, it reorders the adjacency matrix of each graph based

Proceedings of the 26th International Conference on Artificial Intelligence and Statistics (AISTATS) 2023, Valencia, Spain. PMLR: Volume 206.

on those permutation matrices, and compares the aligned adjacency matrices to each other. Unfortunately, aligning the vertices of a set of graphs is hard. Therefore, we relax the above problem, and we allow the kernel to match vertices to each other if the Weisfeiler-Leman test of isomorphism assigns the same label to them. We evaluate the proposed kernel on several benchmark datasets for graph classification and graph regression tasks. We find that the proposed kernel leads to performance gains on several of those datasets. More precisely, our main contributions can be summarized as follows:

- We present the Weisfeiler-Leman graph alignment kernel (GAWL), a new kernel between graphs which capitalizes on the Weisfeiler-Leman test of isomorphism to reorder the vertices of the graphs and then, compares the emerging adjacency matrices to each other.
- We propose an efficient computation scheme which calculates the GAWL kernel skipping the costly construction of the new adjacency matrices of the graphs.
- We evaluate the performance of the proposed kernel on several graph classification and regression datasets where it achieves performance comparable with state-of-the-art graph kernels and graph neural networks.

The rest of this paper is organized as follows. Section 2 provides an overview of the related work. Section 3 introduces some preliminary concepts and gives details about color refinement in graphs. Section 4 presents the proposed graph kernel. Section 5 evaluates the proposed kernel on different datasets. Finally, Section 6 concludes.

2 Related Work

Graph kernels have been studied extensively in the past 20 years and have been applied with great success to many graph classification problems (Kriege et al., 2020; Borgwardt et al., 2020; Nikolentzos et al., 2021). A graph kernel is a symmetric positive semidefinite function, which generates implicitly (or explicitly) graph representations and enables the application of kernel methods such as the SVM classifier to graphs. Most graph kernels are instances of the $\mathcal{R}$-convolution framework. These kernels decompose graphs into substructures which they compare to each other to compute local similarities that are then aggregated. Such substructures include random walks (Kashima et al., 2003; Gärtner et al., 2003; Mahé et al., 2004; Vishwanathan et al., 2010; Sugiyama and Borgwardt, 2015; Kalofolias et al., 2021), shortest paths (Borgwardt and Kriegel, 2005; Feragen et al., 2013), cycles (Horváth et al., 2004), subtrees (Ramon and Gärtner, 2003), small subgraphs (Shervashidze et al., 2009; Kriege and Mutzel, 2012; Johansson et al., 2014), among others. The Weisfeiler-Leman framework builds on ideas from the Weisfeiler-Leman test of isomorphism and relabels the vertices of the input graphs which can then be compared to each other using any graph kernel (Shervashidze et al., 2011). This framework was further generalized to higher order variants of the Weisfeiler-Leman algorithm (Morris et al., 2017). However, the computational cost of the k-dimensional Weisfeiler-Leman algorithm is prohibitive for large values of k, and several recent works focused on improving efficiency by leveraging the sparse and local nature of graphs (Morris et al., 2020b, 2022). There are also kernels that can capture similarity at different granularity levels such as the multiscale Laplacian graph kernel which builds a hierarchy of subgraphs centered at vertices (Kondor and Pan, 2016). These subgraphs are then compared to each other to compute the kernel.

Assignment kernels, another family of graph kernels, have recently received more attention than $\mathcal{R}$-convolution kernels. Those kernels do not compare all substructures of an object against all substructures of another object. Instead, they compute a matching between substructures of one object and substructures of a second object such that the overall similarity of the two objects is maximized. This family of graph kernels is less studied than that of $\mathcal{R}$-convolution kernels, mainly because not all assignment functions are positive semidefinite (Vert, 2008), and thus more effort is required to design such kind of kernels. Despite those difficulties, some assignment kernels have been developed so far such as the pyramid match graph kernel which computes a correspondence between the embeddings of the vertices of graphs (Nikolentzos et al., 2017), and the transitive alignment kernel which computes correspondences between substructures that satisfy transitivity (Schiavinato et al., 2015). More importantly, it has been shown that there exists a class of base kernels used to compare substructures that guarantees positive semidefinite optimal assignment kernels, while three different graph kernels (including the Weisfeiler-Leman optimal assignment kernel) were derived from that methodology (Kriege et al., 2016).

Recently, optimal transport approaches have been widely used for comparing set-structured data. Since graphs can be represented as sets of node embeddings, these methods have also found their way into the problem of graph comparison. For instance, the Wasserstein Weisfeiler-Leman graph kernels compare node embeddings of two graphs via the Wasserstein distance (Togninalli et al., 2019). Petric Maretic et al. (2019) map graphs into Gaussian distributions based on their Laplacian matrices and compare graphs by computing the Wasserstein distance between their distributions. To compare graphs to each other, Vayer et al. (2019) combine the Gromov-Wasserstein distance with the Wasserstein distance. The proposed approach can take into account both the graph structure and node feature information. Xu et al. (2019a) propose an algorithm that is based

on optimal transport and which jointly aligns graphs and learns node embeddings. Kolouri et al. (2021) propose a method that computes an approximate Euclidean embedding for the Wasserstein distance in a reproducing kernel Hilbert space. Dong and Sawin (2020) propose a coordinated optimal transport algorithm for comparing graphs to each other and apply the proposed algorithm to graph sketching. Ma et al. (2020) first embed the graphs into a pyramid structure and then use optimal transport to compare two graphs at each level of the pyramid. Wijesinghe et al. (2021) propose an optimal transport algorithm which uses two regularization terms that guarantee the convergence and numerical stability in finding an optimal assignment between graphs. Furthermore, they compute feature similarity matrices to preserve features and their local variations.

3 Preliminaries

3.1 Notation

Let $\mathbb{N}$ denote the set of natural numbers, i. e., $\{1, 2, \ldots\}$. Then, $[n] = \{1, \ldots, n\} \subset \mathbb{N}$ for $n \geq 1$. Let also $\{\!\{\}\!\}$ denote a multiset, i. e., a generalized concept of a set that allows multiple instances for its elements. A partition of a set X is a set P of non-empty subsets of X such that the union of the sets in P is equal to X, i. e., $\bigcup_{A \in P} A = X$, and the intersection of any two distinct sets in P is empty, i. e., for all $A, B \in P$ with $A \neq B$, it holds that $A \cap B = \emptyset$. Given two partitions P and P' of the same set X, P' is finer than P (we denote this by $P' \sqsubseteq P$) if every element of P' is either a subset or equal to an element of P. If $P' \sqsubseteq P$ holds, we can equivalently say that P is coarser than P'. If both $P \sqsubseteq P'$ and $P' \sqsubseteq P$ hold, we denote this by $P \equiv P'$.

Let $G = (V, E)$ be an undirected, unweighted graph, where V is the vertex set and E is the edge set. We will denote by n the number of vertices and by m the number of edges, i. e., $n = |V|$ and $m = |E|$. The adjacency matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ is a symmetric matrix used to encode edge information in a graph. The element of the i^{th} row and j^{th} column is equal to 1 if there is an edge between v_i and v_j, and 0 otherwise. Let $\mathcal{N}(v)$ denote the neighbourhood of vertex v, i. e., the set $\{u \mid \{v, u\} \in E\}$. The degree of a vertex v is $\deg(v) = |\mathcal{N}(v)|$. Two graphs $G = (V, E)$ and $G' = (V', E')$ are isomorphic (denoted by $G \cong G'$) if there is a bijective mapping $f : V \to V'$ such that $(v, u) \in E$ iff $(f(v), f(u)) \in E'$. A colored graph is a tuple (G, c), where G is a graph and $c \colon V \to \Sigma$ is a function that assigns colors (i. e., elements from a particular set Σ) to the vertices of the graph. We interpret all graphs treated in this paper as colored graphs and just write G instead of (G, c) when c is clear from the context. If the coloring is not specified, we assume either a monochromatic coloring, i. e., all vertices have the same color or we create one color for each value of degree. For a graph G endowed with a node coloring c, a color class of G is a maximal set of vertices that all have the same color. Every graph coloring c induces a partition $\pi(c)$ of V into the vertex color classes with respect to c.

3.2 Weisfeiler-Leman Algorithm

The Weisfeiler-Leman algorithm and its variants have attracted a lot of attention recently since several kernels and neural architectures are based on them (Morris et al., 2021). Next, we present the 1-dimensional Weisfeiler-Leman (WL) algorithm for colored graphs, also known as color refinement and naive vertex classification. The algorithm proceeds by iteratively refining a partition of the vertices of its input graph until the partition is stable with respect to the refinement criterion.

Let (G, c) be a colored graph, and let also $\chi^0_G = c$. The WL algorithm runs for a number of iterations, and in each iteration, it computes a vertex coloring $\chi^i_G \colon V \to \Sigma$. Specifically, for $i \in \mathbb{N}$, the coloring χ^i_G computed by WL after i iterations on G is defined as $\chi^i_G(v) = \big(\chi^{i-1}_G(v), \{\!\{\chi^{i-1}_G(w) \mid w \in N(v)\}\!\}\big)$. That is, $\chi^i_G(v)$ consists of the color of v from the previous iteration as well as the multiset of colors of neighbors of v from the previous iteration. It is clear that each vertex colouring χ^i_G naturally partitions V into colour classes, i. e., sets of vertices with the same colour. Since the refinement takes the colour $\chi^{i-1}_G(v)$ of a vertex v into account when computing $\chi^i_G(v)$, the implication $\chi^{i-1}_G(v) \neq \chi^{i-1}_G(u) \Rightarrow \chi^i_G(v) \neq \chi^i_G(u)$ holds for all $u, v \in V$. Hence, the colour classes induced by χ^i_G are at least as fine as those induced by χ^{i-1}_G, i. e., $\pi(\chi^i_G) \sqsubseteq \pi(\chi^{i-1}_G)$ holds for every graph G and every $i \in \mathbb{N}$.

Traditionally, the WL algorithm is used to test two graphs for isomorphism. If for some $i \in \mathbb{N}$, the numbers of vertices of color $\sigma \in \Sigma$ differ in the two graphs, then the two graphs are non-isomorphic. The algorithm terminates when the number of colors between two successive iterations does not change, i. e., the cardinalities of the images of χ^{i-1}_G and χ^i_G are equal. Termination is guaranteed after at most a number of iterations equal to the number of vertices of the larger of the two graphs. With regards to its expressive power, it is well-known that there exist pairs of non-isomorphic graphs that the WL algorithm cannot distinguish (Arvind et al., 2015). For instance, it cannot distinguish a cycle with six vertices from two triangles with three vertices (Kriege et al., 2018). Still, the algorithm is very useful since it has been shown that it can successfully test isomorphism for a broad class of graphs (Babai and Kucera, 1979). High-dimensional variants of the WL algorithm can distinguish more pairs of non-isomorphic graphs. Specifically, the k-dimensional Weisfeiler-Leman algorithm (k-WL), for $k \geq 2$, is a generalization of the WL which colors tuples from V^k instead of nodes. That is,

the algorithm computes a coloring $\chi^i_G \colon V^k \to \Sigma$. However, as k increases, the computational complexity of the algorithm also increases, thus rendering the algorithm inefficient for practical use. Therefore, in this paper, we focus on the simple WL algorithm presented above. It should be mentioned though that the proposed kernel can benefit from higher-order variants of the WL algorithm, since finer node colorings can be derived from their output.

4 Weisfeiler-Leman Graph Alignment Kernel

Graph-level machine learning problems are usually concerned with predicting the class labels or the targets values (in regression tasks) of graphs contained in a finite set $\{G_1, \ldots, G_M\} \subset \mathfrak{G}$ where $\mathfrak{G}$ is the space of graphs. An interesting question is whether we can somehow align the graphs contained in this set. A natural idea is to look for a permutation matrix $\mathbf{P}^*_i$ for each graph G_i of the dataset where $i \in [M]$ such that the overall distance between graphs is minimized. This gives rise to the following problem:

$$\mathbf{P}^*_1, \ldots, \mathbf{P}^*_M = \operatorname*{arg\,min}_{\mathbf{P}_1,\ldots,\mathbf{P}_M \in \Pi} \sum_{i=1}^{M} \sum_{j=1}^{M} ||\mathbf{P}_i \, \mathbf{A}_i \, \mathbf{P}_i^\top - \mathbf{P}_j \, \mathbf{A}_j \, \mathbf{P}_j^\top|| \tag{1}$$

where Π is the set of $n \times n$ permutation matrices, $||\cdot||$ is the Frobenius norm, $\mathbf{A}_1, \ldots \mathbf{A}_M \in \mathbb{R}^{n \times n}$, n being the number of vertices of the largest graph of the dataset, and zero rows and columns have been appended to the adjacency matrices of the rest of the graphs to match the size of the largest graph. Once we solve the above problem and compute matrices $\mathbf{P}^*_1, \ldots, \mathbf{P}^*_M$, we can then compute the following graph alignment kernel:

$$\begin{aligned} k(G_1, G_2) &= \sum_{i=1}^{n} \sum_{j=1}^{n} \left[\mathbf{P}^*_1 \, \mathbf{A}_1 \, \mathbf{P}^{*\top}_1 \odot \mathbf{P}^*_2 \, \mathbf{A}_2 \, \mathbf{P}^{*\top}_2\right]_{ij} \\ &= \sum_{i=1}^{n} \sum_{j=1}^{n} \left[\sqrt{\mathbf{P}^*_1 \, \mathbf{A}_1 \, \mathbf{P}^{*\top}_1} \odot \sqrt{\mathbf{P}^*_2 \, \mathbf{A}_2 \, \mathbf{P}^{*\top}_2}\right]_{ij} \end{aligned} \tag{2}$$

where $\odot$ denotes elementwise multiplication. Note that the second equality holds since the input graphs are unweighted.

Proposition 1. *The graph alignment kernel defined in Equation* (2) *is positive semidefinite.*

Even though the kernel defined in Equation (2) is a valid kernel, the kernel is not computable in polynomial time since solving the alignment problem defined in Equation (1) is hard. Indeed, the problem defined in Equation (1) is a more general form of the well-known Frobenius distance which cannot be computed in polynomial time (Arvind et al., 2012; Grohe et al., 2018). Therefore, the above kernel is not useful for practical applications. Even if we relax the objective function of Equation (1) from permutations to doubly-stochastic matrices, the problem still remains hard.

4.1 WL-based Vertex Ordering

The problem defined in Equation (1) aligns the graphs' adjacency matrices and imposes an ordering on their vertices. However, as discussed above, this problem has no polynomial time solution. We next capitalize on the WL algorithm to impose an ordering on the vertices of each graph of the input collection of graphs $\mathcal{G} = \{G_1, \ldots, G_M\}$. A naive approach would be to impose an arbitrary ordering on the color classes produced by the WL algorithm, and then sort the vertices of each graph based on that ordering. This would be equivalent to applying $n \times n$ permutation matrices to the input graphs, thus resulting into new adjacency matrices of dimension $n \times n$. However, this can lead two vertices (of different graphs) colored differently from each other into being placed in corresponding positions of the adjacency matrices, even though these two colors might represent structurally dissimilar neighborhoods.

Instead, we propose to treat vertices of a given color independently. Thus, each vertex is mapped only to identically colored vertices of other graphs. This leads to adjacency matrices of size at least equal to that of the largest graph (but usually of much larger size). Formally, let Σ be the set of colors that emerge in the t^{th} iteration of the WL algorithm. For clarity of presentation we will next omit the iteration number t (e. g., instead of $\chi^t_G(v)$, we just use $\chi_G(v)$) since the analysis applies to all iterations. Let also $|\Sigma| = s$, and without loss of generality, we impose an arbitrary ordering on the colors of the t^{th} iteration of the algorithm, i. e., $\{\sigma_1, \ldots, \sigma_s\} = \Sigma$. Then, $\nu_{\sigma_i} \colon G \to \mathbb{N}$ is a function that counts the number of vertices colored σ_i in a graph, and is defined as:

$$\nu_{\sigma_i}(G) = \left|\{\chi_G(v) \,|\, v \in V \text{ and } \chi_G(v) = \sigma_i\}\right|$$

Then, given a color $\sigma_i \in \Sigma$, there exists at least one graph in the input set of graphs $\mathcal{G}$ that contains the largest number of vertices colored σ_i. We denote that number by N_{σ_i}, and is formally defined as:

$$N_{\sigma_i} = \max_{G \in \mathcal{G}} \nu_{\sigma_i}(G)$$

Let N denote the sum of the above number for all colors, i. e., $N = \sum_{i=1}^{s} N_{\sigma_i}$. We then define s sets, one set for each color class, as follows:

$$I_{\sigma_i} = \left\{ j \in \mathbb{N} \,\middle|\, \sum_{k=1}^{i-1} N_{\sigma_k} < j \leq \sum_{k=1}^{i} N_{\sigma_k} \right\}$$

The sets are pairwise disjoint, i. e., $I_\sigma \cap I_{\sigma'} = \emptyset$ for all $\sigma, \sigma' \in \Sigma$ with $\sigma \neq \sigma'$, while $\bigcup_{i=1}^{s} I_{\sigma_i} = [N]$ holds. Each

set contains the indices of the rows and columns of the adjacency matrices of the graphs that correspond to each color class. The vertices that belong to a given color class will then be mapped to the indices associated with that color class.

Before comparing the adjacency matrices of the graphs to each other, they will be first transformed into $N \times N$ matrices. Let $G \in \mathcal{G}$ be an input graph and n denote its number of vertices. For each input graph, we construct a rectangular matrix $\mathbf{D} \in \mathbb{R}^{N \times n}$ whose columns sum to 1, i. e., matrix $\mathbf{D}$ satisfies the following constraint $\mathbf{1}_N^\top \mathbf{D} = \mathbf{1}_n$. Each column of matrix $\mathbf{D}$ corresponds to one vertex of the input graph G. Each vertex of the input graph belongs to some color class $\sigma_\ell \in \Sigma$ and the $\nu_{\sigma_\ell}(G)$ first rows of $\mathbf{D}$ that correspond to that color are set equal to $1/\nu_{\sigma_\ell}(G)$. In other words, let v_j denote the vertex of G that corresponds to the j^{th} row and column of its adjacency matrix (and thus the j^{th} column of $\mathbf{D}$). Then, the elements of the j^{th} column of $\mathbf{D}$ are set equal to the following values:

$$\mathbf{D}_{ij} = \begin{cases} \dfrac{1}{\nu_{\sigma_\ell}(G)}, & \text{if } \chi_G(v_j) = \sigma_\ell \text{ and } i \in I(\sigma_\ell) \\ & \quad \text{and } i - \min\big(I(\sigma_\ell)\big) < \nu_{\sigma_\ell}(G) \\ 0, & \text{otherwise} \end{cases}$$

Then, given matrix $\mathbf{D}$, we transform the adjacency matrix $\mathbf{A}$ of input graph G as follows $\tilde{\mathbf{A}} = \sqrt{\mathbf{D}\,\mathbf{A}\,\mathbf{D}^\top} \in \mathbb{R}^{N \times N}$. Matrix $\tilde{\mathbf{A}}$ has the following form:

$$\tilde{\mathbf{A}} = \begin{bmatrix} \mathbf{B}_{11} & \mathbf{B}_{12} & \dots & \mathbf{B}_{1s} \\ \mathbf{B}_{21} & \mathbf{B}_{22} & \dots & \mathbf{B}_{2s} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{B}_{s1} & \mathbf{B}_{s2} & \dots & \mathbf{B}_{ss} \end{bmatrix}$$

where $\mathbf{B}_{ij}$ is an $N_{\sigma_i} \times N_{\sigma_j}$ matrix whose sum of squared elements is equal to twice the number of edges between vertices colored σ_i and vertices colored σ_j in G. Let $\mu_{\sigma_i,\sigma_j} \colon G \to \mathbb{N}$ be a function that counts the number of edges from vertices colored σ_i to vertices colored σ_j in a graph. Note that edges between identically colored vertices are considered twice. Then, given the transformed adjacency matrix $\tilde{\mathbf{A}}$ of a graph G, $\sum_{i=0}^{N_{\sigma_i}} \sum_{j=0}^{N_{\sigma_j}} [\mathbf{B}_{ij}]_{ij}^2 = \mu_{\sigma_i,\sigma_j}(G)$ holds for all $i, j \in [s]$. In fact, this value is equally distributed among the elements of the $\nu_{\sigma_i}(G) \times \nu_{\sigma_j}(G)$ upper left submatrix of matrix $\mathbf{B}_{ij}^{\odot 2}$ ($\odot$ denotes Hadamard power). The construction of matrix $\tilde{\mathbf{A}}$ for some example graph G is illustrated in Figure 1. The above formulation also admits a probabilistic interpretation since it measures the probability of a vertex colored σ_i to be connected to a vertex colored σ_j.

Finally, given two graphs G, G' and the respective matrices $\mathbf{D}, \mathbf{D}'$ (produced after t iterations of WL), we can compute the kernel between the two graphs as follows:

$$\begin{aligned} k^{(t)}(G, G') &= \sum_{i=1}^{N} \sum_{j=1}^{N} \Big[\sqrt{\mathbf{D}\,\mathbf{A}\,\mathbf{D}^\top} \odot \sqrt{\mathbf{D}'\,\mathbf{A}'\,\mathbf{D}'^\top} \Big]_{ij} \\ &= \sum_{i=1}^{N} \sum_{j=1}^{N} \Big[\tilde{\mathbf{A}} \odot \tilde{\mathbf{A}}' \Big]_{ij} \end{aligned} \tag{3}$$

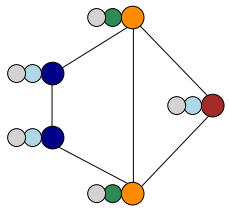
Then the Weisfeiler-Leman graph alignment kernel (with T iterations of the WL algorithm) is defined as: $k(G, G') = k^{(1)}(G, G') + \ldots + k^{(T)}(G, G')$. In Appendix C, we also present a variant of the above definition which takes all colors that emerge in the T iterations of the WL algorithm into account to transform the adjacency matrices of the graphs. Based on Proposition 1, $k^{(t)}$ is a valid kernel function, and thus the sum of such kernels for $t \in \{1, \ldots, T\}$ is also a valid kernel. It should be mentioned that the proposed kernel is closely related to the Weisfeiler-Leman optimal assignment kernel (Kriege et al., 2016) (more details are given in the Appendix). However, it addresses one of the main limitations of the Weisfeiler-Leman optimal assignment kernel, since, as discussed next, it can also handle vertex and edge attributes.

In case of vertex-attributed graphs, let $\mathbf{X} \in \mathbb{R}^{n \times d}$ denote the vertex features where d is the feature dimensionality. The feature of a given vertex v_i is stored in the i^{th} row of $\mathbf{X}$. Then, we can transform the matrix of features $\mathbf{X}$ of the graph as follows $\tilde{\mathbf{X}} = \mathbf{D}\,\mathbf{X}$. Given two graphs G and G', let $\tilde{\mathbf{X}}, \tilde{\mathbf{X}}'$ denote their new matrices of features. Then, the following kernel can be added to the one of Equation (3) $\sum_{i=1}^{N} \sum_{j=1}^{d} \Big[\tilde{\mathbf{X}} \odot \tilde{\mathbf{X}}' \Big]_{ij}$. Note that the kernel puts more emphasis on the graph structure than on the vertex features since matrix $\mathbf{D}$ has emerged from the WL algorithm which captures structural properties of vertices. In cases where edge attributes are available, matrix $\mathbf{A}$ is a tensor, i. e., $\mathbf{A} \in \mathbb{R}^{n \times n \times d}$ where d is the dimension of the edge attributes. We can then use matrix $\mathbf{D}$ to transform $\mathbf{A}$ into a new tensor $\tilde{\mathbf{A}} \in \mathbb{R}^{N \times N \times d}$. Then, the kernel can be computed as follows: $k^{(t)}(G, G') = \sum_{i=1}^{N} \sum_{j=1}^{N} \sum_{k=1}^{d} \Big[\tilde{\mathbf{A}} \odot \tilde{\mathbf{A}}' \Big]_{ijk}$.

4.2 Efficient Computation

Unfortunately, the Wesfeiler-Leman graph alignment kernel is not very efficient. Indeed, N can take large values. In the worst case, it can be equal to the sum of the number of vertices of all graphs (typically in the order of hundreds of thousands or millions). Therefore, computing the transformed adjacency matrix $\tilde{\mathbf{A}} = \sqrt{\mathbf{D}\,\mathbf{A}\,\mathbf{D}^\top} \in \mathbb{R}^{N \times N}$ of a graph G is impractical even if the emerging matrix is stored as a sparse matrix. Instead, we show next that the kernel value between two graphs can in fact be computed more efficiently.

Given two graphs G, G', the kernel computes the following



(a) A graph G and the largest number of vertices colored with each color

$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \end{bmatrix}$$

(b) The adjacency matrix of G

$$\mathbf{D} = \left[\begin{array}{ccccc} 0 & 0 & 1/2 & 1/2 & 0 \\ 0 & 0 & 1/2 & 1/2 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ \hline 1/2 & 1/2 & 0 & 0 & 0 \\ 1/2 & 1/2 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{array}\right]$$

(c) The permutation matrix associated with G

$$\tilde{\mathbf{A}} = \sqrt{\mathbf{D}\,\mathbf{A}\,\mathbf{D}^\top} = \left[\begin{array}{cccc|cc|cc} \sqrt{1/2} & \sqrt{1/2} & 0 & 0 & \sqrt{1/2} & \sqrt{1/2} & 0 & 0 \\ \sqrt{1/2} & \sqrt{1/2} & 0 & 0 & \sqrt{1/2} & \sqrt{1/2} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline \sqrt{1/2} & \sqrt{1/2} & 0 & 0 & \sqrt{1/2} & \sqrt{1/2} & 1 & 0 \\ \sqrt{1/2} & \sqrt{1/2} & 0 & 0 & \sqrt{1/2} & \sqrt{1/2} & 1 & 0 \\ \hline 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right]$$

(d) The transformed adjacency matrix of G

Figure 1: An illustration of how the transformed adjacency matrix $\tilde{\mathbf{A}}$ of a graph G is constructed based on the colors produced by WL.

elementwise product:

$$\tilde{\mathbf{A}} \odot \tilde{\mathbf{A}}' = \begin{bmatrix} \mathbf{B}_{11} \odot \mathbf{B}'_{11} & \mathbf{B}_{12} \odot \mathbf{B}'_{12} & \dots & \mathbf{B}_{1s} \odot \mathbf{B}'_{1s} \\ \mathbf{B}_{21} \odot \mathbf{B}'_{21} & \mathbf{B}_{22} \odot \mathbf{B}'_{22} & \dots & \mathbf{B}_{2s} \odot \mathbf{B}'_{2s} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{B}_{s1} \odot \mathbf{B}'_{s1} & \mathbf{B}_{s2} \odot \mathbf{B}'_{s2} & \dots & \mathbf{B}_{ss} \odot \mathbf{B}'_{ss} \end{bmatrix}$$

For $i, j \in [s]$, as mentioned above, the $\nu_{\sigma_i}(G) \times \nu_{\sigma_j}(G)$ upper left submatrix of $\mathbf{B}_{ij}$ contains nonzero elements. Likewise, the $\nu_{\sigma_i}(G') \times \nu_{\sigma_j}(G')$ upper left submatrix of $\mathbf{B}'_{ij}$ contains nonzero elements. Therefore, only the first $\min\left(\nu_{\sigma_i}(G), \nu_{\sigma_i}(G')\right)$ rows and first $\min\left(\nu_{\sigma_j}(G), \nu_{\sigma_j}(G')\right)$ columns of $\mathbf{B}_{ij}$ and $\mathbf{B}'_{ij}$ need to be multiplied to each other since the rest of the elements end up having values equal to zero (after the elementwise multiplication). The nonzero elements of $\mathbf{B}_{ij}$ are equal to each other, and the same applies to the nonzero elements of $\mathbf{B}'_{ij}$. As already mentioned, we also have that $\sum_{i=0}^{N_{\sigma_i}} \sum_{j=0}^{N_{\sigma_j}} [\mathbf{B}_{ij}]_{ij}^2 = \mu_{\sigma_i,\sigma_j}(G)$ holds. Therefore, each nonzero element of $\mathbf{B}_{ij}$ is equal to $\sqrt{\frac{\mu_{\sigma_i,\sigma_j}}{\nu_{\sigma_i}(G)\,\nu_{\sigma_j}(G)}}$. Based on the above, the kernel can be computed as follows:

$$k^{(t)}(G, G') = \sum_{\sigma_i \in \Sigma} \sum_{\sigma_j \in \Sigma} \left(\sqrt{\frac{\mu_{\sigma_i,\sigma_j}(G)\,\mu_{\sigma_i,\sigma_j}(G')}{\nu_{\sigma_i}(G)\,\nu_{\sigma_j}(G)\,\nu_{\sigma_i}(G')\,\nu_{\sigma_j}(G')}} \right.$$
$$\left. \min\left(\nu_{\sigma_i}(G), \nu_{\sigma_i}(G')\right) \min\left(\nu_{\sigma_j}(G), \nu_{\sigma_j}(G')\right) \right)$$

The runtime complexity of the WL algorithm with T iterations is $\mathcal{O}(Tm)$. Given a colored graph G, $\nu_{\sigma_i}(G)$ and $\mu_{\sigma_i,\sigma_j}(G)$ can be computed in time linear in the number of vertices and edges of G, respectively. Therefore, the kernel can be computed in linear time.

4.3 Expressive Power

We next investigate the expressive power of the Weisfeiler-Leman graph alignment kernel. Since the kernel imposes an ordering on the vertices of each graph based on the color classes to which they belong, it is likely that its expressive power is related to that of WL.

Proposition 2. *Let $G = (V, E)$ and $G' = (V', E')$ be two graphs where the vertices of both graphs have a degree at least equal to 1, i. e., $deg(v) > 0$, $\forall v \in V \cup V'$. Then, if WL decides G and G' are not isomorphic, the Weisfeiler-Leman graph alignment kernel maps G and G' to different embeddings.*

5 Experimental Evaluation

5.1 Datasets

For the classification task, we evaluated the proposed kernel on standard datasets derived from bioinformatics and chemoinformatics (MUTAG, D&D, NCI1, PROTEINS, ENZYMES), and from social networks (IMDB-BINARY, IMDB-MULTI, REDDIT-BINARY, REDDIT-MULTI-5K, COLLAB) (Morris et al., 2020a). Note that the social network graphs are unlabeled, while all other graph datasets come with vertex labels or vertex attributes. We additionally evaluated the proposed kernel on the ogbg-molhiv dataset, a molecular property prediction dataset from the Open Graph Benchmark (OGB) (Hu et al., 2020), a collection of challenging large-scale datasets.

For the regression task, we conducted an experiment on the ZINC 12K dataset (Dwivedi et al., 2020). ZINC is one

of the most popular molecular datasets where the task is to predict the constrained solubility of molecules, an important chemical property for designing generative graph neural networks for molecules.

5.2 Experimental Setup

In the case of the standard graph classification datasets, we compare the proposed kernel against the following five graph kernels: (1) graphlet kernel (GL) (Shervashidze et al., 2009); (2) shortest path kernel (SP) (Borgwardt and Kriegel, 2005); (3) Weisfeiler-Leman subtree kernel (WL) (Shervashidze et al., 2011); (4) Weisfeiler-Leman optimal assignment kernel (WL-OA) (Kriege et al., 2016); (5) Wasserstein Weisfeiler-Leman graph kernel (WWL) (Togninalli et al., 2019). For the first four kernels, we use the implementations of the kernels contained in the GraKeL library (Siglidis et al., 2020), while for the WWL kernel, we use the implementation provided by the authors. We also compare the proposed kernel against the following five graph neural networks: (1) DGCNN (Zhang et al., 2018); (2) DiffPool (Ying et al., 2018); (3) ECC (Simonovsky and Komodakis, 2017); (4) GIN (Xu et al., 2019b); and (5) GraphSAGE (Hamilton et al., 2017). To evaluate the different methods, we employ the framework proposed by Errica et al. (2020). Therefore, we perform 10-fold cross-validation to obtain an estimate of the generalization performance of each method, while within each fold a model is selected based on a $90\%/10\%$ split of the training set. We use exactly the same splits as the ones employed by Errica et al. (2020), hence, for the common datasets, we use the results reported in that paper. For the remaining datasets, we use the code provided by Errica et al. (2020) to evaluate the five graph neural networks. For graph kernels, to perform graph classification, we employed a C-Support Vector Machine (SVM) classifier. The parameter C of the SVM and the hyperparameters of the kernels were optimized on the training set of each fold only. Specifically, we chose parameters for the graph kernels as follows. For the WL, WL-OA and GAWL kernels, we chose the number of iterations h from $\{2, \ldots, 8\}$. For the GL kernel, we set the number of graphlets to be sampled from each graph equal to 500. For the WWL kernel, the number of iterations was chosen from $\{1, \ldots, 5\}$ and the parameter λ from $\{10^{-4}, \ldots, 10^{1}\}$.

In the case of the ogbg-molhiv dataset, we compare the proposed kernel against the following eight graph neural networks: (1) GCN (Kipf and Welling, 2017); (2) GIN (Xu et al., 2019b); (3) GCN-FLAG (Kong et al., 2020); (4) GIN-FLAG (Kong et al., 2020); (5) PNA (Corso et al., 2020); (6) GSN (Bouritsas et al., 2020); (7) HIMP (Fey et al., 2020); and (8) DGN (Beaini et al., 2020). For all models, we use the results that are reported in the respective papers. All reported results are averaged over 10 runs. Finally, in the case of the ZINC 12K dataset, our list of baselines includes the following eight graph neural network models: (1) GCN (Kipf and Welling, 2017); (2) GraphSAGE (Hamilton et al., 2017); (3) MoNet (Monti et al., 2017); (4) GAT (Veličković et al., 2017); (5) GIN (Xu et al., 2019b); (6) GatedGCN (Bresson and Laurent, 2017); (7) RingGNN (Chen et al., 2019); and (8) 3WLGNN (Maron et al., 2019). For all models, we use the results reported by Dwivedi et al. (2020). All reported results are averaged over 4 runs. More details about the experimental setup are provided in the Appendix.

5.3 Results and Discussion

Graph Classification. We report in Table 1 average prediction accuracies and standard deviations. We observe that the proposed kernel outperforms the baselines on 4 out of the 10 datasets, while it provides the second or third best accuracy on 4 out of the remaining 6 datasets. On some datasets, the proposed kernel outperforms the other approaches with quite wide margins. For instance, on the IMDB-BINARY and COLLAB datasets, our kernel offers respective absolute improvements of 1.7% and 1.0% in accuracy over the second best approach. WL-OA, WWL and GIN are the best performing baselines. Each one of these three methods outperforms all the other approaches on two datasets. Kernel methods achieve better performance than graph neural networks on most datasets. In fact, graph neural networks outperform kernels only on two datasets (i. e., REDDIT-BINARY and REDDIT-MULTI-5K). This demonstrates that kernel methods can still achieve high levels of performance on small and medium-sized datasets. Overall, the proposed kernel exhibits highly competitive performance on the graph classification datasets, while the achieved accuracies follow different patterns from all the baseline methods.

Classification results for the ogbg-molhiv dataset are presented in Table 2. It is interesting to mention that even though the proposed kernel does not take into account the edge attributes of the graphs contained in this dataset, still it outperforms most of the baselines. Specifically, it achieves a ROC-AUC score of 78.34 which is comparable to that of the best performing approach (the DGN model which achieves a score of 79.70). Overall, the results indicate that the proposed kernel can generate useful graph representations even for larger datasets, and that it is very competitive with the state-of-the-art in the graph classification task.

Graph Regression. Table 3 reports the mean absolute error achieved by the different approaches on the ZINC 12K dataset. We observe that the proposed approach achieves high levels of performance. In fact, it is the second best performing approach since it is only outperformed by 3WLGNN-E, one of the most expressive graph neural networks. It is interesting to mention that in this task, the proposed kernel manages to outperform graph neural network

Table 1: Classification accuracy ($\pm$ standard deviation) of the different approaches on the 10 graph classification datasets. OOR means Out of Resources, either time (> 72 hours for a single training) or GPU memory.

Method	MUTAG	D&D	NCI1	PROTEINS	ENZYMES
SP	80.2 ($\pm$ 6.5)	78.1 ($\pm$ 4.1)	72.7 ($\pm$ 1.4)	75.3 ($\pm$ 3.8)	38.3 ($\pm$ 8.0)
GL	80.8 ($\pm$ 6.4)	75.4 ($\pm$ 3.4)	61.8 ($\pm$ 1.7)	71.6 ($\pm$ 3.1)	25.1 ($\pm$ 4.4)
WL	84.6 ($\pm$ 8.3)	78.1 ($\pm$ 2.4)	84.8 ($\pm$ 2.5)	73.8 ($\pm$ 4.4)	50.3 ($\pm$ 5.7)
WL-OA	87.2 ($\pm$ 5.4)	77.6 ($\pm$ 3.0)	**86.3** ($\pm$ 1.6)	**76.2** ($\pm$ 3.9)	58.0 ($\pm$ 5.0)
WWL	86.8 ($\pm$ 6.7)	**79.5** ($\pm$ 3.0)	85.3 ($\pm$ 2.0)	72.6 ($\pm$ 4.8)	**72.8** ($\pm$ 4.8)
DGCNN	84.0 ($\pm$ 6.7)	76.6 ($\pm$ 4.3)	76.4 ($\pm$ 1.7)	72.9 ($\pm$ 3.5)	38.9 ($\pm$ 5.7)
DiffPool	79.8 ($\pm$ 7.1)	75.0 ($\pm$ 3.5)	76.9 ($\pm$ 1.9)	73.7 ($\pm$ 3.5)	59.5 ($\pm$ 5.6)
ECC	75.4 ($\pm$ 6.2)	72.6 ($\pm$ 4.1)	76.2 ($\pm$ 1.4)	72.3 ($\pm$ 3.4)	29.5 ($\pm$ 8.2)
GIN	84.7 ($\pm$ 6.7)	75.3 ($\pm$ 2.9)	80.0 ($\pm$ 1.4)	73.3 ($\pm$ 4.0)	59.6 ($\pm$ 4.5)
GraphSAGE	83.6 ($\pm$ 9.6)	72.9 ($\pm$ 2.0)	76.0 ($\pm$ 1.8)	73.0 ($\pm$ 4.5)	58.2 ($\pm$ 6.0)
GAWL	**87.3** ($\pm$ 6.3)	78.7 ($\pm$ 2.8)	85.9 ($\pm$ 1.2)	74.7 ($\pm$ 3.0)	67.6 ($\pm$ 4.2)

Method	IMDB BINARY	IMDB MULTI	REDDIT BINARY	REDDIT MULTI-5K	COLLAB
SP	57.7 ($\pm$ 4.1)	39.8 ($\pm$ 3.7)	89.0 ($\pm$ 1.0)	51.1 ($\pm$ 2.2)	79.9 ($\pm$ 2.7)
GL	63.3 ($\pm$ 2.7)	39.6 ($\pm$ 3.0)	76.6 ($\pm$ 3.3)	38.1 ($\pm$ 2.3)	71.1 ($\pm$ 1.4)
WL	72.8 ($\pm$ 4.5)	51.2 ($\pm$ 6.5)	74.9 ($\pm$ 1.8)	49.6 ($\pm$ 2.0)	78.0 ($\pm$ 2.0)
WL-OA	72.6 ($\pm$ 5.5)	51.1 ($\pm$ 4.3)	89.0 ($\pm$ 1.3)	54.0 ($\pm$ 1.2)	80.5 ($\pm$ 2.0)
WWL	71.0 ($\pm$ 4.9)	50.0 ($\pm$ 4.6)	86.4 ($\pm$ 0.9)	OOR	OOR
DGCNN	69.2 ($\pm$ 3.0)	45.6 ($\pm$ 3.4)	87.8 ($\pm$ 2.5)	49.2 ($\pm$ 1.2)	71.2 ($\pm$ 1.9)
DiffPool	68.4 ($\pm$ 3.3)	45.6 ($\pm$ 3.4)	89.1 ($\pm$ 1.6)	53.8 ($\pm$ 1.4)	68.9 ($\pm$ 2.0)
ECC	67.7 ($\pm$ 2.8)	43.5 ($\pm$ 3.1)	OOR	OOR	OOR
GIN	71.2 ($\pm$ 3.9)	48.5 ($\pm$ 3.3)	**89.9** ($\pm$ 1.9)	**56.1** ($\pm$ 1.7)	75.6 ($\pm$ 2.3)
GraphSAGE	68.8 ($\pm$ 4.5)	47.6 ($\pm$ 3.5)	84.3 ($\pm$ 1.9)	50.0 ($\pm$ 1.3)	73.9 ($\pm$ 1.7)
GAWL	**74.5** ($\pm$ 4.1)	**51.7** ($\pm$ 5.2)	88.0 ($\pm$ 1.4)	53.4 ($\pm$ 1.3)	**81.5** ($\pm$ 2.0)

Table 2: ROC-AUC score ($\pm$ standard deviation) of the different approaches on the ogbg-molhiv dataset.

Method	**Dataset**
	ogbg-molhiv
GCN	76.06 $\pm$ 0.97
GIN	75.58 $\pm$ 1.40
GCN+FLAG	76.83 $\pm$ 1.02
GIN+FLAG	76.54 $\pm$ 1.14
GSN	77.99 $\pm$ 1.00
HIMP	78.80 $\pm$ 0.82
PNA	79.05 $\pm$ 1.32
DGN	**79.70** $\pm$ 0.97
GAWL	78.34 $\pm$ 0.39

Table 3: Mean absolute error ($\pm$ standard deviation) of the different approaches on the ZINC 12K dataset.

Method	**Dataset**
	ZINC
GCN	0.367 $\pm$ 0.011
GraphSAGE	0.398 $\pm$ 0.002
MoNet	0.292 $\pm$ 0.006
GAT	0.384 $\pm$ 0.007
GIN	0.387 $\pm$ 0.015
GatedGCN	0.435 $\pm$ 0.011
GatedGCN-E	0.282 $\pm$ 0.015
RingGNN-E	0.353 $\pm$ 0.019
3WLGNN	0.407 $\pm$ 0.028
3WLGNN-E	**0.256** $\pm$ 0.054
GAWL	0.277 $\pm$ 0.003

models which are considered state-of-the-art for many machine learning problems on graphs.

Node Attributes. As discussed above, besides discrete node labels, GAWL can also handle vertex attributes. This is one of the main improvements the proposed kernel provides over other kernels such as WL and WL-OA. However, GAWL puts more emphasis on the graph structure since matrix $\mathbf{D}$ emerges from it. We performed the following experiment: we generated 500 graphs. Each graph is either a complete graph or an Erdös-Rényi graph $G_{n,p}$ where $n \in \{10, 11, \ldots, 30\}$ and $p \in \{0.4, 0.5, 0.6, 0.7, 0.8\}$ (both and randomly sampled). Each vertex is annotated with a single attribute randomly chosen from $(0, 1)$ with uniform probability. For some of the generated graphs (sampled with probability 0.5), the attribute of a randomly sampled vertex is replaced with a value equal to -1. Those graphs belong to class 1, while the rest of the graphs (where the attributes of all vertices are from $(0, 1)$) belong to class 0. We split the 500 graphs into training, validation and test sets, compute the kernel matrices (for attributed graphs), and use SVM to classify them. We achieved an accuracy equal to 90% (the dataset is almost perfectly balanced), which demonstrates that the GAWL kernel can perform well even when vertex attributes are more important than the graph structure itself.

Runtime Analysis. We next empirically measure the running time of the proposed kernel and compare it against those of three baseline kernels, namely WL, WL-OA and WWL. We report in Table 4 below CPU runtimes for computing each kernel matrix as measured on a Intel Xeon W-2123 CPU @ 3.60GHz with 64Gb of RAM. We observe that the WL is the fastest kernel followed by GAWL, WL-OA and WWL in that order. WWL is the slowest kernel,

Table 4: CPU runtime for kernel matrix computation of the proposed kernel and three baseline kernels on 5 graph classification datasets.

Dataset	Method			
	WL	WL-OA	WWL	GAWL
MUTAG	0.05s	0.54s	13.05s	1.15s
DD	11.99s	1h 32m 55s	8h 27m 43s	10m 10s
NCI1	4s	54m 36s	2h 27m 1s	6m 44s
PROTEINS	1.44s	4m 11s	27m 28s	37.81s
ENZYMES	0.57s	35.08s	5m 23s	25.77s

and on some datasets, its running time is much higher than that of the other three kernels. It is worth mentioning that even if the proposed kernel is slower than the WL kernel, its computing times are by no means prohibitive. Overall, the experiments demonstrate that the running time of the proposed kernel is attractive.

Limitations. To deal with learning problems, the proposed method needs first to construct the Gram matrix, i. e., precompute the kernel value for all pairs of training samples, which can be done in time quadratic in the number of samples. Thus, the kernel becomes highly inefficient in case the number of samples is very large. In such scenarios, approximation methods can be employed. For instance, on ogbg-molhiv, we utilized the Nyström method (Williams and Seeger, 2001).

6 Conclusion

In this paper, we proposed a novel kernel between graphs which permutes the adjacency matrices of input graphs such that structurally equivalent vertices are in corresponding positions of the adjacency matrices. The structural equivalence of vertices is determined based on the output of the WL algorithm. Then, the kernel compares the adjacency matrices of graphs to each other using a linear kernel. We also proposed an efficient computation scheme which makes the time complexity of the kernel very attractive. The effectiveness of the proposed kernel was empirically tested on standard datasets in the tasks of graph classification and graph regression. The obtained results indicate that the kernel is competitive with traditional and state-of-the-art methods.

In terms of future directions of research, we would like to apply the proposed method to align graphs to each other. Given two graphs G_1, G_2 and their respective soft permutation matrices $\mathbf{D}_1 \in \mathbb{R}^{N \times n_1}$ and $\mathbf{D}_2 \in \mathbb{R}^{N \times n_2}$ (where n_1, n_2 denote the number of nodes of G_1 and G_2), we could compute a matrix $\mathbf{T} = \mathbf{D}_1^\top \mathbf{D}_2 \in \mathbb{R}^{n_1 \times n_2}$ which contains the similarities of the nodes of the two graphs. Then, we could seek for the optimal matching between the nodes of the two graphs using some optimal transport technique.

Acknowledgements

G.N. is supported by the French National research agency via the AML-HELAS (ANR-19-CHIA-0020) project.

References

Arvind, V., Köbler, J., Kuhnert, S., and Vasudev, Y. (2012). Approximate Graph Isomorphism. In *International Symposium on Mathematical Foundations of Computer Science*, pages 100–111.

Arvind, V., Köbler, J., Rattan, G., and Verbitsky, O. (2015). On the power of color refinement. In *International Symposium on Fundamentals of Computation Theory*, pages 339–350.

Babai, L. and Kucera, L. (1979). Canonical labelling of graphs in linear average time. In *20th Annual Symposium on Foundations of Computer Science*, pages 39–46.

Beaini, D., Passaro, S., Létourneau, V., Hamilton, W. L., Corso, G., and Liò, P. (2020). Directional Graph Networks. *arXiv preprint arXiv:2010.02863*.

Borgwardt, K., Ghisu, E., Llinares-López, F., O'Bray, L., Rieck, B., et al. (2020). Graph Kernels: State-of-the-Art and Future Challenges. *Foundations and Trends® in Machine Learning*, 13(5-6).

Borgwardt, K., Ong, C., Schönauer, S., Vishwanathan, S., Smola, A., and Kriegel, H. (2005). Protein function prediction via graph kernels. *Bioinformatics*, 21(Suppl. 1):i47–i56.

Borgwardt, K. M. and Kriegel, H. (2005). Shortest-path kernels on graphs. In *Proceedings of the 5th International Conference on Data Mining*, pages 74–81.

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

Bresson, X. and Laurent, T. (2017). Residual Gated Graph ConvNets. *arXiv preprint arXiv:1711.07553*.

Chen, Z., Villar, S., Chen, L., and Bruna, J. (2019). On the equivalence between graph isomorphismtesting and function approximation with GNNs. In *Advances in Neural Information Processing Systems*.

Corso, G., Cavalleri, L., Beaini, D., Liò, P., and Veličković, P. (2020). Principal Neighbourhood Aggregation for Graph Nets. In *Advances in Neural Information Processing Systems*, volume 33.

Debnath, A., Lopez de Compadre, R., Debnath, G., Shusterman, A., and Hansch, C. (1991). Structure-Activity Relationship of Mutagenic Aromatic and Heteroaromatic Nitro Compounds. Correlation with Molecular Orbital Energies and Hydrophobicity. *Journal of Medicinal Chemistry*, 34(2):786–797.

Dobson, P. and Doig, A. (2003). Distinguishing Enzyme Structures from Non-enzymes Without Alignments. *Journal of Molecular Biology*, 330(4):771–783.

Dong, Y. and Sawin, W. (2020). COPT: Coordinated Optimal Transport on Graphs. In *Advances in Neural Information Processing Systems*, pages 19327–19338.

Dwivedi, V. P., Joshi, C. K., Laurent, T., Bengio, Y., and Bresson, X. (2020). Benchmarking Graph Neural Networks. *arXiv preprint arXiv:2003.00982*.

Errica, F., Podda, M., Bacciu, D., and Micheli, A. (2020). A Fair Comparison of Graph Neural Networks for Graph Classification. In *Proceedings of the International Conference on Learning Representations*.

Feragen, A., Kasenburg, N., Petersen, J., de Bruijne, M., and Borgwardt, K. M. (2013). Scalable kernels for graphs with continuous attributes. In *Advances in Neural Information Processing Systems*, pages 216–224.

Fey, M., Yuen, J.-G., and Weichert, F. (2020). Hierarchical Inter-Message Passing for Learning on Molecular Graphs. *arXiv preprint arXiv:2006.12179*.

Fröhlich, H., Wegner, J. K., Sieker, F., and Zell, A. (2005). Optimal Assignment Kernels for Attributed Molecular Graphs. In *Proceedings of the 22nd International Conference on Machine Learning*, pages 225–232.

Gärtner, T., Flach, P., and Wrobel, S. (2003). On Graph Kernels: Hardness Results and Efficient Alternatives. In *Learning Theory and Kernel Machines*, pages 129–143.

Grohe, M., Rattan, G., and Woeginger, G. J. (2018). Graph Similarity and Approximate Isomorphism. In *Proceedings of the 43rd International Symposium on Mathematical Foundations of Computer Science*.

Hamilton, W., Ying, Z., and Leskovec, J. (2017). Inductive Representation Learning on Large Graphs. In *Advances in Neural Information Processing Systems*, pages 1024–1034.

Haussler, D. (1999). Convolution kernels on discrete structures. Technical report, Technical report, Department of Computer Science, University of California ....

Horváth, T., Gärtner, T., and Wrobel, S. (2004). Cyclic Pattern Kernels for Predictive Graph Mining. In *Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 158–167.

Hu, W., Fey, M., Zitnik, M., Dong, Y., Ren, H., Liu, B., Catasta, M., and Leskovec, J. (2020). Open Graph Benchmark: Datasets for Machine Learning on Graphs. *arXiv preprint arXiv:2005.00687*.

Johansson, F., Jethava, V., Dubhashi, D., and Bhattacharyya, C. (2014). Global graph kernels using geometric embeddings. In *Proceedings of the 31st International Conference on Machine Learning*, pages 694–702.

Kalofolias, J., Welke, P., and Vreeken, J. (2021). SUSAN: The Structural Similarity Random Walk Kernel. In *Proceedings of the 2021 SIAM International Conference on Data Mining*, pages 298–306.

Kashima, H., Tsuda, K., and Inokuchi, A. (2003). Marginalized Kernels between Labeled Graphs. In *Proceedings of the 20th International Conference on Machine Learning*, pages 321–328.

Kipf, T. N. and Welling, M. (2017). Semi-supervised classification with graph convolutional networks. In *In 5th International Conference on Learning Representations*.

Kolouri, S., Naderializadeh, N., Rohde, G. K., and Hoffmann, H. (2021). Wasserstein embedding for graph learning. In *9th International Conference on Learning Representations*.

Kondor, R. and Pan, H. (2016). The Multiscale Laplacian Graph Kernel. In *Advances in Neural Information Processing Systems*, pages 2982–2990.

Kong, K., Li, G., Ding, M., Wu, Z., Zhu, C., Ghanem, B., Taylor, G., and Goldstein, T. (2020). Flag: Adversarial Data Augmentation for Graph Neural Networks. *arXiv preprint arXiv:2010.09891*.

Kriege, N. and Mutzel, P. (2012). Subgraph Matching Kernels for Attributed Graphs. In *Proceedings of the 29th International Coference on International Conference on Machine Learning*, pages 291–298.

Kriege, N. M., Giscard, P.-L., and Wilson, R. (2016). On Valid Optimal Assignment Kernels andApplications to Graph Classification. In *Advances in Neural Information Processing Systems*, pages 1623–1631.

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

Kriege, N. M., Morris, C., Rey, A., and Sohler, C. (2018). A Property Testing Framework for the Theoretical Expressivity of Graph Kernels. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, pages 2348–2354.

Ma, K., Wan, P., and Zhang, D. (2020). Transport based Graph Kernels. *arXiv preprint arXiv:2011.00745*.

Mahé, P., Ueda, N., Akutsu, T., Perret, J.-L., and Vert, J.-P. (2004). Extensions of Marginalized Graph Kernels. In *Proceedings of the 21st International Conference on Machine Learning*.

Maron, H., Ben-Hamu, H., Serviansky, H., and Lipman, Y. (2019). Provably Powerful Graph Networks. In *Advances in Neural Information Processing Systems*, pages 2156–2167.

Monti, F., Boscaini, D., Masci, J., Rodola, E., Svoboda, J., and Bronstein, M. M. (2017). Geometric deep learning on graphs and manifolds using mixture model CNNs. In

Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 5115–5124.

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

Morris, C., Kriege, N. M., Bause, F., Kersting, K., Mutzel, P., and Neumann, M. (2020a). TUDataset: A collection of benchmark datasets for learning with graphs. *arXiv preprint arXiv:2007.08663*.

Morris, C., Lipman, Y., Maron, H., Rieck, B., Kriege, N. M., Grohe, M., Fey, M., and Borgwardt, K. (2021). Weisfeiler and Leman go Machine Learning: The Story so far. *arXiv preprint arXiv:2112.09992*.

Morris, C., Rattan, G., Kiefer, S., and Ravanbakhsh, S. (2022). SpeqNets: Sparsity-aware Permutation-equivariant Graph Networks. In *Proceedings of the 39th International Conference on Machine Learning*, pages 16017–16042.

Morris, C., Rattan, G., and Mutzel, P. (2020b). Weisfeiler and Leman go sparse: Towards scalable higher-order graph embeddings. In *Advances in Neural Information Processing Systems*, pages 21824–21840.

Nikolentzos, G., Meladianos, P., and Vazirgiannis, M. (2017). Matching Node Embeddings for Graph Similarity. In *Proceedings of the 31st AAAI Conference on Artificial Intelligence*, pages 2429–2435.

Nikolentzos, G., Siglidis, G., and Vazirgiannis, M. (2021). Graph Kernels: A Survey. *Journal of Artificial Intelligence Research*, 72:943–1027.

Petric Maretic, H., El Gheche, M., Chierchia, G., and Frossard, P. (2019). GOT: An Optimal Transport framework for Graph comparison. In *Advances in Neural Information Processing Systems*, pages 13876–13887.

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

Schiavinato, M., Gasparetto, A., and Torsello, A. (2015). Transitive Assignment Kernels for Structural Classification. In *International Workshop on Similarity-Based Pattern Recognition*, pages 146–159.

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

Shervashidze, N., Vishwanathan, S., Petri, T., Mehlhorn, K., and Borgwardt, K. M. (2009). Efficient graphlet kernels for large graph comparison. In *Proceedings of the 12th International Conference on Artificial Intelligence and Statistics*, pages 488–495.

Siglidis, G., Nikolentzos, G., Limnios, S., Giatsidis, C., Skianis, K., and Vazirgiannis, M. (2020). GraKeL: A Graph Kernel Library in Python. *Journal of Machine Learning Research*, 21:1–5.

Simonovsky, M. and Komodakis, N. (2017). Dynamic Edge-Conditioned Filters in Convolutional Neural Networks on Graphs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3693–3702.

Sugiyama, M. and Borgwardt, K. (2015). Halting in Random Walk Kernels. In *Advances in Neural Information Processing Systems*, pages 1639–1647.

Togninalli, M., Ghisu, E., Llinares-López, F., Rieck, B., and Borgwardt, K. (2019). Wasserstein Weisfeiler-Lehman Graph Kernels. In *Advances in Neural Information Processing Systems*, pages 6439–6449.

Vayer, T., Courty, N., Tavenard, R., and Flamary, R. (2019). Optimal Transport for structured data with application on graphs. In *Proceedings of the 36th International Conference on Machine Learning*, pages 6275–6284.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., and Bengio, Y. (2017). Graph Attention Networks. In *6th International Conference on Learning Representations*.

Vert, J.-P. (2008). The optimal assignment kernel is not positive definite. *arXiv preprint arXiv:0801.4061*.

Vishwanathan, S. V. N., Schraudolph, N. N., Kondor, R., and Borgwardt, K. M. (2010). Graph Kernels. *The Journal of Machine Learning Research*, 11:1201–1242.

Wale, N., Watson, I., and Karypis, G. (2008). Comparison of descriptor spaces for chemical compound retrieval and classification. *Knowledge and Information Systems*, 14(3):347–375.

Wijesinghe, A., Wang, Q., and Gould, S. (2021). A Regularized Wasserstein Framework for Graph Kernels. *arXiv preprint arXiv:2110.02554*.

Williams, C. and Seeger, M. (2001). Using the Nyström Method to Speed up Kernel Machines. In *Advances in Neural Information Processing Systems*, pages 682–688.

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

Xu, H., Luo, D., Zha, H., and Duke, L. C. (2019a). Gromov-Wasserstein Learning for Graph Matching and Node Embedding. In *Proceedings of the 36th International Conference on Machine Learning*, pages 6932–6941.

Xu, K., Hu, W., Leskovec, J., and Jegelka, S. (2019b). How Powerful are Graph Neural Networks? In *7th International Conference on Learning Representations*.

Yanardag, P. and Vishwanathan, S. (2015). Deep Graph Kernels. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1365–1374.

Ying, Z., You, J., Morris, C., Ren, X., Hamilton, W., and Leskovec, J. (2018). Hierarchical Graph Representation Learning with Differentiable Pooling. In *Advances in Neural Information Processing Systems*, pages 4801–4811.

Zhang, M., Cui, Z., Neumann, M., and Chen, Y. (2018). An End-to-End Deep Learning Architecture for Graph Classification. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence*, pages 4438–4445.

The Appendix is organized as follows. In sections A and B, we prove Propositions 1 and 2, respectively. In section C, we present a variant of the proposed kernel which takes colors of all iterations into account, while in section D, we discuss how the proposed kernel is different from the Weisfeiler-Leman optimal assignment kernel. In section E, we provide further details about the experimental setup. Finally, in section F, we give more details about the datasets we used in our experiments.

A Proof of Proposition 1

Let vec denote the vectorization operator which transforms a matrix into a vector by stacking its columns. Then, given a graph G_i, its adjacency matrix $\mathbf{A}_i \in \mathbb{R}^{n \times n}$ and a permutation matrix $\mathbf{P}_i^* \in \mathbb{R}^{n \times n}$, let $\mathbf{v}_i = \text{vec}(\mathbf{P}_i^* \, \mathbf{A}_i \, \mathbf{P}_i^{*\top})$ where $\mathbf{v}_i \in \mathbb{R}^{n^2}$. Then, we have:

$$k(G_1, G_2) = \sum_{i=1}^{n} \sum_{j=1}^{n} \left[\mathbf{P}_i^* \, \mathbf{A}_i \, \mathbf{P}_i^{*\top} \odot \mathbf{P}_j^* \, \mathbf{A}_j \, \mathbf{P}_j^{*\top} \right]_{ij} = \langle \mathbf{v}_i, \mathbf{v}_j \rangle$$

and thus the above function corresponds to an inner product between the vector representations of the two graphs, and is therefore a valid kernel. The kernel computes an explicit mapping from graphs to vector-valued feature space and thus corresponds to the dot product of explicitly computed feature maps.

B Proof of Proposition 2

Assume that WL decides G and G' are not isomorphic. Then, the multiplicity of at least one color in the first graph is different from the multiplicity of the same color in the second graph. Without loss of generality, suppose that this color is σ_i. Then, we have that $\nu_{\sigma_i}(G) \neq \nu_{\sigma_i}(G')$. Since, we have assumed that G and G' are both connected (i. e., each vertex has a degree at least equal to 1), there exists at least one vertex of G colored σ_j that is connected to the vertex (or vertices) colored σ_i. Therefore, the $\nu_{\sigma_i}(G) \times \nu_{\sigma_j}(G)$ upper left submatrix of matrix $\mathbf{B}_{ij}$ will contain nonzero values. Likewise, the $\nu_{\sigma_i}(G') \times \nu_{\sigma_j}(G')$ upper left submatrix of matrix $\mathbf{B}'_{ij}$ will contain nonzero values (in case at least a vertex colored σ_i is connected to a vertex colored σ_j in G'). The rest of the elements of both $\mathbf{B}_{ij}$ and $\mathbf{B}'_{ij}$ are equal to zero. Since $\nu_{\sigma_i}(G) \neq \nu_{\sigma_i}(G')$, it turns out that $\mathbf{B}_{ij} \neq \mathbf{B}'_{ij}$. Thus, the two graphs have different transformed adjacency matrices, i. e., $\tilde{\mathbf{A}} \neq \tilde{\mathbf{A}}'$ since at least one of their blocks (i. e., submatrices) are different from each other. Thus, the kernel maps the two graphs to different embeddings.

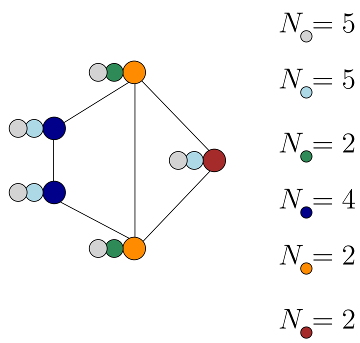
C Using Colors of All Iterations

Let G, G' be two graphs, and let also $\mathbf{D}_1, \mathbf{D}'_1, \ldots, \mathbf{D}_T, \mathbf{D}'_T$ denote the transformation matrices of the two graphs that emerge after $1, \ldots, T$ iterations of WL, respectively. Then, $\mathbf{D}_t, \mathbf{D}'_t \in \mathbb{R}^{N_t \times n}$, and the kernel between the two graphs is computed as follows:

$$\begin{aligned} k(G, G') = k^{(1)}(G, G') + \ldots + k^{(T)}(G, G') &= \sum_{t=1}^{T} \sum_{i=1}^{N_t} \sum_{j=1}^{N_t} \left[\sqrt{\mathbf{D}_t \, \mathbf{A} \, \mathbf{D}_t^\top} \odot \sqrt{\mathbf{D}'_t \, \mathbf{A}' \, \mathbf{D}_t'^\top} \right]_{ij} \\ &= \sum_{t=1}^{T} \sum_{i=1}^{N_t} \sum_{j=1}^{N_t} \left[\tilde{\mathbf{A}}_t \odot \tilde{\mathbf{A}}'_t \right]_{ij} \\ &= \sum_{i=1}^{N} \sum_{j=1}^{N} \left[\tilde{\mathbf{A}}_{block\_diag} \odot \tilde{\mathbf{A}}'_{block\_diag} \right]_{ij} \end{aligned}$$

where $N = N_0 + \ldots + N_T$ and $\tilde{\mathbf{A}}_{block\_diag} \in \mathbb{R}^{N \times N}$ is a block diagonal matrix of the following form:

$$\tilde{\mathbf{A}}_{block\_diag} = \begin{bmatrix} \tilde{\mathbf{A}}_1 & 0 & \ldots & 0 \\ 0 & \tilde{\mathbf{A}}_2 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & \tilde{\mathbf{A}}_T \end{bmatrix}$$



(a) A graph G and the largest number of vertices colored with each color

$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \end{bmatrix}$$

(b) The adjacency matrix of G

$$\mathbf{D} = \begin{bmatrix} 1/5 & 1/5 & 1/5 & 1/5 & 1/5 \\ 1/5 & 1/5 & 1/5 & 1/5 & 1/5 \\ 1/5 & 1/5 & 1/5 & 1/5 & 1/5 \\ 1/5 & 1/5 & 1/5 & 1/5 & 1/5 \\ 1/5 & 1/5 & 1/5 & 1/5 & 1/5 \\ \hline 0 & 0 & 1/3 & 1/3 & 1/3 \\ 0 & 0 & 1/3 & 1/3 & 1/3 \\ 0 & 0 & 1/3 & 1/3 & 1/3 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ \hline 1/2 & 1/2 & 0 & 0 & 0 \\ 1/2 & 1/2 & 0 & 0 & 0 \\ \hline 0 & 0 & 1/2 & 1/2 & 0 \\ 0 & 0 & 1/2 & 1/2 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ \hline 1/2 & 1/2 & 0 & 0 & 0 \\ 1/2 & 1/2 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

(c) The permutation matrix associated with G

$$\tilde{\mathbf{A}} = \left[\begin{array}{ccccc|ccccc|cc|cccc|cc|cc}
\sqrt{\frac{12}{25}} & \sqrt{\frac{12}{25}} & \sqrt{\frac{12}{25}} & \sqrt{\frac{12}{25}} & \sqrt{\frac{12}{25}} & \sqrt{\frac{2}{5}} & \sqrt{\frac{2}{5}} & \sqrt{\frac{2}{5}} & 0 & 0 & \sqrt{\frac{3}{5}} & \sqrt{\frac{3}{5}} & \sqrt{\frac{2}{5}} & \sqrt{\frac{2}{5}} & 0 & 0 & \sqrt{\frac{3}{5}} & \sqrt{\frac{3}{5}} & \sqrt{\frac{2}{5}} & 0 \\
\sqrt{\frac{12}{25}} & \sqrt{\frac{12}{25}} & \sqrt{\frac{12}{25}} & \sqrt{\frac{12}{25}} & \sqrt{\frac{12}{25}} & \sqrt{\frac{2}{5}} & \sqrt{\frac{2}{5}} & \sqrt{\frac{2}{5}} & 0 & 0 & \sqrt{\frac{3}{5}} & \sqrt{\frac{3}{5}} & \sqrt{\frac{2}{5}} & \sqrt{\frac{2}{5}} & 0 & 0 & \sqrt{\frac{3}{5}} & \sqrt{\frac{3}{5}} & \sqrt{\frac{2}{5}} & 0 \\
\sqrt{\frac{12}{25}} & \sqrt{\frac{12}{25}} & \sqrt{\frac{12}{25}} & \sqrt{\frac{12}{25}} & \sqrt{\frac{12}{25}} & \sqrt{\frac{2}{5}} & \sqrt{\frac{2}{5}} & \sqrt{\frac{2}{5}} & 0 & 0 & \sqrt{\frac{3}{5}} & \sqrt{\frac{3}{5}} & \sqrt{\frac{2}{5}} & \sqrt{\frac{2}{5}} & 0 & 0 & \sqrt{\frac{3}{5}} & \sqrt{\frac{3}{5}} & \sqrt{\frac{2}{5}} & 0 \\
\sqrt{\frac{12}{25}} & \sqrt{\frac{12}{25}} & \sqrt{\frac{12}{25}} & \sqrt{\frac{12}{25}} & \sqrt{\frac{12}{25}} & \sqrt{\frac{2}{5}} & \sqrt{\frac{2}{5}} & \sqrt{\frac{2}{5}} & 0 & 0 & \sqrt{\frac{3}{5}} & \sqrt{\frac{3}{5}} & \sqrt{\frac{2}{5}} & \sqrt{\frac{2}{5}} & 0 & 0 & \sqrt{\frac{3}{5}} & \sqrt{\frac{3}{5}} & \sqrt{\frac{2}{5}} & 0 \\
\sqrt{\frac{12}{25}} & \sqrt{\frac{12}{25}} & \sqrt{\frac{12}{25}} & \sqrt{\frac{12}{25}} & \sqrt{\frac{12}{25}} & \sqrt{\frac{2}{5}} & \sqrt{\frac{2}{5}} & \sqrt{\frac{2}{5}} & 0 & 0 & \sqrt{\frac{3}{5}} & \sqrt{\frac{3}{5}} & \sqrt{\frac{2}{5}} & \sqrt{\frac{2}{5}} & 0 & 0 & \sqrt{\frac{3}{5}} & \sqrt{\frac{3}{5}} & \sqrt{\frac{2}{5}} & 0 \\
\hline
\sqrt{\frac{2}{5}} & \sqrt{\frac{2}{5}} & \sqrt{\frac{2}{5}} & \sqrt{\frac{2}{5}} & \sqrt{\frac{2}{5}} & \sqrt{\frac{2}{9}} & \sqrt{\frac{2}{9}} & \sqrt{\frac{2}{9}} & 0 & 0 & \sqrt{\frac{2}{3}} & \sqrt{\frac{2}{3}} & \sqrt{\frac{1}{3}} & \sqrt{\frac{1}{3}} & 0 & 0 & \sqrt{\frac{2}{3}} & \sqrt{\frac{2}{3}} & 0 & 0 \\
\sqrt{\frac{2}{5}} & \sqrt{\frac{2}{5}} & \sqrt{\frac{2}{5}} & \sqrt{\frac{2}{5}} & \sqrt{\frac{2}{5}} & \sqrt{\frac{2}{9}} & \sqrt{\frac{2}{9}} & \sqrt{\frac{2}{9}} & 0 & 0 & \sqrt{\frac{2}{3}} & \sqrt{\frac{2}{3}} & \sqrt{\frac{1}{3}} & \sqrt{\frac{1}{3}} & 0 & 0 & \sqrt{\frac{2}{3}} & \sqrt{\frac{2}{3}} & 0 & 0 \\
\sqrt{\frac{2}{5}} & \sqrt{\frac{2}{5}} & \sqrt{\frac{2}{5}} & \sqrt{\frac{2}{5}} & \sqrt{\frac{2}{5}} & \sqrt{\frac{2}{9}} & \sqrt{\frac{2}{9}} & \sqrt{\frac{2}{9}} & 0 & 0 & \sqrt{\frac{2}{3}} & \sqrt{\frac{2}{3}} & \sqrt{\frac{1}{3}} & \sqrt{\frac{1}{3}} & 0 & 0 & \sqrt{\frac{2}{3}} & \sqrt{\frac{2}{3}} & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\hline
\sqrt{\frac{3}{5}} & \sqrt{\frac{3}{5}} & \sqrt{\frac{3}{5}} & \sqrt{\frac{3}{5}} & \sqrt{\frac{3}{5}} & \sqrt{\frac{2}{3}} & \sqrt{\frac{2}{3}} & \sqrt{\frac{2}{3}} & 0 & 0 & \sqrt{\frac{1}{2}} & \sqrt{\frac{1}{2}} & \sqrt{\frac{1}{2}} & \sqrt{\frac{1}{2}} & 0 & 0 & \sqrt{\frac{1}{2}} & \sqrt{\frac{1}{2}} & 1 & 0 \\
\sqrt{\frac{3}{5}} & \sqrt{\frac{3}{5}} & \sqrt{\frac{3}{5}} & \sqrt{\frac{3}{5}} & \sqrt{\frac{3}{5}} & \sqrt{\frac{2}{3}} & \sqrt{\frac{2}{3}} & \sqrt{\frac{2}{3}} & 0 & 0 & \sqrt{\frac{1}{2}} & \sqrt{\frac{1}{2}} & \sqrt{\frac{1}{2}} & \sqrt{\frac{1}{2}} & 0 & 0 & \sqrt{\frac{1}{2}} & \sqrt{\frac{1}{2}} & 1 & 0 \\
\hline
\sqrt{\frac{2}{5}} & \sqrt{\frac{2}{5}} & \sqrt{\frac{2}{5}} & \sqrt{\frac{2}{5}} & \sqrt{\frac{2}{5}} & \sqrt{\frac{1}{3}} & \sqrt{\frac{1}{3}} & \sqrt{\frac{1}{3}} & 0 & 0 & \sqrt{\frac{1}{2}} & \sqrt{\frac{1}{2}} & \sqrt{\frac{1}{2}} & \sqrt{\frac{1}{2}} & 0 & 0 & \sqrt{\frac{1}{2}} & \sqrt{\frac{1}{2}} & 0 & 0 \\
\sqrt{\frac{2}{5}} & \sqrt{\frac{2}{5}} & \sqrt{\frac{2}{5}} & \sqrt{\frac{2}{5}} & \sqrt{\frac{2}{5}} & \sqrt{\frac{1}{3}} & \sqrt{\frac{1}{3}} & \sqrt{\frac{1}{3}} & 0 & 0 & \sqrt{\frac{1}{2}} & \sqrt{\frac{1}{2}} & \sqrt{\frac{1}{2}} & \sqrt{\frac{1}{2}} & 0 & 0 & \sqrt{\frac{1}{2}} & \sqrt{\frac{1}{2}} & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\hline
\sqrt{\frac{3}{5}} & \sqrt{\frac{3}{5}} & \sqrt{\frac{3}{5}} & \sqrt{\frac{3}{5}} & \sqrt{\frac{3}{5}} & \sqrt{\frac{2}{3}} & \sqrt{\frac{2}{3}} & \sqrt{\frac{2}{3}} & 0 & 0 & \sqrt{\frac{1}{2}} & \sqrt{\frac{1}{2}} & \sqrt{\frac{1}{2}} & \sqrt{\frac{1}{2}} & 0 & 0 & \sqrt{\frac{1}{2}} & \sqrt{\frac{1}{2}} & 1 & 0 \\
\sqrt{\frac{3}{5}} & \sqrt{\frac{3}{5}} & \sqrt{\frac{3}{5}} & \sqrt{\frac{3}{5}} & \sqrt{\frac{3}{5}} & \sqrt{\frac{2}{3}} & \sqrt{\frac{2}{3}} & \sqrt{\frac{2}{3}} & 0 & 0 & \sqrt{\frac{1}{2}} & \sqrt{\frac{1}{2}} & \sqrt{\frac{1}{2}} & \sqrt{\frac{1}{2}} & 0 & 0 & \sqrt{\frac{1}{2}} & \sqrt{\frac{1}{2}} & 1 & 0 \\
\hline
\sqrt{\frac{2}{5}} & \sqrt{\frac{2}{5}} & \sqrt{\frac{2}{5}} & \sqrt{\frac{2}{5}} & \sqrt{\frac{2}{5}} & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{array}\right]$$

(d) The transformed adjacency matrix of G

Figure 2: An illustration of how the transformed adjacency matrix $\tilde{\mathbf{A}}$ of a graph G is constructed based on the initial colors and the colors produced by all iterations of WL.

Based on the above, it is clear that given some number of WL iterations T, the proposed kernel takes also the colors of the previous iterations into account to compute the kernel value.

An alternative computation scheme would be to construct a single transformation matrix $\mathbf{D} \in \mathbb{R}^{N \times n}$ by concatenating vertically matrices $\mathbf{D}_1, \ldots, \mathbf{D}_T$, and then to apply this matrix to the adjacency matrix of the graph (which will give rise to a matrix $\tilde{\mathbf{A}} \in \mathbb{R}^{N \times N}$), and compute the kernel between two graphs as follows:

$$k(G, G') = \sum_{i=1}^{N} \sum_{j=1}^{N} \left[\tilde{\mathbf{A}} \odot \tilde{\mathbf{A}}' \right]_{ij}$$

The above computation scheme is illustrated in Figure 2 and is more expressive than the previous one since it takes into account relationships between colors that emerge from different iterations of the WL algorithm, but is computationally more expensive since it leads to transformed adjacency matrices that contain more non-zero elements.

D Link to Weisfeiler-Leman Optimal Assignment Kernel

The Weisfeiler-Leman optimal assignment kernel (WL-OA) belongs to the family of assignment kernels and its formulation is similar to the final formulation of the proposed GAWL kernel. However, the motivation behind the proposed kernel is fundamentally different from that of WL-OA since it deals with the graph comparison problem from a different perspective: that of graph alignment.

WL-OA maps each graph into a histogram that measures the frequencies (i. e., number of occurrences) of the colors that emerge in the different iterations of the WL algorithm. Let $\nu_{\sigma_i}(G)$ denote the number of vertices of G colored σ_i. Given two graphs G, G' and the set of colors $\{\sigma_1, \sigma_2, \ldots, \sigma_{|\Sigma|}\}$ that are produced by WL, the WL-OA kernel only needs $\nu_{\sigma_1}(G), \nu_{\sigma_2}(G), \ldots, \nu_{\sigma_{|\Sigma|}}(G)$ and $\nu_{\sigma_1}(G'), \nu_{\sigma_2}(G'), \ldots, \nu_{\sigma_{|\Sigma|}}(G')$ to compute the kernel between the two graphs by histogram intersection. Thus, the kernel compares color frequencies using the histogram intersection kernel. On the other hand, the proposed kernel is different from WL-OA since it considers pairs of colors instead of single colors and also the number of edges between vertices of these colors. Indeed, this is clear from the kernel's definition provided in subsection 4.2:

$$k^{(t)}(G, G') = \sum_{\sigma_i \in \Sigma} \sum_{\sigma_j \in \Sigma} \left(\sqrt{\frac{\mu_{\sigma_i,\sigma_j}(G)\, \mu_{\sigma_i,\sigma_j}(G')}{\nu_{\sigma_i}(G)\, \nu_{\sigma_j}(G)\, \nu_{\sigma_i}(G')\, \nu_{\sigma_j}(G')}} \min\left(\nu_{\sigma_i}(G), \nu_{\sigma_i}(G')\right) \min\left(\nu_{\sigma_j}(G), \nu_{\sigma_j}(G')\right) \right)$$

Even though the two kernels are similar to each other, it turns out that they perform differently in the experiments of Table 1.

We also performed an experiment where we computed the kernel matrix for the graphs of the IMDB-BINARY dataset using WL-OA, WL and the proposed GAWL kernel for different iterations of the WL algorithm (from 1 to 4). We then computed the Pearson correlation between the kernel values generated by the different kernels. We found that GAWL is more correlated to WL than to WL-OA (correlations between $0.84 - 0.93$ vs. correlations between $0.69 - 0.79$). Furthermore, as discussed above, the proposed kernel can handle node and edge attributes, while its running time is smaller than that of WL-OA (implementation from GraKeL package (Siglidis et al., 2020)).

E Details about Experimental Setup

For both ogbg-molhiv and ZINC 12K, we used the available predefined splits. For ogbg-molhiv, a different kernel was computed for each one of the 9 categorical vertex attributes and the emerging values were summed to produce a single kernel value. Edge attributes were ignored. We used the Nyström method (Williams and Seeger, 2001) to obtain a low-rank approximation of the kernel matrix. We set the rank equal to $2,000$. For ZINC 12K, we computed the entire kernel matrix, and we used kernel PCA to obtain a vector of dimension $4,000$ for each graph. The WL algorithm took edge labels into account while updating vertex colors. For both datasets, the emerging graph representations were fed to a multi-layer perceptron consisting of two hidden layers with a hidden-dimension size of $1,024$ and 512, respectively. For the classification and regression tasks, we used the binary cross entropy and l_1 loss for the optimization, respectively. We used a dropout layer with $p = 0.6$ between the two hidden layers in the case of ogbg-molhiv and no dropout in the case of ZINC 12K. Moreover, we used the Adam optimizer with a learning rate of 10^{-3} and trained the network for 10 epochs. The number of WL iterations of the proposed GAWL kernel was chosen from $\{1, \ldots, 4\}$. The whole training and evaluation procedure was repeated 10 times in the case of the ogbg-molhiv dataset and 4 times in the case of the ZINC 12K dataset, respectively.

F Datasets

We evaluated the proposed kernel on 10 publicly available graph classification datasets including 5 bio/chemo-informatics datasets: MUTAG, D&D, NCI1, PROTEINS and ENZYMES, as well as 5 social interaction datasets: IMDB-BINARY, IMDB-MULTI, REDDIT-BINARY, REDDIT-MULTI-5K and COLLAB (Morris et al., 2020a). A summary of the 10 datasets is given in Table 5. MUTAG consists of 188 mutagenic aromatic and heteroaromatic nitro compounds. The task is to predict whether or not each chemical compound has mutagenic effect on the Gram-negative bacterium *Salmonella typhimurium* (Debnath et al., 1991). ENZYMES contains 600 protein tertiary structures represented as graphs obtained from the BRENDA enzyme database. Each enzyme is a member of one of the Enzyme Commission top level enzyme classes (EC classes) and the task is to correctly assign the enzymes to their classes (Borgwardt et al., 2005). NCI1 contains more than four thousand chemical compounds screened for activity against non-small cell lung cancer and ovarian cancer

Table 5: Summary of the 10 datasets that were used in our experiments.

Dataset	MUTAG	D&D	NCI1	PROTEINS	ENZYMES	IMDB BINARY	IMDB MULTI	REDDIT BINARY	REDDIT MULTI-5K	COLLAB
Max # vertices	28	5,748	111	620	126	136	89	3,782	3,648	492
Min # vertices	10	30	3	4	2	12	7	6	22	32
Average # vertices	17.93	284.32	29.87	39.05	32.63	19.77	13.00	429.61	508.50	74.49
Max # edges	33	14,267	119	1,049	149	1,249	1,467	4,071	4,783	40,119
Min # edges	10	63	2	5	1	26	12	4	21	60
Average # edges	19.79	715.66	32.30	72.81	62.14	96.53	65.93	497.75	594.87	2,457.34
# labels	7	82	37	3	–	–	–	–	–	–
# attributes	–	–	–	–	18	–	–	–	–	–
# graphs	188	1,178	4,110	1,113	600	1,000	1,500	2,000	4,999	5,000
# classes	2	2	2	2	6	2	3	2	5	3

cell lines (Wale et al., 2008). PROTEINS contains proteins represented as graphs where vertices are secondary structure elements and there is an edge between two vertices if they are neighbors in the amino-acid sequence or in 3D space. The task is to classify proteins into enzymes and non-enzymes (Borgwardt et al., 2005). D&D contains over a thousand protein structures. Each protein is a graph whose nodes correspond to amino acids and a pair of amino acids are connected by an edge if they are less than 6 Ångstroms apart. The task is to predict if a protein is an enzyme or not (Dobson and Doig, 2003). IMDB-BINARY and IMDB-MULTI were created from IMDb, an online database of information related to movies and television programs. The graphs contained in the two datasets correspond to movie collaborations. The vertices of each graph represent actors/actresses and two vertices are connected by an edge if the corresponding actors/actresses appear in the same movie. Each graph is the ego-network of an actor/actress, and the task is to predict which genre an ego-network belongs to (Yanardag and Vishwanathan, 2015). REDDIT-BINARY and REDDIT-MULTI-5K contain graphs that model the social interactions between users of Reddit. Each graph represents an online discussion thread. Specifically, each vertex corresponds to a user, and two users are connected by an edge if one of them responded to at least one of the other's comments. The task is to classify graphs into either communities or subreddits (Yanardag and Vishwanathan, 2015). COLLAB is a scientific collaboration dataset that consists of the ego-networks of several researchers from three subfields of Physics (High Energy Physics, Condensed Matter Physics and Astro Physics). The task is to determine the subfield of Physics to which the ego-network of each researcher belongs (Yanardag and Vishwanathan, 2015).

We also evaluated the proposed kernel on one dataset from the Open Graph Benchmark (OGB) (Hu et al., 2020), a collection of large-scale and diverse benchmark datasets for machine learning on graphs. The `ogbg-molhiv` dataset is a molecular property prediction dataset that is adopted from the MoleculeNet (Wu et al., 2018). The dataset consists of $41,127$ molecules and corresponds to a binary classification dataset where the task is to predict whether a molecule inhibits HIV virus replication or not. The average number of vertices per graph is equal to 25.5, while the average number of edges is equal to 27.5. The dataset is split into training/validation/test sets with a ratio of $80/10/10$. The molecules in the training, validation and test sets are divided using a scaffold splitting procedure that splits the molecules based on their two-dimensional structural frameworks. The scaffold splitting attempts to separate structurally different molecules into different subsets.

Besides the above graph classification datasets, we evaluated the proposed kernel on the ZINC 12K graph regression dataset (Dwivedi et al., 2020). The dataset consists of $12,000$ molecules and the task is to predict the constrained solubility of each molecule, an important chemical property for designing generative graph neural networks for molecules. The average number of vertices per graph is equal to 23.16, while the average number of edges is equal to 49.83. Vertices are annotated with the types of heavy atoms, while edges are annotated with types of bonds between their endpoints. The dataset is split into training/validation/test sets consisting of $10,000/1,000/1,000$ graphs, respectively.