

Convolutional Kernel Networks for Graph-Structured Data

Dexiong Chen, Laurent Jacob, Julien Mairal

▶ **To cite this version:**

Dexiong Chen, Laurent Jacob, Julien Mairal. Convolutional Kernel Networks for Graph-Structured Data. ICML 2020 - 37th International Conference on Machine Learning, Jul 2020, Vienna, Austria. pp.1576-1586. hal-02504965v2

HAL Id: hal-02504965
https://hal.science/hal-02504965v2

Submitted on 29 Jun 2020

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Convolutional Kernel Networks for Graph-Structured Data

Dexiong Chen [1] **Laurent Jacob** [2] **Julien Mairal** [1]

Abstract

We introduce a family of multilayer graph kernels and establish new links between graph convolutional neural networks and kernel methods. Our approach generalizes convolutional kernel networks to graph-structured data, by representing graphs as a sequence of kernel feature maps, where each node carries information about local graph substructures. On the one hand, the kernel point of view offers an unsupervised, expressive, and easy-to-regularize data representation, which is useful when limited samples are available. On the other hand, our model can also be trained end-to-end on large-scale data, leading to new types of graph convolutional neural networks. We show that our method achieves competitive performance on several graph classification benchmarks, while offering simple model interpretation. Our code is freely available at `https://github.com/claying/GCKN`.

1. Introduction

Graph kernels are classical tools for representing graph-structured data (see Kriege et al., 2020, for a survey). Most successful examples represent graphs as very-high-dimensional feature vectors that enumerate and count occurences of local graph sub-structures. In order to perform well, a graph kernel should be as expressive as possible, *i.e.*, able to distinguish graphs with different topological properties (Kriege et al., 2018), while admitting polynomial-time algorithms for its evaluation. Common sub-structures include walks (Gärtner et al., 2003), shortest paths (Borgwardt et al., 2005), subtrees (Shervashidze et al., 2011), or graphlets (Shervashidze et al., 2009).

[1]Univ. Grenoble Alpes, Inria, CNRS, Grenoble INP, LJK, 38000 Grenoble, France [2]Univ. Lyon, Université Lyon 1, CNRS, Laboratoire de Biométrie et Biologie Evolutive UMR 5558, 69000 Lyon, France. Correspondence to: Dexiong Chen, Julien Mairal <firstname.lastname@inria.fr>, Laurent Jacob <laurent.jacob@univ-lyon1.fr>.

Proceedings of the 37^{th} International Conference on Machine Learning, Vienna, Austria, PMLR 119, 2020. Copyright 2020 by the author(s).

Graph kernels have shown to be expressive enough to yield good empirical results, but decouple data representation and model learning. In order to obtain task-adaptive representations, another line of research based on neural networks has been developed recently (Niepert et al., 2016; Kipf & Welling, 2017; Xu et al., 2019; Verma et al., 2018). The resulting tools, called graph neural networks (GNNs), are conceptually similar to convolutional neural networks (CNNs) for images; they provide graph-structured multilayer models, where each layer operates on the previous layer by aggregating local neighbor information. Even though harder to regularize than kernel methods, these models are trained end-to-end and are able to extract features adapted to a specific task. In a recent work, Xu et al. (2019) have shown that the class of GNNs based on neighborhood aggregation is at most as powerful as the Weisfeiler-Lehman (WL) graph isomorphism test, on which the WL kernel is based (Shervashidze et al., 2011), and other types of network architectures than simple neighborhood aggregation are needed for more powerful features.

Since GNNs and kernel methods seem to benefit from different characteristics, several links have been drawn between both worlds in the context of graph modeling. For instance, Lei et al. (2017) introduce a class of GNNs whose output lives in the reproducing kernel Hilbert space (RKHS) of a WL kernel. In this line of research, the kernel framework is essentially used to design the architecture of the GNN since the final model is trained as a classical neural network. This is also the approach used by Zhang et al. (2018a) and Morris et al. (2019). By contrast, Du et al. (2019) adopt an opposite strategy and leverage a GNN architecture to design new graph kernels, which are equivalent to infinitely-wide GNNs initialized with random weights and trained with gradient descent. Other attempts to merge neural networks and graph kernels involve using the metric induced by graph kernels to initialize a GNN (Navarin et al., 2018), or using graph kernels to obtain continuous embeddings that are plugged to neural networks (Nikolentzos et al., 2018).

In this paper, we go a step further in bridging graph neural networks and kernel methods by proposing an explicit multilayer kernel representation, which can be used either as a traditional kernel method, or trained end-to-end as a GNN when enough labeled data are available. The multilayer construction allows to compute a series of maps

which account for local sub-structures ("receptive fields") of increasing size. The graph representation is obtained by pooling the final representations of its nodes. The resulting kernel extends to graph-structured data the concept of convolutional kernel networks (CKNs), which was originally designed for images and sequences (Mairal, 2016; Chen et al., 2019a). As our representation of nodes is built by iteratively aggregating representations of their outgoing paths, our model can also be seen as a multilayer extension of path kernels. Relying on paths rather than neighbors for the aggregation step makes our approach more expressive than the GNNs considered in Xu et al. (2019), which implicitly rely on walks and whose power cannot exceed the Weisfeiler-Lehman (WL) graph isomorphism test. Even with medium/small path lengths (which leads to reasonable computational complexity in practice), we show that the resulting representation outperforms walk or WL kernels.

Our model called graph convolutional kernel network (GCKN) relies on the successive uses of the Nyström method (Williams & Seeger, 2001) to approximate the feature map at each layer, which makes our approach scalable. GCKNs can then be interpreted as a new type of graph neural network whose filters may be learned without supervision, by following kernel approximation principles. Such unsupervised graph representation is known to be particularly effective when small amounts of labeled data are available. Similar to CKNs, our model can also be trained end-to-end, as a GNN, leading to task-adaptive representations, with a computational complexity similar to that of a GNN when the path lengths are small enough.

Notation. A graph G is defined as a triplet $(\mathcal{V}, \mathcal{E}, a)$, where $\mathcal{V}$ is the set of vertices, $\mathcal{E}$ is the set of edges, and $a : \mathcal{V} \to \Sigma$ is a function that assigns attributes, either discrete or continous, from a set Σ to nodes in the graph. A path is a sequence of distinct vertices linked by edges and we denote by $\mathcal{P}(G)$ and $\mathcal{P}_k(G)$ the set of paths and paths of length k in G, respectively. In particular, $\mathcal{P}_0(G)$ is reduced to $\mathcal{V}$. We also denote by $\mathcal{P}_k(G, u) \subset \mathcal{P}_k(G)$ the set of paths of length k starting from u in $\mathcal{V}$. For any path p in $\mathcal{P}(G)$, we denote by $a(p)$ in $\Sigma^{|p|+1}$ the concatenation of node attributes in this path. We replace $\mathcal{P}$ with $\mathcal{W}$ to denote the corresponding sets of walks by allowing repeated nodes.

2. Related Work on Graph Kernels

Graph kernels were originally introduced by Gärtner et al. (2003) and Kashima et al. (2003), and have been the subject of intense research during the last twenty years (see the reviews of Vishwanathan et al., 2010; Kriege et al., 2020).

In this paper, we consider graph kernels that represent a graph as a feature vector counting the number of occurrences of some local connected sub-structure. Enumerating common local sub-structures between two graphs is unfortunately often intractable; for instance, enumerating common subgraphs or common paths is known to be NP-hard (Gärtner et al., 2003). For this reason, the literature on graph kernels has focused on alternative structures allowing for polynomial-time algorithms, *e.g.*, walks.

More specifically, we consider graph kernels that perform pairwise comparisons between local sub-structures centered at every node. Given two graphs $G = (\mathcal{V}, \mathcal{E}, a)$ and $G' = (\mathcal{V}', \mathcal{E}', a')$, we consider the kernel

$$K(G, G') = \sum_{u \in \mathcal{V}} \sum_{u' \in \mathcal{V}'} \kappa_{\text{base}}(l_G(u), l_{G'}(u')), \qquad (1)$$

where the base kernel κ_{base} compares a set of local patterns centered at nodes u and u', denoted by $l_G(u)$ and $l_{G'}(u')$, respectively. For simplicity, we will omit the notation $l_G(u)$ in the rest of the paper, and the base kernel will be simply written $\kappa_{\text{base}}(u, u')$ with an abuse of notation. As noted by Lei et al. (2017); Kriege et al. (2020), this class of kernels covers most of the examples mentioned in the introduction.

Walks and path kernels. Since computing all path co-occurences between graphs is NP-hard, it is possible instead to consider paths of length k, which can be reasonably enumerated if k is small enough, or the graphs are sparse. Then, we may define the kernel $K^{(k)}_{\text{path}}$ as (1) with

$$\kappa_{\text{base}}(u, u') = \sum_{p \in \mathcal{P}_k(G,u)} \sum_{p' \in \mathcal{P}_k(G',u')} \delta(a(p), a'(p')), \quad (2)$$

where $a(p)$ represents the attributes for path p in G, and δ is the Dirac kernel such that $\delta(a(p), a'(p')) = 1$ if $a(p) = a'(p')$ and 0 otherwise.

It is also possible to define a variant that enumerates all paths up to length k, by simply adding the kernels $K^{(i)}_{\text{path}}$:

$$K_{\text{path}}(G, G') = \sum_{i=0}^{k} K^{(i)}_{\text{path}}(G, G'). \qquad (3)$$

Similarly, one may also consider using walks by simply replacing the notation $\mathcal{P}$ by $\mathcal{W}$ in the previous definitions.

Weisfeiler-Lehman subtree kernels. A subtree is a subgraph with a tree structure. It can be extended to subtree patterns (Shervashidze et al., 2011; Bach, 2008) by allowing nodes to be repeated, just as the notion of walks extends that of paths. All previous subtree kernels compare subtree patterns instead of subtrees. Among them, the Weisfeiler-Lehman (WL) subtree kernel is one of the most widely used graph kernels to capture such patterns. It is essentially based on a mechanism to augment node attributes by iteratively aggregating and hashing the attributes of each node's neighborhoods. After i iterations, we denote by a_i the new node

attributes for graph $G = (\mathcal{V}, \mathcal{E}, a)$, which is defined in Algorithm 1 of Shervashidze et al. (2011) and then the WL subtree kernel after k iterations is defined, for two graphs $G = (\mathcal{V}, \mathcal{E}, a)$ and $G' = (\mathcal{V}', \mathcal{E}', a')$, as

$$K_{WL}(G, G') = \sum_{i=0}^{k} K^{(i)}_{\text{subtree}}(G, G'), \tag{4}$$

where

$$K^{(i)}_{\text{subtree}}(G, G') = \sum_{u \in \mathcal{V}} \sum_{u' \in \mathcal{V}'} \kappa^{(i)}_{\text{subtree}}(u, u'), \tag{5}$$

with $\kappa^{(i)}_{\text{subtree}}(u, u') = \delta(a_i(u), a'_i(u'))$ and the attributes $a_i(u)$ capture subtree patterns of depth i rooted at node u.

3. Graph Convolutional Kernel Networks

In this section, we introduce our model, which builds upon the concept of graph-structured feature maps, following the terminology of convolutional neural networks.

Definition 1 (Graph feature map)**.** *Given a graph $G = (\mathcal{V}, \mathcal{E}, a)$ and a RKHS $\mathcal{H}$, a graph feature map is a mapping $\varphi : \mathcal{V} \to \mathcal{H}$, which associates to every node a point in $\mathcal{H}$ representing information about local graph substructures.*

We note that the definition matches that of convolutional kernel networks (Mairal, 2016) when the graph is a two-dimensional grid. Generally, the map φ depends on the graph G, and can be seen as a collection of $|\mathcal{V}|$ elements of $\mathcal{H}$ describing its nodes. The kernel associated to the feature maps φ, φ' for two graphs G, G', is defined as

$$K(G, G') = \sum_{u \in \mathcal{V}} \sum_{u' \in \mathcal{V}'} \langle \varphi(u), \varphi'(u') \rangle_{\mathcal{H}} = \langle \Phi(G), \Phi(G') \rangle_{\mathcal{H}}, \tag{6}$$

with

$$\Phi(G) = \sum_{u \in \mathcal{V}} \varphi(u) \quad \text{and} \quad \Phi(G') = \sum_{u \in \mathcal{V}'} \varphi'(u). \tag{7}$$

The RKHS of K can be characterized by using Theorem 2 in Appendix A. It is the space of functions $f_z : G \mapsto \langle z, \Phi(G) \rangle_{\mathcal{H}}$ for all z in $\mathcal{H}$ endowed with a particular norm.

Note that even though graph feature maps φ, φ' are graph-dependent, learning with K is possible as long as they all map nodes to the same RKHS $\mathcal{H}$—as Φ will then also map all graphs to the same space $\mathcal{H}$. We now detail the full construction of the kernel, starting with a single layer.

3.1. Single-Layer Construction of the Feature Map

We propose a single-layer model corresponding to a continuous relaxation of the path kernel. We assume that the input attributes $a(u)$ live in $\mathbb{R}^{q_0}$, such that a graph $G = (\mathcal{V}, \mathcal{E}, a)$ admits a graph feature map $\varphi_0 : \mathcal{V} \to \mathcal{H}_0$ with $\mathcal{H}_0 = \mathbb{R}^{q_0}$ and $\varphi_0(u) = a(u)$. Note that this assumption also allows us to handle discrete labels by using a one-hot encoding strategy—that is *e.g.*, four labels $\{A, B, C, D\}$ are represented by four-dimensional vectors $(1,0,0,0), (0,1,0,0), (0,0,1,0), (0,0,0,1)$, respectively.

Continuous relaxation of the path kernel. We rely on paths of length k, and introduce the kernel K_1 for graphs G, G' with feature maps φ_0, φ'_0 of the form (1) with

$$\kappa_{\text{base}}(u, u') = \sum_{p \in \mathcal{P}_k(G, u)} \sum_{p' \in \mathcal{P}_k(G', u')} \kappa_1(\varphi_0(p), \varphi'_0(p')), \tag{8}$$

where $\varphi_0(p) = [\varphi_0(p_i)]_{i=0}^{k}$ denotes the concatenation of $k+1$ attributes along path p, which is an element of $\mathcal{H}_0^{k+1}$, p_i is the i-th node on path p starting from index 0, and κ_1 is a Gaussian kernel comparing such attributes:

$$\kappa_1(\varphi_0(p), \varphi'_0(p')) = e^{-\frac{\alpha_1}{2} \sum_{i=0}^{k} \|\varphi_0(p_i) - \varphi'_0(p'_i)\|^2_{\mathcal{H}_0}}. \tag{9}$$

This is an extension of the path kernel, obtained by replacing the hard matching function δ in (2) by κ_1, as done for instance by Togninalli et al. (2019) for the WL kernel. This replacement not only allows us to use continuous attributes, but also has important consequences in the discrete case since it allows to perform inexact matching between paths. For instance, when the graph is a chain with discrete attributes—in other words, a string—then, paths are simply k-mers, and the path kernel (with matching function δ) becomes the spectrum kernel for sequences (Leslie et al., 2001). By using κ_1 instead, we obtain the single-layer CKN kernel of Chen et al. (2019a), which performs inexact matching, as the mismatch kernel does (Leslie et al., 2004), and leads to better performances in many tasks involving biological sequences.

From graph feature map φ_0 to graph feature map φ_1. The kernel κ_1 acts on pairs of paths in potentially different graphs, but only through their mappings to the same space $\mathcal{H}_0^{k+1}$. Since κ_1 is positive definite, we denote by $\mathcal{H}_1$ its RKHS and consider its mapping $\phi_1^{\text{path}} : \mathcal{H}_0^{k+1} \to \mathcal{H}_1$ such that

$$\kappa_1(\varphi_0(p), \varphi'_0(p')) = \langle \phi_1^{\text{path}}(\varphi_0(p)), \phi_1^{\text{path}}(\varphi'_0(p')) \rangle_{\mathcal{H}_1}.$$

For any graph G, we can now define a graph feature map $\varphi_1 : \mathcal{V} \to \mathcal{H}_1$, operating on nodes u in $\mathcal{V}$, as

$$\varphi_1(u) = \sum_{p \in \mathcal{P}_k(G, u)} \phi_1^{\text{path}}(\varphi_0(p)). \tag{10}$$

Then, the continuous relaxation of the path kernel, denoted by $K_1(G, G')$, can also be written as (6) with $\varphi = \varphi_1$, and its underlying kernel representation Φ_1 is given by (7). The construction of φ_1 from φ_0 is illustrated in Figure 1.

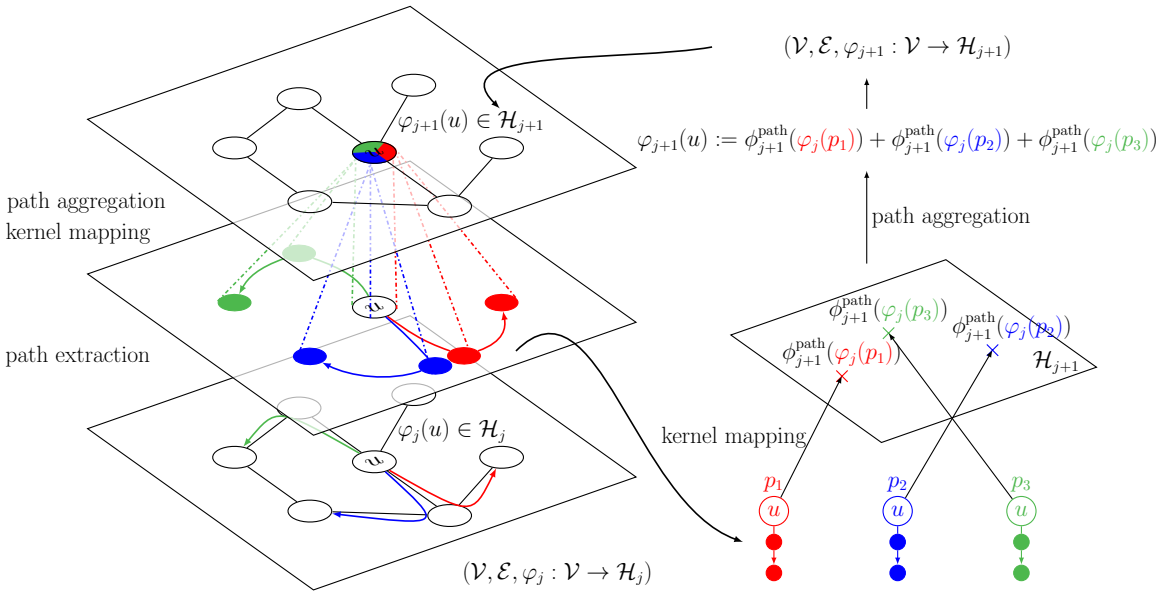


Figure 1. Construction of the graph feature map φ_{j+1} from φ_j given a graph $(\mathcal{V}, \mathcal{E})$. The first step extracts paths of length k (here colored by red, blue and green) from node u, then (on the right panel) maps them to a RKHS $\mathcal{H}_{j+1}$ via the Gaussian kernel mapping. The new map φ_{j+1} at u is obtained by local path aggregation (pooling) of their representations in $\mathcal{H}_{j+1}$. The representations for other nodes can be obtained in the same way. In practice, such a model is implemented by using finite-dimensional embeddings approximating the feature maps, see Section 3.2.

The graph feature map φ_0 maps a node (resp a path) to $\mathcal{H}_0$ (resp $\mathcal{H}_0^{k+1}$) which is typically a Euclidean space describing its attributes. By contrast, ϕ_1^{path} is the kernel mapping of the Gaussian kernel κ_1, and maps each path p to a Gaussian function centered at $\varphi_0(p)$—remember indeed that for kernel function $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ with RKHS $\mathcal{H}$, the kernel mapping is of a data point x is the function $K(x, .) : \mathcal{X} \to \mathbb{R}$. Finally, φ_1 maps each node u to a mixture of Gaussians, each Gaussian function corresponding to a path starting at u.

3.2. Concrete Implementation and GCKNs

We now discuss algorithmic aspects, leading to the graph convolutional kernel network (GCKN) model, which consists in building a finite-dimensional embedding $\Psi(G)$ that may be used in various learning tasks without scalability issues. We start here with the single-layer case.

The Nyström method and the single-layer model. A naive computation of the path kernel K_1 requires comparing all pairs of paths in each pair of graphs. To gain scalability, a key component of the CKN model is the Nyström method (Williams & Seeger, 2001), which computes finite-dimensional approximate kernel embeddings. We discuss here the use of such a technique to define finite-dimensional maps $\psi_1 : \mathcal{V} \to \mathbb{R}^{q_1}$ and $\psi_1' : \mathcal{V}' \to \mathbb{R}^{q_1}$ for graphs G, G' such that for all pairs of nodes u, u' in $\mathcal{V}, \mathcal{V}'$, respectively,

$$\langle \varphi_1(u), \varphi_1'(u') \rangle_{\mathcal{H}_1} \approx \langle \psi_1(u), \psi_1'(u') \rangle_{\mathbb{R}^{q_1}}.$$

The consequence of such an approximation is that it provides a finite-dimensional approximation Ψ_1 of Φ_1:

$$K_1(G, G') \approx \langle \Psi_1(G), \Psi_1(G') \rangle_{\mathbb{R}^{q_1}}$$
$$\text{with } \Psi_1(G) = \sum_{u \in \mathcal{V}} \psi_1(u).$$

Then, a supervised learning problem with kernel K_1 on a dataset $(G_i, y_i)_{i=1,\dots,n}$, where y_i are labels in $\mathbb{R}$, can be solved by minimizing the regularized empirical risk

$$\min_{w \in \mathbb{R}^{q_1}} \sum_{i=1}^{n} L(y_i, \langle \Psi_1(G_i), w \rangle) + \lambda \|w\|^2, \qquad (11)$$

where L is a convex loss function. Next, we show that using the Nyström method to approximate the kernel κ_1 yields a new type of GNN, represented by $\Psi_1(G)$, whose filters can be obtained without supervision, or, as discussed later, with back-propagation in a task-adaptive manner.

Specifically, the Nyström method projects points from a given RKHS onto a finite-dimensional subspace and performs all subsequent operations within that subspace. In the context of κ_1, whose RKHS is $\mathcal{H}_1$ with mapping function ϕ_1^{path}, we consider a collection $Z = \{z_1, \dots, z_{q_1}\}$ of q_1 prototype paths represented by attributes in $\mathcal{H}_0^{k+1}$, and we define the subspace $\mathcal{E}_1 = \text{Span}(\phi_1^{\text{path}}(z_1), \dots, \phi_1^{\text{path}}(z_{q_1}))$. Given a new path with attributes z, it is then possible to show (see Chen et al., 2019a) that the projection of path

attributes z onto $\mathcal{E}_1$ leads to the q_1-dimensional mapping

$$\psi_1^{\text{path}}(z) = [\kappa_1(z_i, z_j)]_{ij}^{-\frac{1}{2}} [\kappa_1(z_1, z), \dots, \kappa_1(z_{q_1}, z)]^\top,$$

where $[\kappa_1(z_i, z_j)]_{ij}$ is a $q_1 \times q_1$ Gram matrix. Then, the approximate graph feature map ψ_1 is obtained by pooling

$$\psi_1(u) = \sum_{p \in \mathcal{P}_k(G,u)} \psi_1^{\text{path}}(\psi_0(p)) \quad \text{for all} \quad u \in \mathcal{V},$$

where $\psi_0 = \varphi_0$ and $\psi_0(p) = [\psi_0(p_i)]_{i=0,\dots,k}$ in $\mathbb{R}^{q_0(k+1)}$ represents the attributes of path p, with an abuse of notation.

Interpretation as a GNN. When input attributes $\psi_0(u)$ have unit-norm, which is the case if we use one-hot encoding on discrete attributes, the Gaussian kernel κ_1 between two path attributes z, z' in $\mathbb{R}^{q_0(k+1)}$ may be written

$$\kappa_1(z, z') = e^{-\frac{\alpha_1}{2}\|z - z'\|^2} = e^{\alpha_1(z^\top z' - k - 1)} = \sigma_1(z^\top z'), \tag{12}$$

which is a dot-product kernel with a non-linear function σ_1. Then, calling Z in $\mathbb{R}^{q_0(k+1) \times q_1}$ the matrix of prototype path attributes, we have

$$\psi_1(u) = \sum_{p \in \mathcal{P}_k(G,u)} \sigma_1(Z^\top Z)^{-\frac{1}{2}} \sigma_1(Z^\top \psi_0(p)), \tag{13}$$

where, with an abuse of notation, the non-linear function σ_1 is applied pointwise. Then, the map ψ_1 is build from ψ_0 with the following steps (i) feature aggregation along the paths, (ii) encoding of the paths with a linear operation followed by point-wise non-linearity, (iii) multiplication by the $q_1 \times q_1$ matrix $\sigma_1(Z^\top Z)^{-\frac{1}{2}}$, and (iv) linear pooling. The major difference with a classical GNN is that the "filtering" operation may be interpreted as an orthogonal projection onto a linear subspace, due to the matrix $\sigma_1(Z^\top Z)^{-\frac{1}{2}}$. Unlike the Dirac function, the exponential function σ_1 is differentiable. A useful consequence is the possibility of optimizing the filters Z with back-propagation as detailed below. Note that in practice we add a small regularization term to the diagonal for stability reason: $(\sigma_1(Z^\top Z) + \varepsilon I)^{-\frac{1}{2}}$ with $\varepsilon = 0.01$.

Learning without supervision. Learning the "filters" Z with Nyström can be achieved by simply running a K-means algorithm on path attributes extracted from training data (Zhang et al., 2008). This is the approach adopted for CKNs by Mairal (2016); Chen et al. (2019a), which proved to be very effective as shown in the experimental section.

End-to-end learning with back-propagation. While the previous unsupervised learning strategy consists in finding a good kernel approximation that is independent of labels, it is also possible to learn the parameters Z end-to-end, by minimizing (11) jointly with respect to Z and w. The main observations from Chen et al. (2019a) in the context of biological sequences is that such a supervised learning approach may yield good models with much fewer filters q_1 than with the unsupervised learning strategy. We refer the reader to Chen et al. (2019a;b) for how to perform back-propagation with the inverse square root matrix $\sigma_1(Z^\top Z)^{-\frac{1}{2}}$.

Algorithm 1 Forward pass for multilayer GCKN

Input: graph $G = (\mathcal{V}, \mathcal{E}, \psi_0 : \mathcal{V} \to \mathbb{R}^{q_0})$, set of anchor points (filters) $Z_j \in \mathbb{R}^{(k+1)q_{j-1} \times q_j}$ for $j = 1, \dots, J$.
for $j = 1, \dots, J$ **do**
  for u **in** $\mathcal{V}$ **do**
    $\psi_j(u) = \sum_{p \in \mathcal{P}_k(G,u)} \psi_j^{\text{path}}(\psi_{j-1}(p))$;
  end for
end for
Global pooling: $\Psi(G) = \sum_{u \in \mathcal{V}} \psi_J(u)$;

Complexity. The complexity for computing the feature map ψ_1 is dominated by the complexity of finding all the paths of length k from each node. This can be done by simply using a depth first search algorithm, whose worst-case complexity for each graph is $O(|\mathcal{V}|d^k)$, where d is the maximum degree of each node, meaning that large k may be used only for sparse graphs. Then, each path is encoded in $O(q_1 q_0(k+1))$ operations; When learning with back-propagation, each gradient step requires computing the eigenvalue decomposition of $\sigma_1(Z^\top Z)^{-\frac{1}{2}}$ whose complexity is $O(q_1^3)$, which is not a computational bottleneck when using mini-batches of order $O(q_1)$, where typical practical values for q_1 are reasonably small, *e.g.*, less than 128.

3.3. Multilayer Extensions

The mechanism to build the feature map φ_1 from φ_0 can be iterated, as illustrated in Figure 1 which shows how to build a feature map φ_{j+1} from a previous one φ_j. As discussed by Mairal (2016) for CKNs, the Nyström method may then be extended to build a sequence of finite-dimensional maps $\psi_0, \dots, \psi_J$, and the final graph representation is given by

$$\Psi_J(G) = \sum_{u \in \mathcal{V}} \psi_J(u). \tag{14}$$

The computation of $\Psi_J(G)$ is illustrated in Algorithm 1. Here we discuss two possible uses for these additional layers, either to account for more complex structures than paths, or to extend the receptive field of the representation without resorting to the enumeration of long paths.We will denote by k_j the path length used at layer j.

A simple two-layer model to account for subtrees. As emphasized in (7), GCKN relies on a representation $\Phi(G)$ of graphs, which is a sum of node-level representations provided by a graph feature map φ. If φ is a sum over

paths starting at the represented node, $\Phi(G)$ can simply be written as a sum over all paths in G, consistently with our observation that (6) recovers the path kernel when using a Dirac kernel to compare paths in κ_1. The path kernel often leads to good performances, but it is also blind to more complex structures. Figure 2 provides a simple example of this phenomenon, using $k = 1$: G_1 and G_3 differ by a single edge, while G_4 has a different set of nodes and a rather different structure. Yet $\mathcal{P}_1(G_3) = \mathcal{P}_1(G_4)$, making $K_1(G_1, G_3) = K_1(G_1, G_4)$ for the path kernel.

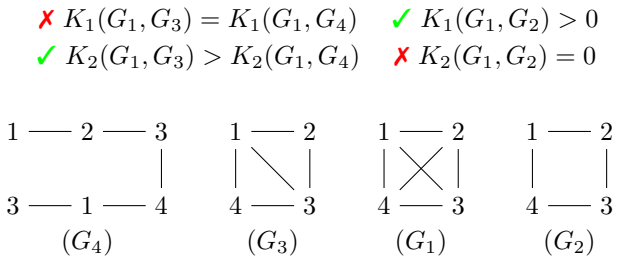


Figure 2. Example cases using $\kappa_1 = \kappa_2 = \delta$, with path lengths $k_1 = 1$ and $k_2 = 0$; The one-layer kernel K_1 counts the number of common edges while the two-layer K_2 counts the number of nodes with the same set of outgoing edges. The figure suggests using $K_1 + K_2$ to gain expressiveness.

Expressing more complex structures requires breaking the succession of linearities introduced in (7) and (10)—much like pointwise nonlinearities are used in neural networks. Concretely, this effect can simply be obtained by using a second layer with path length $k_2 = 0$—paths are then identified to vertices—which produces the feature map $\varphi_2(u) = \phi_2^{\text{path}}(\varphi_1(u))$, where $\phi_2^{\text{path}} : \mathcal{H}_1 \to \mathcal{H}_2$ is a non-linear kernel mapping. The resulting kernel is then

$$K_2(G, G') = \sum_{u \in \mathcal{V}} \sum_{u' \in \mathcal{V}'} \langle \varphi_2(u), \varphi_2'(u') \rangle_{\mathcal{H}_2} = \sum_{u \in \mathcal{V}} \sum_{u' \in \mathcal{V}'} \kappa_2(\varphi_1(u), \varphi_1'(u')). \quad (15)$$

When κ_1 and κ_2 are both Dirac kernels, K_2 counts the number of nodes in G and G' with the exact same set of outgoing paths $\mathcal{P}(G, u)$, as illustrated in Figure 2.

Theorem 1 further illustrates the effect of using a nonlinear ϕ_2^{path} on the feature map φ_1, by formally linking the walk and WL subtree kernel through our framework.

Theorem 1. *Let $G = (\mathcal{V}, \mathcal{E})$, $G' = (\mathcal{V}', \mathcal{E}')$, $\mathcal{M}$ be the set of exact matchings of subsets of the neighborhoods of two nodes, as defined in Shervashidze et al. (2011), and φ defined as in (10) with $\kappa_1 = \delta$ and replacing paths by walks. For any $u \in \mathcal{V}$ and $u' \in \mathcal{V}'$ such that $|\mathcal{M}(u, u')| = 1$,*

$$\delta(\varphi_1(u), \varphi_1'(u')) = \kappa_{\text{subtree}}^{(k)}(u, u'). \quad (16)$$

Recall that when using (8) with walks instead of paths and a Dirac kernel for κ_1, the kernel (6) with $\varphi = \varphi_1$ is the walk kernel. The condition $|\mathcal{M}(u, u')| = 1$ indicates that u and u' have the same degrees and each of them has distinct neighbors. This can be always ensured by including degree information and adding noise to node attributes. For a large class of graphs, both the walk and WL subtree kernels can therefore be written as (6) with the same first layer φ_1 representing nodes by their walk histogram. While walk kernels use a single layer, WL subtree kernels rely on a second layer φ_2 mapping nodes to the indicator function of $\varphi_1(u)$.

Theorem 1 also shows that the kernel built in (15) is a path-based version of WL subtree kernels, therefore more expressive as it captures subtrees rather than subtree patterns. However, the Dirac kernel lacks flexibility, as it only accounts for pairs of nodes with identical $\mathcal{P}(G, u)$. For example, in Figure 2, $K_2(G_1, G_2) = 0$ even though G_1 only differs from G_2 by two edges, because these two edges belong to the set $\mathcal{P}(G, u)$ of all nodes in the graph. In order to retain the stratification by node of (15) while allowing for a softer comparison between sets of outgoing paths, we replace δ by the kernel $\kappa_2(\varphi_1(u), \varphi_1'(u')) = e^{-\alpha_2 \|\varphi_1(u) - \varphi_1'(u')\|_{\mathcal{H}_1}^2}$. Large values of α_2 recover the behavior of the Dirac, while smaller values gives non-zero values for similar $\mathcal{P}(G, u)$.

A multilayer model to account for longer paths. In the previous paragraph, we have seen that adding a second layer could bring some benefits in terms of expressiveness, even when using path lengths $k_2 = 0$. Yet, a major limitation of this model is the exponential complexity of path enumeration, which is required to compute the feature map φ_1, preventing us to use large values of k as soon as the graph is dense. Representing large receptive fields while relying on path enumerations with small k, *e.g.*, $k \leq 3$, is nevertheless possible with a multilayer model. To account for a receptive field of size k, the previous model requires a path enumeration with complexity $O(|\mathcal{V}| d^k)$, whereas the complexity of a multilayer model is linear in k.

3.4. Practical Variants

Summing the kernels for different k and different scales. As noted in Section 2, summing the kernels corresponding to different values of k provides a richer representation. We also adopt such a strategy, which corresponds to concatenating the feature vectors $\Psi(G)$ obtained for various path lengths k. When considering a multilayer model, it is also possible to concatenate the feature representations obtained at every layer j, allowing to obtain a multi-scale feature representation of the graph and gain expressiveness.

Use of homogeneous dot-product kernel. Instead of the Gaussian kernel (9), it is possible to use a homogeneous dot-

product kernel, as suggested by Mairal (2016) for CKNs:

$$\kappa_1(z, z') = \|z\|\|z'\|\sigma_1\left(\frac{\langle z, z'\rangle}{\|z\|\|z'\|}\right),$$

where σ_1 is defined in (12). Note that when z, z' have unit-norm, we recover the Gaussian kernel (9). In our paper, we use such a kernel for upper layers, or for continuous input attributes when they do not have unit norm. For multilayer models, this homogenization is useful for preventing vanishing or exponentially growing representations. Note that ReLU is also a homogeneous non-linear mapping.

Other types of pooling operations. Another variant consists in replacing the sum pooling operation in (13) and (14) by a mean or a max pooling. While using max pooling as a heuristic seems to be effective on some datasets, it is hard to justify from a RKHS point of view since max operations typically do not yield positive definite kernels. Yet, such a heuristic is widely adopted in the kernel literature, *e.g.*, for string alignment kernels (Saigo et al., 2004). In order to solve such a discrepancy between theory and practice, Chen et al. (2019b) propose to use the generalized max pooling operator of Murray & Perronnin (2014), which is compatible with the RKHS point of view. Applying the same ideas to GCKNs is straightforward.

Using walk kernel instead of path kernel. One can use a relaxed walk kernel instead of the path kernel in (8), at the cost of losing some expressiveness but gaining some time complexity. Indeed, there exists a very efficient recursive way to enumerate walks and thus to compute the resulting approximate feature map in (13) for the walk kernel. Specifically, if we denote the k-walk kernel by $\kappa_{\text{walk}}^{(k)}$, then its value between two nodes can be decomposed as the product of the 0-walk kernel between the nodes and the sum of the $(k-1)$-walk kernel between their neighbors

$$\kappa_{\text{walk}}^{(k)}(u, u') = \kappa_{\text{walk}}^{(0)}(u, u') \sum_{v \in \mathcal{N}(u)} \sum_{v' \in \mathcal{N}(u')} \kappa_{\text{walk}}^{(k-1)}(v, v'),$$

where $\kappa_{\text{walk}}^{(0)}(u, u') = \kappa_1(\varphi_0(u), \varphi_0'(u'))$. After applying the Nyström method, the approximate feature map of the walk kernel is written, similar to (13), as

$$\psi_1(u) = \sigma_1(Z^\top Z)^{-\frac{1}{2}} \underbrace{\sum_{p \in \mathcal{W}_k(G,u)} \sigma_1(Z^\top \psi_0(p))}_{c_k(u) :=}.$$

Based on the above observation and following similar induction arguments as Chen et al. (2019b), it is not hard to show that $(c_j(u))_{j=1,\dots,k}$ obeys the following recursion

$$c_j(u) = b_j(u) \odot \sum_{v \in \mathcal{N}(u)} c_{j-1}(v), \ 1 \leq j \leq k,$$

where $\odot$ denotes the element-wise product and $b_j(u)$ is a vector in $\mathbb{R}^{q_1}$ whose entry i in $\{1, \dots, q_1\}$ is $\kappa_1(u, z_i^{(k+1-j)})$ and $z_i^{(k+1-j)}$ denotes the $k+1-j$-th column vector of z_i in $\mathbb{R}^{q_0}$. More details can be found in Appendix C.

4. Model Interpretation

Ying et al. (2019) introduced an approach to interpret trained GNN models, by finding a subgraph of an input graph G maximizing the mutual information with its predicted label (note that this approach depends on a specific input graph). We show here how to adapt similar ideas to our framework.

Interpreting GCKN-path and GCKN-subtree. We call GCKN-path our model Ψ_1 with a single layer, and GCKN-subtree our model Ψ_2 with two layers but with $k_2 = 0$, which is the first model presented in Section 3.3 that accounts for subtree structures. As these models are built upon path enumeration, we extend the method of Ying et al. (2019) by identifying a small subset of paths in an input graph G preserving the prediction. We then reconstruct a subgraph by merging the selected paths. For simplicity, let us consider a one-layer model. As $\Psi_1(G)$ only depends on G through its set of paths $\mathcal{P}_k(G)$, we note $\Psi_1(\mathcal{P})$ with an abuse of notation for any subset of $\mathcal{P}$ of paths in G, to emphasize the dependency in this set of paths. For a trained model (Ψ_1, w) and a graph G, our objective is to solve

$$\min_{\mathcal{P}' \subseteq \mathcal{P}_k(G)} L(\hat{y}, \langle \Psi_1(\mathcal{P}'), w\rangle) + \mu|\mathcal{P}'|, \tag{17}$$

where $\hat{y}$ is the predicted label of G and μ a regularization parameter controlling the number of paths to select. This problem is combinatorial and can be computationally intractable when $\mathcal{P}(G)$ is large. Following Ying et al. (2019), we relax it by using a mask M with values in $[0; 1]$ over the set of paths, and replace the number of paths $|\mathcal{P}'|$ by the ℓ_1-norm of M, which is known to have a sparsity-inducing effect (Tibshirani, 1996). The problem then becomes

$$\min_{M \in [0;1]^{|\mathcal{P}_k(G)|}} L(\hat{y}, \langle \Psi_1(\mathcal{P}_k(G) \odot M), w\rangle) + \mu\|M\|_1, \tag{18}$$

where $\mathcal{P}_k(G) \odot M$ denotes the use of $M(p)a(p)$ instead of $a(p)$ in the computation of Ψ_1 for all p in $\mathcal{P}_k(G)$. Even though the problem is non-convex due to the non-linear mapping Ψ_1, it may still be solved approximately by using projected gradient-based optimization techniques.

Interpreting multilayer models. By noting that $\Psi_j(G)$ only depends on the union of the set of paths $\mathcal{P}_{k_l}(G)$, for all layers $l \leq j$, we introduce a collection of masks M_l at each layer, and then optimize the same objective as (18) over all masks $(M_l)_{l=1,\dots,j}$, with the regularization $\sum_{l=1}^{j} \|M_l\|_1$.

5. Experiments

We evaluate GCKN and compare its variants to state-of-the-art methods, including GNNs and graph kernels, on several real-world graph classification datasets, involving either discrete or continuous attributes.

5.1. Implementation Details

We follow the same protocols as (Du et al., 2019; Xu et al., 2019), and report the average accuracy and standard deviation over a 10-fold cross validation on each dataset. We use the same data splits as Xu et al. (2019), using their code. Note that performing nested 10-fold cross validation would have provided better estimates of test accuracy for all models, but it would have unfortunately required 10 times more computation, which we could not afford for many of the baselines we considered.

Considered models. We consider two single-layer models called GCKN-walk and GCKN-path, corresponding to the continuous relaxation of the walk and path kernels respectively. We also consider the two-layer model GCKN-subtree introduced in Section 3.3 with path length $k_2 = 0$, which accounts for subtrees. Finally, we consider a 3-layer model GCKN-3layers with path length $k_2 = 2$ (which enumerates paths with three vertices for the second layer), and $k_3 = 0$, which introduces a non-linear mapping before global pooling, as in GCKN-subtree. We use the same parameters α_j and q_j (number of filters) across layers. Our comparisons include state-of-the-art graph kernels such as WL kernel (Shervashidze et al., 2011), AWL (Ivanov & Burnaev, 2018), RetGK (Zhang et al., 2018b), GNTK (Du et al., 2019), WWL (Togninalli et al., 2019) and recent GNNs including GCN (Kipf & Welling, 2017), PatchySAN (Niepert et al., 2016) and GIN (Xu et al., 2019). We also include a simple baseline method LDP (Cai & Wang, 2018) based on node degree information and a Gaussian SVM.

Learning unsupervised models. Following Mairal (2016), we learn the anchor points Z_j for each layer by K-means over 300000 extracted paths from each training fold. The resulting graph representations are then mean-centered, standardized, and used within a linear SVM classifier (11) with squared hinge loss. In practice, we use the SVM implementation of the Cyanure toolbox (Mairal, 2019).[1] For each 10-fold cross validation, we tune the bandwidth of the Gaussian kernel (identical for all layers), pooling operation (local (13) or global (14)), path size k_1 at the first layer, number of filters (identical for all layers) and regularization parameter λ in (11). More details are provided in Appendix B, as well as a study of the model robustness to hyperparameters.

[1] http://julien.mairal.org/cyanure/

Learning supervised models. Following Xu et al. (2019), we use an Adam optimizer (Kingma & Ba, 2015) with the initial learning rate equal to 0.01 and halved every 50 epochs, and fix the batch size to 32. We use the unsupervised model based described above for initialization. We select the best model based on the same hyperparameters as for unsupervised models, with the number of epochs as an additional hyperparameter as used in Xu et al. (2019). Note that we do not use DropOut or batch normalization, which are typically used in GNNs such as Xu et al. (2019). Importantly, the number of filters needed for supervised models is always much smaller (*e.g.*, 32 vs 512) than that for unsupervised models to achieve comparable performance.

5.2. Results

Graphs with categorical node labels We use the same benchmark datasets as in Du et al. (2019), including 4 biochemical datasets MUTAG, PROTEINS, PTC and NCI1 and 3 social network datasets IMDB-B, IMDB-MULTI and COLLAB. All the biochemical datasets have categorical node labels while none of the social network datasets has node features. We use degrees as node labels for these datasets, following the protocols of previous works (Du et al., 2019; Xu et al., 2019; Togninalli et al., 2019). Similarly, we also transform all the categorical node labels to one-hot representations. The results are reported in Table 1. With a few exceptions, GCKN-walk has a small edge on graph kernels and GNNs—both implicitly relying on walks too—probably because of the soft structure comparison allowed by the Gaussian kernel. GCKN-path often brings some further improvement, which can be explained by its increasing the expressivity. Both multilayer GCKNs bring a stronger increase, whereas supervising the filter learning of GCKN-subtree does not help. Yet, the number of filters selected by GCKN-subtree-sup is smaller than GCKN-subtree-unsup (see Appendix B), allowing for faster classification at test time. GCKN-3layers-unsup performs in the same ballpark as GCKN-subtree-unsup, but benefits from lower complexity due to smaller path length k_1.

Graphs with continuous node attributes We use 4 real-world graph classification datasets with continuous node attributes: ENZYMES, PROTEINS_full, BZR, COX2. All datasets and size information about the graphs can be found in Kersting et al. (2016). The node attributes are preprocessed with standardization as in Togninalli et al. (2019). To make a fair comparison, we follow the same protocol as used in Togninalli et al. (2019). Specifically, we perform 10 different 10-fold cross validations, using the same hyperparameters that give the best average validation accuracy. The hyperparameter search grids remain the same as for training graphs with categorical node labels. The results are shown in Table 2. They are comparable to the

Table 1. Classification accuracies on graphs with discrete node attributes. The accuracies of other models are taken from Du et al. (2019) except LDP, which we evaluate on our splits and for which we tune bin size, the regularization parameter in the SVM and Gaussian kernel bandwidth. Note that RetGK uses a different protocol, performing 10-fold cross-validation 10 times and reporting the average accuracy.

Dataset	MUTAG	PROTEINS	PTC	NCI1	IMDB-B	IMDB-M	COLLAB
size	188	1113	344	4110	1000	1500	5000
classes	2	2	2	2	2	3	3
avg ♯nodes	18	39	26	30	20	13	74
avg ♯edges	20	73	51	32	97	66	2458
LDP	88.9 ± 9.6	73.3 ± 5.7	63.8 ± 6.6	72.0 ± 2.0	68.5 ± 4.0	42.9 ± 3.7	76.1 ± 1.4
WL subtree	90.4 ± 5.7	75.0 ± 3.1	59.9 ± 4.3	$\mathbf{86.0 \pm 1.8}$	73.8 ± 3.9	50.9 ± 3.8	78.9 ± 1.9
AWL	87.9 ± 9.8	-	-	-	74.5 ± 5.9	51.5 ± 3.6	73.9 ± 1.9
RetGK	90.3 ± 1.1	75.8 ± 0.6	62.5 ± 1.6	84.5 ± 0.2	71.9 ± 1.0	47.7 ± 0.3	81.0 ± 0.3
GNTK	90.0 ± 8.5	75.6 ± 4.2	67.9 ± 6.9	84.2 ± 1.5	76.9 ± 3.6	52.8 ± 4.6	$\mathbf{83.6 \pm 1.0}$
GCN	85.6 ± 5.8	76.0 ± 3.2	64.2 ± 4.3	80.2 ± 2.0	74.0 ± 3.4	51.9 ± 3.8	79.0 ± 1.8
PatchySAN	92.6 ± 4.2	75.9 ± 2.8	60.0 ± 4.8	78.6 ± 1.9	71.0 ± 2.2	45.2 ± 2.8	72.6 ± 2.2
GIN	89.4 ± 5.6	76.2 ± 2.8	64.6 ± 7.0	82.7 ± 1.7	75.1 ± 5.1	52.3 ± 2.8	80.2 ± 1.9
GCKN-walk-unsup	92.8 ± 6.1	75.7 ± 4.0	65.9 ± 2.0	80.1 ± 1.8	75.9 ± 3.7	53.4 ± 4.7	81.7 ± 1.4
GCKN-path-unsup	92.8 ± 6.1	76.0 ± 3.4	67.3 ± 5.0	81.4 ± 1.6	75.9 ± 3.7	53.0 ± 3.1	82.3 ± 1.1
GCKN-subtree-unsup	95.0 ± 5.2	$\mathbf{76.4 \pm 3.9}$	$\mathbf{70.8 \pm 4.6}$	83.9 ± 1.6	$\mathbf{77.8 \pm 2.6}$	$\mathbf{53.5 \pm 4.1}$	83.2 ± 1.1
GCKN-3layer-unsup	$\mathbf{97.2 \pm 2.8}$	75.9 ± 3.2	69.4 ± 3.5	83.9 ± 1.2	77.2 ± 3.8	53.4 ± 3.6	83.4 ± 1.5
GCKN-subtree-sup	91.6 ± 6.7	76.2 ± 2.5	68.4 ± 7.4	82.0 ± 1.2	76.5 ± 5.7	53.3 ± 3.9	82.9 ± 1.6

ones obtained with categorical attributes, except that in 2/4 datasets, the multilayer versions of GCKN underperform compared to GCKN-path, but they achieve lower computational complexity. Paths were indeed presumably predictive enough for these datasets. Besides, the supervised version of GCKN-subtree outperforms its unsupervised counterpart in 2/4 datasets.

Table 2. Classification accuracies on graphs with continuous attributes. The accuracies of other models except GNTK are taken from Togninalli et al. (2019). The accuracies of GNTK are obtained by running the code of Du et al. (2019) on a similar setting.

Dataset	ENZYMES	PROTEINS	BZR	COX2
size	600	1113	405	467
classes	6	2	2	2
attr. dim.	18	29	3	3
avg ♯nodes	32.6	39.0	35.8	41.2
avg ♯edges	62.1	72.8	38.3	43.5
RBF-WL	68.4 ± 1.5	75.4 ± 0.3	81.0 ± 1.7	75.5 ± 1.5
HGK-WL	63.0 ± 0.7	75.9 ± 0.2	78.6 ± 0.6	78.1 ± 0.5
HGK-SP	66.4 ± 0.4	75.8 ± 0.2	76.4 ± 0.7	72.6 ± 1.2
WWL	73.3 ± 0.9	$\mathbf{77.9 \pm 0.8}$	84.4 ± 2.0	78.3 ± 0.5
GNTK	69.6 ± 0.9	75.7 ± 0.2	85.5 ± 0.8	79.6 ± 0.4
GCKN-walk-unsup	73.5 ± 0.5	76.5 ± 0.3	85.3 ± 0.5	80.6 ± 1.2
GCKN-path-unsup	$\mathbf{75.7 \pm 1.1}$	76.3 ± 0.5	85.9 ± 0.5	81.2 ± 0.8
GCKN-subtree-unsup	74.8 ± 0.7	77.5 ± 0.3	85.8 ± 0.9	81.8 ± 0.8
GCKN-3layer-unsup	74.6 ± 0.8	77.5 ± 0.4	84.7 ± 1.0	$\mathbf{82.0 \pm 0.6}$
GCKN-subtree-sup	72.8 ± 1.0	77.6 ± 0.4	$\mathbf{86.4 \pm 0.5}$	81.7 ± 0.7

5.3. Model Interpretation

We train a supervised GCKN-subtree model on the Mutagenicity dataset (Kersting et al., 2016), and use our method described in Section 4 to identify important subgraphs. Figure 3 shows examples of detected subgraphs. Our method is able to identify chemical groups known for their mutagenicity such as Polycyclic aromatic hydrocarbon (top row left), Diphenyl ether (top row middle) or NO_2 (top row right), thus admitting simple model interpretation. We also find some groups whose mutagenicity is not known, such as polyphenylene sulfide (bottom row middle) and 2-chloroethyl- (bottom row right). More details and additional results are provided in Appendix B.

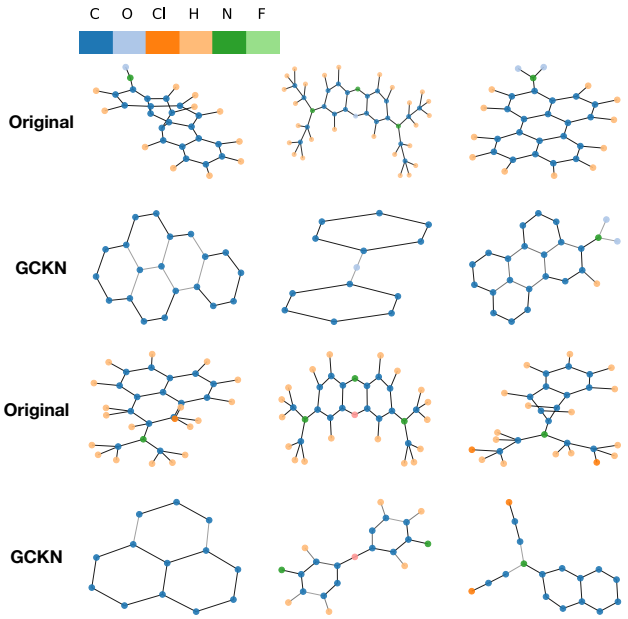


Figure 3. Motifs extracted by GCKN on the Mutagenicity dataset.

Acknowledgements

This work has been supported by the grants from ANR (FAST-BIG project ANR-17 CE23-0011-01), by the ERC grant number 714381 (SOLARIS), and by ANR 3IA MIAI@Grenoble Alpes, (ANR-19-P3IA-0003).

References

Bach, F. Graph kernels between point clouds. In *International Conference on Machine Learning (ICML)*, 2008.

Borgwardt, K. M., Ong, C. S., Schönauer, S., Vishwanathan, S., Smola, A. J., and Kriegel, H.-P. Protein function prediction via graph kernels. *Bioinformatics*, 21:47–56, 2005.

Cai, C. and Wang, Y. A simple yet effective baseline for non-attributed graph classification. *arXiv preprint arXiv:1811.03508*, 2018.

Chen, D., Jacob, L., and Mairal, J. Biological sequence modeling with convolutional kernel networks. 35(18): 3294–3302, 2019a.

Chen, D., Jacob, L., and Mairal, J. Recurrent kernel networks. In *Adv. Neural Information Processing Systems (NeurIPS)*, 2019b.

Du, S. S., Hou, K., Salakhutdinov, R. R., Poczos, B., Wang, R., and Xu, K. Graph neural tangent kernel: Fusing graph neural networks with graph kernels. In *Adv. Neural Information Processing Systems (NeurIPS)*, 2019.

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

Ivanov, S. and Burnaev, E. Anonymous walk embeddings. In *International Conference on Machine Learning*, pp. 2186–2195, 2018.

Kashima, H., Tsuda, K., and Inokuchi, A. Marginalized kernels between labeled graphs. In *International Conference on Machine Learning (ICML)*, 2003.

Kersting, K., Kriege, N. M., Morris, C., Mutzel, P., and Neumann, M. Benchmark data sets for graph kernels, 2016. http://graphkernels.cs.tu-dortmund.de.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, 2015.

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

Kriege, N. M., Morris, C., Rey, A., and Sohler, C. A property testing framework for the theoretical expressivity of graph kernels. In *International Joint Conferences on Artificial Intelligence (IJCAI)*, 2018.

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

Lei, T., Jin, W., Barzilay, R., and Jaakkola, T. Deriving neural architectures from sequence and graph kernels. In *International Conference on Machine Learning (ICML)*, 2017.

Leslie, C., Eskin, E., and Noble, W. S. The spectrum kernel: A string kernel for svm protein classification. In *Biocomputing 2002*, pp. 564–575. 2001.

Leslie, C. S., Eskin, E., Cohen, A., Weston, J., and Noble, W. S. Mismatch string kernels for discriminative protein classification. *Bioinformatics*, 20(4):467–476, 2004.

Mairal, J. End-to-end kernel learning with supervised convolutional kernel networks. In *Adv. Neural Information Processing Systems (NIPS)*, 2016.

Mairal, J. Cyanure: An open-source toolbox for empirical risk minimization for Python, C++, and soon more. *preprint arXiv:1912.08165*, 2019.

Morris, C., Ritzert, M., Fey, M., Hamilton, W. L., Lenssen, J. E., Rattan, G., and Grohe, M. Weisfeiler and Leman go neural: Higher-order graph neural networks. In *AAAI Conference on Artificial Intelligence*, 2019.

Murray, N. and Perronnin, F. Generalized max pooling. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014.

Navarin, N., Tran, D. V., and Sperduti, A. Pre-training graph neural networks with kernels. *preprint arXiv:1811.06930*, 2018.

Niepert, M., Ahmed, M., and Kutzkov, K. Learning convolutional neural networks for graphs. In *International Conference on Machine Learning (ICML)*, 2016.

Nikolentzos, G., Meladianos, P., Tixier, A. J.-P., Skianis, K., and Vazirgiannis, M. Kernel graph convolutional neural networks. In *International Conference on Artificial Neural Networks (ICANN)*, 2018.

Saigo, H., Vert, J.-P., Ueda, N., and Akutsu, T. Protein homology detection using string alignment kernels. *Bioinformatics*, 20(11):1682–1689, 2004.

Saitoh, S. *Integral transforms, reproducing kernels and their applications*, volume 369. CRC Press, 1997.

Shervashidze, N., Vishwanathan, S., Petri, T., Mehlhorn, K., and Borgwardt, K. Efficient graphlet kernels for large graph comparison. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2009.

Shervashidze, N., Schweitzer, P., Leeuwen, E. J. v., Mehlhorn, K., and Borgwardt, K. M. Weisfeiler-Lehman graph kernels. *Journal of Machine Learning Research (JMLR)*, 12:2539–2561, 2011.

Tibshirani, R. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1):267–288, 1996.

Togninalli, M., Ghisu, E., Llinares-López, F., Rieck, B., and Borgwardt, K. Wasserstein Weisfeiler-Lehman graph kernels. In *Adv. Neural Information Processing Systems (NeurIPS)*, 2019.

Verma, N., Boyer, E., and Verbeek, J. Feastnet: Feature-steered graph convolutions for 3d shape analysis. In *IEEE conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

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

Williams, C. K. and Seeger, M. Using the Nyström method to speed up kernel machines. In *Adv. Neural Information Processing Systems (NIPS)*, 2001.

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

Ying, Z., Bourgeois, D., You, J., Zitnik, M., and Leskovec, J. Gnnexplainer: Generating explanations for graph neural networks. In *Adv. Neural Information Processing Systems (NeurIPS)*, 2019.

Zhang, K., Tsang, I. W., and Kwok, J. T. Improved Nyström low-rank approximation and error analysis. In *International Conference on Machine Learning (ICML)*, 2008.

Zhang, M., Cui, Z., Neumann, M., and Chen, Y. An end-to-end deep learning architecture for graph classification. In *AAAI Conference on Artificial Intelligence*, 2018a.

Zhang, Z., Wang, M., Xiang, Y., Huang, Y., and Nehorai, A. Retgk: Graph kernels based on return probabilities of random walks. In *Adv. Neural Information Processing Systems (NeurIPS)*, 2018b.

Appendix

This appendix provides both theoretical and experimental material and is organized as follows: Appendix A presents a classical result, allowing us to characterize the RKHS of the graph kernels we introduce. Appendix B provides additional experimental details that are useful to reproduce our results and additional experimental results. Then, Appendix C explains how to accelerate the computation of GCKN when using walks instead of paths (at the cost of lower expressiveness), and Appendix D presents a proof of Theorem 1 on the expressiveness of WL and walk kernels.

A. Useful Result about RKHSs

The following result characterizes the RKHS of a kernel function when an explicit mapping to a Hilbert space is available. It may be found in classical textbooks (see, *e.g.*, Saitoh, 1997, §2.1).

Theorem 2. *Let $\Phi : \mathcal{X} \to \mathcal{F}$ be a mapping from a data space $\mathcal{X}$ to a Hilbert space $\mathcal{F}$, and let $K(x, x') := \langle \Phi(x), \psi(x') \rangle_{\mathcal{F}}$ for x, x' in $\mathcal{X}$. Consider the Hilbert space*

$$\mathcal{H} := \{ f_z \; ; \; z \in \mathcal{F} \} \quad \textit{s.t.} \quad f_z : x \mapsto \langle z, \Phi(x) \rangle_{\mathcal{F}},$$

endowed with the norm

$$\|f\|_{\mathcal{H}}^2 := \inf_{z \in \mathcal{F}} \left\{ \|z\|_{\mathcal{F}}^2 \quad \textit{s.t.} \quad f = f_z \right\}.$$

Then, $\mathcal{H}$ is the reproducing kernel Hilbert space associated to kernel K.

B. Details on Experimental Setup and Additional Experiments

In this section, we provide additional details and more experimental results. In Section B.1, we provide additional experimental details; in Section **??**, we present a benchmark on graph classification with continuous attributes by using the protocol of Togninalli et al. (2019); in Section B.2, we perform a hyperparameter study for unsupervised GCKN on three datasets, showing that our approach is relatively robust to the choice of hyperparameters. In particular, the number of filters controls the quality of Nyström's kernel approximation: more filters means a better approximation and better results, at the cost of more computation. This is in contrast with a traditional (supervised) GNN, where more filters may lead to overfitting. Finally, Section B.3 provides motif discovery results.

B.1. Experimental Setup and Reproducibility

Hyperparameter search grids. In our experiments for supervised models, we use an Adam optimizer (Kingma & Ba, 2015) for at most 350 epochs with an initial learning rate equal to 0.01 and halved every 50 epochs with a batch size fixed to 32 throughout all datasets; the number of epochs is selected using cross validation following Xu et al. (2019). The full hyperparameter search range is given in Table 3 for both unsupervised and supervised models on all tasks. Note that we include some large values (1.5 and 2.0) for σ to simulate the linear kernel as we discussed in Section 3.3. In fact, the function $\sigma_1(x) = e^{\alpha(x-1)}$ defined in (12) is upper bounded by $e^{-\alpha} + (1 - e^{-\alpha})x$ and lower bounded by $1 + \alpha(x - 1)$ by its convexity at 0 and 1. Their difference is increasing with α and converges to zero when α tends to 0. Hence, when α is small, σ_1 behaves as an affine kernel with a small slope.

Computing infrastructure. Experiments for unsupervised models were conducted by using a shared CPU cluster composed of 2 Intel Xeon E5-2470v2 @2.4GHz CPUs with 16 cores and 192GB of RAM. Supervised models were trained by using a shared GPU cluster, in large parts built with Nvidia gamer cards (Titan X, GTX1080TI). About 20 of these CPUs and 10 of these GPUs were used simultaneously to perform the experiments of this paper.

B.2. Hyperparameter Study

We show here that both unsupervised and supervised models are generally robust to different hyperparameters, including path size k_1, bandwidth parameter σ, regularization parameter λ and their performance grows increasingly with the number

Table 3. Hyperparameter search range

Hyperparameter	Search range
σ ($\alpha = 1/\sigma^2$)	$[0.3; 0.4; 0.5; 0.6; 1.0; 1.5; 2.0]$
local/global pooling	[sum, mean, max]
path size k_1	integers between 2 and 12
number of filters (unsup)	[32; 128; 512; 1024]
number of filters (sup)	[32; 64] and 256 for ENZYMES
λ (unsup)	$1/n \times$ np.logspace(-3, 4, 60)
λ (sup)	[0.01; 0.001; 0.0001; 1e-05; 1e-06; 1e-07]

of filters q. The accuracies for NCI1, PROTEINS and IMDBMULTI are given in Figure 4, by varying respectively the number of filters, the path size, the bandwidth parameter and regularization parameter when fixing other parameters which give the best accuracy. Supervised models generally require fewer number of filters to achieve similar performance to its unsupervised counterpart. In particular on the NCI1 dataset, the supervised GCKN outperforms its unsupervised counterpart by a significant margin when using a small number of filters.

B.3. Model Interpretation

Implementation details. We use a similar experimental setting as Ying et al. (2019) to train a supervised GCKN-subtree model on Mutagenicity dataset, consisting of 4337 molecule graphs labeled according to their mutagenic effect. Specifically, we use the same split for train and validation set and train a GCKN-subtree model with $k_1 = 3$, which is similar to a 3-layer GNN model. The number of filters is fixed to 20, the same as Ying et al. (2019). The bandwidth parameter σ is fixed to 0.4, local and global pooling are fixed to mean pooling, the regularization parameter λ is fixed to 1e-05. We use an Adam optimizer with initial learning equal to 0.01 and halved every 50 epochs, the same as previously. The accuracy of the trained model is assured to be more than 80% on the test set as Ying et al. (2019). Then we use the procedure described in Section 4 to interpret our trained model. We use an LBFGS optimizer and fixed μ to 0.01. The final subgraph for each given graph is obtained by extracting the maximal connected component formed by the selected paths. A contribution score for each edge can also be obtained by gathering the weights M of all the selected paths that pass through this edge.

More results. More motifs extracted by GCKN are shown in Figure 5 for the Mutagenicity dataset. We recovered some benzene ring or polycyclic aromatic groups which are known to be mutagenic. We also found some groups whose mutagenicity is not known, such as polyphenylene sulfide in the fourth subgraph and 2-chloroethyl- in the last subgraph.

C. Fast Computation of GCKN with Walks

Here we discuss an efficient computational variant using walk kernel instead of path kernel, at the cost of losing some expressive power. Let us consider a relaxed walk kernel by analogy to (8) with

$$\kappa_{\text{base}}^{(k)}(u, u') = \sum_{p \in \mathcal{W}_k(G,u)} \sum_{p' \in \mathcal{W}_k(G',u')} \kappa_1(\varphi_0(p), \varphi_0'(p')), \tag{19}$$

using walks instead of paths and with κ_1 the Gaussian kernel defined in (9). As Gaussian kernel can be decomposed as a product of the Gaussian kernel on pair of nodes at each position

$$\kappa_1(\varphi_0(p), \varphi_0'(p')) = \prod_{j=1}^{k} \kappa_1(\varphi_0(p_j), \varphi_0'(p_j')),$$

We can obtain similar recursive relation as for the original walk kernel in Lemma 2

$$\kappa_{\text{base}}^{(k)}(u, u') = \kappa_1(\varphi_0(u), \varphi_0'(u')) \sum_{v \in \mathcal{N}(u)} \sum_{v' \in \mathcal{N}(u')} \kappa_{\text{base}}^{(k-1)}(v, v'). \tag{20}$$

After applying the Nyström method, the approximate feature map in (13) becomes

$$\psi_1(u) = \sigma_1(Z^\top Z)^{-\frac{1}{2}} c_k(u),$$

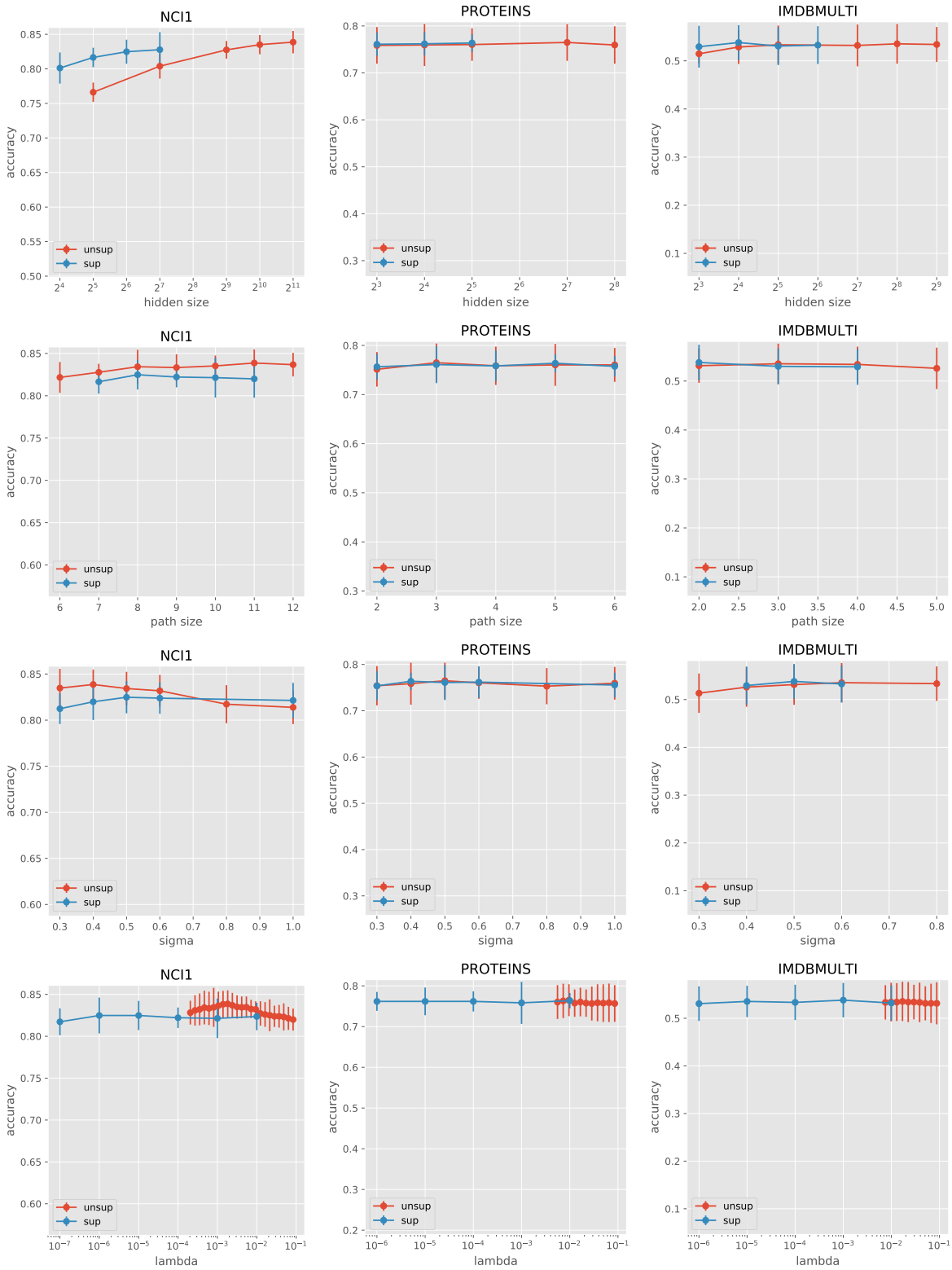


Figure 4. Hyperparamter study: sensibility to different hyperparameters for unsupervised and supervised GCKN-subtree models. The row from top to bottom respectively corresponds to number of filters q_1, path size k_1, bandwidth parameter σ and regularization parameter λ. The column from left to right corresponds to different datasets: NC11, PROTEINS and IMDBMULTI.

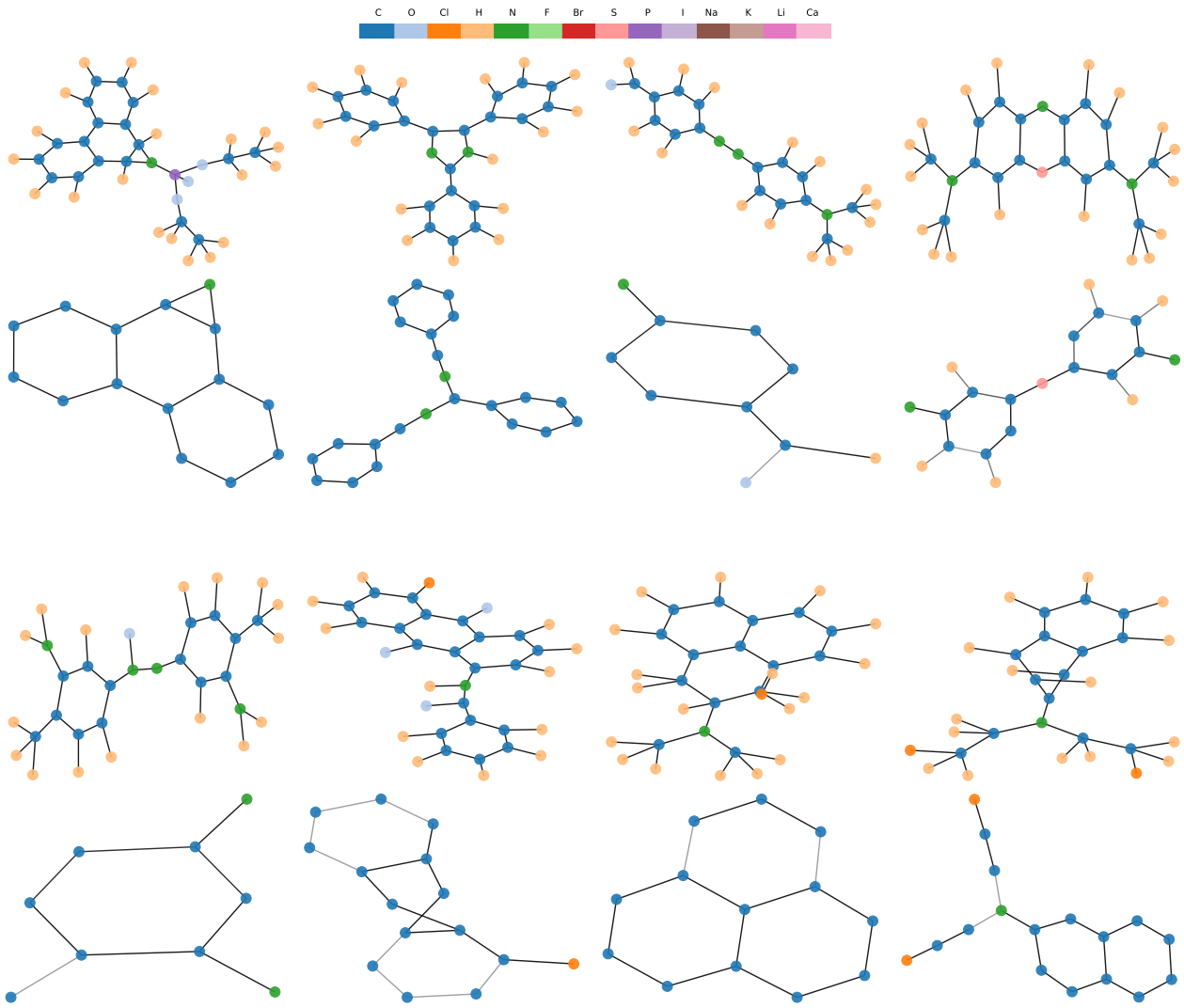


Figure 5. More motifs extracted by GCKN on Mutagenicity dataset. First and third rows are original graphs; second and fourth rows are corresponding motifs. Some benzene ring or polycyclic aromatic groups are identified, which are known to be mutagenic. In addition, Some chemical groups whose mutagenicity is not known are also identified, such as polyphenylene sulfide in the fourth subgraph and 2-chloroethyl- in the last subgraph.

where for any $0 \leq j \leq k$, $c_j(u) := \sum_{p \in \mathcal{W}_j(G,u)} \sigma_1(Z_j^\top \psi_0(p))$ and Z_j in $\mathbb{R}^{q_0(j+1) \times q_1}$ denotes the matrix consisting of the $j+1$ last columns of q_1 anchor points. Using the above recursive relation (20) and similar arguments in *e.g.* (Chen et al., 2019b), we can show c_j obeys the following recursive relation

$$c_j(u) = b_j(u) \odot \sum_{v \in \mathcal{N}(u)} c_{j-1}(v), \ 1 \leq j \leq k, \tag{21}$$

where $\odot$ denotes the element-wise product and $b_j(u)$ is a vector in $\mathbb{R}^{q_1}$ whose entry i in $\{1, \dots, q_1\}$ is $\kappa_1(u, z_i^{(k+1-j)})$ and $z_i^{(k+1-j)}$ denotes the $k+1-j$-th column vector of z_i in $\mathbb{R}^{q_0}$. In practice, $\sum_{v \in \mathcal{N}(u)} c_{j-1}(v)$ can be computed efficiently by multiplying the adjacency matrix with the $|\mathcal{V}|$-dimensional vector with entries $c_{j-1}(v)$ for $v \in \mathcal{V}$.

D. Proof of Theorem 1

Before presenting and proving the link between the WL subtree kernel and the walk kernel, we start by reminding and showing some useful results about the WL subtree kernel and the walk kernel.

D.1. Useful results for the WL subtree kernel

We first recall a recursive relation of the WL subtree kernel, given in the Theorem 8 of Shervashidze et al. (2011). Let us denote by $\mathcal{M}(u, u')$ the set of exact matchings of subsets of the neighbors of u and u', formally given by

$$\mathcal{M}(u,u') = \Big\{ R \subseteq \mathcal{N}(u) \times \mathcal{N}(u') \ \Big| \ |R| = |\mathcal{N}(u)| = |\mathcal{N}(u')| \wedge \\ (\forall (v,v'), (w,w') \in R : u = w \Leftrightarrow u' = w') \wedge (\forall (u,u') \in R : a(u) = a'(u')) \Big\}. \tag{22}$$

Then we have the following recursive relation for $\kappa_{\text{subtree}}^{(k)}(u,u') := \delta(a_k(u), a'_k(u'))$

$$\kappa_{\text{subtree}}^{(k+1)}(u,u') = \begin{cases} \kappa_{\text{subtree}}^{(k)}(u,u') \max\limits_{R \in \mathcal{M}(u,u')} \prod\limits_{(v,v') \in R} \kappa_{\text{subtree}}^{(k)}(v,v'), & \text{if } \mathcal{M}(u,u') \neq \emptyset, \\ 0, & \text{otherwise.} \end{cases} \tag{23}$$

We can further simply the above recursion using the following Lemma

Lemma 1. *If $\mathcal{M}(u,u') \neq \emptyset$, we have*

$$\kappa_{\textit{subtree}}^{(k+1)}(u,u') = \delta(a(u), a'(u')) \max_{R \in \mathcal{M}(u,u')} \prod_{(v,v') \in R} \kappa_{\textit{subtree}}^{(k)}(v,v').$$

Proof. We prove this by induction on $k \geq 0$. For $k = 0$, this is true by the definition of $\kappa_{\text{subtree}}^{(0)}$. For $k \geq 1$, we suppose that $\kappa_{\text{subtree}}^{(k)}(u,u') = \delta(a(u), a'(u')) \max_{R \in \mathcal{M}(u,u')} \prod_{(v,v') \in R} \kappa_{\text{subtree}}^{(k-1)}(v,v')$. We have

$$\begin{aligned} \kappa_{\text{subtree}}^{(k+1)}(u,u') &= \kappa_{\text{subtree}}^{(k)}(u,u') \max_{R \in \mathcal{M}(u,u')} \prod_{(v,v') \in R} \kappa_{\text{subtree}}^{(k)}(v,v') \\ &= \delta(a(u), a'(u')) \max_{R \in \mathcal{M}(u,u')} \prod_{(v,v') \in R} \kappa_{\text{subtree}}^{(k-1)}(v,v') \max_{R \in \mathcal{M}(u,u')} \prod_{(v,v') \in R} \kappa_{\text{subtree}}^{(k)}(v,v'). \end{aligned}$$

It suffices to show

$$\max_{R \in \mathcal{M}(u,u')} \prod_{(v,v') \in R} \kappa_{\text{subtree}}^{(k-1)}(v,v') \max_{R \in \mathcal{M}(u,u')} \prod_{(v,v') \in R} \kappa_{\text{subtree}}^{(k)}(v,v') = \max_{R \in \mathcal{M}(u,u')} \prod_{(v,v') \in R} \kappa_{\text{subtree}}^{(k)}(v,v').$$

Since the only values can take for $\kappa_{\text{subtree}}^{(k-1)}$ is 0 and 1, the only values that $\max_{R \in \mathcal{M}(u,u')} \prod_{(v,v') \in R} \kappa_{\text{subtree}}^{(k-1)}(v,v')$ can take is also 0 and 1. Then we can split the proof on these two conditions. It is obvious if this term is equal to 1. If this term is equal to 0, then

$$\max_{R \in \mathcal{M}(u,u')} \prod_{(v,v') \in R} \kappa_{\text{subtree}}^{(k)}(v,v') \leq \max_{R \in \mathcal{M}(u,u')} \prod_{(v,v') \in R} \kappa_{\text{subtree}}^{(k-1)}(v,v') = 0,$$

as all terms are not negative and $\kappa^{(k)}_{\text{subtree}}(v, v')$ is not creasing on k. Then $\max_{R \in \mathcal{M}(u,u')} \prod_{(v,v') \in R} \kappa^{(k)}_{\text{subtree}}(v, v') = 0$ and we have 0 for both sides. □

D.2. Recursive relation for the walk kernel

We recall that the k-walk kernel is defined as

$$K(G, G') = \sum_{u \in \mathcal{V}} \sum_{u' \in \mathcal{V}'} \kappa^{(k)}_{\text{walk}}(u, u'),$$

where

$$\kappa^{(k)}_{\text{walk}}(u, u') = \sum_{p \in \mathcal{W}_k(G,u)} \sum_{p' \in \mathcal{W}_k(G',u')} \delta(a(p), a'(p')).$$

The feature map of this kernel is given by

$$\varphi^{(k)}_{\text{walk}}(u) = \sum_{p \in \mathcal{W}_k(G,u)} \varphi_\delta(a(p)),$$

where φ_δ is the feature map associated with δ. We give here a recursive relation for the walk kernel on the size of walks, thanks to its allowance of nodes to repeat.

Lemma 2. *For any $k \geq 0$, we have*

$$\kappa^{(k+1)}_{walk}(u, u') = \delta(a(u), a'(u')) \sum_{v \in \mathcal{N}(u)} \sum_{v' \in \mathcal{N}(u')} \kappa^{(k)}_{walk}(v, v'). \tag{24}$$

Proof. Noticing that we can always decompose a path $p \in \mathcal{W}_{k+1}(G, u)$, with (u, v) the first edge that it passes and $v \in \mathcal{N}(u)$, into (u, q) with $q \in \mathcal{W}_k(G, v)$, then we have

$$\begin{aligned}
\kappa^{(k+1)}_{\text{walk}}(u, u') &= \sum_{p \in \mathcal{W}_{k+1}(G,u)} \sum_{p' \in \mathcal{W}_{k+1}(G',u')} \delta(a(p), a'(p')) \\
&= \sum_{v \in \mathcal{N}(u)} \sum_{p \in \mathcal{W}_k(G,v)} \sum_{v' \in \mathcal{N}(u')} \sum_{p' \in \mathcal{W}_k(G,v')} \delta(a(u), a'(u')) \delta(a(p), a'(p')) \\
&= \delta(a(u), a'(u')) \sum_{v \in \mathcal{N}(u)} \sum_{v' \in \mathcal{N}(u')} \sum_{p \in \mathcal{W}_k(G,v)} \sum_{p' \in \mathcal{W}_k(G',v')} \delta(a(p), a'(p')) \\
&= \delta(a(u), a'(u')) \sum_{v \in \mathcal{N}(u)} \sum_{v' \in \mathcal{N}(u')} \kappa^{(k)}_{\text{walk}}(v, v').
\end{aligned}$$

□

This relation also provides us a recursive relation for the feature maps of the walk kernel

$$\varphi^{(k+1)}_{\text{walk}}(u) = \varphi_\delta(a(u)) \otimes \sum_{v \in \mathcal{N}(u)} \varphi^{(k)}_{\text{walk}}(v),$$

where $\otimes$ denotes the tensor product.

D.3. Discriminative power between walk kernel and WL subtree kernel

Before proving the Theorem 1, let us first show that the WL subtree kernel is always more discriminative than the walk kernel.

Proposition 1. *For any node u in graph G and u' in graph G' and any $k \geq 0$, then $d_{\kappa^{(k)}_{subtree}}(u, u') = 0 \implies d_{\kappa^{(k)}_{walk}}(u, u') = 0$.*

This proposition suggests that though both of their feature maps are not injective (see e.g. Kriege et al. (2018)), the feature map of $\kappa^{(k)}_{\text{subtree}}$ is more injective in the sense that for a node u, its collision set $\{u' \in \mathcal{V} \,|\, \varphi(u') = \varphi(u)\}$ for $\kappa^{(k)}_{\text{subtree}}$, with φ the corresponding feature map, is included in that for $\kappa^{(k)}_{\text{walk}}$. Furthermore, if we denote by $\hat{\kappa}$ the normalized kernel of κ such that $\hat{\kappa}(u,u') = \kappa(u,u')/\sqrt{\kappa(u,u)\kappa(u',u')}$, then we have

Corolary 1. *For any node u in graph G and u' in graph G' and any $k \geq 0$, $d_{\kappa^{(k)}_{subtree}}(u,u') \geq d_{\hat{\kappa}^{(k)}_{walk}}(u,u')$.*

Proof. We prove by induction on k. It is clear for $k = 0$ as both kernels are equal to the Dirac kernel on the node attributes. Let us suppose this is true for $k \geq 0$, we will show this is also true for $k+1$. We suppose $d_{\kappa^{(k+1)}_{\text{subtree}}}(u,u') = 0$. Since $\kappa^{(k+1)}_{\text{subtree}}(u,u) = 1$, by equality (23) we have

$$1 = \kappa^{(k+1)}_{\text{subtree}}(u,u') = \kappa^{(k)}_{\text{subtree}}(u,u') \max_{R \in \mathcal{M}(u,u')} \prod_{(v,v') \in R} \kappa^{(k)}_{\text{subtree}}(v,v'),$$

which implies that $\kappa^{(k)}_{\text{subtree}}(u,u') = 1$ and $\max_{R \in \mathcal{M}(u,u')} \prod_{(v,v') \in R} \kappa^{(k)}_{\text{subtree}}(v,v') = 1$. Then $\delta(a(u), a'(u)) = 1$ by the non-growth of $\kappa^{(k)}_{\text{subtree}}(u,u')$ on k and it exists an exact matching $R^\star \in \mathcal{M}(u,u')$ such that $|\mathcal{N}(u)| = |\mathcal{N}(u')| = |R^\star|$ and $\forall (v,v') \in R^\star$, $\kappa^{(k)}_{\text{subtree}}(v,v') = 1$. Therefore, we have $d_{\kappa^{(k)}_{\text{walk}}}(v,v') = 0$ for all $(v,v') \in R^\star$ by the induction hypothesis.

On the other hand, by Lemma 2 we have

$$\begin{aligned}
\kappa^{(k+1)}_{\text{walk}}(u,u') &= \delta(a(u), a'(u')) \sum_{v \in \mathcal{N}(u)} \sum_{v' \in \mathcal{N}(u')} \kappa^{(k)}_{\text{walk}}(v,v') \\
&= \sum_{v \in \mathcal{N}(u)} \sum_{v' \in \mathcal{N}(u')} \kappa^{(k)}_{\text{walk}}(v,v'),
\end{aligned}$$

which suggest that the feature map of $\kappa^{(k+1)}_{\text{walk}}$ can be written as $\varphi^{(k+1)}_{\text{walk}}(u) = \sum_{v \in \mathcal{N}(u)} \varphi^{(k)}_{\text{walk}}(v)$. Then we have

$$\begin{aligned}
d_{\kappa^{(k+1)}_{\text{walk}}}(u,u') &= \left\| \sum_{v \in \mathcal{N}(u)} \varphi^{(k)}_{\text{walk}}(v) - \sum_{v' \in \mathcal{N}(u')} \varphi^{(k)}_{\text{walk}}(v') \right\| \\
&= \left\| \sum_{(v,v') \in R^\star} \varphi^{(k)}_{\text{walk}}(v) - \varphi^{(k)}_{\text{walk}}(v') \right\| \\
&\leq \sum_{(v,v') \in R^\star} \|\varphi^{(k)}_{\text{walk}}(v) - \varphi^{(k)}_{\text{walk}}(v')\| \\
&= \sum_{(v,v') \in R^\star} d_{\kappa^{(k)}_{\text{walk}}}(v,v') = 0.
\end{aligned}$$

We conclude that $d_{\kappa^{(k+1)}_{\text{walk}}}(u,u') = 0$.

Now let us prove the Corollary 1. The only values that $d_{\kappa^{(k)}_{\text{subtree}}}(u,u')$ can take are 0 and 1. Since $d_{\hat{\kappa}^{(k)}_{\text{walk}}}(u,u')$ is always not larger than 1, we only need to prove $d_{\kappa^{(k)}_{\text{subtree}}}(u,u') = 0 \implies d_{\hat{\kappa}^{(k)}_{\text{walk}}}(u,u') = 0$, which has been shown above. □

D.4. Proof of Theorem 1

Note that using our notation here, $\varphi_1 = \varphi^{(k)}_{\text{walk}}$

Proof. We prove by induction on k. For $k = 0$, we have for any $u \in \mathcal{V}$ and $u' \in \mathcal{V}'$

$$\kappa^{(0)}_{\text{subtree}}(u,u') = \delta(a(u), a'(u')) = \delta(\varphi^{(0)}_{\text{walk}}(u), \varphi^{(0)}_{\text{walk}}(u')).$$

Assume that (16) is true for $k \geq 0$. We want to show this is also true for $k+1$. As the only values that the δ kernel can take is 0 and 1, it suffices to show the equality between $\kappa^{(k+1)}_{\text{subtree}}(u,u')$ and $\delta(\varphi^{(k+1)}_{\text{walk}}(u), \varphi^{(k+1)}_{\text{walk}}(u'))$ in these two situations.

- If $\kappa_{\text{subtree}}^{(k+1)}(u, u') = 1$, by Proposition 1 we have $\varphi_{\text{walk}}^{(k+1)}(u) = \varphi_{\text{walk}}^{(k+1)}(u')$, and thus $\delta(\varphi_{\text{walk}}^{(k+1)}(u), \varphi_{\text{walk}}^{(k+1)}(u')) = 1$.
- If $\kappa_{\text{subtree}}^{(k+1)}(u, u') = 0$, by the recursive relation of the feature maps in Lemma 2, we have

$$\delta(\varphi_{\text{walk}}^{(k+1)}(u), \varphi_{\text{walk}}^{(k+1)}(u')) = \delta(a(u), a'(u'))\delta\left(\sum_{v\in\mathcal{N}(u)} \varphi_{\text{walk}}^{(k)}(v), \sum_{v'\in\mathcal{N}(u')} \varphi_{\text{walk}}^{(k)}(v')\right).$$

By Lemma 1, it suffices to show that

$$\max_{R\in\mathcal{M}(u,u')} \prod_{(v,v')\in R} \kappa_{\text{subtree}}^{(k)}(u, u') = 0 \implies \delta\left(\sum_{v\in\mathcal{N}(u)} \varphi_{\text{walk}}^{(k)}(v), \sum_{v'\in\mathcal{N}(u')} \varphi_{\text{walk}}^{(k)}(v')\right) = 0.$$

The condition $|\mathcal{M}(u, u')| = 1$ suggests that there exists exactly one matching of the neighbors of u and u'. Let us denote this matching by R. The left equality implies that there exists a non-empty subset of neighbor pairs $S \subseteq R$ such that $\kappa_{\text{subtree}}^{(k)}(v, v') = 0$ for any $(v, v') \in S$ and $\kappa_{\text{subtree}}^{(k)}(v, v') = 1$ for all $(v, v') \notin S$. Then by the induction hypothesis, $\varphi_{\text{walk}}^{(k)}(v) = \varphi_{\text{walk}}^{(k)}(v')$ for all $(v, v') \notin S$ and $\varphi_{\text{walk}}^{(k)}(v) \neq \varphi_{\text{walk}}^{(k)}(v')$ for all $(v, v') \in S$. Consequently, $\sum_{(v,v')\notin S} \varphi_{\text{walk}}^{(k)}(v) - \varphi_{\text{walk}}^{(k)}(v') = 0$. Now we will show $\sum_{(v,v')\in S} \varphi_{\text{walk}}^{(k)}(v) - \varphi_{\text{walk}}^{(k)}(v') \neq 0$ since all neighbors of either u or u' have distinct attributes. Then

$$\begin{aligned}
&\| \sum_{v\in\mathcal{N}(u)} \varphi_{\text{walk}}^{(k)}(v) - \sum_{v'\in\mathcal{N}(u')} \varphi_{\text{walk}}^{(k)}(v')\| \\
=&\| \sum_{(v,v')\in R} \varphi_{\text{walk}}^{(k)}(v) - \varphi_{\text{walk}}^{(k)}(v')\| \\
=&\| \sum_{(v,v')\in S} \varphi_{\text{walk}}^{(k)}(v) - \varphi_{\text{walk}}^{(i)}(v')\| > 0.
\end{aligned}$$

Therefore, $\delta\left(\sum_{v\in\mathcal{N}(u)} \varphi_{\text{walk}}^{(k)}(v), \sum_{v'\in\mathcal{N}(u')} \varphi_{\text{walk}}^{(k)}(v')\right) = 0$.

□