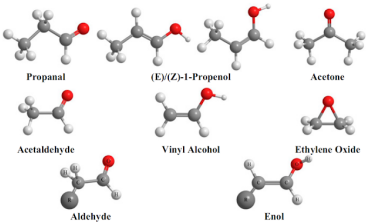
Convolutional Kernel Networks for Graph-Structured Data

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

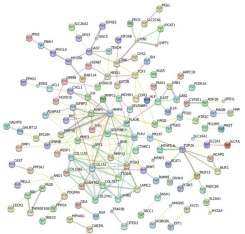
[1]Inria Grenoble [2]CNRS/LBBE Lyon

ICML 2020

Graph-structured data are ubiquitous

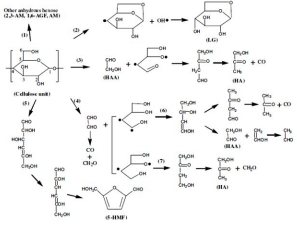


(a) molecules



(b) protein regulation

(c) social networks



(d) chemical pathways

Learning graph representations

State-of-the-art models for representing graphs

- **Deep learning for graphs**: graph neural networks (GNNs)
- **Graph kernels**: Weisfeiler-Lehman (WL) graph kernels
- **Hybrid models** attempt to bridge both worlds: graph neural tangent kernels

Learning graph representations

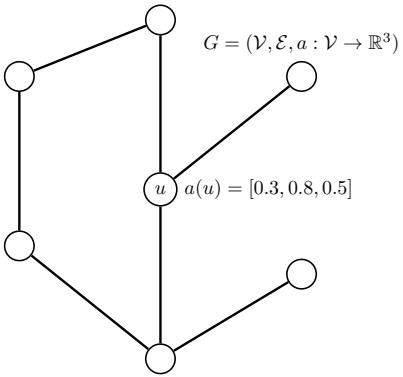
State-of-the-art models for representing graphs

- **Deep learning for graphs**: graph neural networks (GNNs)
- **Graph kernels**: Weisfeiler-Lehman (WL) graph kernels
- **Hybrid models** attempt to bridge both worlds: graph neural tangent kernels

Our model:

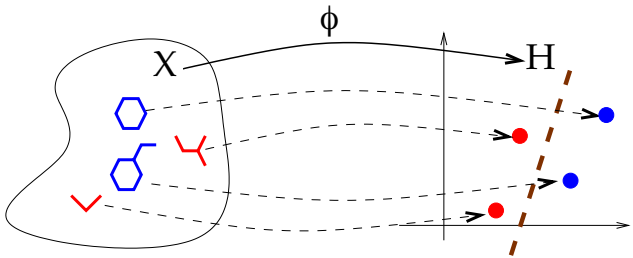
- A new type of **multilayer** graph kernel: more **expressive** than WL kernels
- Learning easy-to-regularize and scalable **unsupervised** graph representations
- Learning **supervised** graph representations like GNNs

Graphs with node attributes



- A graph is defined as a triplet $(\mathcal{V}, \mathcal{E}, a)$;
- $\mathcal{V}$ and $\mathcal{E}$ correspond to the set of vertices and edges;
- $a : \mathcal{V} \to \mathbb{R}^d$ is a function assigning attributes to each node.

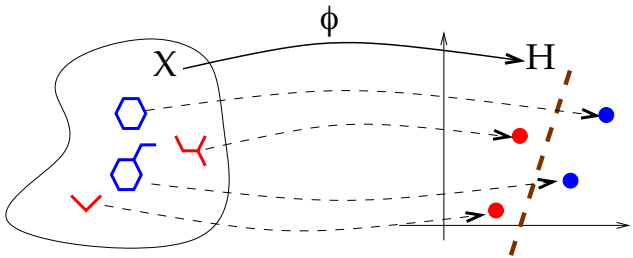
Graph kernel mappings



- Map each graph G in $\mathcal{X}$ to a vector $\varphi(G)$ in $\mathcal{H}$, which lends itself to learning tasks.

[Lei et al., 2017, Kriege et al., 2019]

Graph kernel mappings

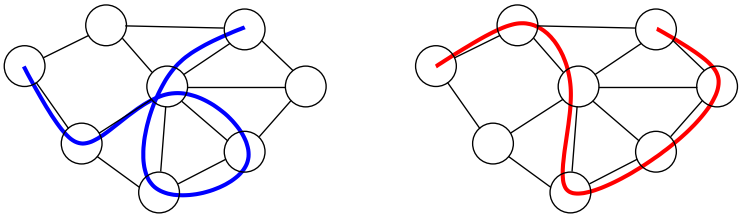


- Map each graph G in $\mathcal{X}$ to a vector $\varphi(G)$ in $\mathcal{H}$, which lends itself to learning tasks.
- A large class of graph kernel mappings can be written in the form

$$\varphi(G) := \sum_{u \in \mathcal{V}} \varphi_{\text{base}}(\ell_G(u)) \quad \text{where } \varphi_{\text{base}} \text{ embeds some local patterns } \ell_G(u) \text{ to } \mathcal{H}.$$

[Lei et al., 2017, Kriege et al., 2019]

Basic kernels: walk and path kernel mappings

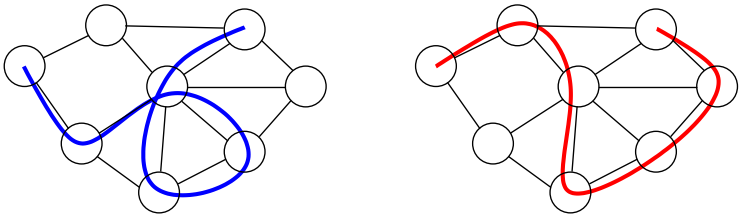


- $\mathcal{P}_k(G, u) :=$ paths of length k from node u in G. The **k-path** mapping is

$$\varphi_{\text{path}}(u) := \sum_{p \in \mathcal{P}_k(G,u)} \delta_{a(p)}(\cdot)$$

- $a(p)$: concatenated attributes in p; δ: the Dirac function.
- $\varphi_{\text{path}}(u)$ can be interpreted as a **histogram** of paths occurrences.

Basic kernels: walk and path kernel mappings

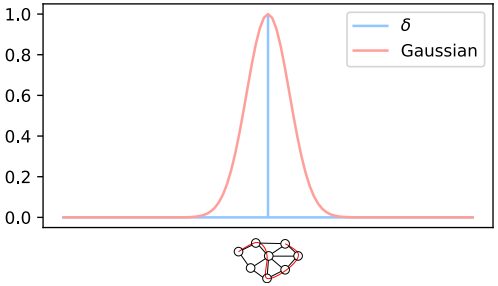


- $\mathcal{P}_k(G, u) :=$ paths of length k from node u in G. The **k-path** mapping is

$$\varphi_{\text{path}}(u) := \sum_{p \in \mathcal{P}_k(G,u)} \delta_{a(p)}(\cdot)$$

- $a(p)$: concatenated attributes in p; δ: the Dirac function.
- $\varphi_{\text{path}}(u)$ can be interpreted as a **histogram** of paths occurrences.
- Path kernels are more **expressive** than walk kernels, but less preferred for **computational** reasons.

A relaxed path kernel

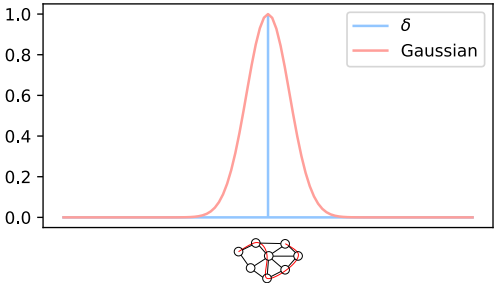


$$\varphi_{\text{path}}(u) = \sum_{p \in \mathcal{P}_k(G,u)} \delta_{a(p)}(\cdot)$$

Issues of the path kernel mapping:

- δ allows hard comparison between paths thus only works for discrete attributes.
- δ is not differentiable, which cannot be "optimized" with back-propagation.

A relaxed path kernel



$$\varphi_{\text{path}}(u) = \sum_{p \in \mathcal{P}_k(G,u)} \delta_{a(p)}(\cdot)$$

$$\Longrightarrow \sum_{p \in \mathcal{P}_k(G,u)} e^{-\frac{\alpha}{2}\|a(p)-\cdot\|^2}.$$

Issues of the path kernel mapping:

- δ allows hard comparison between paths thus only works for discrete attributes.
- δ is not differentiable, which cannot be "optimized" with back-propagation.

Relax it with a "soft" and differentiable mapping

- interpreted as the sum of Gaussians centered at each path features from u.

One-layer GCKN: a closer look on the relaxed path kernel

- We define the one-layer GCKN as the relaxed path kernel mapping

$$\varphi_1(u) := \sum_{p \in \mathcal{P}_k(G,u)} e^{-\frac{\alpha_1}{2}\|a(p)-\cdot\|^2} = \sum_{p \in \mathcal{P}_k(G,u)} \Phi_1(a(p)) \in \mathcal{H}_1.$$

- This formula can be divided into **3 steps**:
  - path extraction: enumerating all $\mathcal{P}_k(G, u)$
  - kernel mapping: evaluating Gaussian embedding Φ_1 of path features
  - path aggregation: aggregating the path embeddings

One-layer GCKN: a closer look on the relaxed path kernel

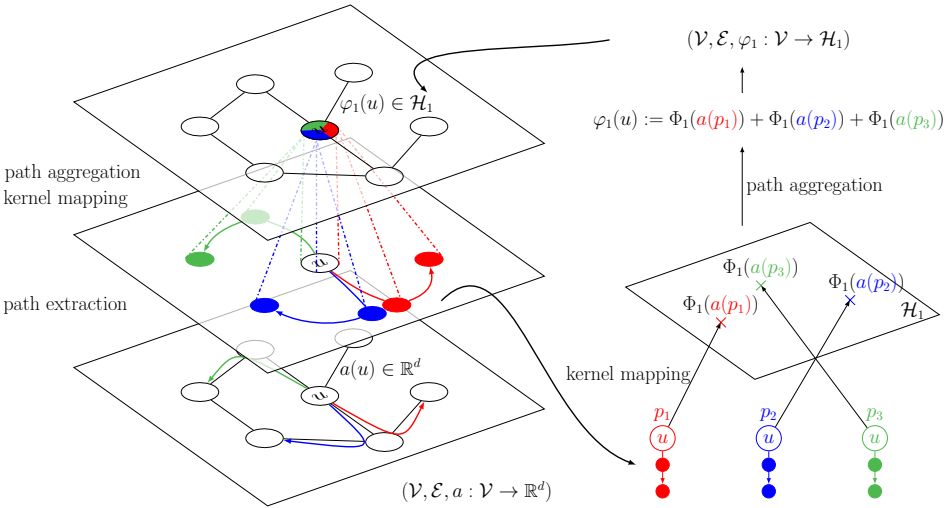
- We define the one-layer GCKN as the relaxed path kernel mapping

$$\varphi_1(u) := \sum_{p \in \mathcal{P}_k(G,u)} e^{-\frac{\alpha_1}{2}\|a(p)-\cdot\|^2} = \sum_{p \in \mathcal{P}_k(G,u)} \Phi_1(a(p)) \in \mathcal{H}_1.$$

- This formula can be divided into **3 steps**:
  - path extraction: enumerating all $\mathcal{P}_k(G, u)$
  - kernel mapping: evaluating Gaussian embedding Φ_1 of path features
  - path aggregation: aggregating the path embeddings
- We obtain a new graph with the same topology but different features

$$(\mathcal{V}, \mathcal{E}, a) \xrightarrow{\varphi_{\text{path}}} (\mathcal{V}, \mathcal{E}, \varphi_1)$$

Construction of one-layer GCKN



From one-layer to multilayer GCKN

- We can repeat applying φ_{path} to the new graph

$$(\mathcal{V}, \mathcal{E}, a) \xrightarrow{\varphi_{\text{path}}} (\mathcal{V}, \mathcal{E}, \varphi_1) \xrightarrow{\varphi_{\text{path}}} (\mathcal{V}, \mathcal{E}, \varphi_2) \xrightarrow{\varphi_{\text{path}}} \ldots \xrightarrow{\varphi_{\text{path}}} (\mathcal{V}, \mathcal{E}, \varphi_j).$$

- $\varphi_j(u)$ represents the information about a neighborhood of u.
- Final graph representation at layer j, $\varphi_j(G) = \sum_{u \in \mathcal{V}} \varphi_j(u)$.

From one-layer to multilayer GCKN

- We can repeat applying φ_{path} to the new graph

$$(\mathcal{V}, \mathcal{E}, a) \xrightarrow{\varphi_{\text{path}}} (\mathcal{V}, \mathcal{E}, \varphi_1) \xrightarrow{\varphi_{\text{path}}} (\mathcal{V}, \mathcal{E}, \varphi_2) \xrightarrow{\varphi_{\text{path}}} \dots \xrightarrow{\varphi_{\text{path}}} (\mathcal{V}, \mathcal{E}, \varphi_j).$$

- $\varphi_j(u)$ represents the information about a neighborhood of u.
- Final graph representation at layer j, $\varphi_j(G) = \sum_{u \in \mathcal{V}} \varphi_j(u)$.
- Why is the multilayer model interesting ?
  - applying φ_{path} once can capture **paths**: GCKN-path;
  - applying twice can capture **subtrees**: GCKN-subtree;
  - so applying even more times may capture **higher-order structures** ?
  - **Long paths** cannot be enumerated due to computational complexity, yet multilayer model can capture **long-range substructures**.

Scalable approximation of Gaussian kernel mapping

$$\varphi_{\text{path}}(u) = \sum_{p \in \mathcal{P}_k(G,u)} \Phi(a(p))$$

- $\Phi(x) = e^{-\frac{\alpha}{2}\|x-\cdot\|^2} \in \mathcal{H}$ is infinite-dimensional and can be expensive to compute.

[Chen et al., 2019a,b]

Scalable approximation of Gaussian kernel mapping

$$\varphi_{\text{path}}(u) = \sum_{p \in \mathcal{P}_k(G,u)} \Phi(a(p))$$

- $\Phi(x) = e^{-\frac{\alpha}{2}\|x-\cdot\|^2} \in \mathcal{H}$ is infinite-dimensional and can be expensive to compute.
- **Nyström** provides a **finite-dimensional** approximation $\Psi(x) \in \mathbb{R}^q$ by orthogonally projecting $\Phi(x)$ onto some finite-dimensional subspace:

  $$\text{span}(\Phi(z_1), \ldots, \Phi(z_q)) \text{ parametrized by } Z = \{z_1, \ldots, z_q\},$$

  where $z_j \in \mathbb{R}^{dk}$ can be interpreted as path features.

[Chen et al., 2019a,b]

Scalable approximation of Gaussian kernel mapping

$$\varphi_{\text{path}}(u) = \sum_{p \in \mathcal{P}_k(G,u)} \Phi(a(p))$$

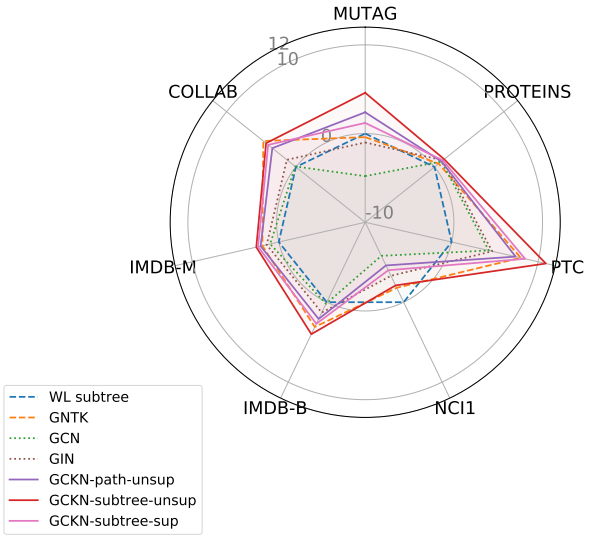
- $\Phi(x) = e^{-\frac{\alpha}{2}\|x-\cdot\|^2} \in \mathcal{H}$ is infinite-dimensional and can be expensive to compute.
- **Nyström** provides a **finite-dimensional** approximation $\Psi(x) \in \mathbb{R}^q$ by orthogonally projecting $\Phi(x)$ onto some finite-dimensional subspace:

$$\text{span}(\Phi(z_1), \ldots, \Phi(z_q)) \text{ parametrized by } Z = \{z_1, \ldots, z_q\},$$

  where $z_j \in \mathbb{R}^{dk}$ can be interpreted as path features.
- The parameters Z can be learned by
  - (unsupervised) K-means on the set of path features;
  - (supervised) end-to-end learning with back-propagation.

[Chen et al., 2019a,b]

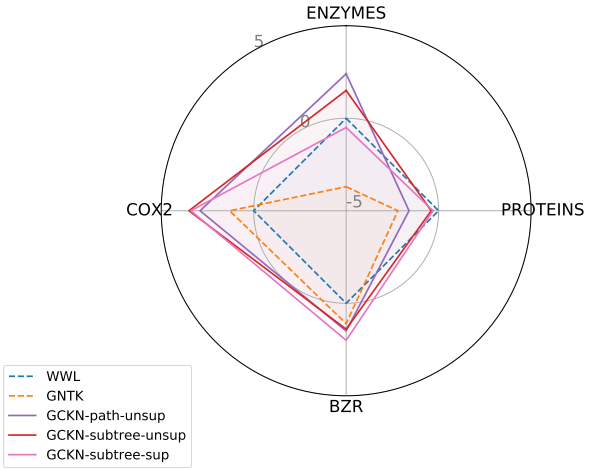
Experiments on graphs with discrete attributes



- Accuracy improvement with respect to the WL subtree kernel.
- GCKN-path already outperforms the baselines.
- Increasing number of layers brings larger improvement.
- Supervised learning does not improve performance, but leads to more compact representations.

[Shervashidze et al., 2011, Du et al., 2019, Xu et al., 2019, Kipf and Welling, 2017]

Experiments on graphs with continuous attributes

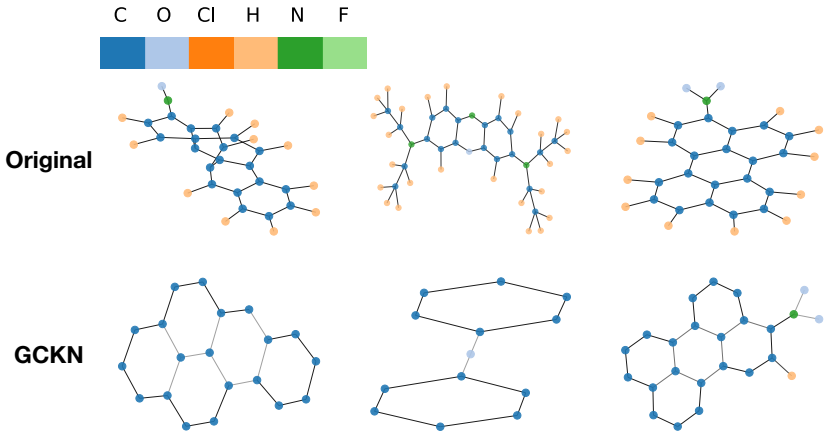


- Accuracy improvement with respect to the WWL kernel.
- Results similar to discrete case.
- Path features seem presumably predictive enough.

[Du et al., 2019, Togninalli et al., 2019]

Model interpretation for mutagenicity prediction

- Idea: find the minimal connected component that preserves the prediction.



[Ying et al., 2019]

Take-home messages

- GCKN is a **multilayer kernel** for graphs based on **paths**, which allows to control the trade-off between **computation** and **expressiveness**.
- Its graph representations can be learned in both **supervised** and **unsupervised** fashions. Unsupervised models are **easy-to-regularize** and **scalable**.
- A straightforward model **interpretation** is also provided.
- **Our code is freely available at** `https://github.com/claying/GCKN`.

References I

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

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

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

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

N. M. Kriege, M. Neumann, C. Morris, K. Kersting, and P. Mutzel. A unifying view of explicit and implicit feature maps of graph kernels. *Data Mining and Knowledge Discovery*, 33(6):1505–1547, 2019.

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

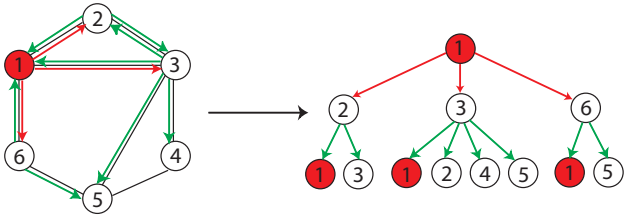
N. Shervashidze, P. Schweitzer, E. J. v. Leeuwen, K. Mehlhorn, and K. M. Borgwardt. Weisfeiler-Lehman graph kernels. *Journal of Machine Learning Research (JMLR)*, 12:2539–2561, 2011.

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

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

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

Weisfeiler-Lehman subtree kernel



- Enumerating **subtree patterns** can be exponentially costly. Is there a fast way ?
- WL algorithm: **iterative enumeration** for graphs with discrete node labels.
  - We define **a sequence of node labels** initialized with $a_0 = a$.
  - At iteration $i \geq 1$, $a_i(u) = \text{hash}([a_{i-1}(u), \text{sort}(\{a_{i-1}(v) \mid v \in \mathcal{N}(u)\})])$.
- WL subtree kernel at depth k is defined as

$$\kappa_{\text{subtree}}(u, u') = \delta(a_i(u), a'_i(u'))$$

[Shervashidze et al., 2011]

Motivation: link between walk and WL subtree kernels

Is there some relation between the base kernels κ_{walk} and κ_{subtree} ?

WL subtree kernel as a 2-layer walk kernel

Let $\mathcal{M}(u, u')$ be the set of exact matchings of subsets of the neighborhoods of two nodes u and u'. For any $u \in G$ and $u' \in G'$ such that $|\mathcal{M}(u, u')| = 1$,

$$\kappa_{\text{subtree}}(u, u') = \delta(\varphi_{\text{walk}}(u), \varphi'_{\text{walk}}(u')), \tag{1}$$

where φ_{walk} is the feature map of κ_{walk} satisfying $\varphi_{\text{walk}}(u) = \sum_{p \in \mathcal{W}_k(G,u)} \varphi_\delta(p)$.

- A sufficient condition for $|\mathcal{M}(u, u')| = 1$: u and u' have **same degrees** and both of them have **distinct neighbors**.
- If we replace φ_{path} instead of φ_{walk} we capture **subtrees** without repeated nodes !

Motivation: link between walk and WL subtree kernels

Is there some relation between the base kernels κ_{walk} and κ_{subtree} ?

WL subtree kernel as a 2-layer walk kernel

Let $\mathcal{M}(u, u')$ be the set of exact matchings of subsets of the neighborhoods of two nodes u and u'. For any $u \in G$ and $u' \in G'$ such that $|\mathcal{M}(u, u')| = 1$,

$$\kappa_{\text{subtree}}(u, u') = \delta(\varphi_{\text{walk}}(u), \varphi'_{\text{walk}}(u')), \tag{1}$$

where φ_{walk} is the feature map of κ_{walk} satisfying $\varphi_{\text{walk}}(u) = \sum_{p \in \mathcal{W}_k(G,u)} \varphi_\delta(p)$.

Can we go beyond subtrees to higher order patterns ?

Motivation: link between walk and WL subtree kernels

Is there some relation between the base kernels κ_{walk} and κ_{subtree} ?

WL subtree kernel as a 2-layer walk kernel

Let $\mathcal{M}(u, u')$ be the set of exact matchings of subsets of the neighborhoods of two nodes u and u'. For any $u \in G$ and $u' \in G'$ such that $|\mathcal{M}(u, u')| = 1$,

$$\kappa_{\text{subtree}}(u, u') = \delta(\varphi_{\text{walk}}(u), \varphi'_{\text{walk}}(u')), \tag{1}$$

where φ_{walk} is the feature map of κ_{walk} satisfying $\varphi_{\text{walk}}(u) = \sum_{p \in \mathcal{W}_k(G,u)} \varphi_\delta(p)$.

Can we go beyond subtrees to higher order patterns ?
Composing path kernels !