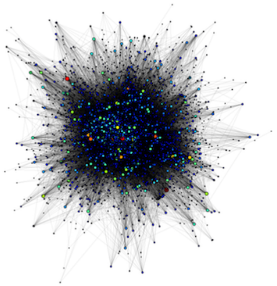
Using Graph Kernels to Address the Graph Similarity and Learning Problems

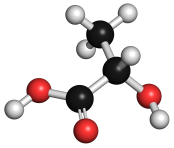
Giannis Nikolentzos

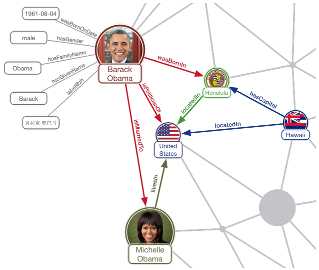
ÉCOLE POLYTECHNIQUE
UNIVERSITÉ PARIS-SACLAY

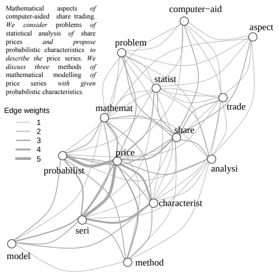
31 May 2018

Graphs Are Everywhere








Mathematical aspects of computer-aided share trading. We consider problems of statistical analysis of share prices and propose probabilistic characteristics to describe the price series. We discuss three methods of mathematical modelling of price series with given probabilistic characteristics.

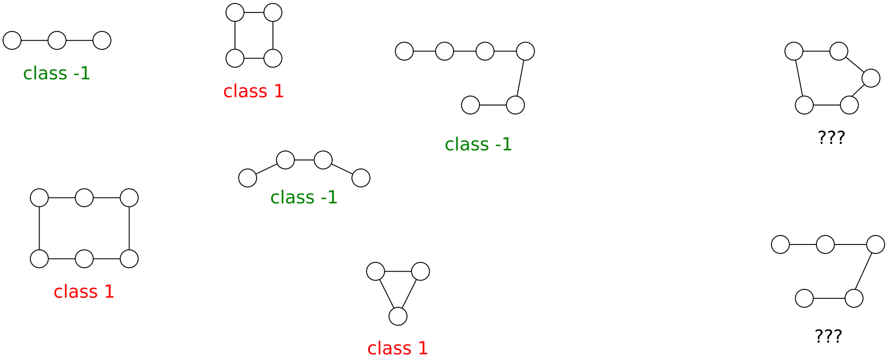
Why graphs?

Machine Learning on Graphs

Machine learning tasks on graphs:

- Node classification: given a graph with labels on some nodes, provide a high quality labeling for the rest of the nodes
- Graph clustering: given a graph, group its vertices into clusters taking into account its edge structure in such a way that there are many edges within each cluster and relatively few between the clusters
- Link Prediction: given a pair of vertices, predict if they should be linked with an edge
- **Graph classification**: given a set of graphs with known class labels for some of them, decide to which class the rest of the graphs belong

Graph Classification

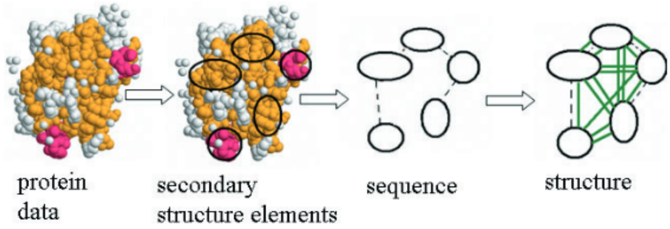


- Input data $x \in \mathcal{X}$
- Output $y \in \{-1, 1\}$
- Training set $\mathcal{S} = \{(x_1, y_1), \dots, (x_n, y_n)\}$
- Goal: estimate a function $f : \mathcal{X} \rightarrow \mathbb{R}$ to predict y from f(x)

Motivation - Protein Function Prediction

For each protein, create a graph that contains information about its

- structure
- sequence
- chemical properties

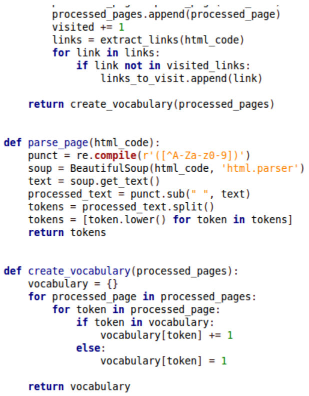


Perform **graph classification** to predict the function of proteins

Borgwardt et al. "Protein function prediction via graph kernels". Bioinformatics 21

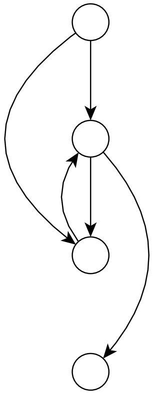
Motivation - Malware Detection

Given a computer program, create its control flow graph



```
        processed_pages.append(processed_page)
        visited += 1
        links = extract_links(html_code)
        for link in links:
            if link not in visited_links:
                links_to_visit.append(link)

    return create_vocabulary(processed_pages)


def parse_page(html_code):
    punct = re.compile(r'([^A-Za-z0-9])')
    soup = BeautifulSoup(html_code, 'html.parser')
    text = soup.get_text()
    processed_text = punct.sub(" ", text)
    tokens = processed_text.split()
    tokens = [token.lower() for token in tokens]
    return tokens


def create_vocabulary(processed_pages):
    vocabulary = {}
    for processed_page in processed_pages:
        for token in processed_page:
            if token in vocabulary:
                vocabulary[token] += 1
            else:
                vocabulary[token] = 1

    return vocabulary
```

$\rightarrow$



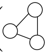
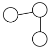
Perform **graph classification** to predict if there is malicious code inside the program or not

Gascon et al. "Structural detection of android malware using embedded call graphs". In AISec'13

Graph Comparison

Graph classification very related to graph comparison

Example

$f($  ,  $)$

$+$

$k{-}nn$

$=$ graph classification

Although graph comparison seems a tractable problem, it is very **complex**

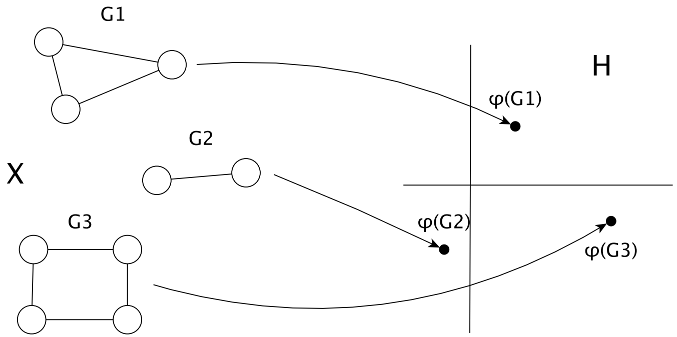
We are interested in algorithms capable of measuring the similarity between two graphs in **polynomial** time

Graph Kernels

Definition (Graph Kernel)

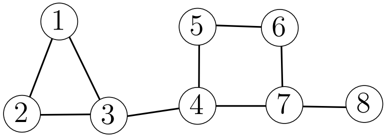
A graph kernel $k : \mathcal{G} \times \mathcal{G} \rightarrow \mathcal{R}$ is a kernel function over a set of graphs $\mathcal{G}$

- It is equivalent to an inner product of the embeddings $\phi : \mathcal{X} \rightarrow \mathbb{H}$ of a pair of graphs into a Hilbert space: $k(G_1, G_2) = \langle \phi(G_1), \phi(G_2) \rangle$
- Makes the whole family of kernel methods (e.g. SVMs) applicable to graphs



Substructure-based Kernels

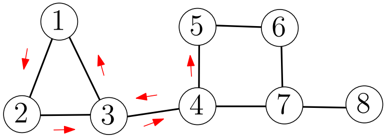
A large number of graph kernels compare substructures of graphs that are computable in polynomial time:



Substructure-based Kernels

A large number of graph kernels compare substructures of graphs that are computable in polynomial time:

- walks



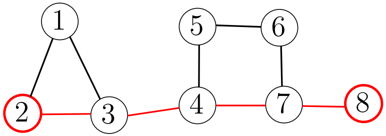
Walk: $4 \to 3 \to 1 \to 2 \to 3 \to 4 \to 5$

Vishwanathan et al. "Graph Kernels". JMLR 11, 2010

Substructure-based Kernels

A large number of graph kernels compare substructures of graphs that are computable in polynomial time:

- walks
- shortest path lengths



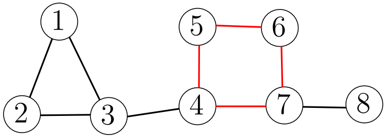
SP length between vertices 2 and 8 : 4

Borgwardt and Kriegel. "Shortest-path kernels on graphs". In ICDM'05

Substructure-based Kernels

A large number of graph kernels compare substructures of graphs that are computable in polynomial time:

- walks
- shortest path lengths
- cyclic patterns



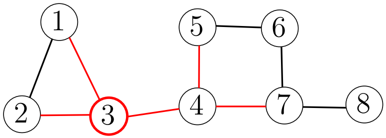
Cycle: $4 \rightarrow 7 \rightarrow 6 \rightarrow 5 \rightarrow 4$

Horváth et al. "Cyclic pattern kernels for predictive graph mining". In KDD'04

Substructure-based Kernels

A large number of graph kernels compare substructures of graphs that are computable in polynomial time:

- walks
- shortest path lengths
- cyclic patterns
- rooted subtrees



Subtree rooted at vertex 3

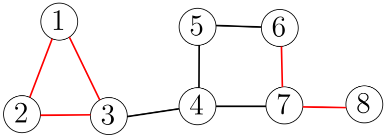
Shervashidze et al. "Weisfeiler-Lehman Graph Kernels". JMLR 12, 2011

Substructure-based Kernels

A large number of graph kernels compare substructures of graphs that are computable in polynomial time:

- walks
- shortest path lengths
- cyclic patterns
- rooted subtrees
- graphlets

⋮



Shervashidze et al. "Efficient graphlet kernels for large graph comparison.". In AISTATS'09

A Degeneracy Framework for Graph Comparison

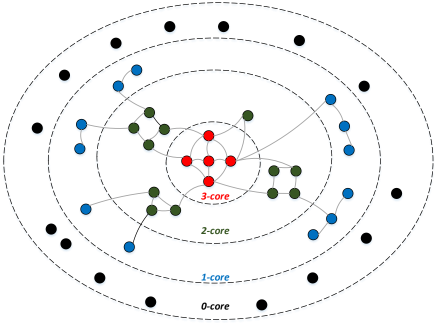
- a framework for increasing the expressive power of existing algorithms
- can be applied to any algorithm that compares graphs
- utilizes *k*-core decomposition to build a hierarchy of nested subgraphs

k-core

Definition (*k*-core)

The *k*-core of a graph is defined as a maximal subgraph in which every vertex is connected to at least *k* other vertices within that subgraph

A *k-core decomposition* of a graph consists of finding the set of all *k*-cores



The set of all *k*-cores forms a nested sequence of subgraphs

The degeneracy $\delta^*(G)$ is defined as the maximum k for which graph G contains a non-empty k-core subgraph

Degeneracy Framework for Graph Comparison

Idea: use the nested sequence of subgraphs generated by k-core decomposition to capture structure at multiple different scales

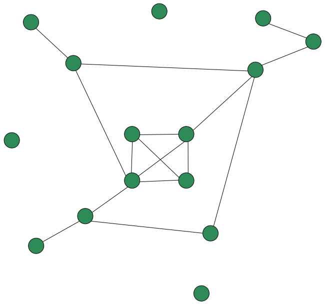
Definition (core kernel)

Let $G = (V, E)$ and $G' = (V', E')$ be two graphs. Let also k be any kernel for graphs. Then, the core variant of the base kernel k is defined as

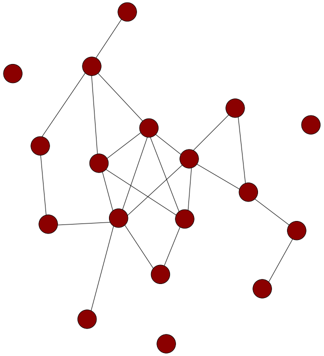
$$k_c(G, G') = k(C_0, C'_0) + k(C_1, C'_1) + \ldots + k(C_{\delta^*_{min}}, C'_{\delta^*_{min}})$$

where δ^*_{min} is the minimum of the degeneracies of the two graphs, and $C_0, C_1, \ldots, C_{\delta^*_{min}}$ and $C'_0, C'_1, \ldots, C'_{\delta^*_{min}}$ are the 0-core, 1-core,..., δ^*_{min}-core subgraphs of G and G' respectively

Example

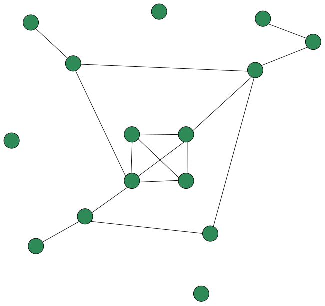


G

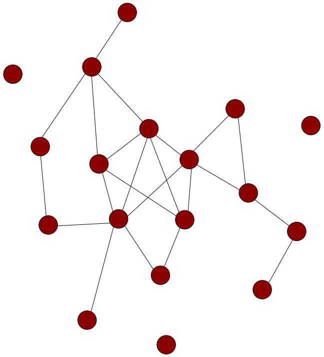


G'

Example



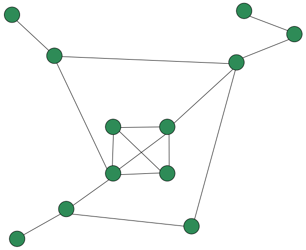
C_0



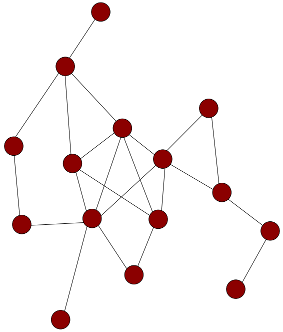
C_0'

$$k_c(G, G') = k(C_0, C_0')$$

Example



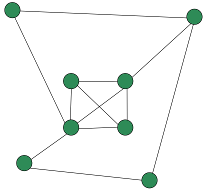
C_1

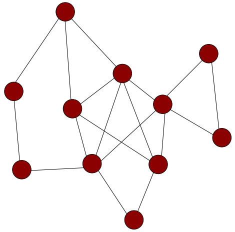


C_1'

$$k_c(G, G') = k(C_0, C_0') + k(C_1, C_1')$$

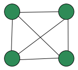
Example

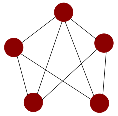

C_2


C'_2

$$k_c(G, G') = k(C_0, C'_0) + k(C_1, C'_1) + k(C_2, C'_2)$$

Example





C_3 C_3'

$$k_c(G, G') = k(C_0, C_0') + k(C_1, C_1') + k(C_2, C_2') + k(C_3, C_3')$$

Computational Complexity

Computational complexity depends on:

- the properties of the base kernel
- the degeneracy of the graphs under comparison

Given a pair of graphs and an algorithm A for comparing two graphs, computing the core variant requires $\delta^*_{min}\mathcal{O}_A$ time, where $\mathcal{O}_A$ be the time complexity of algorithm A

The degeneracy of a graph is upper bounded by the largest eigenvalue of its adjacency matrix λ_1

In most real-world graphs, $\lambda_1 \ll n$, then $\delta^*_{min} \ll n$, hence time complexity not prohibitive

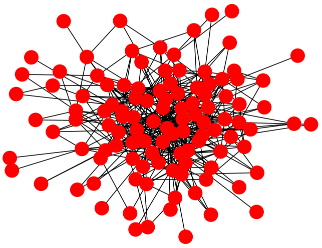
Dimensionality Reduction View

k-core decomposition can be seen as a method for performing dimensionality reduction on graphs

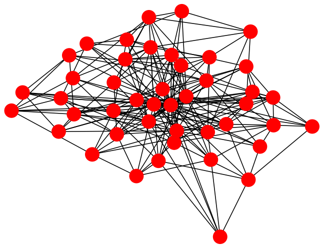
- each core can be considered as an approximation of the graph
- features of low importance are removed

Problem: For very large graphs, the running time of algorithms with high complexity (e.g. shortest path kernel) is prohibitive

Solution: Use high-order cores



$\rightarrow$



Datasets

Task: graph classification → standard datasets from chemoinformatics, bioinformatics and social networks

Dataset	MUTAG	ENZYMES	NCI1	PTC-MR	D&D	IMDB BINARY	IMDB MULTI	REDDIT BINARY	REDDIT MULTI-5K	REDDIT MULTI-12K
Max # vertices	28	126	111	109	5748	136	89	3782	3648	3782
Min # vertices	10	2	3	2	30	12	7	6	22	2
Average # vertices	17.93	32.63	29.87	25.56	284.32	19.77	13.00	429.61	508.50	391.40
Max # edges	33	149	119	108	14267	1249	1467	4071	4783	5171
Min # edges	10	1	2	1	63	26	12	4	21	1
Average # edges	19.79	62.14	32.30	25.96	715.66	96.53	65.93	497.75	594.87	456.89
# graphs	188	600	4110	344	1178	1000	1500	2000	4999	11929
# classes	2	6	2	2	2	2	3	2	5	11

Classification using:

- SVM → precompute kernel matrix
- Hyperparameters of SVM (i. e. C) and kernels optimized on training set using cross-validation

We compare an algorithm's output with the expected outcome:

- *Accuracy*: proportion of good predictions

Base Kernels

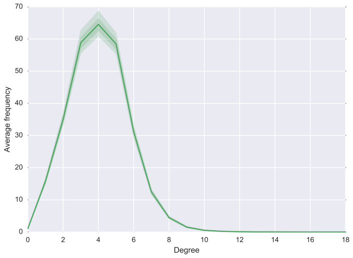
We employed the following kernels:

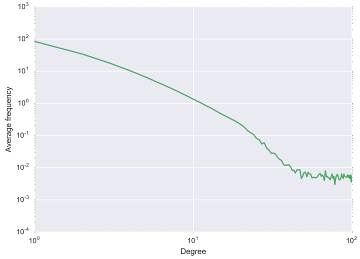
1. **Graphlet kernel (GR)** [Shervashidze et al., 2009]: The graphlet kernel counts identical pairs of graphlets (i.e. subgraphs with k nodes where $k \in 3, 4, 5$) in two graphs
2. **Shortest path kernel (SP)** [Borgwardt and Kriegel, 2005]: The shortest path kernel counts pairs of shortest paths in two graphs having the same source and sink labels and identical length
3. **Weisfeiler-Lehman subtree kernel (WL)** [Shervashidze et al., 2011]: The Weisfeiler-Lehman subtree kernel for a number of iterations counts pairs of matching subtree patterns in two graphs, while at each iteration updates the labels of the vertices of the two graphs
4. **Pyramid match graph kernel (PM)** [Nikolentzos et al., 2017]: The pyramid match graph kernel first embedds the vertices of the graphs in a vector space. It then partitions the feature space into regions of increasingly larger size and takes a weighted sum of the matches that occur at each level

Graph Classification

Method \ Dataset	MUTAG	ENZYMES	NCI1	PTC-MR	D&D
GR	69.97 (± 2.22)	33.08 (± 0.93)	65.47 (± 0.14)	56.63 (± 1.61)	77.77 (± 0.47)
Core GR	**82.34** (± 1.29)	33.66 (± 0.65)	**66.85** (± 0.20)	57.68 (± 1.26)	78.05 (± 0.56)
SP	84.03 (± 1.49)	40.75 (± 0.81)	72.85 (± 0.24)	60.14 (± 1.80)	77.14 (± 0.77)
Core SP	**88.29** (± 1.55)	41.20 (± 1.21)	**73.46** (± 0.32)	59.06 (± 0.93)	77.30 (± 0.80)
WL	83.63 (± 1.57)	51.56 (± 2.75)	84.42 (± 0.25)	61.93 (± 2.35)	79.19 (± 0.39)
Core WL	**87.47** (± 1.08)	47.82 (± 4.62)	**85.01** (± 0.19)	59.43 (± 1.20)	79.24 (± 0.34)
PM	80.66 (± 0.90)	42.17 (± 2.02)	72.27 (± 0.59)	56.41 (± 1.45)	77.34 (± 0.97)
Core PM	**87.19** (± 1.47)	42.42 (± 1.06)	**74.90** (± 0.45)	**61.13** (± 1.44)	77.72 (± 0.71)

Method \ Dataset	IMDB BINARY	IMDB MULTI	REDDIT BINARY	REDDIT MULTI-5K	REDDIT MULTI-12K
GR	59.85 (± 0.41)	35.28 (± 0.14)	76.82 (± 0.15)	35.32 (± 0.09)	22.68 (± 0.18)
Core GR	**69.91** (± 0.19)	**47.34** (± 0.84)	**80.67** (± 0.16)	**46.77** (± 0.09)	**32.41** (± 0.08)
SP	60.65 (± 0.34)	40.10 (± 0.71)	83.10 (± 0.22)	49.48 (± 0.14)	35.79 (± 0.09)
Core SP	**72.62** (± 0.59)	**49.43** (± 0.42)	**90.84** (± 0.14)	**54.35** (± 0.11)	**43.30** (± 0.04)
WL	72.44 (± 0.77)	51.19 (± 0.43)	74.99 (± 0.57)	49.69 (± 0.27)	33.44 (± 0.08)
Core WL	**74.02** (± 0.42)	51.35 (± 0.48)	**78.02** (± 0.23)	50.14 (± 0.21)	**35.23** (± 0.17)
PM	68.53 (± 0.61)	45.75 (± 0.66)	82.70 (± 0.68)	42.91 (± 0.42)	38.16 (± 0.19)
Core PM	**71.04** (± 0.64)	**48.30** (± 1.01)	**87.39** (± 0.55)	**50.63** (± 0.50)	**42.89** (± 0.14)





Degree distribution of D&D (left) and REDDIT-BINARY (right) datasets. Both axis of the right figure are logarithmic.

Runtime Performance

Comparison of running times of base kernels vs their core variants (relative increase in running time)

	MUTAG	ENZYMES	NCI1	PTC-MR	D&D	IMDB BINARY	IMDB MULTI	REDDIT BINARY	REDDIT MULTI-5K	REDDIT MULTI-12K
SP	1.69x	2.52x	1.62x	1.65x	3.00x	12.42x	17.34x	1.04x	1.05x	1.18x
GR	1.85x	2.94x	1.75x	1.50x	3.44x	7.95x	8.20x	2.24x	2.37x	2.80x
WL	1.76x	2.77x	1.68x	1.62x	3.34x	7.13x	6.84x	1.52x	1.58x	1.54x
PM	1.87x	2.79x	1.68x	1.50x	3.67x	6.92x	6.33x	1.90x	1.98x	1.96x
δ^*	2	4	3	2	7	29	37	6	8	8

- In most cases, extra computational cost is negligible
- Extra computational cost is very related to the maximum of the degeneracies of the graphs of the dataset δ^*

Conclusion

- Graph kernels have shown good performance on several tasks
- We defined a general framework for improving the performance of graph comparison algorithms
- The proposed framework allows existing algorithms to compare structure in graphs at multiple different scales
- The conducted experiments highlight the superiority in terms of accuracy of the core variants over their base kernels at the expense of only a slight increase in computational time

Thank you!

Selected Publications I

Borgwardt, K. M. and Kriegel, H. (2005).

Shortest-path kernels on graphs.

In *Proceedings of the 5th International Conference on Data Mining*, pages 74–81.

Borgwardt, K. M., Ong, C. S., Schönauer, S., Vishwanathan, S., Smola, A. J., and Kriegel, H.-P. (2005).

Protein function prediction via graph kernels.

Bioinformatics, 21(suppl 1):i47–i56.

Gascon, H., Yamaguchi, F., Arp, D., and Rieck, K. (2013).

Structural Detection of Android Malware using Embedded Call Graphs.

In *Proceedings of the 2013 ACM Workshop on Artificial Intelligence and Security*, pages 45–54.

Horváth, T., Gärtner, T., and Wrobel, S. (2004).

Cyclic Pattern Kernels for Predictive Graph Mining.

In *Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 158–167.

Mahé, P., Ueda, N., Akutsu, T., Perret, J.-L., and Vert, J.-P. (2004).

Extensions of Marginalized Graph Kernels.

In *Proceedings of the 21st International Conference on Machine Learning*, pages 552–559.

Selected Publications II

Nikolentzos, G., Meladianos, P., and Vazirgiannis, M. (2017).

Matching Node Embeddings for Graph Similarity.

In *Proceedings of the 31st AAAI Conference on Artificial Intelligence*.

Shervashidze, N., Petri, T., Mehlhorn, K., Borgwardt, K. M., and Vishwanathan, S. (2009).

Efficient Graphlet Kernels for Large Graph Comparison.

In *Proceedings of the International Conference on Artificial Intelligence and Statistics*, pages 488–495.

Shervashidze, N., Schweitzer, P., Van Leeuwen, E. J., Mehlhorn, K., and Borgwardt, K. M. (2011).

Weisfeiler-Lehman Graph Kernels.

The Journal of Machine Learning Research, 12:2539–2561.

Vishwanathan, S. V. N., Schraudolph, N. N., Kondor, R., and Borgwardt, K. M. (2010).

Graph Kernels.

The Journal of Machine Learning Research, 11:1201–1242.