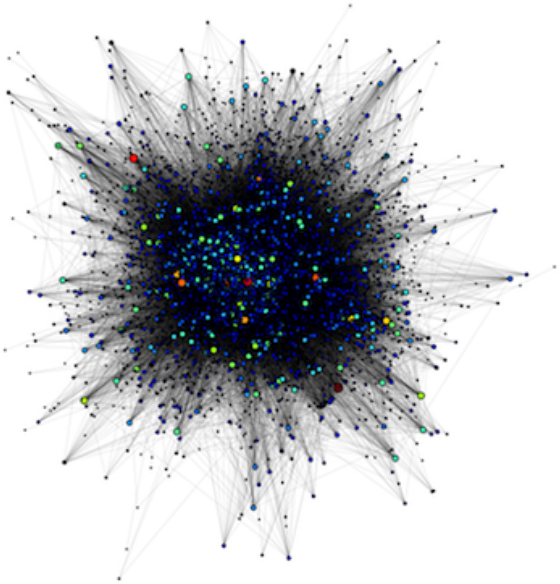
Graph Similarity and Classification @ DaSciM

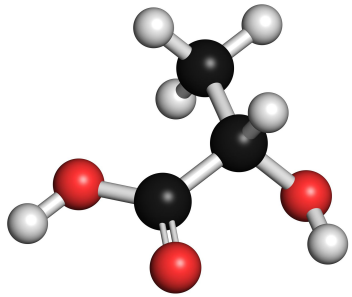
M. Vazirgiannis

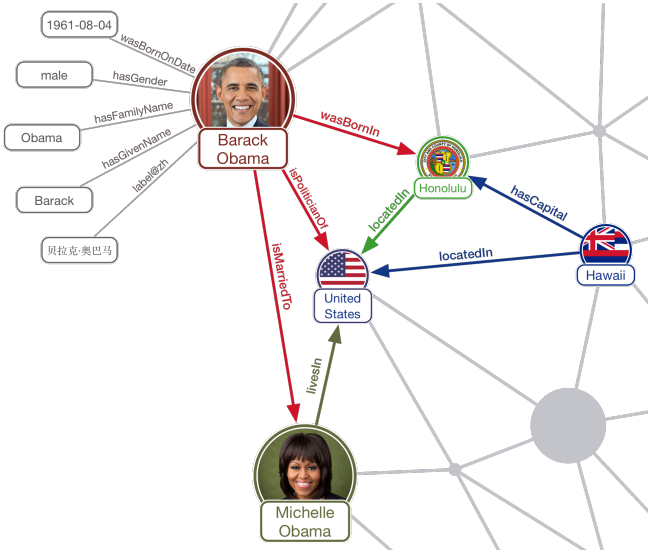
DaSciM, LIX, École Polytechnique

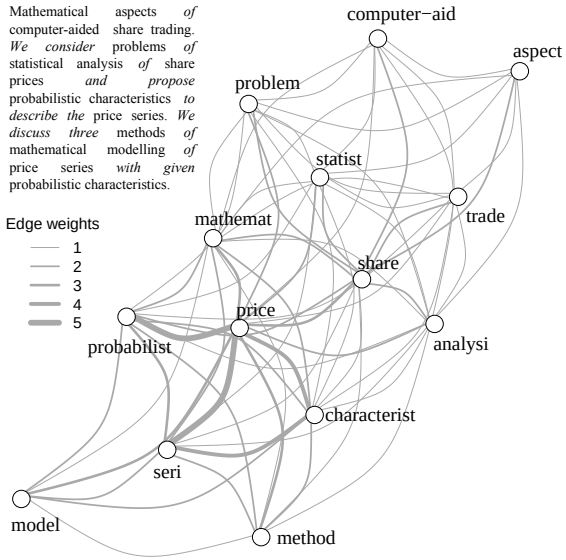
February 11, 2018

Graphs Are Everywhere



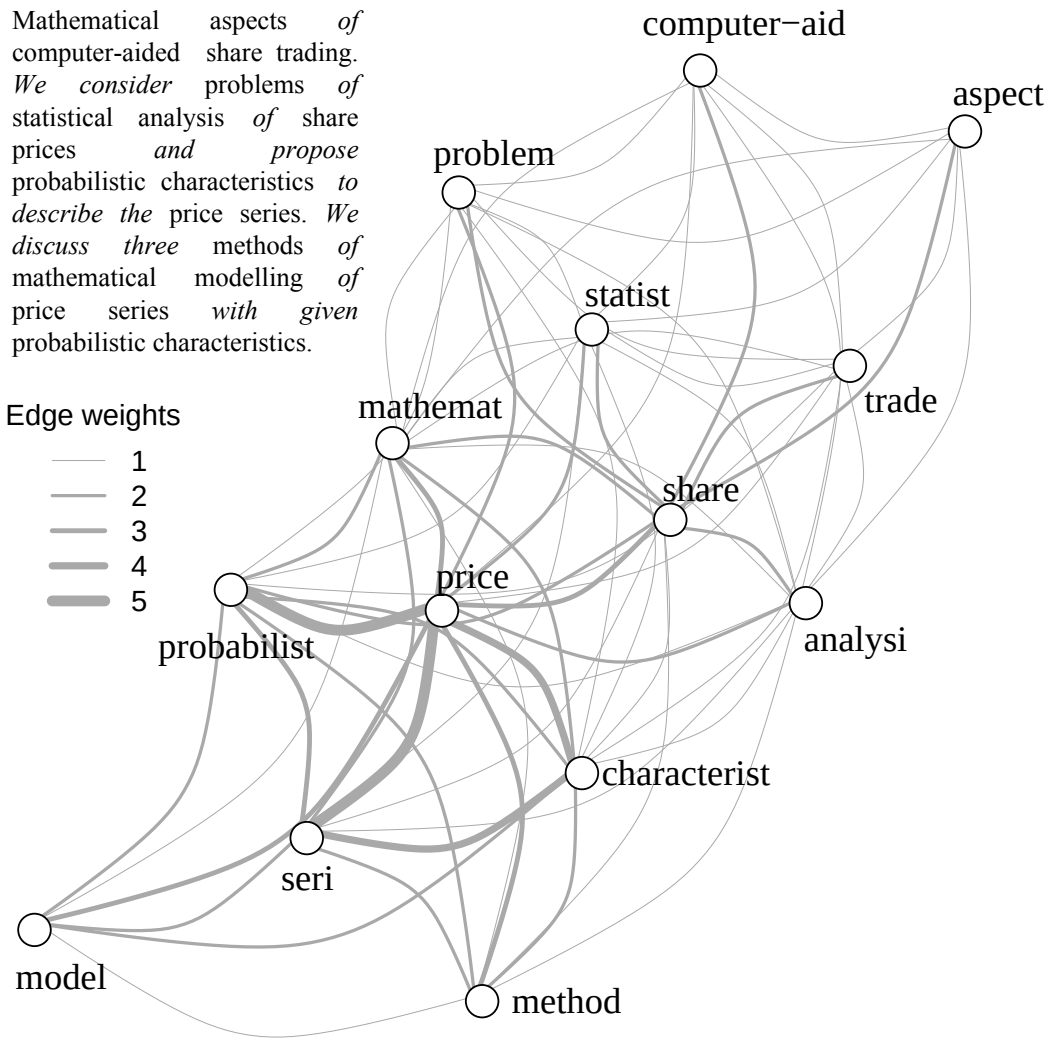






Why graphs?

Motivation - Text Categorization



Given a text, create a graph where

- vertices correpond to terms
- two terms are linked to each other if they co-occur within a fixed-size sliding window

Rousseau et al. "Text categorization as a graph classification problem.". ACL'15

Motivation - Text Categorization

Intuition: documents sharing same subgraphs belong to the same class

Given a set of documents and their graph representations:

Extract frequent subgraphs

- from the set of graphs

  or

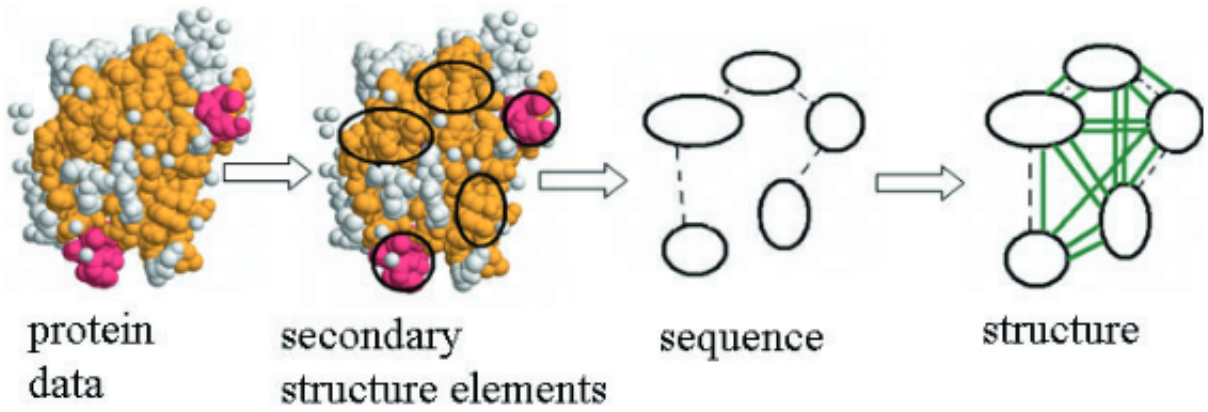
- from the set of the main cores of the graphs

Then, use frequent subgraphs as features for classification

Motivation - Protein Function Prediction

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

- structure
- sequence
- chemical properties



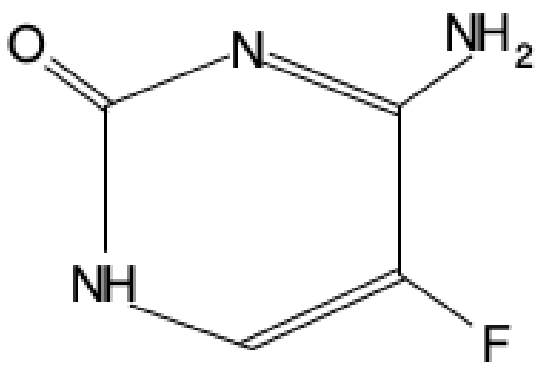
Use graph kernels to

- measure structural similarity between proteins
- predict the function of proteins

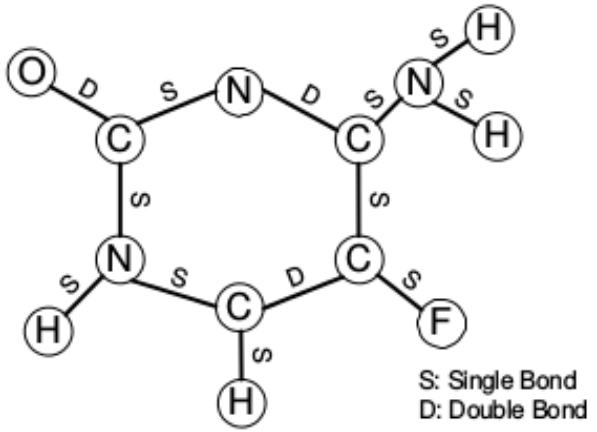
Borgwardt et al. "Protein function prediction via graph kernels". Bioinformatics 21

Motivation - Chemical Compound Classification

Represent each chemical compound as a graph



→



Use a frequent subgraph discovery algorithm to discover the substructures that occur above a certain support constraint

Perform feature selection

Use the remaining substructures as features for classification

Deshpande et al. "Frequent substructure-based approaches for classifying chemical compounds". TKDE 17(8)

Motivation - Anomaly Detection for the Web Graph

Search engines create snapshots of the web $\rightarrow$ web graphs

These are necessary for

- monitoring the evolution of the web
- computing global properties such as PageRank

Identify anomalies in a single snapshot by comparing it with previous snapshots

Employed similarity mesaures:

- vertex/edge overlap
- vertex ranking
- vertex/edge vector similarity
- etc

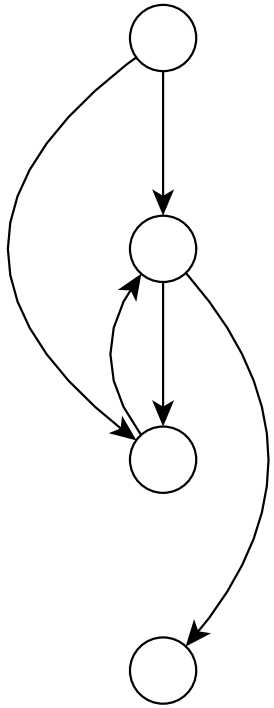
Papadimitriou et al. "Web graph similarity for anomaly detection". JISA 1(1)

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
```

→



Compare the control flow graph of the problem against the set of control flow graphs of known malware

If it contains a subgraph isomporphic to these graphs → malicious code inside the program

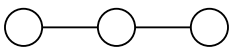
Gascon et al. "Structural detection of android malware using embedded call graphs". In AISec'13

Machine Learning on Graphs

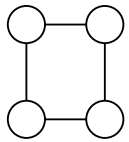
Machine learning tasks on graphs:

- 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

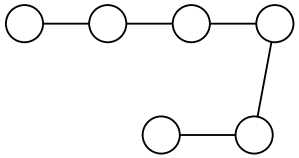
Graph Classification



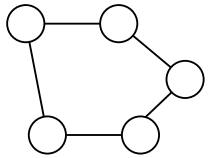
class -1



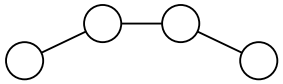
class 1



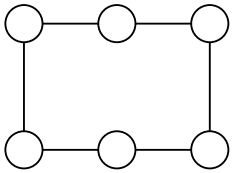
class -1



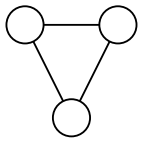
???



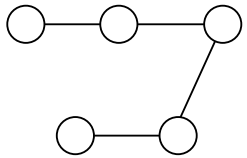
class -1



class 1



class 1



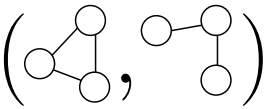
???

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

Graph Comparison

Graph classification very related to graph comparison

Example

$f($  $)$

$+$

$k\text{-}nn$

$=$ graph classification

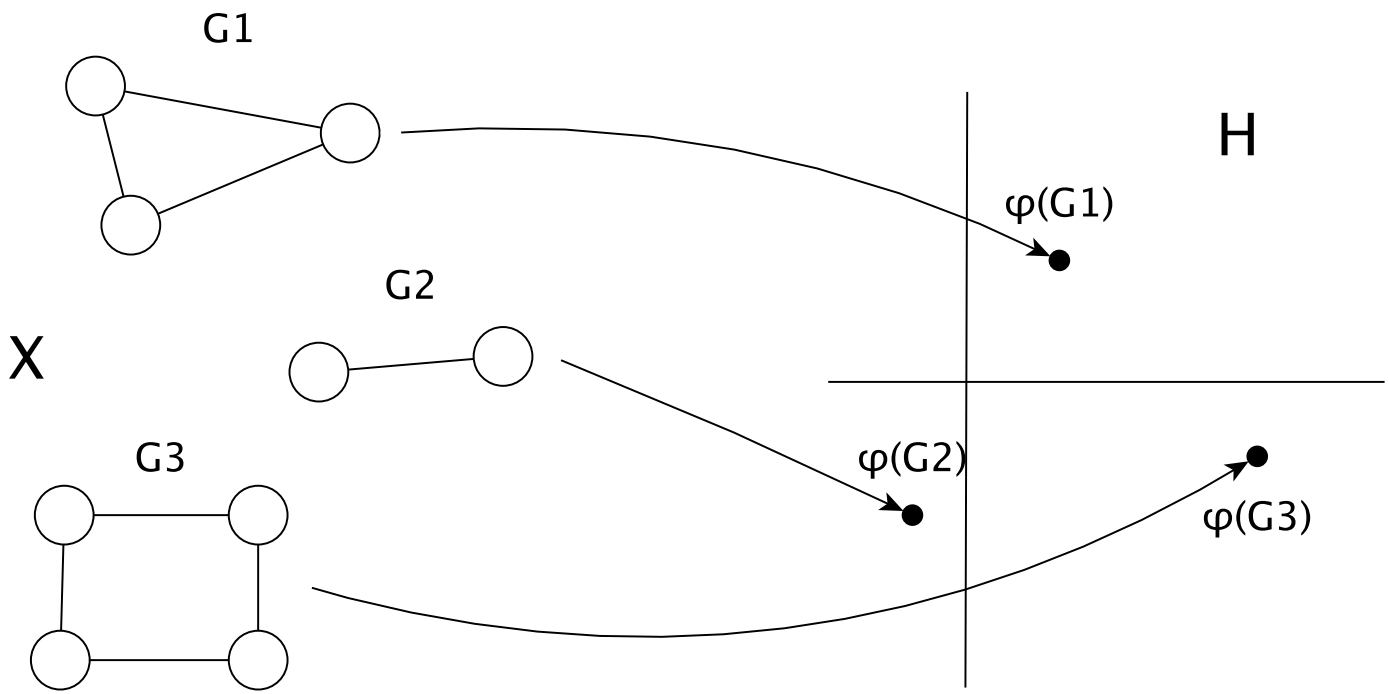
Although graph comparison seems a tractable problem, it is very **complex**

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

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



Graph Invariants

We saw that proving that two graphs are isomorphic is not a simple task

It is much simpler to show that two graphs are not isomorphic by finding a property that only one of the two graphs has. Such a property is called a *graph invariant*

Definition (Graph Invariant)

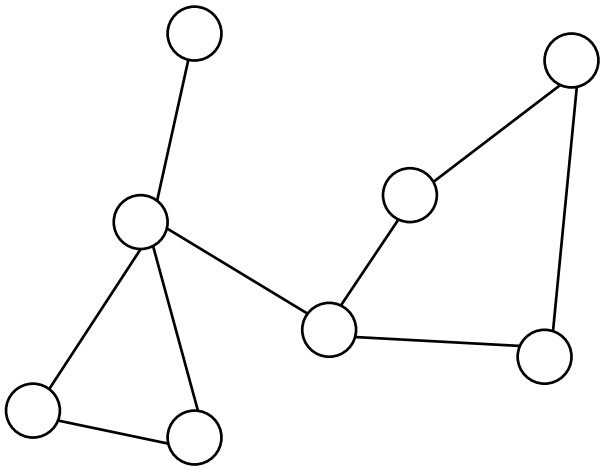
A graph invariant is a numerical property of graphs for which any two isomorphic graphs must have the same value

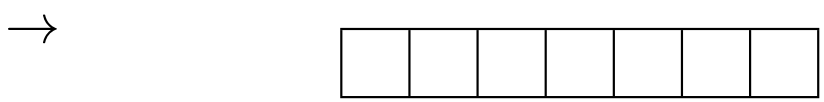
Some examples of graph invariants include:

1. number of vertices
2. number of edges
3. number of spanning trees
4. degree sequence
5. spectrum

Learning on Graphs

- Most machine learning algorithms require the input to be represented as a fixed-length feature vector
- Graphs cannot be naturally represented as vectors
- Typical vector-based classifiers (e. g., logistic regression) are not applicable



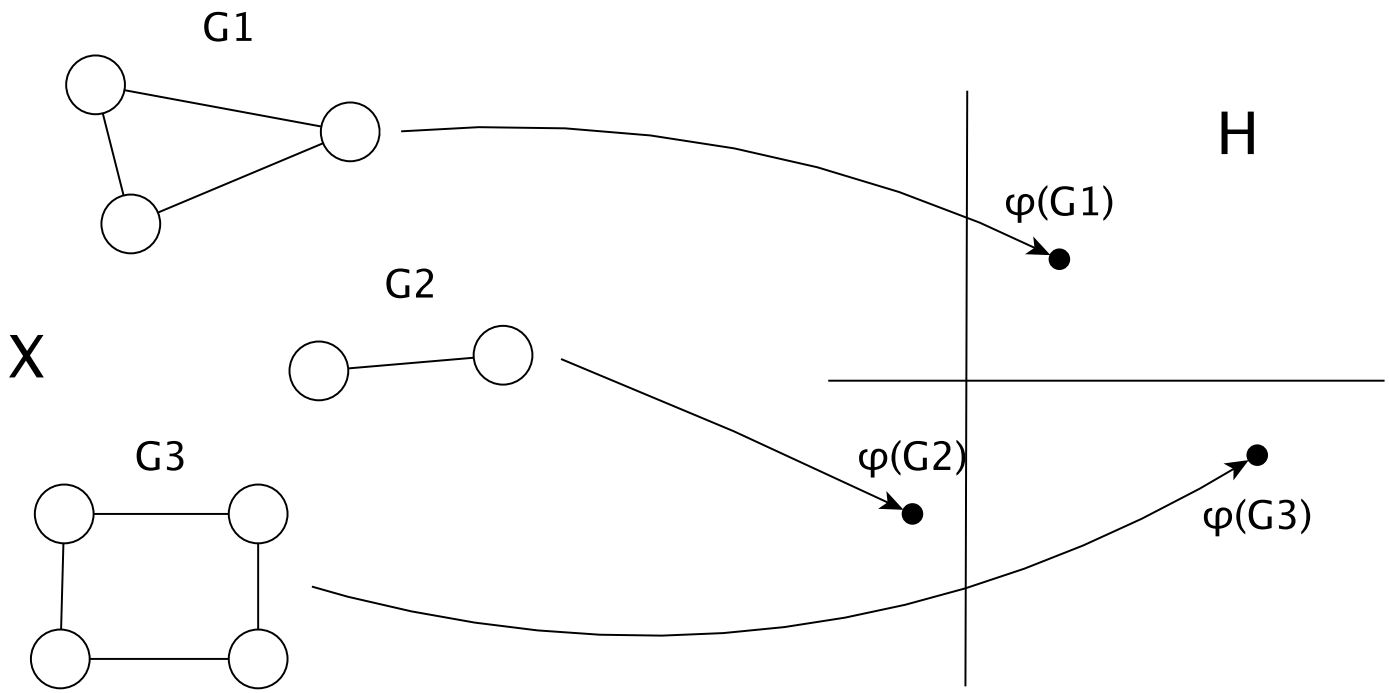


?

Definition (Graph Kernel)

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

- It is equivalent to an inner product of the embeddings $\phi : \mathcal{X} \rightarrow \mathcal{H}$ of a pair of graphs into a Hilbert space
- Makes the whole family of kernel methods applicable to graphs.



Graph Kernels - Expressiveness vs. Efficiency

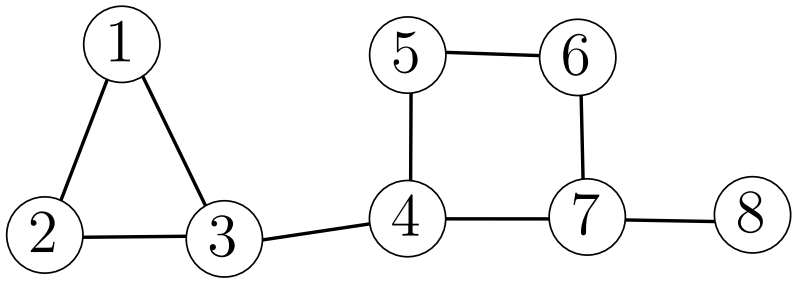
Definition (Complete Graph Kernel)

A graph kernel $k(G_i, G_j) = \phi(Gi), \phi(Gj) >$ is complete iff the transformation ϕ is injective.

- Computing any complete graph kernel is of same complexity as graph isomorphism [Gartner et.al., 2003]
- Complete graph kernels prohibitive in practical applications.
- More efficient kernels do not guarrantee that non-isomorphic graphs will not be mapped into the same point in the feature space.
- Trade-off between efficiency and effectiveness is a vital issue designing a graph kernel

Substructure-based Kernels

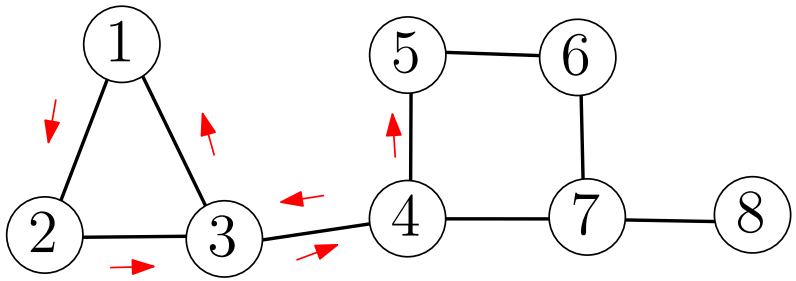
A large number of graph kernels compare substructures of graphs that are computable in polynomial time:



Substructure-based Kernels

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

- walks



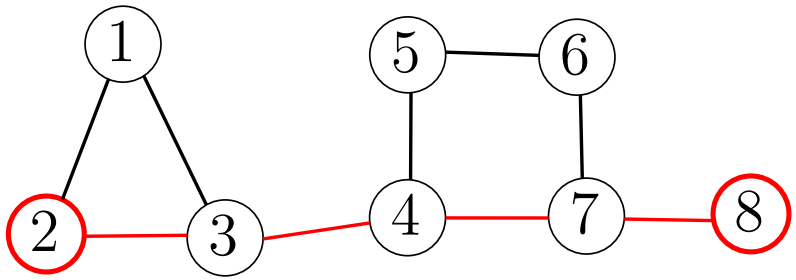
Walk: $4 \to 3 \to 1 \to 2 \to 3 \to 4 \to 5$

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



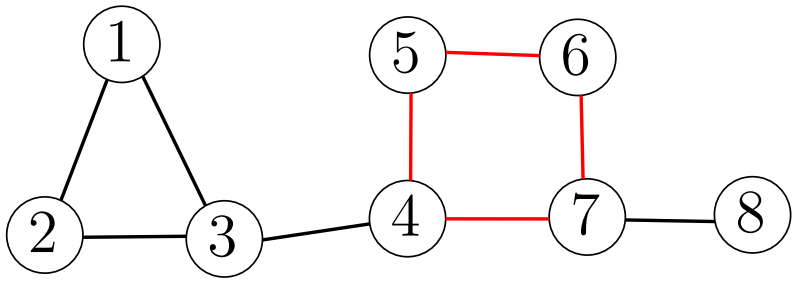
SP length between vertices 2 and 8 : 4

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



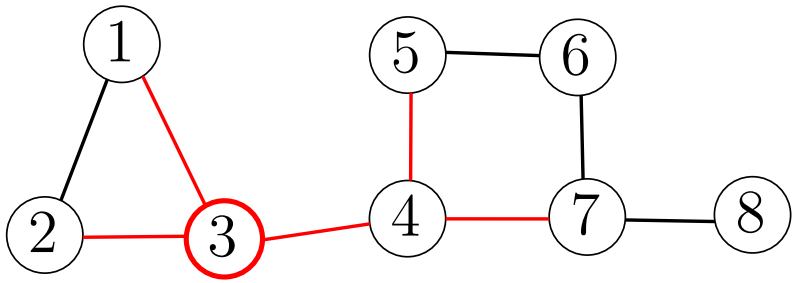
Cycle: $4 \rightarrow 7 \rightarrow 6 \rightarrow 5 \rightarrow 4$

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



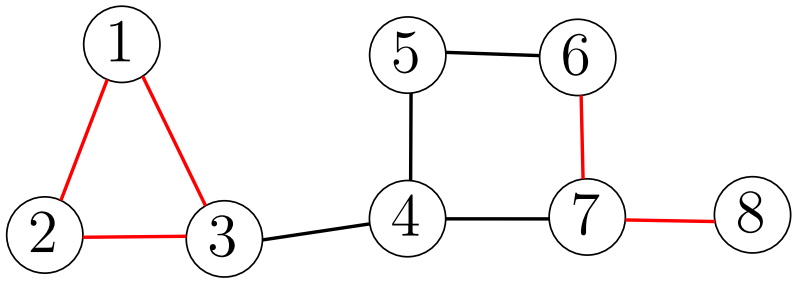
Subtree rooted at vertex 3

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

Weisfeiler-Lehman Framework

Uses the Weisfeiler-Lehman isomorphism test to improve the performance of existing kernels

- subtree kernel
- shortest path kernel

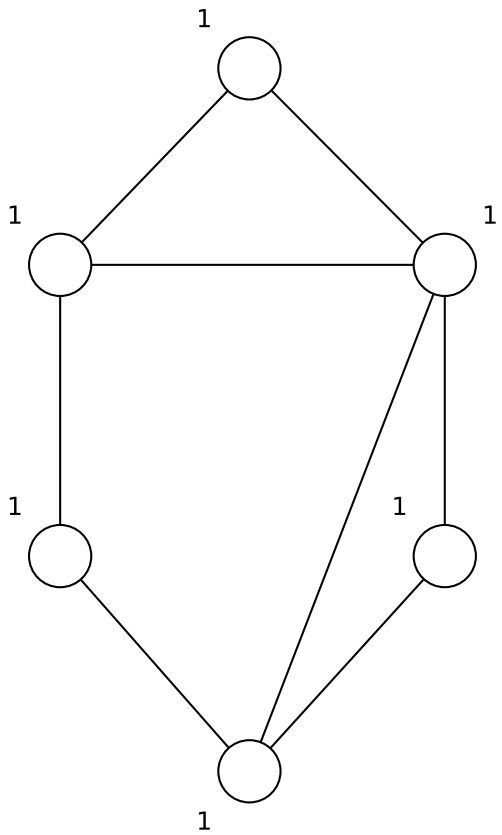
Weisfeiler-Lehman kernels achieve state-of-the-art results

Based on the Weisfeiler-Lehman algorithm: may answer if two graphs are not isomorphic

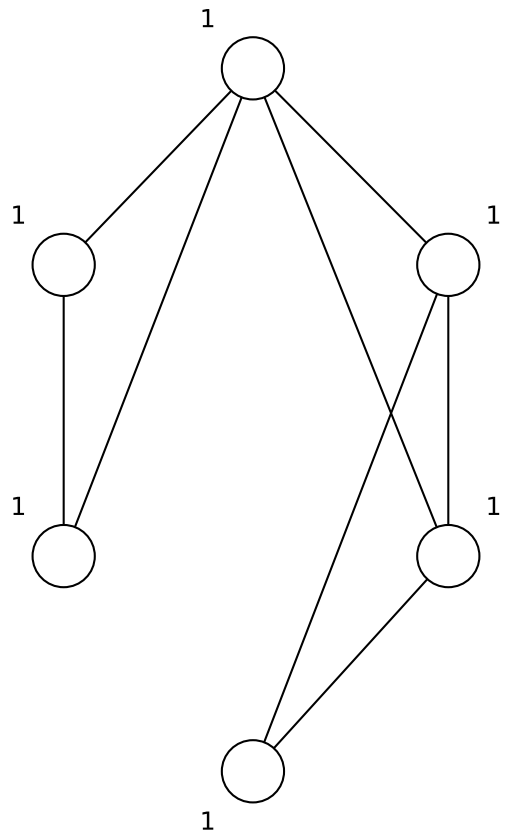
Shervashidze et al. "Weisfeiler-Lehman Graph Kernels". JMLR 12(Sep)

Example

Run the Weisfeiler-Lehman algorithm for the following pair of graphs



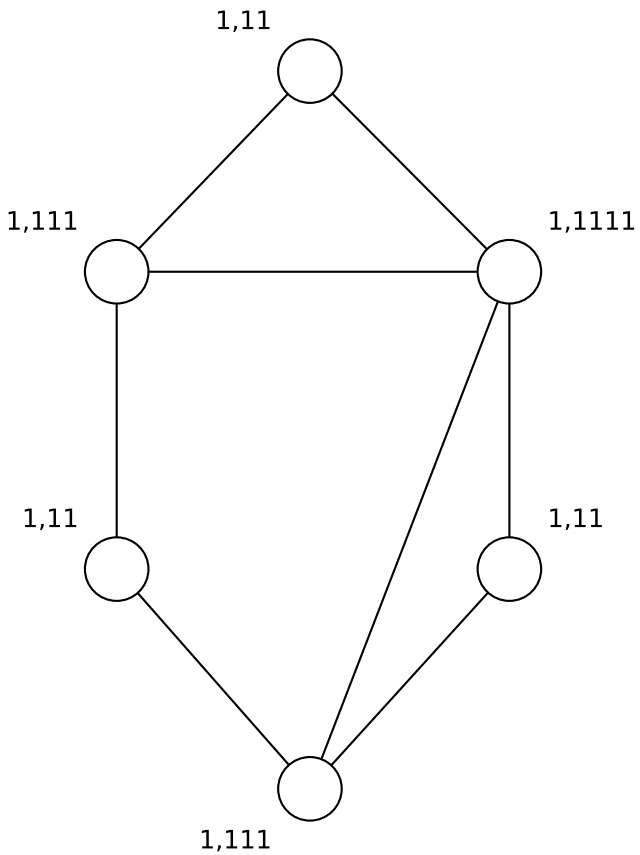
G_1



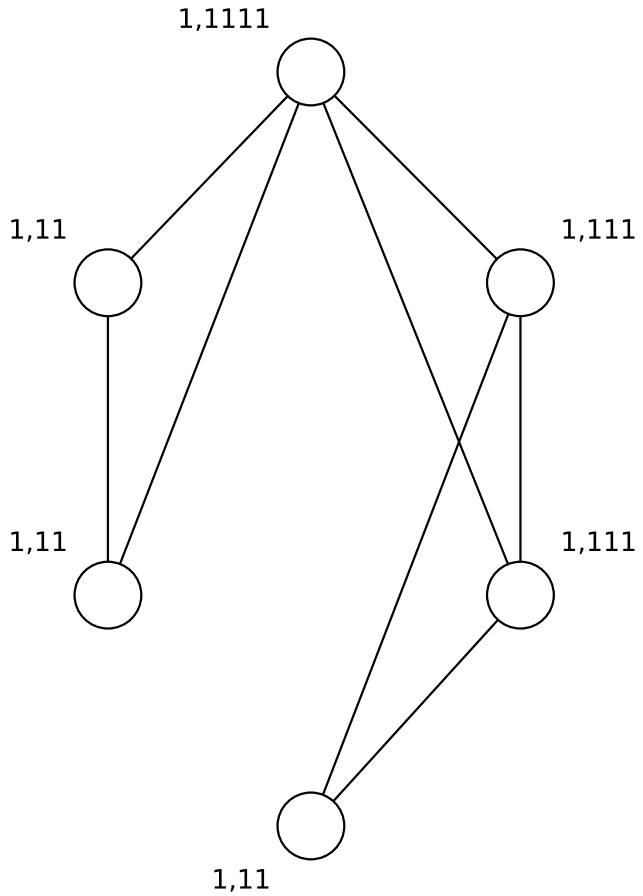
G_2

Iteration 1

First step: Augment the labels of the vertices by the sorted set of labels of neighbouring vertices



G_1

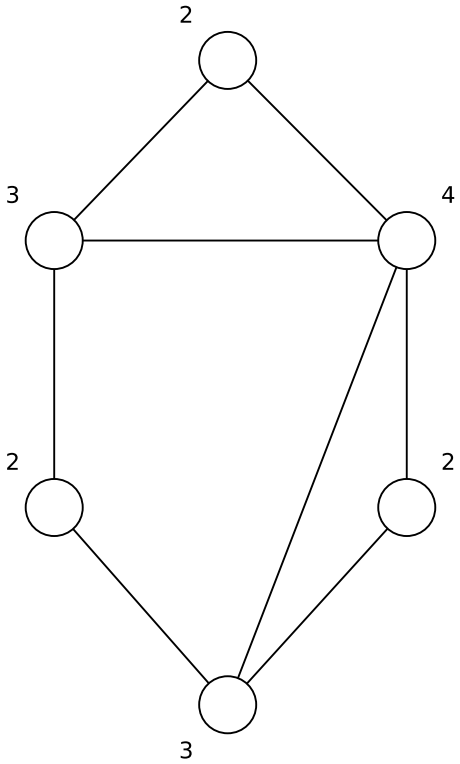


G_2

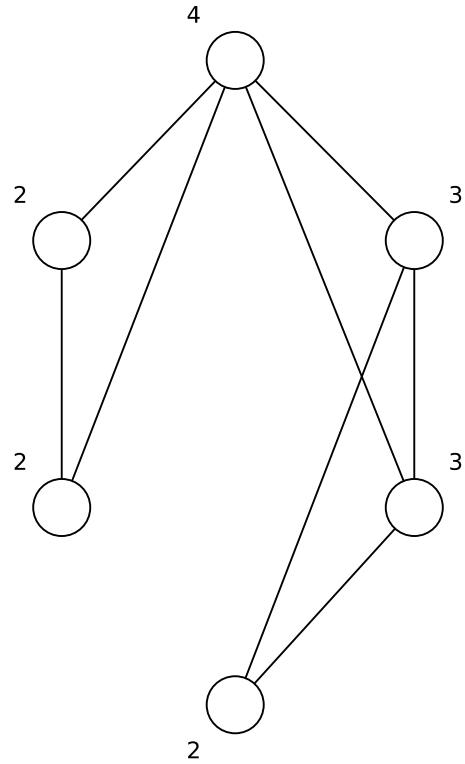
Iteration 1

Second step: Compress the augmented labels into new, short labels:

- $1, 11 \rightarrow 2$
- $1, 111 \rightarrow 3$
- $1, 1111 \rightarrow 4$



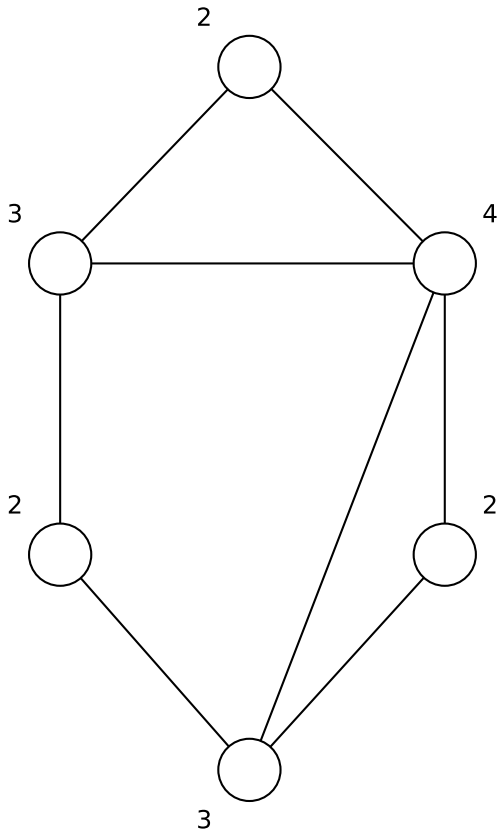
G_1



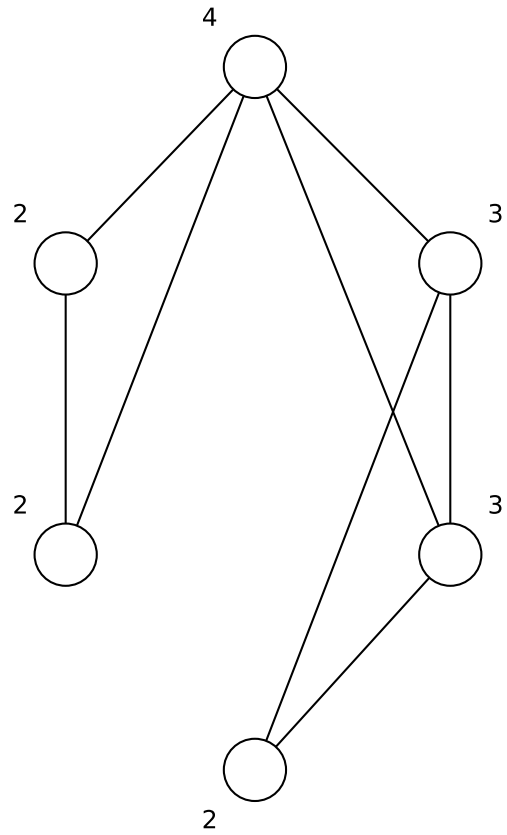
G_2

Iteration 1

Are the label sets of G_1 and G_2 identical?



G_1



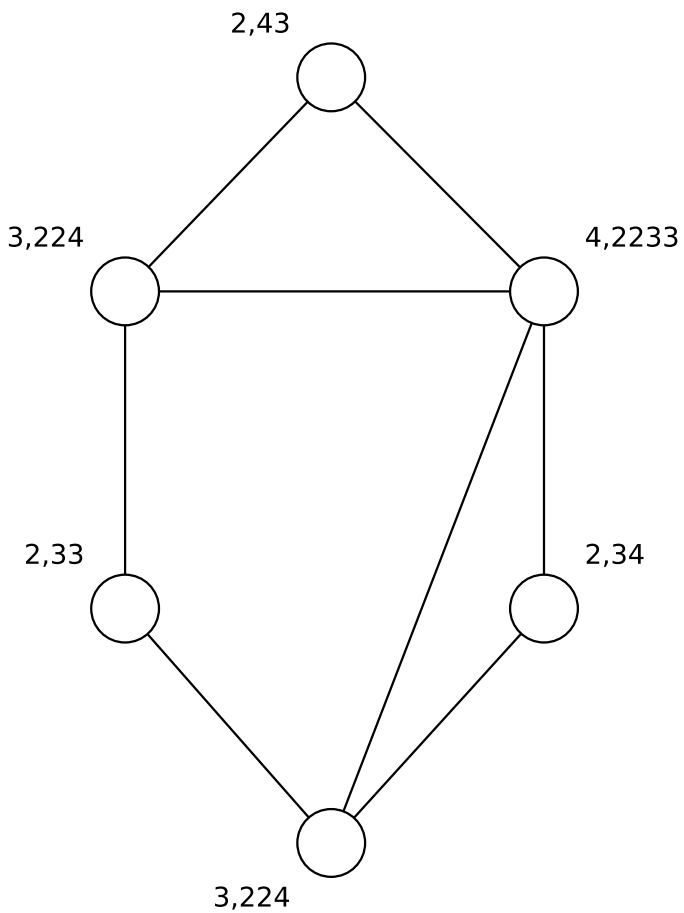
G_2

Yes!!!

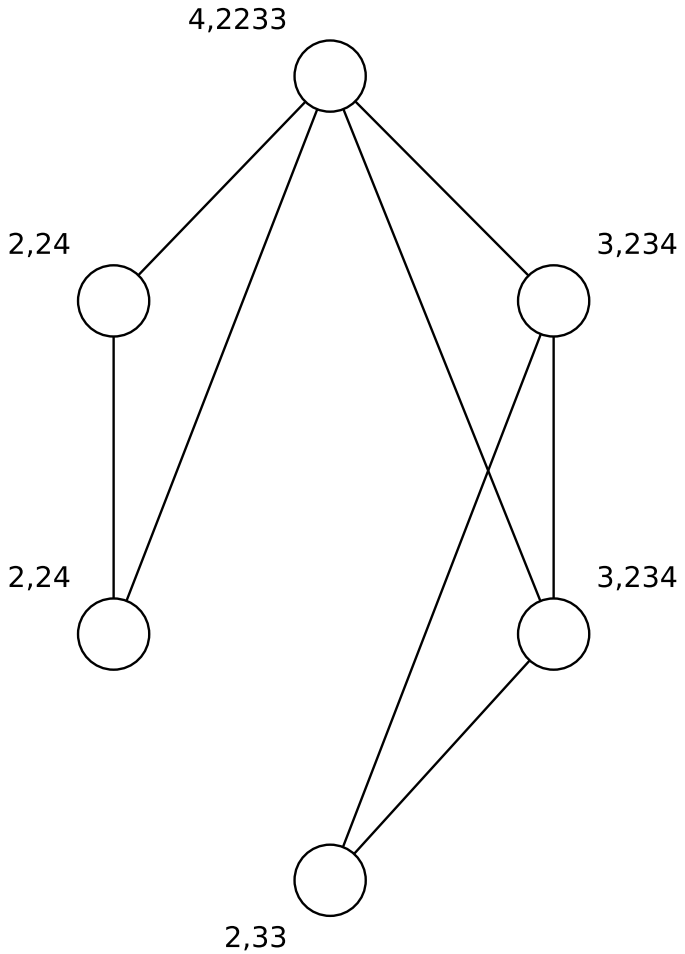
Continue to the next iteration

Iteration 2

First step: Augment the labels of the vertices by the sorted set of labels of neighbouring vertices



G_1

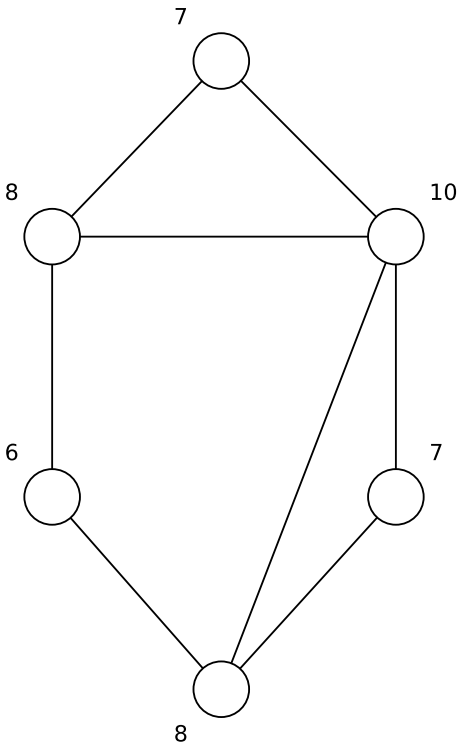


G_2

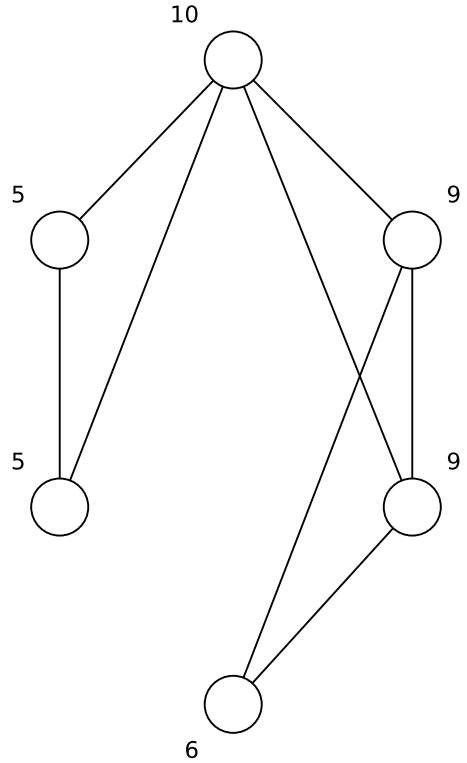
Iteration 2

Second step: Compress the augmented labels into new, short labels:

- $2, 24 \rightarrow 5$
- $2, 33 \rightarrow 6$
- $2, 34 \rightarrow 7$
- $3, 234 \rightarrow 9$
- $4, 2233 \rightarrow 10$



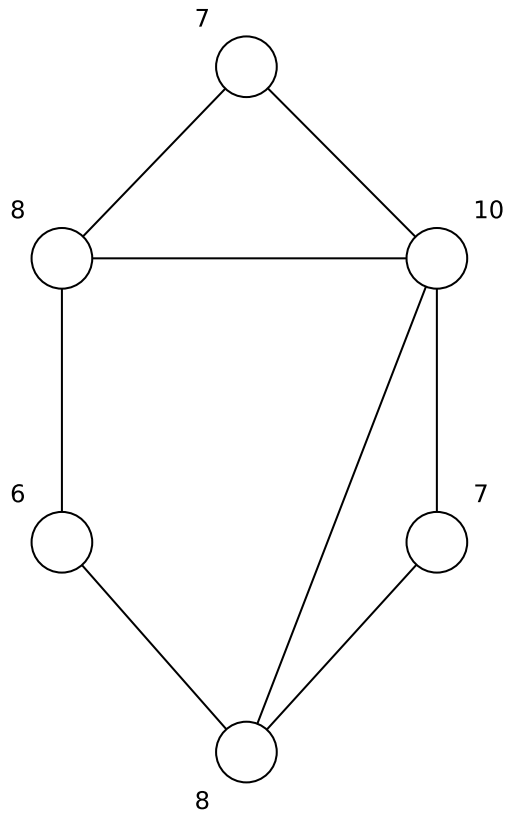
G_1



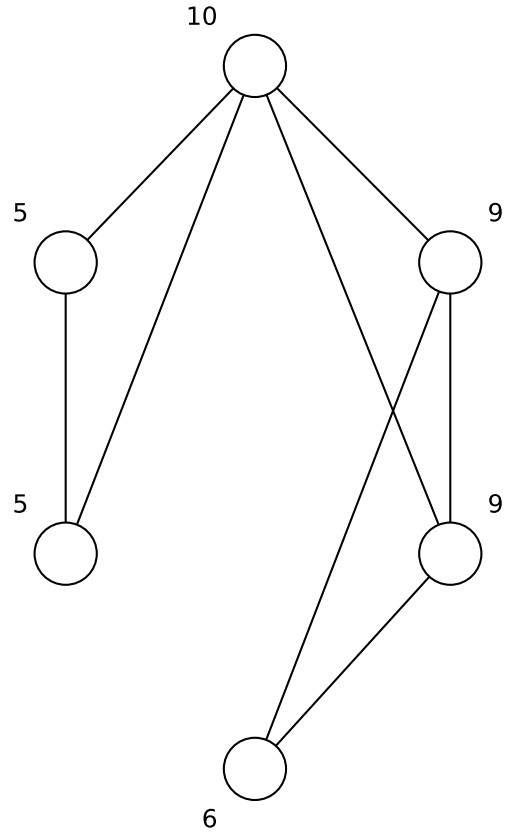
G_2

Iteration 2

Are the label sets of G_1 and G_2 identical?



G_1



G_2

No!!!

Graphs are not isomorphic

Weisfeiler-Lehman Framework

Let $G^1, G^2, \ldots, G^h$ be the graphs emerging from graph G at the iteration $1, 2, \ldots, h$ of the Weisfeiler-Lehman algorithm

Then, the Weisfeiler-Lehman kernel is defined as:

$$k_{WL}^h(G_1, G_2) = k(G_1, G_2) + k(G_1^1, G_2^1) + k(G_1^2, G_2^2) + \ldots + k(G_1^h, G_2^h)$$

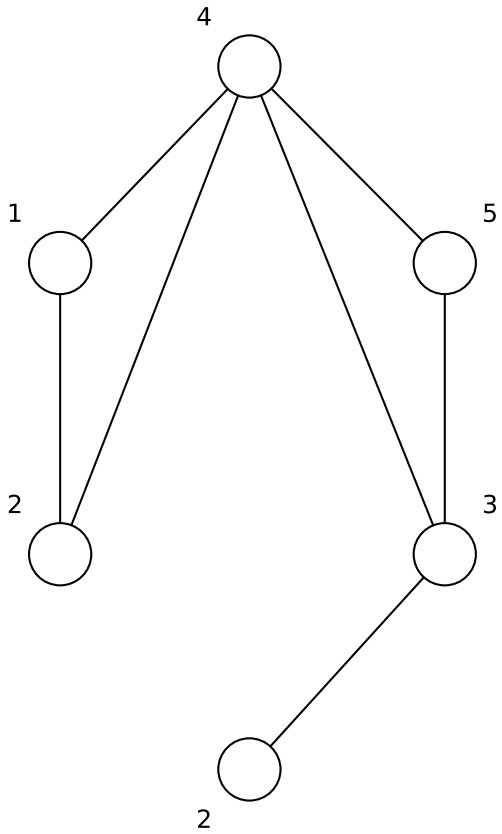
where $k(\cdot, \cdot)$ is a base kernel (e.g. subtree kernel, shortest path kernel, ...)

At each iteration of the Weisfeiler-Lehman algorithm:

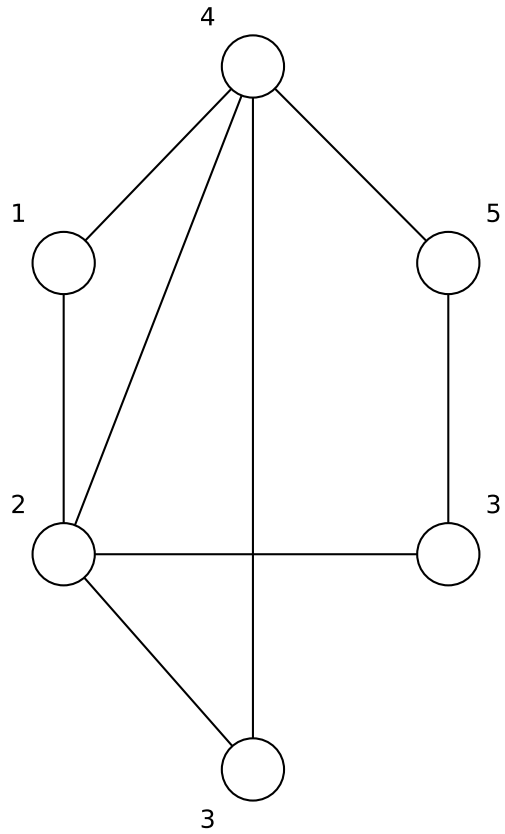
- runs a graph kernel for labeled graphs
- the new kernel values are added to the ones of the previous iteration

Weisfeiler-Lehman Subtree Kernel

Counts matching pairs of labels in two graphs after each iteration



G_1

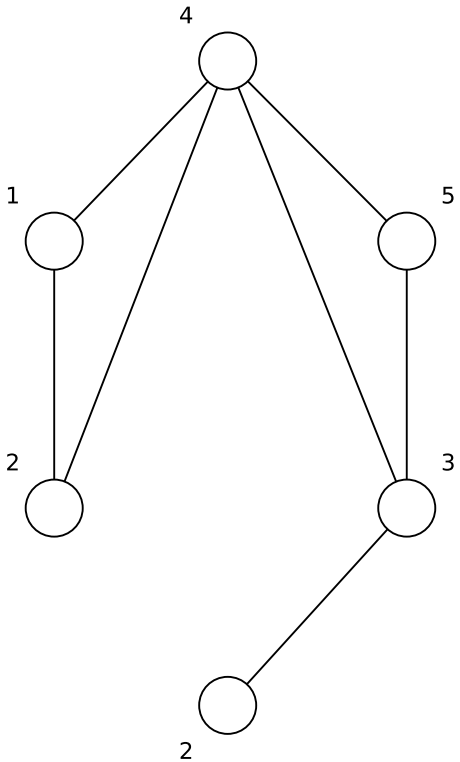


G_2

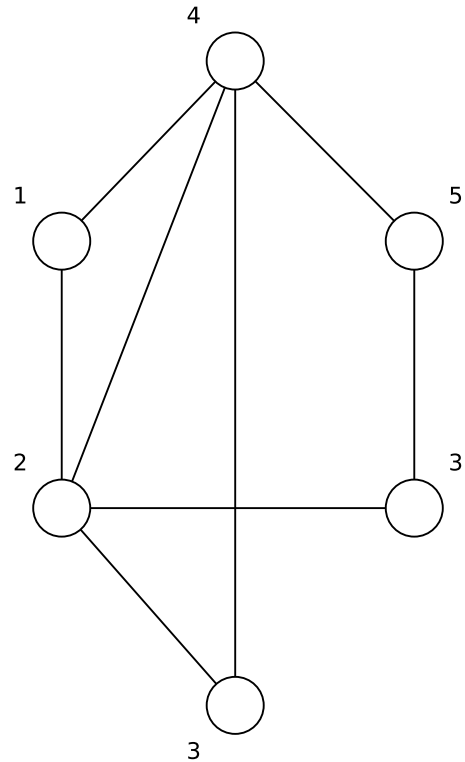
Initialization

Feature vector for a graph G:

$$\phi(G) = \{\#\text{nodes with label 1}, \#\text{nodes with label 2}, \ldots, \#\text{nodes with label } l\}$$



G_1



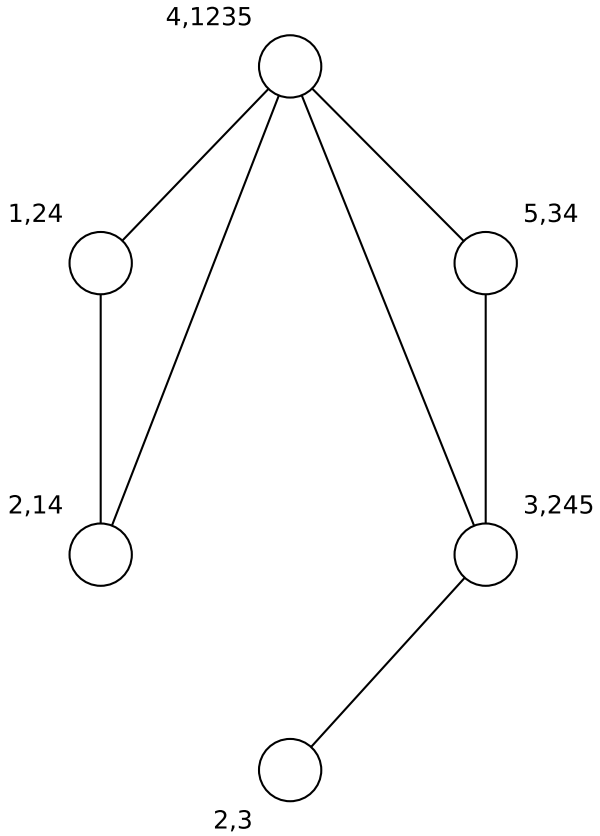
G_2

$$\phi(G_1) = \{1, 2, 1, 1, 1\}^\top \qquad \phi(G_2) = \{1, 1, 2, 1, 1\}^\top$$

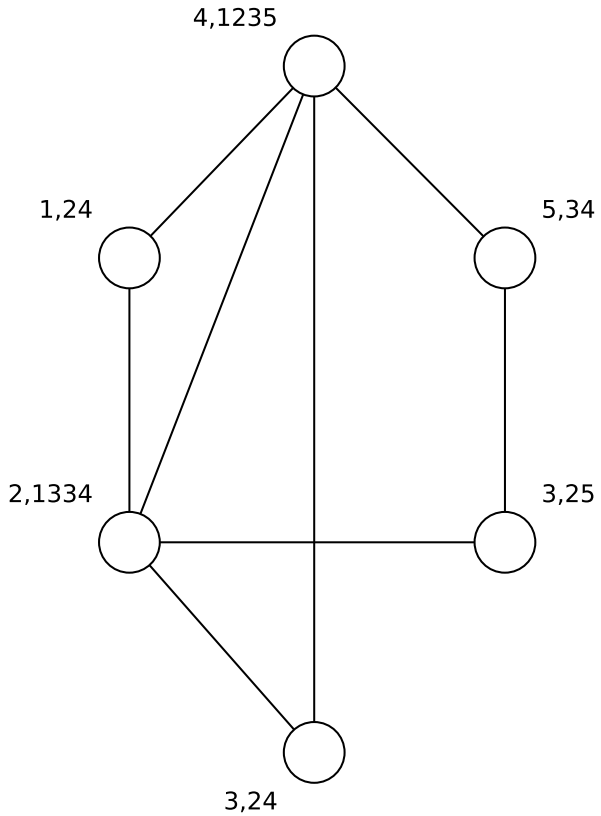
$$k(G_1, G_2) = \phi(G_1)^\top \phi(G_2) = 7$$

Iteration 1

First step: Augment the labels of the vertices by the sorted set of labels of neighbouring vertices



G_1

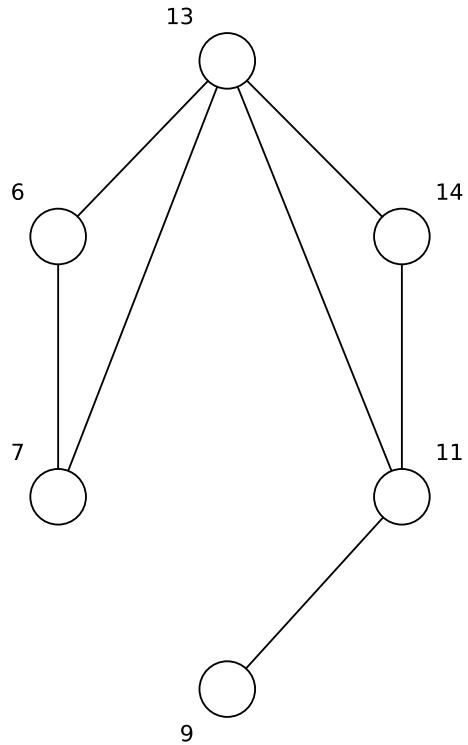


G_2

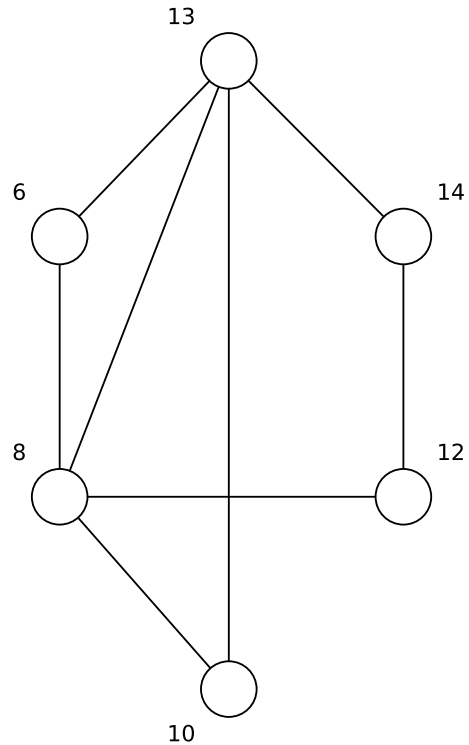
Iteration 1

Second step: Compress the augmented labels into new, short labels:

- $1, 24 \rightarrow 6$
- $2, 14 \rightarrow 7$
- $2, 1334 \rightarrow 8$
- $2, 3 \rightarrow 9$
- $3, 24 \rightarrow 10$
- $3, 245 \rightarrow 11$
- $3, 25 \rightarrow 12$
- $4, 1235 \rightarrow 13$
- $5, 34 \rightarrow 14$



G_1



G_2

Graph Kernels Features

Graph Kernel	Exp. ϕ	Node Labels	Node Attributes	Type
Vertex Histogram	√	√	×	R-convolution
Edge Histogram	√	√	×	R-convolution
Random Walk	×	√	√	R-convolution
Subtree	×	√	√	R-convolution
Cyclic Pattern	√	√	×	intersection
Shortest Path	×	√	√	R-convolution
Graphlet	√	×	×	R-convolution
Weisfeiler-Lehman Subtree	√	√	×	R-convolution
Neighborhood Hash	√	√	×	intersection
Neighborhood Subgraph Pairwise Distance	√	√	×	R-convolution
Lovász ϑ	√	×	×	R-convolution
SVM-ϑ	√	×	×	R-convolution
Ordered Decomposition DAGs	√	√	×	R-convolution
Pyramid Match	×	√	×	assignment
Weisfeiler-Lehman Optimal Assignment	×	√	×	assignment
Subgraph Matching	×	√	√	R-convolution
GraphHopper	×	√	√	R-convolution
Graph Invariant Kernels	×	√	√	R-convolution
Propagation	√	√	√	R-convolution
Multiscale Laplacian	×	√	√	R-convolution

Summary of selected graph kernels regarding computation by explicit feature mapping (Exp.ϕ), support for node-labeled and node-attributed graphs, and type
*$R-convolution$: decompose graphs into their substructures and add up the pairwise similarities between these substructures

Graph Kernels: a Survey, G. Nikolentzos, M. Vazirgiannis, under submission

Graph Similarity with Graph Kernels - DaSciM contributions

New Kernels

- Matching Node Embeddings for Graph Similarity [AAAI 2017]
- Message Passing GKs [arXiv:1808.02510]
- Shortest-path graph kernels for document similarity – [ENMLP 2017] - applications to NLP

Kernel based Similarity / Embedding Frameworks

- Degeneracy framework for graph similarity [IJCAI 2018 - best paper award]
- Enhancing graph kernels via successive embeddings [CIKM 2018]
- Structural Node Embeddings using Graph Kernels [submitted to TKDE]

Software Library

- GraKel: A python library for graph kernels – scikit compatible
  https://github.com/ysig/GraKeL

Matching Node Embeddings for Graph Similarity [AAAI 2017]

Goal: Measure the similarity between pairs of graphs and perform graph classification
Motivation:

1. Graph similarity is a key issue in many applications (e.g. chemoinformatics, bioinformatics)
2. Most algorithms focus on local substructures of graphs (e.g. graphlets, cycles, trees)
3. Several interesting properties of graphs may not be captured in local substructures

Contributions:

- Generate features describing global properties of graphs
- Elaborate two algorithms that utilize these features:
  1. Earth Mover's Distance **[Rubner et al., IJCV '00]**
  2. Pyramid Match Kernel **[Grauman and Darrell, JMLR '07]**

Preliminary Concepts

Node embeddings: represent nodes as points in a vector space

- Generate embeddings using eigenvectors of adjacency matrix
- Such embeddings capture global properties of graphs

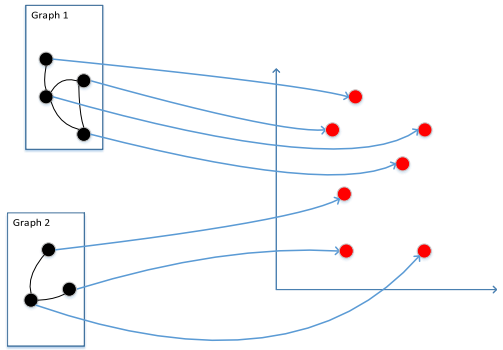
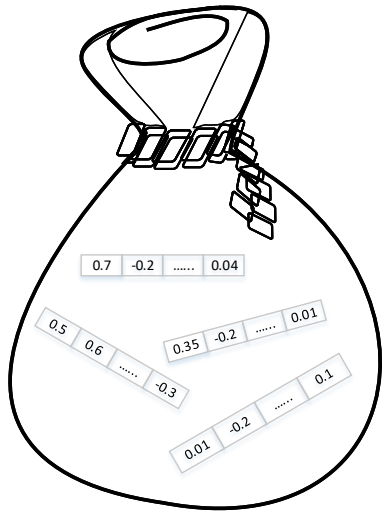


Figure: Node embedding

Graphs represented as **bags-of-vectors**:

- A graph is represented as a set of vectors: $\{\mathbf{u}_1, \ldots, \mathbf{u}_n\}$
- Each vector $\mathbf{u}_i \in \mathbb{R}^d$ represents the embedding of the i^{th} node in the d-dimensional space
- This is a natural representation
  $\hookrightarrow$ There is no canonical ordering for the nodes of a graph



Earth Mover's Distance

Earth Mover's Distance (**EMD**): minimum "travel cost" from $G_1 = (V_1, E_1)$ to $G_2 = (V_2, E_2)$:

$$\min_{\mathbf{T} \geq 0} \sum_{i=1}^{n_1} \sum_{j=1}^{n_2} \mathbf{T}_{ij} ||\mathbf{v}_i - \mathbf{u}_j||_2$$

subject to

$$\sum_{i=1}^{n_1} \mathbf{T}_{ij} = \frac{1}{n_2} \quad \forall j \in \{1, \ldots, n_2\}$$

$$\sum_{j=1}^{n_2} \mathbf{T}_{ij} = \frac{1}{n_1} \quad \forall i \in \{1, \ldots, n_1\}$$

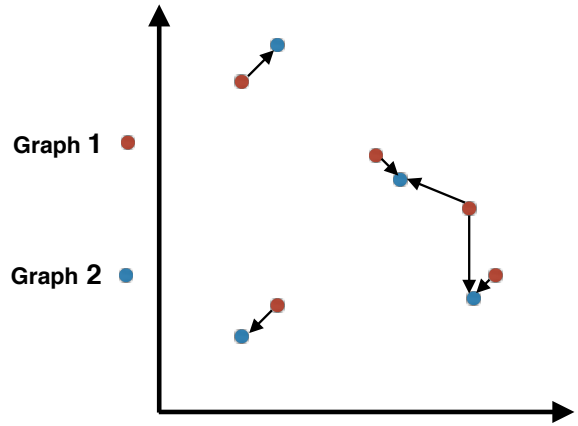


Figure: Minimum cumulative distance between G_1, G_2

where $\mathbf{v}_i, \mathbf{u}_j$ are the embeddings of nodes $v_i \in V_1, u_j \in V_2$ and $\mathbf{T}$ is a flow matrix

- Each vertex $v_i \in V_1$ transformed into any vertex $u_j \in V_2$ in total or in parts
- Outgoing flow from each graph = 1 and equally divided among all vertices
- However, emerging similarity matrices not necessarily positive semidefinite
- Complexity: $\mathcal{O}(n^3 logn)$

However, emerging similarity matrices not necessarily positive semidefinite
Complexity: $\mathcal{O}(n^3 logn)$

Earth Mover's Distance

Earth Mover's Distance (**EMD**): minimum "travel cost" from $G_1 = (V_1, E_1)$ to $G_2 = (V_2, E_2)$:

$$\min_{\mathbf{T} \geq 0} \sum_{i=1}^{n_1} \sum_{j=1}^{n_2} \mathbf{T}_{ij} ||\mathbf{v}_i - \mathbf{u}_j||_2$$

subject to

$$\sum_{i=1}^{n_1} \mathbf{T}_{ij} = \frac{1}{n_2} \quad \forall j \in \{1, \ldots, n_2\}$$

$$\sum_{j=1}^{n_2} \mathbf{T}_{ij} = \frac{1}{n_1} \quad \forall i \in \{1, \ldots, n_1\}$$

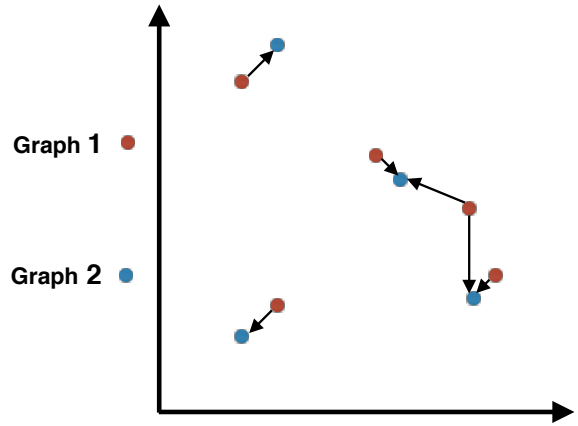


Figure: Minimum cumulative distance between G_1, G_2

where $\mathbf{v}_i, \mathbf{u}_j$ are the embeddings of nodes $v_i \in V_1, u_j \in V_2$ and $\mathbf{T}$ is a flow matrix

- Each vertex $v_i \in V_1$ transformed into any vertex $u_j \in V_2$ in total or in parts
- Outgoing flow from each graph = 1 and equally divided among all vertices
- However, emerging similarity matrices not necessarily positive semidefinite
- Complexity: $\mathcal{O}(n^3 logn)$

However, emerging similarity matrices not necessarily positive semidefinite
Complexity: $\mathcal{O}(n^3 logn)$

Pyramid Match Graph Kernel - need to fit..all

The Pyramid Match Graph Kernel (**PM**)

- partitions feature space into cells
- at level $l \rightarrow 2^l$ cells along each dimension

Number of nodes (i.e. embeddings) that match at l:

$$I(H^l_{G_1}, H^l_{G_2}) = \sum_{i=1}^{2^l d} \min\left(H^l_{G_1}(i), H^l_{G_2}(i)\right)$$

$H^l_G(i)$: number of nodes of G in cell i

PM: weightd sum of the matches occurring at each level (levels 0 to L):

$$k_\Delta(G_1, G_2) = I(H^L_{G_1}, H^L_{G_2}) + \sum_{l=0}^{L-1} \frac{1}{2^{L-l}} \left(I(H^l_{G_1}, H^l_{G_2}) - I(H^{l+1}_{G_1}, H^{l+1}_{G_2})\right)$$

- Matches within lower levels weighted less
- Only new matches are taken into account

Complexity: $\mathcal{O}(dnL)$

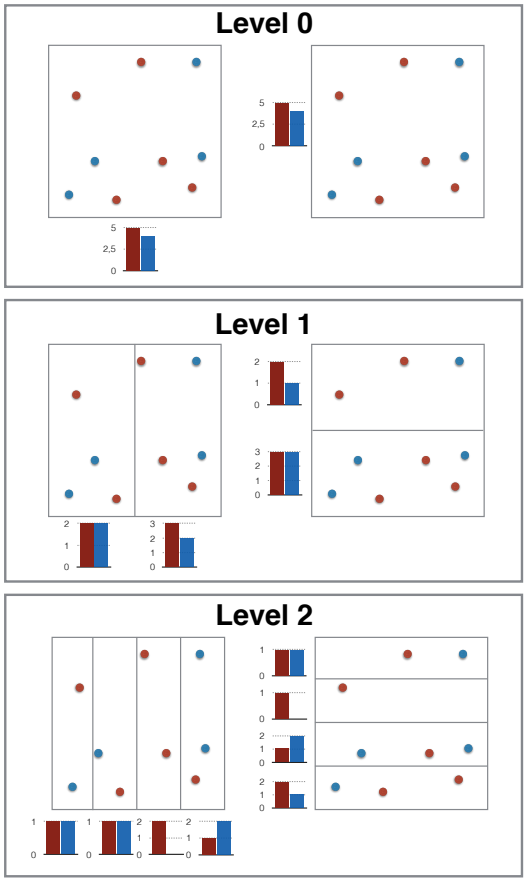


Figure: Histogram creation

Graph Classification Datasets - misssing table

Dataset	MUTAG	ENZYMES	NCI1	NCI109	PTC-MR	D&D
Max # vertices	28	126	111	111	109	5,748
Min # vertices	10	2	3	4	2	30
Average # vertices	17.93	32.63	29.87	29.68	25.56	284.32
Max # edges	33	149	119	119	108	14,267
Min # edges	10	1	2	3	1	63
Average # edges	19.79	62.14	32.30	32.13	25.96	715.66
# labels	7	3	37	38	19	82
# graphs	188	600	4,110	4,127	344	1,178
# classes	2	6	2	2	2	2

Table: Data Sets

Baselines We compare our methods against the following baselines:

- random walk kernel (RW)
- graphlet kernel (GR)
- shortest path kernel (SP)
- Weisfeiler-Lehman subtree and shortest path kernels (WL ST, WL SP)
- Lovász ϑ kernel (Lo-ϑ)
- optimal assignment between node embeddings (OA)

Experimental evaluation - missing table

Method \ Datasets	MUTAG	ENZYMES	NCI1	NCI109	PTC-MR	D&D
Without node labels						
SP	82.22 (± 1.14) 0.16"	28.17 (± 0.64) 1.26"	62.02 (± 0.17) 7.55"	61.41 (± 0.32) 7.32"	56.18 (± 0.56) 0.47"	> 1 day
RW	77.78 (± 0.98) 17.01"	20.17 (± 0.83) 4' 46"	56.89 (± 0.24) 2h 40' 15"	56.13 (± 0.31) 2h 44' 29"	56.18 (± 1.12) 1' 23"	58.71 (± 0.43) 6h 45' 1"
GR	66.11 (± 1.31) 0.07"	18.16 (± 0.47) 2.01"	47.37 (± 0.15) 4.42"	48.39 (± 0.18) 4.37"	57.05 (± 0.83) 0.14"	63.67 (± 0.57) 1' 04"
Lo-ϑ	82.78 (± 0.89) 15' 26"	26.33 (± 0.44) 2h 11' 31"	62.68 (± 0.24) 17h 41' 57"	62.42 (± 0.27) 17h 45' 27"	55.00 (± 0.62) 1h 9' 58"	> 1 day
OA	79.44 (± 1.08) 6.56"	36.33 (± 0.71) 3' 19"	67.81 (± 0.18) 2h 21' 57"	66.94 (± 0.21) 2h 20' 13"	56.17 (± 0.95) 42.94"	> 1 day
EMD	**86.11** (± 0.84) 4.5"	**36.83** (± 0.78) 1' 57"	**72.65** (± 0.24) 1h 11' 31"	**71.70** (± 0.16) 1h 10' 37"	57.65 (± 0.59) 24.28"	> 1 day
PM	85.55 (± 0.63) 1.26"	28.17 (± 0.37) 8.07"	69.73 (± 0.11) 3' 19"	68.37 (± 0.14) 3' 17"	**59.41** (± 0.68) 3.51"	**75.55** (± 0.62) 41.23"
With node labels						
SP	87.78 (± 0.44) 0.16"	41.00 (± 0.26) 1.42"	72.85 (± 0.18) 9.86"	73.20 (± 0.16) 9.46"	60.00 (± 0.72) 0.48"	> 1 day
RW	81.11 (± 1.23) 3' 51"	19.33 (± 0.62) 12' 58"	> 1 day	> 1 day	57.06 (± 0.86) 1h 33' 32"	> 1 day
GR	71.67 (± 0.81) 0.79"	32.00 (± 0.46) 17.77"	65.52 (± 0.35) 40.03"	66.70 (± 0.15) 41.36"	59.41 (± 0.94) 2.21"	**79.40** (± 0.39) 15' 13"
OA	82.22 (± 0.68) 6.03"	43.16 (± 0.56) 1' 57"	69.53 (± 0.20) 1h 33' 35"	68.76 (± 0.15) 1h 32' 51"	58.23 (± 0.82) 28.28"	77.52 (± 0.43) 7h 45' 5"
WL ST	83.33 (± 0.86) 2.91"	52.16 (± 0.61) 17.12"	84.72 (± 0.16) 1' 57"	84.26 (± 0.22) 2' 00"	57.64 (± 0.66) 7.35"	76.83 (± 0.49) 5' 06"
WL SP	84.55 (± 1.08) 1.27"	**61.00** (± 0.69) 12.23"	84.64 (± 0.21) 1' 15"	84.29 (± 0.17) 1' 13"	56.76 (± 0.74) 2.81"	> 1 day
WL OA	85.55 (± 1.02) 27.46"	53.66 (± 0.72) 7' 2"	85.35 (± 0.18) 5h 7' 57"	84.51 (± 0.14) 5h 4' 5"	59.70 (± 1.01) 2' 3"	> 1 day
EMD	**89.44** (± 0.76) 5.61"	43.17 (± 0.48) 1' 53"	76.76 (± 0.17) 1h 15' 44"	73.88 (± 0.18) 1h 16' 10"	58.82 (± 0.83) 25.79"	> 1 day
PM	86.67 (± 0.60) 3.52"	40.33 (± 0.34) 21.50"	72.91 (± 0.53) 5' 48"	71.97 (± 0.15) 5' 50"	60.22 (± 0.86) 9.58"	77.78 (± 0.48) 4' 30"
WL PM	87.77 (± 0.81) 9.20"	55.55 (± 0.56) 1' 25"	**86.40** (± 0.20) 50' 35"	**85.34** (± 0.23) 50' 42"	**61.41** (± 0.81) 29.33"	78.63 (± 0.26) 16' 15"

Figure: Classification accuracy (± standard deviation) and CPU runtime for kernel/similarity matrix computation of the random walk kernel (RW), shortest path kernel (SP), lets of size 3 kernel (GR), Lovász ϑ kernel (Lo-ϑ), optimal assignment similarity (OA), Weisfeiler-Lehman subtree kernel (WL ST), Weisfeiler-Lehman shortest path kernel (WL SP), Weisfeiler-Lehman optimal assignment similarity (WL OA), earth mover's distance similarity (EMD), pyramid match kernel (PM) and Weisfeiler-Lehman pyramid match kernel (WL PM) on the 6 graph classification datasets. > 1 day indicates that the computation did not finish after 1 day.

Degeneracy Framework for Graph Comparison

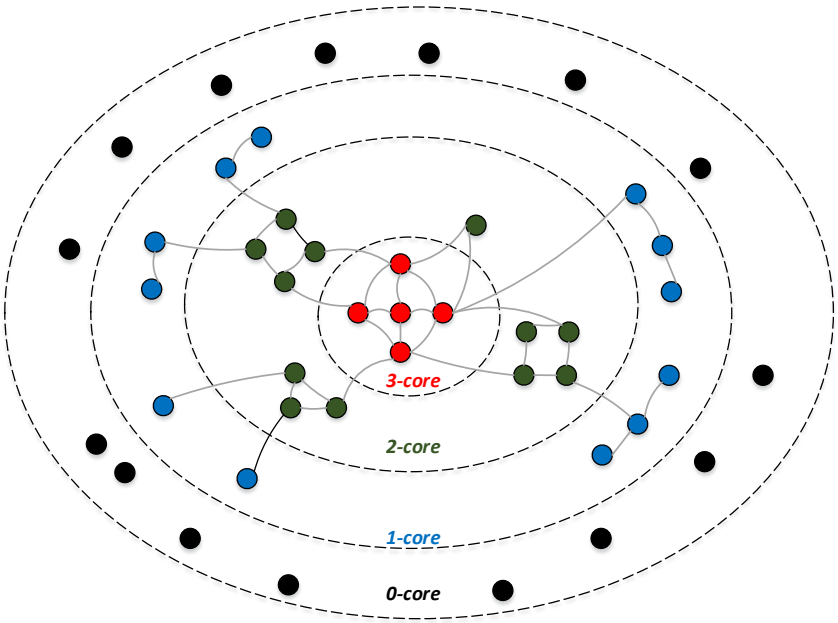
[IJCAI 2018 - best paper award]

A framework that allows graph similarity algorithms to compare structure in graphs at multiple different scales

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

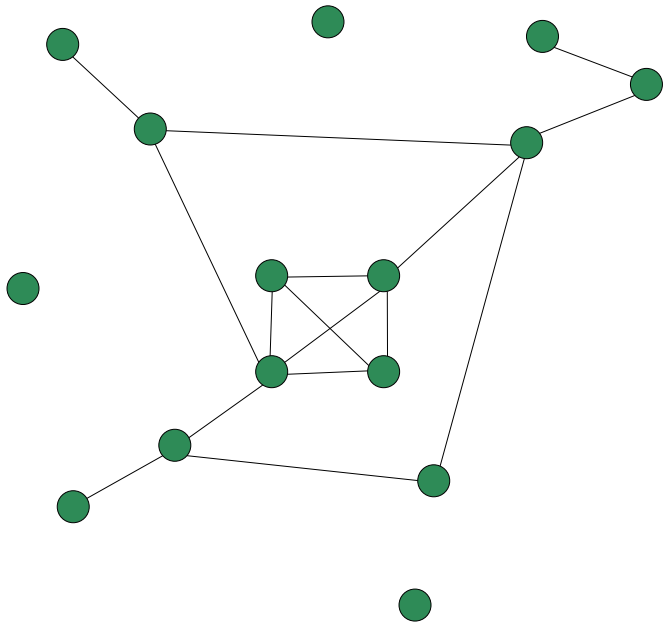
Degeneracy Framework for Graph Comparison

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

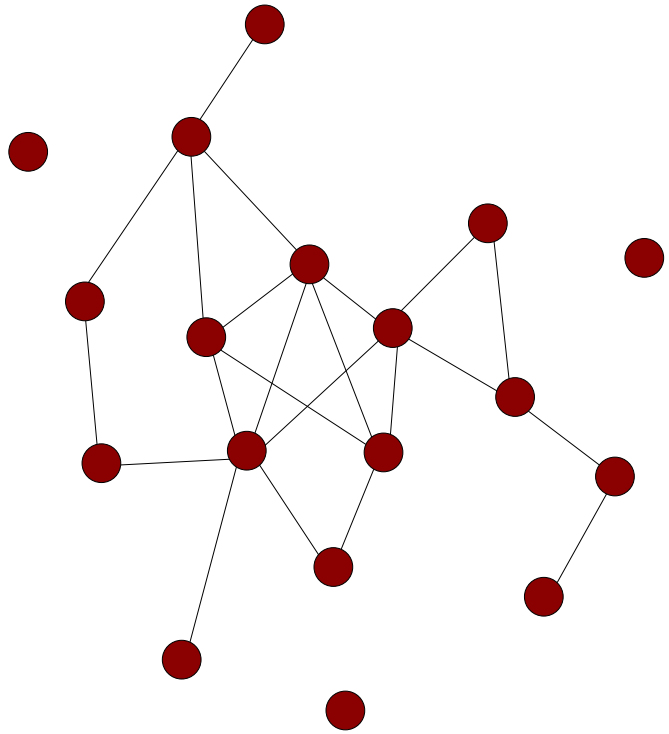
- Let $G = (V, E)$ and $G' = (V', E')$ be two graphs
- Let δ^*_{min} be the minimum of the degeneracies of G, G'
- Let $C_0, C_1, \ldots, C_{\delta^*_{min}}$ and $C'_0, C'_1, \ldots, C'_{\delta^*_{min}}$ be the 0-core, 1-core,..., δ^*_{min}-core subgraphs of G and G' respectively
- Let k be any kernel for graphs
- The **core variant** of the base kernel k is defined as:

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

Degeneracy Framework for Graph Comparison - Example

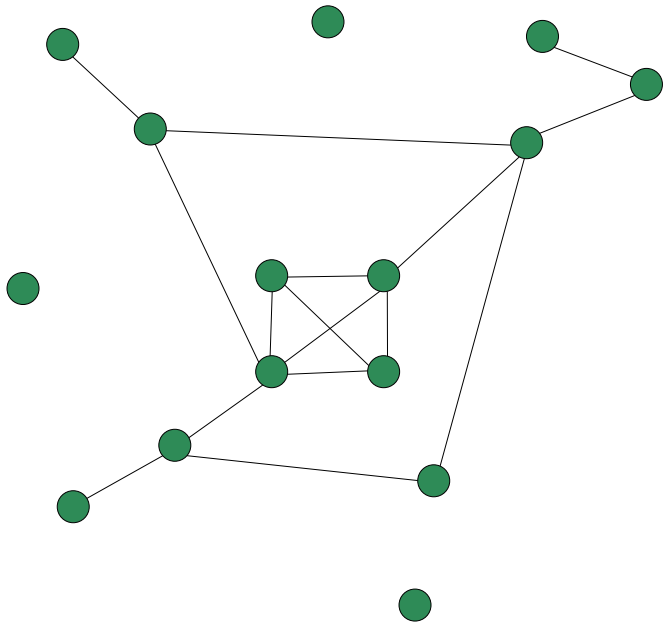


G

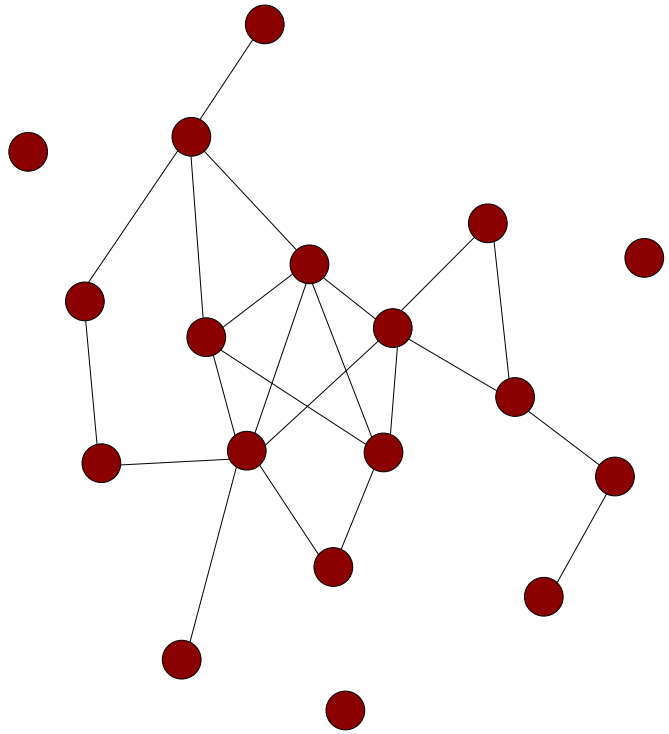


G'

Degeneracy Framework for Graph Comparison - Example



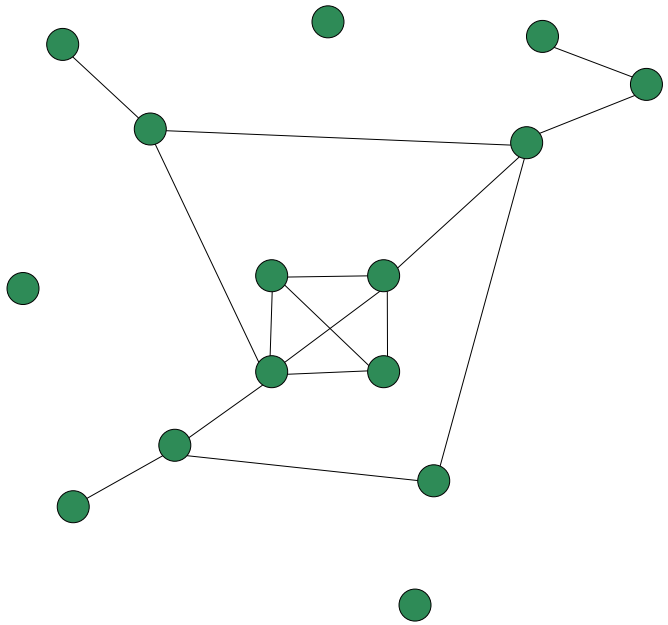
G



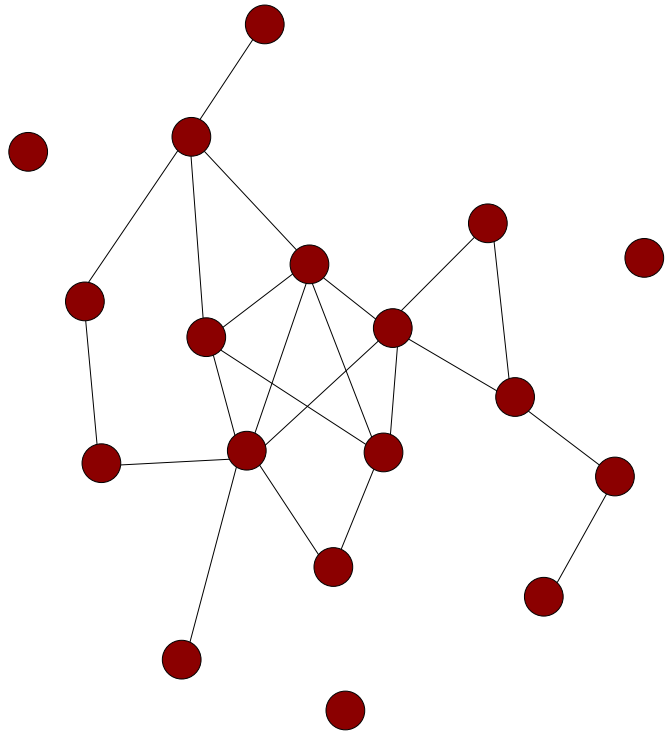
G'

k

Degeneracy Framework for Graph Comparison - Example



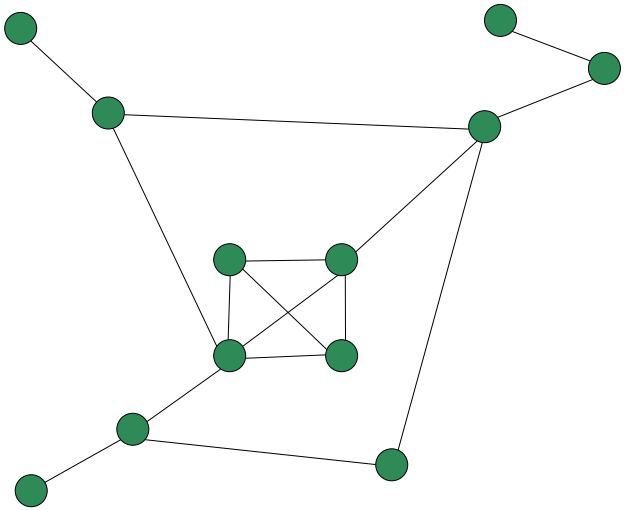
C_0



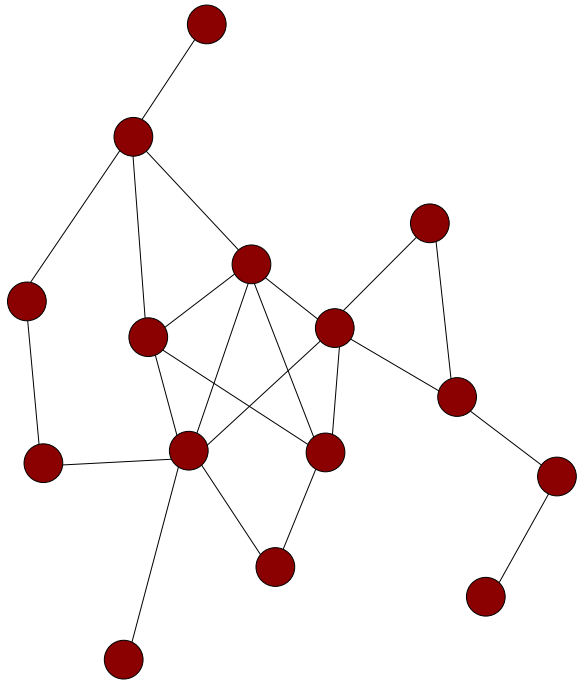
C_0'

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

Degeneracy Framework for Graph Comparison - Example



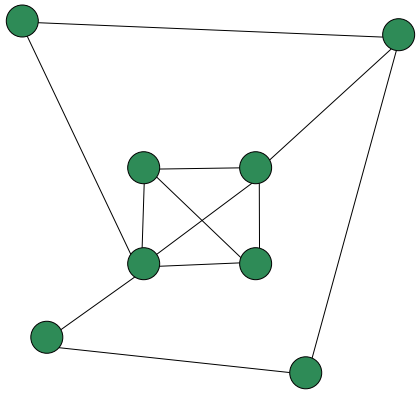
C_1

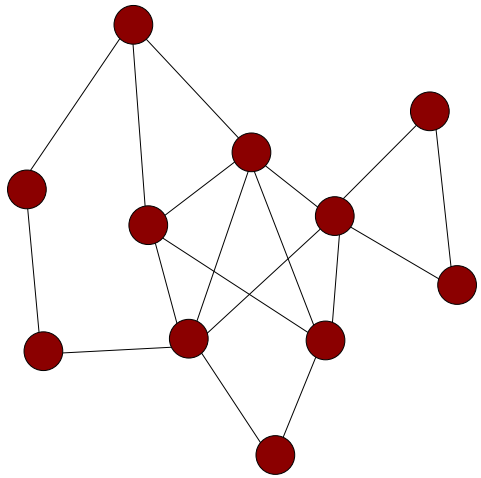


C_1'

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

Degeneracy Framework for Graph Comparison - Example

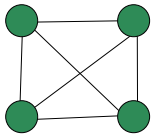


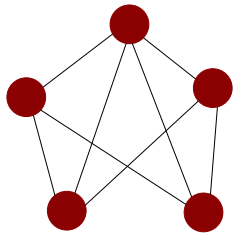


C_2 C_2'

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

Degeneracy Framework for Graph Comparison - 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

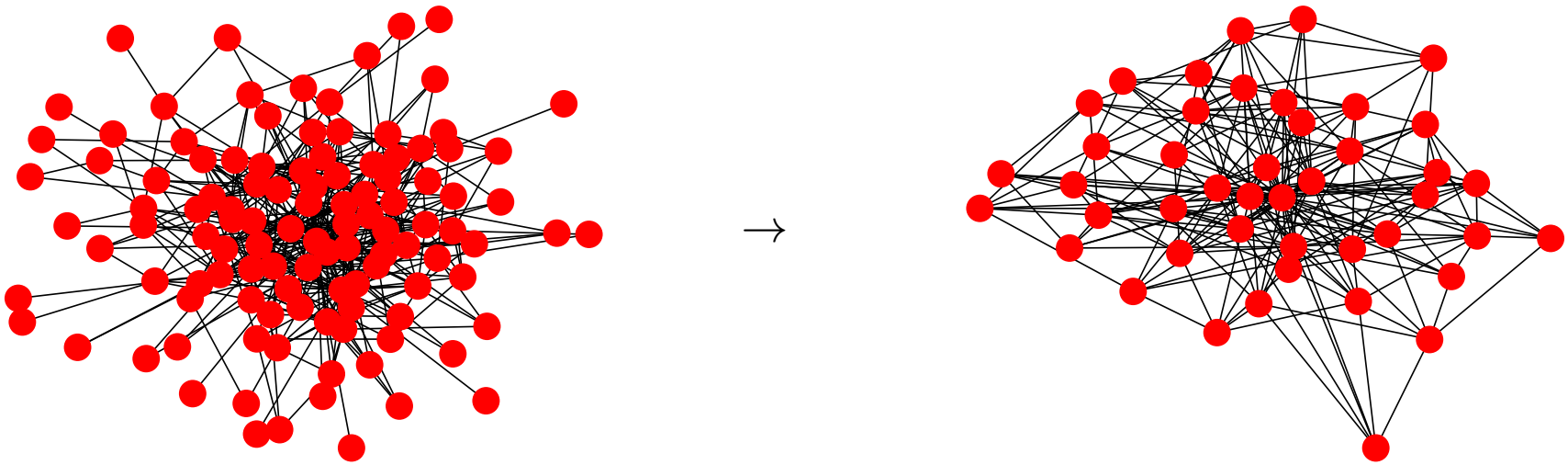
Dimensionality Reduction View

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



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	COLLAB
Max # vertices	28	126	111	111	109	620	5748	136	89	3782	3648	3782	492
Min # vertices	10	2	3	4	2	4	30	12	7	6	22	2	32
Average # vertices	17.93	32.63	29.87	29.68	25.56	39.05	284.32	19.77	13.00	429.61	508.50	391.40	74.49
Max # edges	33	149	119	119	108	1049	14267	1249	1467	4071	4783	5171	40119
Min # edges	10	1	2	3	1	5	63	26	12	4	21	1	60
Average # edges	19.79	62.14	32.30	32.13	25.96	72.81	715.66	96.53	65.93	497.75	594.87	456.89	2457.34
# graphs	188	600	4110	4127	344	1113	1178	1000	1500	2000	4999	11929	5000
# classes	2	6	2	2	2	2	2	2	3	2	5	11	3

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

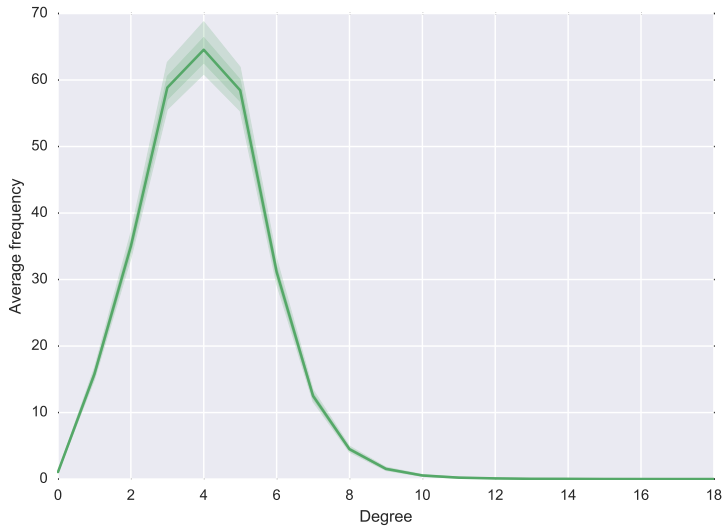
We employed the following kernels:

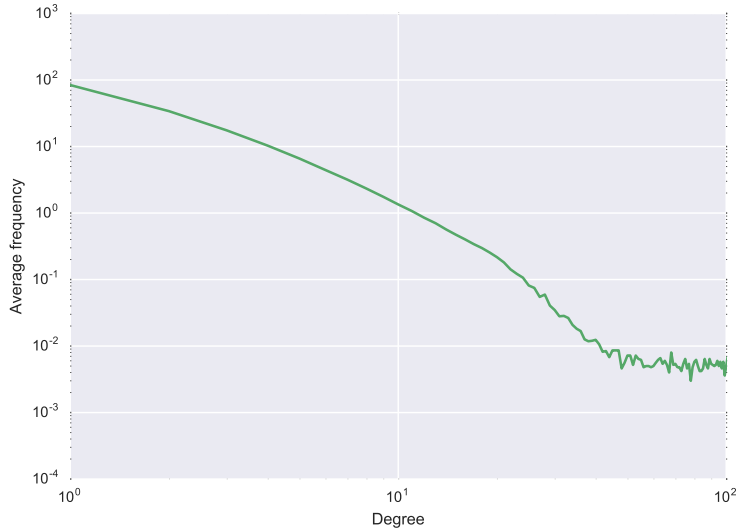
1. **Graphlet kernel (GR)** [?]: The graphlet kernel counts identical pairs of graphlets (i.e. subgraphs with k nodes where $k \in 3, 4, 5$) in two graphs
2. **Shortest path kernel (SP)** [?]: The shortest path kernel counts pairs of shortest paths in two graphs having the same source and sink labels and identical length
3. **Weisfeiler-Lehman subtree kernel (WL)** [?]: 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)** [?]: 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

Enhancing Graph Kernels via Successive Embeddings [CIKM 2018]

Goal: Measure similarity between pairs of graphs for graph classification

Motivation:

- Graph similarity - key issue in many applications (e.g. chemoinformatics, bioinformatics)
- Graph kernels compute implicitly the inner product between the representations of input graphs in $\mathcal{H}$
  - Equivalent to computing the linear kernel on feature space $\mathcal{H}$
  - Linear kernel limits expressiveness of derived representations

Contributions:

1. Increase expressive power of graph kernels by applying kernel functions for vector data to derived representations
2. Embed graphs into a high-dimensional feature space by combining graph kernels with gaussian and polynomial kernels
3. Show how to efficiently compute this sequence in the kernel space

Successive Embeddings

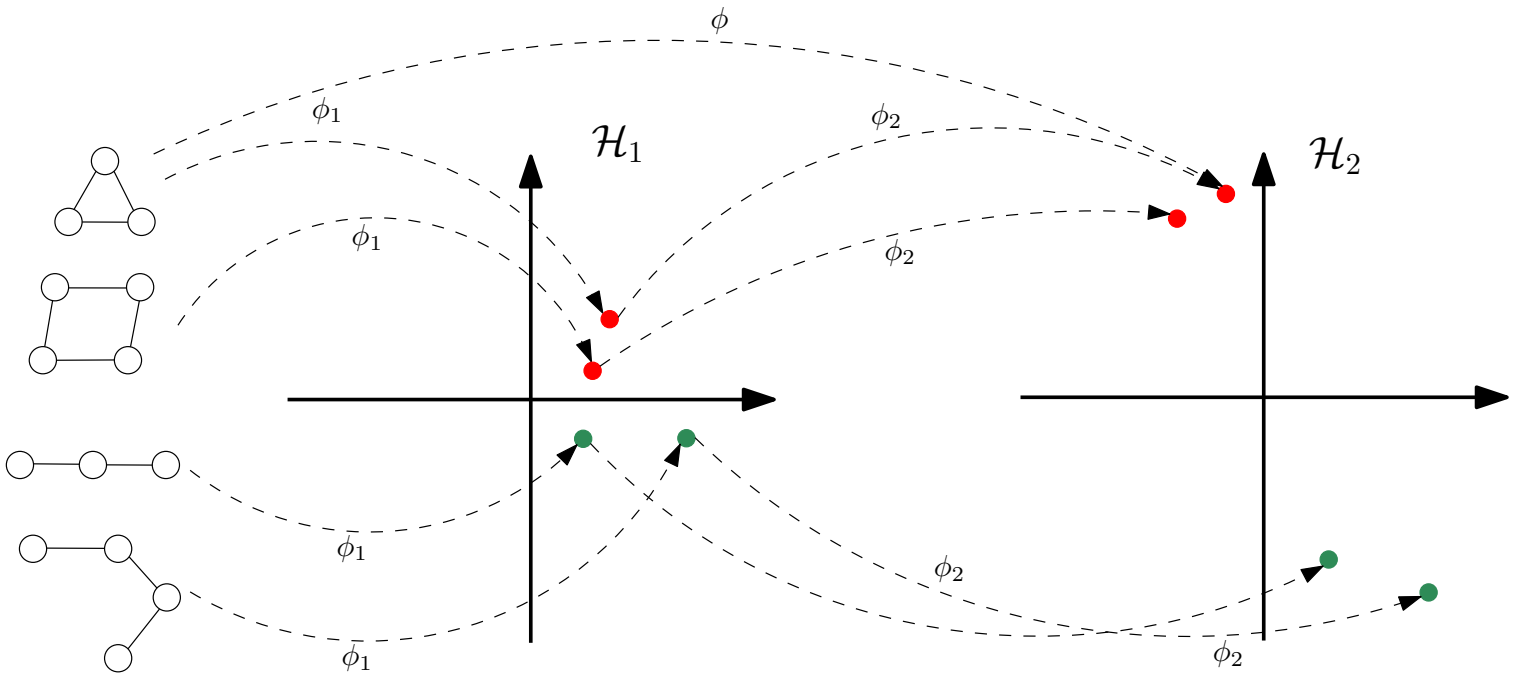
Idea: Obtain complex kernels by stacking simpler kernels on top of one another
Methodology:

- Define a kernel k_1 on set of graphs $\mathcal{G}$
  $\hookrightarrow$ There exists a Hilbert space $\mathcal{H}_1$ and a mapping ϕ_1:
  $k_1(G, G') = \langle \phi_1(G), \phi_1(G') \rangle_{\mathcal{H}_1}$ for all $G, G' \in \mathcal{G}$
- Define a kernel k_2 on feature space $\mathcal{H}_1$
  $\hookrightarrow$ There exists a Hilbert space $\mathcal{H}_2$ and a mapping ϕ_2 such that
  $k_2(\phi_1(G), \phi_1(G')) = \langle \phi_2(\phi_1(G)), \phi_2(\phi_1(G')) \rangle_{\mathcal{H}_2}$ for all $\phi_1(G), \phi_1(G') \in \mathcal{H}_1$
- Define a new kernel k_3 on feature space $\mathcal{H}_2$
  $\hookrightarrow$ There exists a Hilbert space $\mathcal{H}_3$ and a mapping ϕ_3 such that
  $k_3(\phi_2(\phi_1(G)), \phi_2(\phi_1(G'))) = \langle \phi_3(\phi_2(\phi_1(G))), \phi_3(\phi_2(\phi_1(G'))) \rangle_{\mathcal{H}_3}$ for all
  $\phi_2(\phi_1(G)), \phi_2(\phi_1(G')) \in \mathcal{H}_2$

.....

Successive Embeddings - Example

- Figure below illustrates a sequence of two embeddings
- Separation of the data points associated with the two classes progressively increased



Proposed Instances of Kernels

Proposed Instances: Sequences of two embeddings
Embedding 1: Embed graphs in a Hilbert space $\mathcal{H}_1$ using a graph kernel

- *Shortest path kernel (SP)* Borgwardt and Kriegel, ICDM '05
- *Weisfeiler-Lehman subtree kernel (WL)* Shervashidze et al., JMLR '09
- *Pyramid match graph kernel (PM)* Nikolentzos et al., AAAI '17

Embedding 2: Embed emerging representations x, y into a Hilbert space $\mathcal{H}_2$ using kernels for vector data:

1. *Polynomial kernel*: $k_P(x, y) = \big(\langle x, y\rangle\big)^d, \quad d \in \mathbb{N}$
2. *Gaussian kernel*: $k_G(x, y) = exp\Big(-\frac{||x-y||^2}{2\sigma^2}\Big), \quad \sigma > 0$

Problem: Usually x and y not computed explicitly. How to apply **Embedding 2**?
$\hookrightarrow$ Use an implicit computation scheme

Proposed Instances of Kernels

The two kernels for vector data can be computed as:

1. *Polynomial kernel*: $k_P(x, y) = \big(\langle x, y\rangle\big)^d = \big(k(x, y)\big)^d, \quad d \in \mathbb{N}$
2. *Gaussian kernel*: $k_G(x, y) = exp\Big(- \frac{k(x,x)-2k(x,y)+k(y,y)}{2\sigma^2}\Big), \quad \sigma > 0$

where k is the employed graph kernel (i.e. the first kernel in the sequence)

Complexity:

- Increase depends only on size of dataset and not on size of graphs
- Both proposed kernels for vector data implemented using fast matrix operations
- Added complexity is low

Experimental Evaluation

Increase in running time due to the proposed approach negligible compared to running time of graph kernels
Example: largest dataset $\rightarrow$ REDDIT-MULTI-12K

- Computing each one of the 3 graph kernels takes several minutes/hours
- Given the kernel matrix generated by a graph kernel, computing second embedding takes less than 10 seconds

Accuracy

The proposed kernels:

- outperformed the baseline kernels on 34/36 experiments
- led to statistically significant improvement in accuracy on 29/36 experiments

Conclusion

- On most datasets, proposed kernels outperform baseline kernels
- Slight increase in running time

Experimental results

	Method \ Dataset	MUTAG	ENZYMES	AIDS	NCI1	PTC5-MR	D&D
SP	linear	86.46 (±1.47)	42.00 (±0.96)	99.34 (±0.07)	73.66 (±0.36)	59.21 (±1.87)	77.96 (±0.58)
	polynomial	84.18 (±1.89)	**53.15** (±1.03)	**99.53** (±0.03)	**77.89** (±0.25)	58.20 (±2.47)	**80.95** (±0.53)
	gaussian	87.94 (±1.02)	**50.52** (±1.34)	**99.64** (±0.01)	**77.80** (±0.36)	58.52 (±1.95)	**80.29** (±0.42)
WL	linear	80.30 (±1.43)	50.65 (±1.47)	98.05 (±0.10)	84.68 (±0.15)	61.62 (±1.16)	79.25 (±0.31)
	polynomial	**82.92** (±0.90)	**53.73** (±1.37)	**98.48** (±0.05)	**85.63** (±0.16)	**65.26** (±1.39)	79.58 (±0.29)
	gaussian	**85.55** (±0.96)	**54.80** (±0.82)	**99.21** (±0.07)	**86.17** (±0.17)	59.81 (±1.34)	77.91 (±0.33)
PM	linear	85.46 (±1.18)	40.65 (±0.82)	99.68 (±0.02)	69.71 (±0.73)	58.17 (±1.86)	77.05 (±0.96)
	polynomial	87.12 (±1.51)	**48.43** (±1.08)	99.63 (±0.03)	**77.35** (±0.32)	**61.19** (±1.43)	**78.74** (±0.59)
	gaussian	**87.71** (±1.39)	**48.32** (±0.88)	99.69 (±0.03)	**76.84** (±0.26)	**61.47** (±1.97)	77.49 (±0.58)

	Method \ Dataset	IMDB BINARY	IMDB MULTI	REDDIT BINARY	REDDIT MULTI-5K	REDDIT MULTI-12K	COLLAB
SP	linear	60.19 (±0.29)	39.82 (±0.57)	82.54 (±0.22)	49.24 (±0.15)	33.46 (±0.08)	64.58 (±0.60)
	polynomial	**63.99** (±0.83)	39.88 (±0.30)	**84.50** (±0.24)	**49.94** (±0.22)	**34.98** (±0.07)	60.03 (±0.04)
	gaussian	**70.08** (±1.11)	**46.37** (±0.49)	82.43 (±0.15)	46.05 (±0.21)	30.32 (±0.10)	**68.37** (±0.20)
WL	linear	73.51 (±0.43)	51.16 (±0.31)	69.36 (±0.49)	50.27 (±0.26)	36.20 (±0.08)	77.39 (±0.13)
	polynomial	73.49 (±0.33)	49.08 (±0.32)	**73.58** (±0.39)	50.06 (±0.23)	**37.62** (±0.10)	75.82 (±0.11)
	gaussian	74.11 (±0.60)	50.57 (±0.45)	**77.62** (±0.11)	**52.63** (±0.21)	**40.94** (±0.11)	**77.96** (±0.18)
PM	linear	65.25 (±0.97)	42.09 (±0.89)	77.88 (±0.11)	37.12 (±0.62)	30.70 (±0.40)	71.92 (±0.29)
	polynomial	66.35 (±0.68)	43.08 (±0.42)	**79.30** (±0.26)	36.63 (±0.39)	**31.20** (±0.31)	**75.46** (±0.24)
	gaussian	**66.64** (±0.69)	42.74 (±0.30)	78.01 (±0.12)	**40.92** (±0.51)	**34.19** (±0.19)	**73.59** (±0.23)

Table: Classification accuracy (± standard deviation) of the kernels defined using a sequence of two embeddings (*polynomial* and *gaussian*) and a single embedding (*linear*) on the 12 graph classification datasets. Kernels that perform successive embeddings with statistically significant improvements over the corresponding graph kernels are shown in bold as measured by a t-test with a p value of ≤ 0.05.

Graph Similarity with Graph Kernels - DaSciM contributions

New Kernels

- Matching Node Embeddings for Graph Similarity [AAAI 2017]
- Message Passing GKs [arXiv:1808.02510]
- Shortest-path graph kernels for document similarity – [ENMLP 2017] - applications to NLP

Kernel based Similarity / Embedding Frameworks

- Degeneracy framework for graph similarity [IJCAI 2018 - best paper award]
- Enhancing graph kernels via successive embeddings [CIKM 2018]
- Structural Node Embeddings using Graph Kernels [submitted to TKDE]

Software Library

- GraKel: A python library for graph kernels – scikit compatible
  https://github.com/ysig/GraKeL

SP Graph Kernels for Document Similarity [EMNLP2017]

Traditionally, documents are represented as bag of words (BOW) vectors

- I entries correspond to terms
- non-zero for terms appearing in the document

Example

- corpus vocabulary: the, quick, brown, cat, fox, jumped, went, over, lazy, lion, dog
- BOW representation of sentence: "the quick brown fox jumped over the lazy dog"

1	1	1	0	1	1	0	1	1	0	1

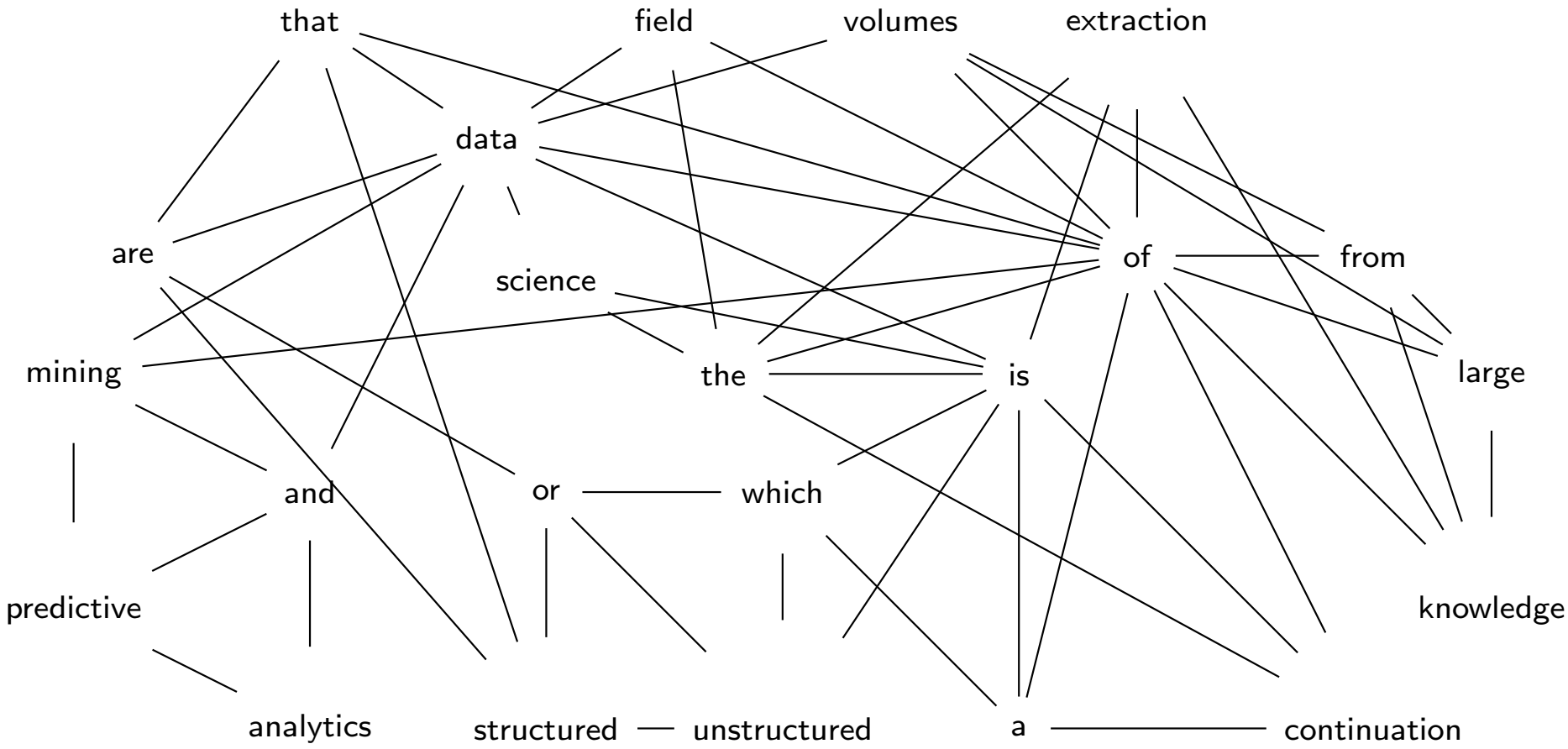
- However, BOW representation disregards word order!!!

Graph of Words [CIKM2013 - best paper mention award]

- Each document is represented as a graph $G = (V, E)$ consisting of a set V of vertices and a set E of edges between them
- vertices: unique terms (i.e. pre-preprocessed words)
- edges: co-occurrences within a fixed-size sliding window no edge weight no edge direction
- Graph representation more flexible than n-grams. It takes into account word inversion
- subset matching

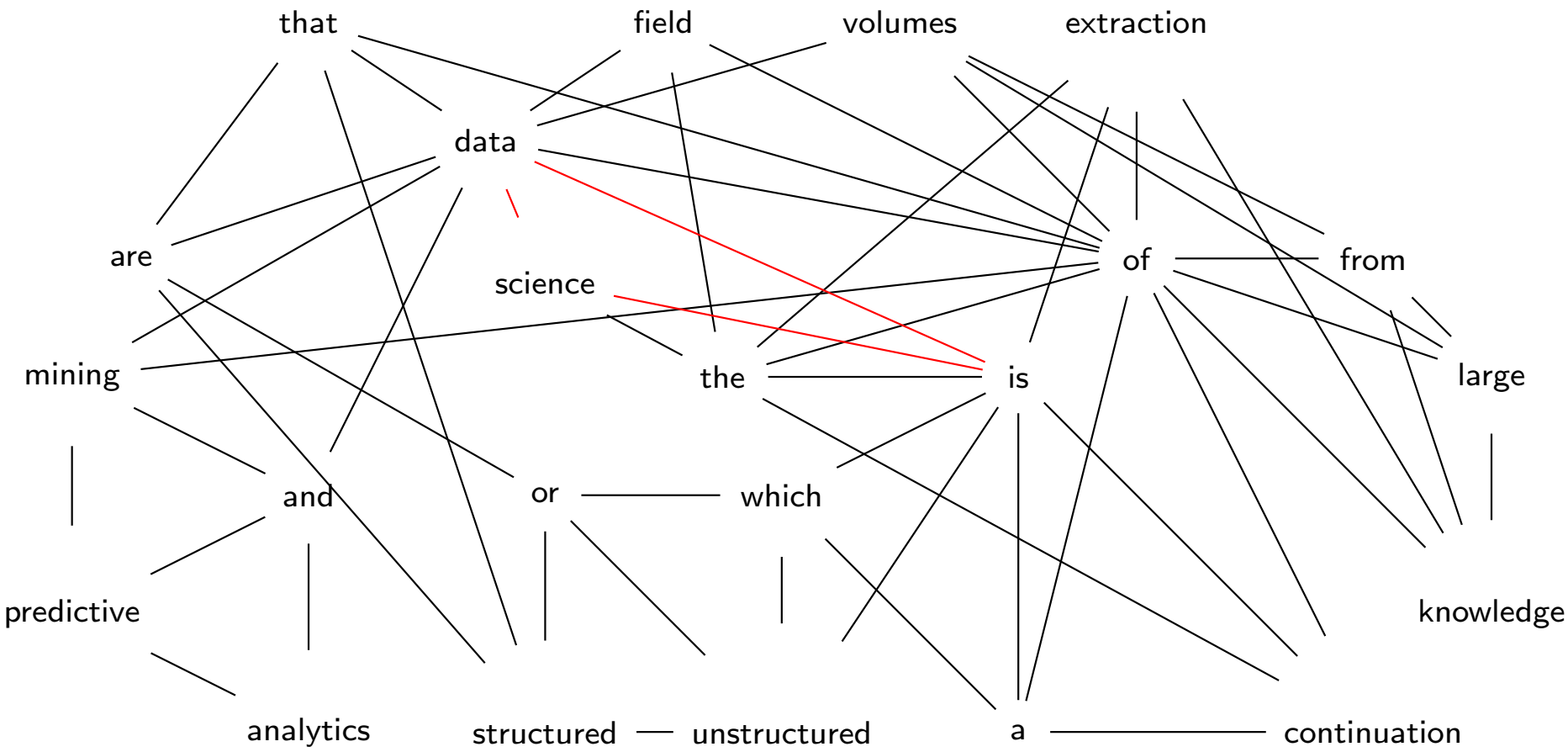
Graph of Words - Example

Data Science is the extraction of knowledge from large volumes of data that are structured or unstructured which is a continuation of the field of data mining and predictive analytics



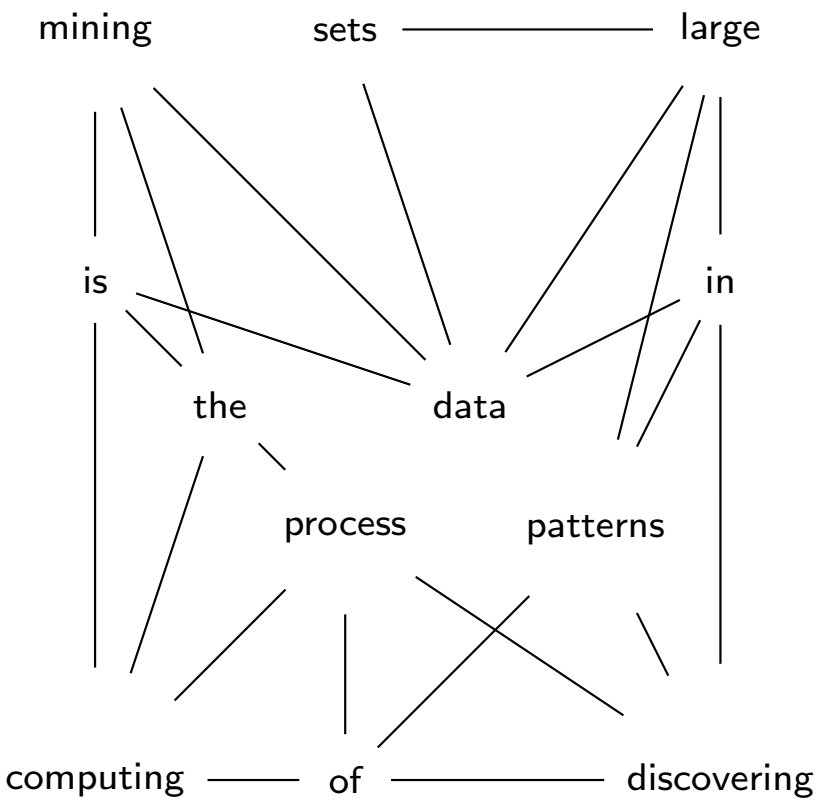
Graph of Words - Example

Data Science is the extraction of knowledge from large volumes of data that are structured or unstructured which is a continuation of the field of data mining and predictive analytics

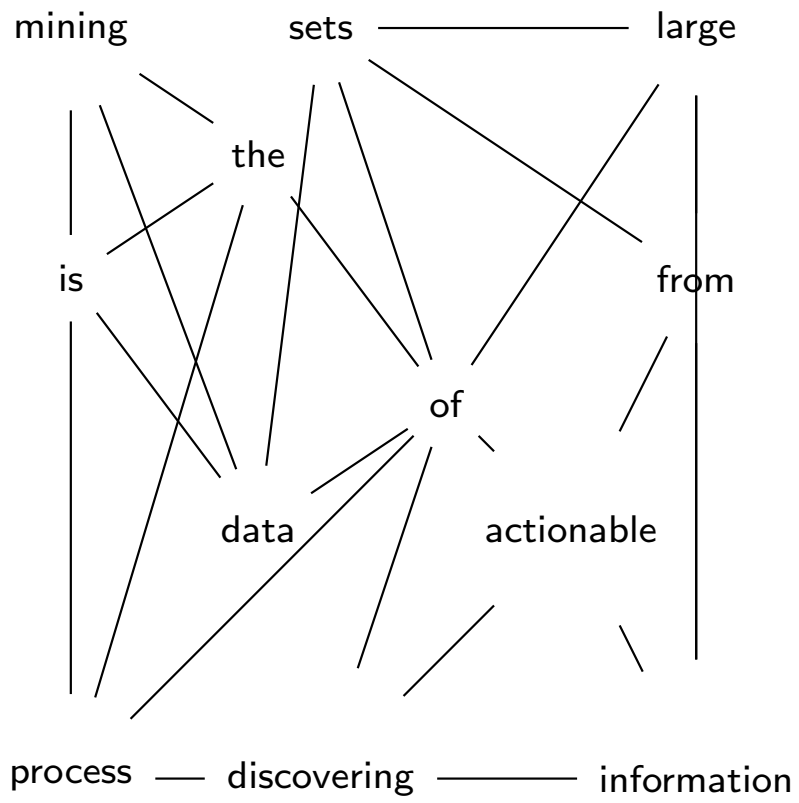


Graph of Words as a Graph similarity problem

Data mining is the computing process of discovering patterns in large data sets



Data mining is the process of discovering actionable information from large sets of data



Hence, document similarity problem $\rightarrow$ graph comparison problem

Custom Shortest Path Kernel

Based on the Shortest Path Kernel proposed by Borgwardt and Kriegel

Compares the length of shortest paths having the same endpoint labels in two graphs-of-words

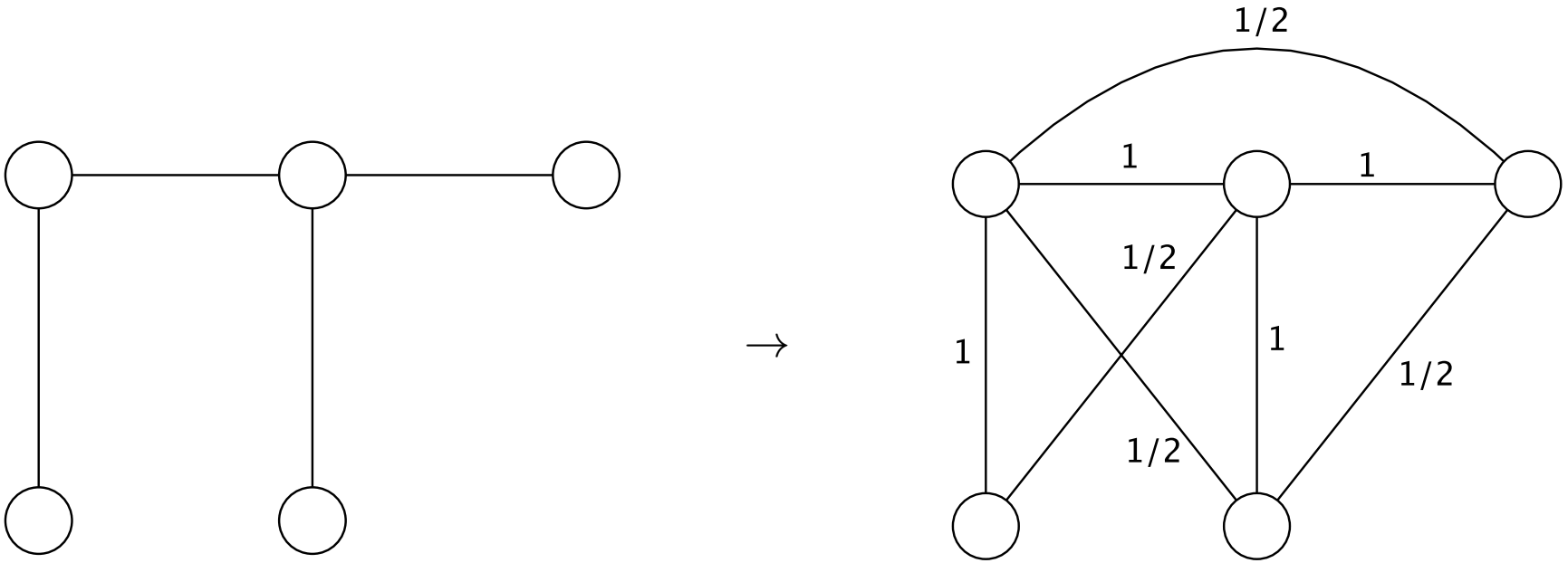
SP-transformation

Transforms the original graphs into shortest-paths graphs

- Create a shortest-path graph C
- Same set of vertices
- Edges correspond to shortest paths of length at most d in original graph

Example

SP-transformation ($d = 2$)



G C

Custom Shortest Path Kernel

Given the SP-transformed graphs $C_1 = (V_1, E_1)$ and $C_2 = (V_2, E_2)$ of G_1 and G_2, the shortest path kernel is defined as:

$$k(G_1, G_2) = \frac{\sum_{v_1 \in V_1, v_2 \in V_2} k_{node}(v_1, v_2) + \sum_{e_1 \in E_1, e_2 \in E_2} k^{(1)}_{walk}(e_1, e_2)}{norm}$$

where k_{node} is a kernel for comparing two vertices, $k^{(1)}_{walk}$ a kernel on edge walks of length 1 and *norm* a normalization factor. Specifically:

$$k_{node}(v_1, v_2) = \begin{cases} 1 & \text{if } \ell(v_1) = \ell(v_2), \\ 0 & \text{otherwise} \end{cases}$$

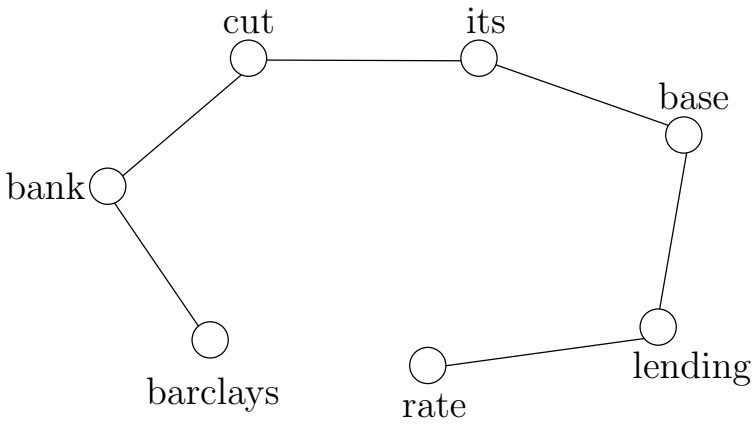
$$k^{(1)}_{walk}(e_1, e_2) = k_{node}(u_1, u_2) \times k_{edge}(e_1, e_2) \times k_{node}(v_1, v_2)$$

$$k_{edge}(e_1, e_2) = \begin{cases} \ell(e_1) \times \ell(e_2) & \text{if } e_1 \in E_1 \wedge e_2 \in E_2, \\ 0 & \text{otherwise} \end{cases}$$

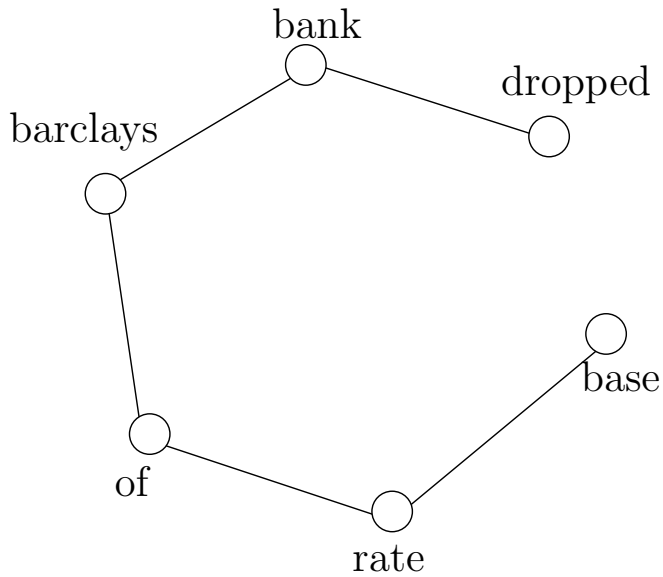
Example

d_1: "*barclays bank cut its base lending rate*"

d_2: "*base rate of barclays bank dropped*"



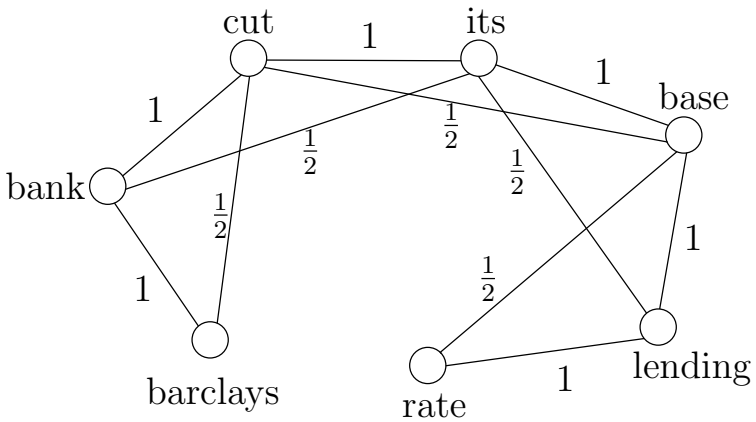
G_1



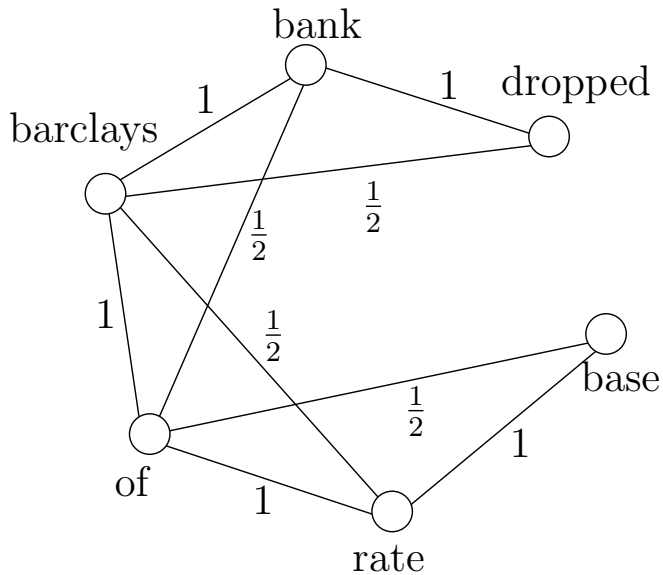
G_2

Example

SP-transformation ($d = 2$)



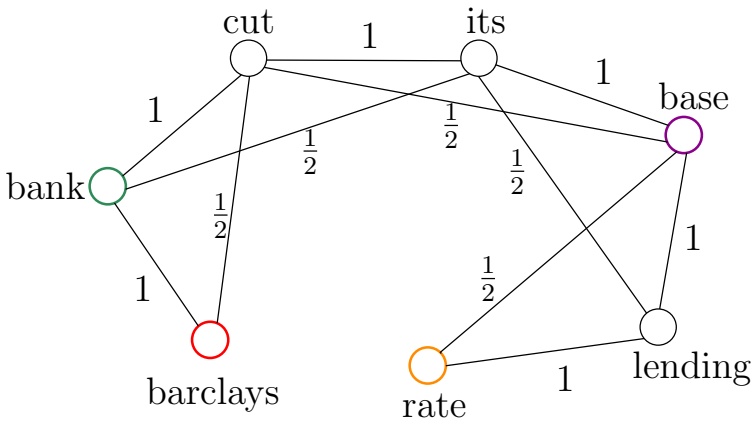
C_1



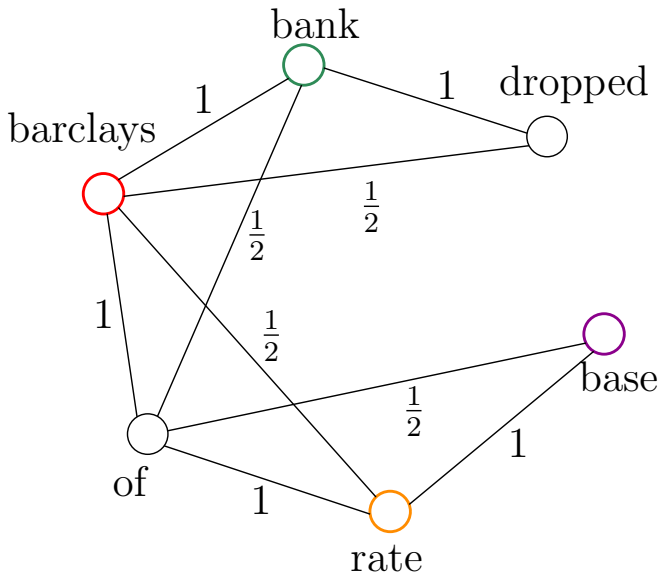
C_2

Example

$\sum_{v_1 \in V_1, v_2 \in V_2} k_{node}(v_1, v_2) = 4$



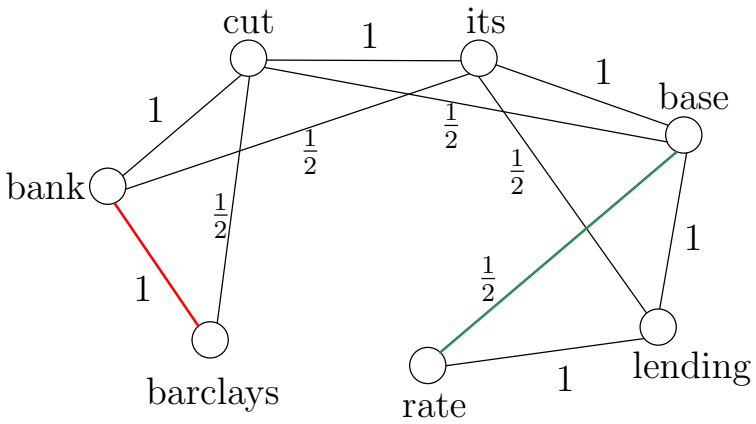
C_1



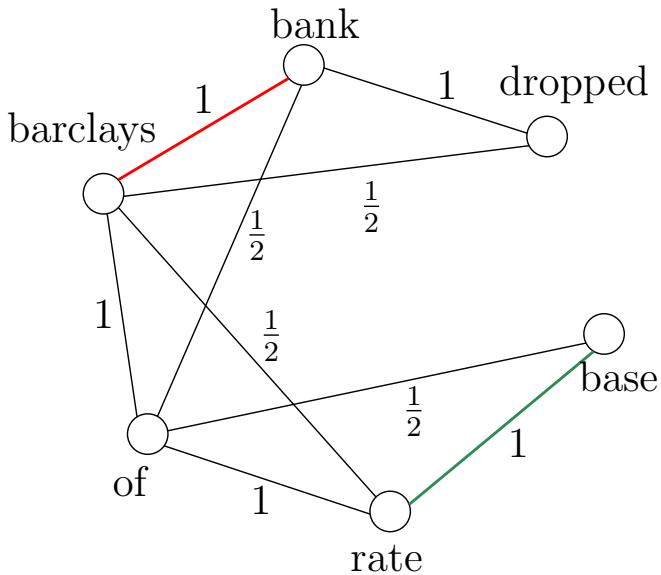
C_2

Example

$$\sum_{e_1 \in E_1, e_2 \in E_2} k^{(1)}_{walk}(e_1, e_2) = 1 + \frac{1}{2} = \frac{3}{2}$$



C_1

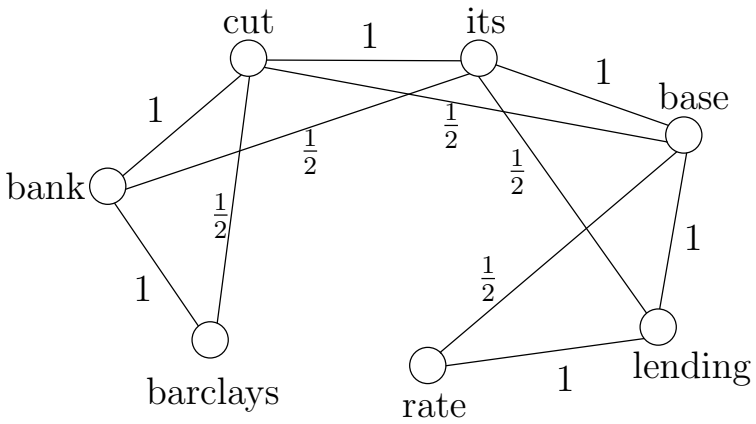


C_2

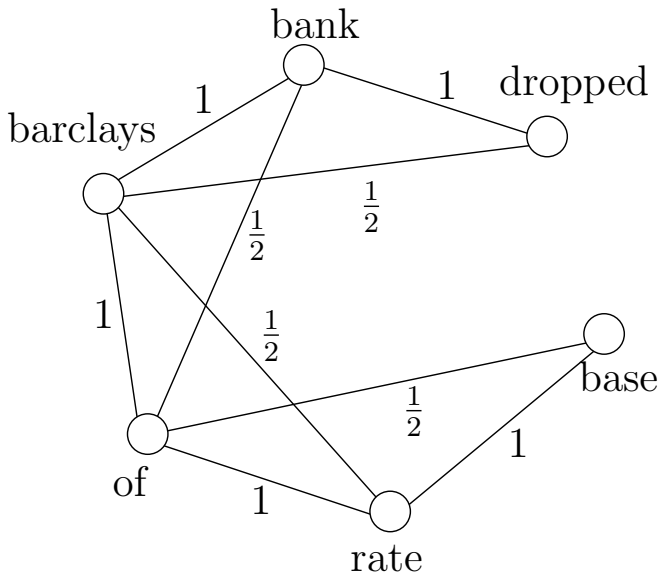
Example

$norm = 13.07$

$k(G_1, G_2) = \frac{4+\frac{3}{2}}{13.07} = 0.42$



C_1



C_2

Run Time Complexity

Standard kernel computation:

- All shortest paths from root: $\mathcal{O}(b^d)$ time (average branching factor b with breadth-first search)
- All shortest paths for n nodes: $\mathcal{O}(nb^d)$ time
- Compare all pairs of shortest paths from C_1 and C_2: $\mathcal{O}(n^4)$
- However, since each node has a unique label, we have to consider n^2 pairs of edges
- Hence, total complexity: $\mathcal{O}(n^2 + nb^d)$

Special case for $d = 1$:

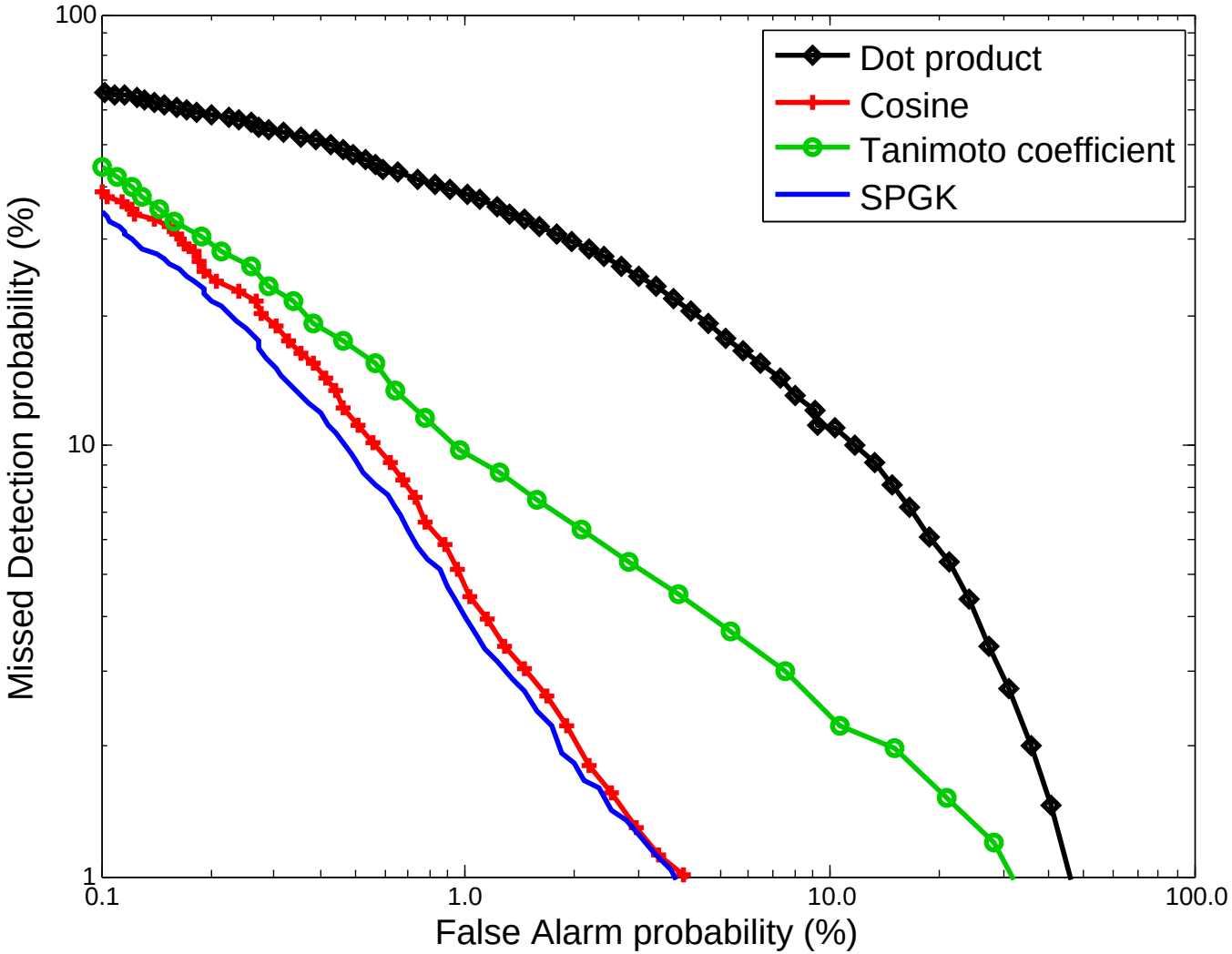
$$k(G_1, G_2) = \frac{\sum \mathbf{M}_1 \circ \mathbf{M}_2}{\|\mathbf{M}_1\|_F \times \|\mathbf{M}_2\|_F}$$

$\mathbf{M}_i = \mathbf{A}_i + \mathbf{D}_i$
$\mathbf{A}_i$: adjacency matrix of the SP-transformed graph
$\mathbf{D}_i$: diagonal matrix with diagonal entries set to 1 if corresponding term exists in corresponding document
$\mathcal{O}(n + m)$ time in the worst case scenario

Text Categorization

Method \ Dataset		WebKB		News		Subjectivity		Amazon		Polarity	
		Acc	F1	Acc	F1	Acc	F1	Acc	F1	Acc	F1
Dot product	$n=1$	90.26	89.23	81.10	77.64	89.92	89.92	91.88	91.88	76.27	76.26
	$n=2$	90.47	89.50	80.91	77.32	91.01	91.01	92.00	92.02	77.46	77.45
	$n=3$	90.26	89.17	80.72	77.10	90.90	90.90	91.81	91.85	77.41	77.40
	$n=4$	89.40	88.13	80.31	76.51	90.39	90.39	91.31	91.33	77.19	77.18
Cosine	$n=1$	92.48	91.88	81.17	77.66	90.03	90.02	94.00	94.00	76.70	76.69
	$n=2$	93.05	92.75	**81.49**	**77.97**	90.94	90.94	94.13	94.13	77.56	77.56
	$n=3$	92.98	92.59	80.97	77.38	90.99	90.99	94.19	94.18	77.65	77.65
	$n=4$	92.48	92.08	80.76	77.09	90.76	90.75	94.13	94.13	77.53	77.53
Tanimoto	$n=1$	90.62	89.83	81.55	78.15	90.94	90.93	92.25	92.26	77.49	77.48
	$n=2$	90.40	89.45	80.75	77.00	90.61	90.60	91.81	91.85	77.35	77.35
	$n=3$	92.41	91.80	79.80	75.75	90.21	90.20	93.44	93.47	76.48	76.48
	$n=4$	91.76	90.84	78.99	74.83	89.53	89.52	93.00	93.00	75.86	75.86
DCNN		89.18	87.99	79.91	76.15	90.26	90.26	91.81	91.81	73.26	73.26
CNN	static,rand	>1 day		77.57	73.37	87.16	87.15	88.81	88.82	71.50	71.50
	non-static,rand	>1 day		81.13	77.49	89.61	89.60	93.56	93.56	76.54	76.53
SPGK	$d=1$	93.27	92.78	81.04	77.49	**91.48**	**91.48**	94.00	94.01	77.76	77.75
	$d=2$	**93.70**	**93.36**	80.89	77.29	91.46	91.46	94.13	94.13	**77.89**	**77.88**
	$d=3$	92.91	92.33	80.78	77.03	91.37	91.37	94.44	94.44	77.61	77.60
	$d=4$	92.91	92.23	80.97	77.30	91.18	91.18	**94.63**	**94.63**	77.80	77.80

Story Link Detection



Graph Similarity with Graph Kernels - DaSciM contributions

New Kernels

- Matching Node Embeddings for Graph Similarity [AAAI 2017]
- Message Passing GKs [arXiv:1808.02510]
- Shortest-path graph kernels for document similarity – [ENMLP 2017] - applications to NLP

Kernel based Similarity / Embedding Frameworks

- Degeneracy framework for graph similarity [IJCAI 2018 - best paper award]
- Enhancing graph kernels via successive embeddings [CIKM 2018]
- Structural Node Embeddings using Graph Kernels [submitted to TKDE]

Software Library

- GraKel: A python library for graph kernels – scikit compatible
  https://github.com/ysig/GraKeL

GraKel: A python library for graph kernels

- Python library for graph similarity computations
- Contains practically all known graph kernels
- Compatible with scikit learn
- Open source - can be extended
- Project repository https://ysig.github.io/GraKeL/dev/

Graph Kernels - Experimental Comparison

Kernels	DATASETS			
	MUTAG	ENZYMES	NCI1	PTC-MR
VH	71.87 (± 1.83)	16.87 (± 1.56)	56.09 (± 0.35)	58.09 (± 0.62)
RW	82.24 (± 2.87)	12.90 (± 1.42)	TIMEOUT	51.26 (± 2.30)
SP	82.54 (± 1.00)	40.13 (± 1.34)	72.25 (± 0.28)	59.26 (± 2.34)
WL-VH	84.00 (± 1.25)	53.15 (± 1.22)	85.03 (± 0.20)	63.28 (± 1.34)
WL-SP	82.29 (± 1.93)	28.23 (± 1.00)	61.43 (± 0.32)	55.51 (± 1.68)
WL-PM	88.60 (± 0.95)	57.72 (± 0.84)	85.31 (± 0.42)	64.52 (± 1.36)
NH	87.74 (± 1.17)	43.43 (± 1.45)	74.81 (± 0.37)	60.50 (± 2.10)
NSPDK	82.46 (± 1.55)	41.97 (± 1.66)	74.36 (± 0.31)	60.04 (± 1.15)
ODD-STh	79.01 (± 2.04)	31.87 (± 1.35)	75.03 (± 0.45)	59.08 (± 1.85)
PM	84.72 (± 1.67)	42.67 (± 1.78)	73.11 (± 0.49)	57.99 (± 2.45)
GH	82.11 (± 2.13)	36.47 (± 2.13)	71.36 (± 0.13)	55.64 (± 2.03)
SM	84.04 (± 1.55)	35.68 (± 0.80)	TIMEOUT	57.91 (± 1.73)
PK	77.23 (± 1.22)	44.48 (± 1.63)	82.12 (± 0.22)	59.30 (± 1.24)
ML	86.11 (± 1.60)	53.08 (± 1.53)	79.40 (± 0.47)	59.95 (± 1.71)
CORE-WL	85.90 (± 1.44)	52.37 (± 1.29)	85.12 (± 0.21)	63.03 (± 1.67)
CORE-SP	85.13 (± 2.46)	41.55 (± 1.66)	73.87 (± 0.19)	58.21 (± 1.87)

Kernels	DATASETS			Avg. Rank
	D&D	PROTEINS	AIDS	
VH	74.83 (± 0.40)	70.93 (± 0.28)	79.78 (± 0.13)	13.7
RW	OUT-OF-MEM	69.31 (± 0.29)	79.52 (± 0.58)	15.0
SP	78.93 (± 0.53)	75.92 (± 0.35)	99.41 (± 0.12)	6.7
WL-VH	78.88 (± 0.46)	75.45 (± 0.33)	98.51 (± 0.05)	4.8
WL-SP	75.66 (± 0.42)	71.88 (± 0.22)	99.36 (± 0.02)	11.8
WL-PM	OUT-OF-MEM	75.63 (± 0.49)	99.37 (± 0.04)	2.1
NH	76.02 (± 0.94)	75.55 (± 1.00)	99.54 (± 0.02)	5.0
NSPDK	78.76 (± 0.56)	73.17 (± 0.76)	98.04 (± 0.20)	8.0
ODD-STh	75.82 (± 0.54)	70.49 (± 0.64)	90.75 (± 0.30)	11.4
PM	76.98 (± 0.84)	71.90 (± 0.79)	99.56 (± 0.08)	8.2
GH	TIMEOUT	74.19 (± 0.42)	99.57 (± 0.02)	9.6
SM	OUT-OF-MEM	OUT-OF-MEM	91.96 (± 0.18)	11.2
PK	78.43 (± 0.55)	72.71 (± 0.62)	96.51 (± 0.38)	8.4
ML	78.28 (± 0.99)	73.89 (± 0.93)	98.48 (± 0.12)	6.0
CORE-WL	78.91 (± 0.50)	75.46 (± 0.38)	98.70 (± 0.09)	4.1
CORE-SP	79.33 (± 0.65)	76.31 (± 0.40)	99.47 (± 0.05)	5.5

- Average classification accuracy (+/-standard deviation) on the 7 classification datasets containing node-labeled graphs.
- "Avg. Rank" column illustrates the averagerank of each kernel. The lower the average rank, the better the overall performance of thekernel

Graph Kernels: a Survey, G. Nikolentzos, M. Vazirgiannis, under submission

Graph Kernels - Experimental Comparison

Kernels	DATASETS			
	MUTAG	ENZYMES	NCI1	PTC-MR
VH	0.01s	0.04s	0.84s	0.02s
RW	1m 46.86s	4h 24m 16.26s	TIMEOUT	6m 41.20s
SP	0.92s	11.03s	1m 9.69s	1.52s
WL-VH	0.21s	3.81s	7m 5.33s	0.55s
WL-SP	7.02s	1m 27.07s	15m 29.50s	12.55s
WL-PM	3m 42.07s	1h 5m 37.26s	13h 31m 34.36s	11m 8.16s
NH	0.40s	11.17s	7m 4.54s	1.31s
NSPDK	4.05s	27.02s	6m 9.81s	7.66s
ODD-STh	1.54s	50.05s	46m 2.13s	4.03s
PM	2.59s	31.38s	37m 37.50s	11.35s
GH	24.70s	15m 38.33s	3h 45m 8.31s	1m 33.90s
SM	1m 57.25s	3h 25m 43.59s	TIMEOUT	4m 19.80s
PK	0.48s	12.05s	10m 27.83s	1.81s
ML	10m 3.15s	56m 43.76s	5h 30m 56.29s	19m 22.43s
CORE-WL	0.55s	12.52s	14m 30.56s	17m 2.27s
CORE-SP	2.69s	48.02s	3m 16.54s	3.97s

Kernels	DATASETS			Avg. Rank
	D&D	PROTEINS	AIDS	
VH	0.24s	0.10s	0.25s	1.0
RW	OUT-OF-MEM	51m 10.11s	1h 51m 56.47s	13.6
SP	55m 58.79s	1m 18.91s	13.93s	4.4
WL-VH	5m 52.96s	32.48s	40.49s	2.8
WL-SP	7h 27m 21.90s	8m 3.68s	1m 33.46s	10.1
WL-PM	OUT-OF-MEM	5h 37m 10.33s	5h 55m 20.37s	14.6
NH	6m 17.21s	41.81s	33.30s	3.5
NSPDK	4h 36m 28.97s	9m 9.80s	1m 12.31s	8.1
ODD-STh	27m 59.18s	4m 7.81s	2m 5.32s	8.7
PM	5m 48.51s	1m 26.82s	2m 48.04s	8.0
GH	TIMEOUT	3h 43m 1.54s	38m 51.78s	12.1
SM	OUT-OF-MEM	OUT-OF-MEM	4h 26m 46.71s	14.0
PK	9m 34.30s	51.20s	1m 43.62s	5.5
ML	3h 40m 30.72s	2h 20m 39.57s	1h 11m 58.23s	13.2
CORE-WL	17m 2.27s	1m 16.74s	54.79s	7.2
CORE-SP	5h 2m 39.71s	3m 31.97s	40.11s	7.2

- Average CPU running time for kernel matrix computation on the 7 classificationdatasets containing node-labeled graphs.
- "Avg. Rank" column illustrates averagerank of each kernel. The lower the average rank, the lower the overall running time of thekernel

Graph Kernels: a Survey, G. Nikolentzos, M. Vazirgiannis, under submission

Graph Similarity - with Deep Learning - Current work

Tasks: Embedding Nodes, Graphs, Sets with Deep Learning for node/graph classification, set comparison etc.

- Kernel graph convolutional neural networks [ICANN 2018]
- Learning Structural Node Representations on Directed Graphs [Complex Networks 2018]
- Graph Matching for learning Set representations [sbmtd ICML 2019]
- Learning Deep Set representations [submitted ICML2019]
- Classifying graphs as images with convolutional neural networks [arXiv:1708.02218]

Challenge: *How do Kernels compare to GNNs ?*

THANK YOU !

Acknowledgements
Dr. I. Nikolentzos, Dr. A. Tixier, Dr. P. Meladianos

http://www.lix.polytechnique.fr/dascim/

Software and data sets: http://www.lix.polytechnique.fr/dascim/software_datasets/