Graph Similarity and Classification © DaSciM

M. Vazirgiannis

DaSciM, LIX, École Polytechnique

February 11, 2018
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?
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.

Given a text, create a graph where

- vertices correspond 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
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 → 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 measures:

- 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 isomorphic to these graphs → malicious code inside the program

Gascon et al. “Structural detection of android malware using embedded call graphs”. In AI Sec’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

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

Graph classification very related to graph comparison

Example

\[ f(\circ, \bigcirc) + k-nn = \text{graph classification} \]

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

- It is equivalent to an inner product of the embeddings $\phi : \mathcal{X} \to \mathcal{H}$ 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
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.
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.
Definition (Complete Graph Kernel)

A graph kernel \( k(G_i, G_j) = \phi(G_i), \phi(G_j) \) is complete iff the transformation \( \phi \) 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 guarantee 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
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

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

Borgwardt and Kriegel. “Shortest-path kernels on graphs”. In ICDM’05
A large number of graph kernels compare substructures of graphs that are computable in polynomial time:

- walks
- shortest path lengths
- cyclic patterns

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

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

$G_1$

$G_2$
**First step:** Augment the labels of the vertices by the sorted set of labels of neighbouring vertices

\[ G_1 \quad G_2 \]
Iteration 1

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

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

Are the label sets of $G_1$ and $G_2$ identical?

$G_1$

$G_2$

Yes!!!

Continue to the next iteration
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:

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

![Graph $G_1$](image1.png)

![Graph $G_2$](image2.png)
Are the label sets of $G_1$ and $G_2$ identical?

No!!!

Graphs are not isomorphic
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:

$$k^h_{WL}(G_1, G_2) = k(G_1, G_2) + k(G^1_1, G^1_2) + k(G^2_1, G^2_2) + \ldots + k(G^h_1, G^h_2)$$

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

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
Counts matching pairs of labels in two graphs after each iteration

$G_1$  

$G_2$  

Weisfeiler-Lehman Subtree Kernel
Feature vector for a graph $G$:

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

$G_1$ and $G_2$ are graphs with nodes labeled 1, 2, 3, 4, and 5. The feature vectors are:

- $\phi(G_1) = \{1, 2, 1, 1, 1\}^T$
- $\phi(G_2) = \{1, 1, 2, 1, 1\}^T$

The similarity measure $k(G_1, G_2)$ is calculated as:

$$
k(G_1, G_2) = \phi(G_1)^T \phi(G_2) = 7
$$
**First step**: Augment the labels of the vertices by the sorted set of labels of neighbouring vertices.
Iteration 1

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

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

<table>
<thead>
<tr>
<th>Graph Kernel</th>
<th>Exp. $\phi$</th>
<th>Node Labels</th>
<th>Node Attributes</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vertex Histogram</td>
<td>$\checkmark$</td>
<td>$\checkmark$</td>
<td>$\times$</td>
<td>$R$-convolution</td>
</tr>
<tr>
<td>Edge Histogram</td>
<td>$\checkmark$</td>
<td>$\checkmark$</td>
<td>$\times$</td>
<td>$R$-convolution</td>
</tr>
<tr>
<td>Random Walk</td>
<td>$\times$</td>
<td>$\checkmark$</td>
<td>$\checkmark$</td>
<td>$R$-convolution</td>
</tr>
<tr>
<td>Subtree</td>
<td>$\times$</td>
<td>$\checkmark$</td>
<td>$\checkmark$</td>
<td>$R$-convolution</td>
</tr>
<tr>
<td>Cyclic Pattern</td>
<td>$\checkmark$</td>
<td>$\checkmark$</td>
<td>$\times$</td>
<td>Intersection</td>
</tr>
<tr>
<td>Shortest Path</td>
<td>$\times$</td>
<td>$\checkmark$</td>
<td>$\checkmark$</td>
<td>$R$-convolution</td>
</tr>
<tr>
<td>Graphlet</td>
<td>$\checkmark$</td>
<td>$\times$</td>
<td>$\times$</td>
<td>$R$-convolution</td>
</tr>
<tr>
<td>Weisfeiler-Lehman Subtree</td>
<td>$\checkmark$</td>
<td>$\checkmark$</td>
<td>$\times$</td>
<td>$R$-convolution</td>
</tr>
<tr>
<td>Neighborhood Hash</td>
<td>$\checkmark$</td>
<td>$\checkmark$</td>
<td>$\times$</td>
<td>Intersection</td>
</tr>
<tr>
<td>Neighborhood Subgraph Pairwise Distance</td>
<td>$\checkmark$</td>
<td>$\checkmark$</td>
<td>$\times$</td>
<td>$R$-convolution</td>
</tr>
<tr>
<td>Lovász $\vartheta$</td>
<td>$\checkmark$</td>
<td>$\times$</td>
<td>$\times$</td>
<td>$R$-convolution</td>
</tr>
<tr>
<td>SVM-$\vartheta$</td>
<td>$\checkmark$</td>
<td>$\times$</td>
<td>$\times$</td>
<td>$R$-convolution</td>
</tr>
<tr>
<td>Ordered Decomposition DAGs</td>
<td>$\checkmark$</td>
<td>$\checkmark$</td>
<td>$\times$</td>
<td>$R$-convolution</td>
</tr>
<tr>
<td>Pyramid Match</td>
<td>$\times$</td>
<td>$\checkmark$</td>
<td>$\times$</td>
<td>Assignment</td>
</tr>
<tr>
<td>Weisfeiler-Lehman Optimal Assignment</td>
<td>$\times$</td>
<td>$\checkmark$</td>
<td>$\times$</td>
<td>Assignment</td>
</tr>
<tr>
<td>Subgraph Matching</td>
<td>$\times$</td>
<td>$\checkmark$</td>
<td>$\checkmark$</td>
<td>$R$-convolution</td>
</tr>
<tr>
<td>GraphHopper</td>
<td>$\times$</td>
<td>$\checkmark$</td>
<td>$\checkmark$</td>
<td>$R$-convolution</td>
</tr>
<tr>
<td>Graph Invariant Kernels</td>
<td>$\times$</td>
<td>$\checkmark$</td>
<td>$\checkmark$</td>
<td>$R$-convolution</td>
</tr>
<tr>
<td>Propagation</td>
<td>$\checkmark$</td>
<td>$\checkmark$</td>
<td>$\checkmark$</td>
<td>$R$-convolution</td>
</tr>
<tr>
<td>Multiscale Laplacian</td>
<td>$\times$</td>
<td>$\checkmark$</td>
<td>$\checkmark$</td>
<td>$R$-convolution</td>
</tr>
</tbody>
</table>

Summary of selected graph kernels regarding computation by explicit feature mapping (Exp.$\phi$), 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
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:
  - Earth Mover’s Distance [Rubner et al., IJCV ’00]
  - Pyramid Match Kernel [Grauman and Darrell, JMLR ’07]
Node embeddings: represent nodes as points in a vector space

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

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
  \( \rightarrow \) There is no canonical ordering for the nodes of a graph
Earth Mover’s Distance (EMD): minimum “travel cost” from $G_1 = (V_1, E_1)$ to $G_2 = (V_2, E_2)$:

\[
\min_{T \geq 0} \sum_{i=1}^{n_1} \sum_{j=1}^{n_2} T_{ij} \| v_i - u_j \|_2
\]

subject to

\[
\sum_{i=1}^{n_1} T_{ij} = \frac{1}{n_2} \quad \forall j \in \{1, \ldots, n_2\}
\]

\[
\sum_{j=1}^{n_2} T_{ij} = \frac{1}{n_1} \quad \forall i \in \{1, \ldots, n_1\}
\]

where $v_i, u_j$ are the embeddings of nodes $v_i \in V_1, u_j \in V_2$ and $T$ 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: $\mathcal{O}(n^3 \log n)$

**However**, emerging similarity matrices not necessarily positive semidefinite.

Complexity: $\mathcal{O}(n^3 \log n)$

**Figure:** Minimum cumulative distance between $G_1, G_2$
Earth Mover’s Distance (EMD): minimum “travel cost” from $G_1 = (V_1, E_1)$ to $G_2 = (V_2, E_2)$:

$$
\min_{T \geq 0} \sum_{i=1}^{n_1} \sum_{j=1}^{n_2} T_{ij} \|v_i - u_j\|_2
$$

subject to

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

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

where $v_i, u_j$ are the embeddings of nodes $v_i \in V_1, u_j \in V_2$ and $T$ 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: $O(n^3 \log n)$

However, emerging similarity matrices not necessarily positive semidefinite

Complexity: $O(n^3 \log n)$
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 $l$:

$$I(H^l_{G_1}, H^l_{G_2}) = \sum_{i=1}^{2^ld} \min (H^l_{G_1}(i), H^l_{G_2}(i))$$

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

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

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

- $I(H^{l+1}_{G_1}, H^{l+1}_{G_2})$)

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

Complexity: $O(dnL)$
### 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 $\vartheta$ kernel (Lo-$\vartheta$)
- optimal assignment between node embeddings (OA)
## Experimental evaluation - missing table

<table>
<thead>
<tr>
<th>Method</th>
<th>Datasets</th>
<th>MUTAG</th>
<th>ENZYMES</th>
<th>NCI1</th>
<th>NCI109</th>
<th>PTC-MR</th>
<th>D&amp;D</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Without node labels</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SP</td>
<td></td>
<td>82.22 (± 1.14) 0.16&quot;</td>
<td>28.17 (± 0.64) 1.26&quot;</td>
<td>62.02 (± 0.17) 7.55&quot;</td>
<td>61.41 (± 0.32) 7.32&quot;</td>
<td>56.18 (± 0.56) 0.47&quot;</td>
<td>&gt; 1 day</td>
</tr>
<tr>
<td>RW</td>
<td></td>
<td>77.78 (± 0.98) 17.01&quot;</td>
<td>20.17 (± 0.83) 4' 46&quot;</td>
<td>56.89 (± 0.24) 2h 40' 15&quot;</td>
<td>56.13 (± 0.31) 2h 44' 25&quot;</td>
<td>56.18 (± 1.12) 1' 23&quot;</td>
<td>58.71 (± 0.43) 6h 45' 1&quot;</td>
</tr>
<tr>
<td>GR</td>
<td></td>
<td>66.11 (± 1.31) 0.07&quot;</td>
<td>18.16 (± 0.47) 4.32&quot;</td>
<td>47.37 (± 0.15) 4.42&quot;</td>
<td>48.39 (± 0.18) 4.37&quot;</td>
<td>57.05 (± 0.83) 0.14&quot;</td>
<td>63.67 (± 0.57) 1' 04&quot;</td>
</tr>
<tr>
<td>Lo-ψ</td>
<td></td>
<td>82.78 (± 0.89) 15' 26&quot;</td>
<td>26.33 (± 0.44) 2h 11' 31&quot;</td>
<td>62.68 (± 0.24) 17h 41' 57&quot;</td>
<td>62.42 (± 0.27) 17h 45' 27&quot;</td>
<td>55.00 (± 0.62) 1h 9' 58&quot;</td>
<td>&gt; 1 day</td>
</tr>
<tr>
<td>OA</td>
<td></td>
<td>79.44 (± 1.08) 6.56&quot;</td>
<td>36.33 (± 0.71) 3' 19&quot;</td>
<td>67.81 (± 0.18) 2h 21' 57&quot;</td>
<td>66.94 (± 0.21) 2h 20' 13&quot;</td>
<td>56.17 (± 0.95) 42.94&quot;</td>
<td>&gt; 1 day</td>
</tr>
<tr>
<td>EMD</td>
<td></td>
<td>86.11 (± 0.84) 4.5&quot;</td>
<td>36.83 (± 0.78) 1' 57&quot;</td>
<td>72.65 (± 0.24) 1h 11' 31&quot;</td>
<td>71.70 (± 0.16) 1h 10' 37&quot;</td>
<td>57.65 (± 0.59) 24.28&quot;</td>
<td>&gt; 1 day</td>
</tr>
<tr>
<td>PM</td>
<td></td>
<td>85.55 (± 0.63) 1.26&quot;</td>
<td>28.17 (± 0.37) 8.07&quot;</td>
<td>69.73 (± 0.11) 3' 19&quot;</td>
<td>68.37 (± 0.14) 3' 17&quot;</td>
<td>59.41 (± 0.68) 3.51&quot;</td>
<td>75.55 (± 0.62) 41.23&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>With node labels</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SP</td>
<td></td>
<td>87.78 (± 0.44) 0.16&quot;</td>
<td>41.00 (± 0.26) 1.42&quot;</td>
<td>72.85 (± 0.18) 9.86&quot;</td>
<td>73.20 (± 0.16) 9.46&quot;</td>
<td>60.00 (± 0.72) 0.48&quot;</td>
<td>&gt; 1 day</td>
</tr>
<tr>
<td>RW</td>
<td></td>
<td>81.11 (± 1.23) 3' 51&quot;</td>
<td>19.33 (± 0.62) 12' 58&quot;</td>
<td>&gt; 1 day</td>
<td>&gt; 1 day</td>
<td>57.06 (± 0.86) 1h 33' 32&quot;</td>
<td>&gt; 1 day</td>
</tr>
<tr>
<td>GR</td>
<td></td>
<td>71.67 (± 0.81) 0.79&quot;</td>
<td>32.00 (± 0.46) 17.77&quot;</td>
<td>65.52 (± 0.35) 40.03&quot;</td>
<td>66.70 (± 0.15) 41.36&quot;</td>
<td>59.41 (± 0.94) 2.21&quot;</td>
<td>79.40 (± 0.39) 15' 13&quot;</td>
</tr>
<tr>
<td>OA</td>
<td></td>
<td>82.22 (± 0.68) 6.03&quot;</td>
<td>43.16 (± 0.56) 1' 57&quot;</td>
<td>69.53 (± 0.20) 1h 33' 35&quot;</td>
<td>68.76 (± 0.15) 1h 32' 51&quot;</td>
<td>58.23 (± 0.82) 28.28&quot;</td>
<td>77.52 (± 0.43) 7h 45' 5&quot;</td>
</tr>
<tr>
<td>WL ST</td>
<td></td>
<td>83.33 (± 0.86) 2.91&quot;</td>
<td>52.16 (± 0.61) 17.12&quot;</td>
<td>84.72 (± 0.16) 1' 57&quot;</td>
<td>84.26 (± 0.22) 2' 00&quot;</td>
<td>57.64 (± 0.56) 7.35&quot;</td>
<td>76.83 (± 0.49) 5' 06&quot;</td>
</tr>
<tr>
<td>WL SP</td>
<td></td>
<td>84.55 (± 1.08) 1.27&quot;</td>
<td>61.00 (± 0.69) 12.23&quot;</td>
<td>84.64 (± 0.21) 1' 15&quot;</td>
<td>84.29 (± 0.17) 1' 13&quot;</td>
<td>56.76 (± 0.74) 2.81&quot;</td>
<td>&gt; 1 day</td>
</tr>
<tr>
<td>WL OA</td>
<td></td>
<td>85.55 (± 1.02) 27.46&quot;</td>
<td>53.66 (± 0.72) 7' 2&quot;</td>
<td>85.35 (± 0.18) 5h 7' 57&quot;</td>
<td>84.51 (± 0.14) 5h 4' 5&quot;</td>
<td>59.70 (± 1.01) 2' 3&quot;</td>
<td>&gt; 1 day</td>
</tr>
<tr>
<td>EMD</td>
<td></td>
<td>89.44 (± 0.76) 5.61&quot;</td>
<td>43.17 (± 0.48) 1' 15&quot;</td>
<td>76.76 (± 0.17) 1h 15' 44&quot;</td>
<td>73.88 (± 0.18) 1h 16' 10&quot;</td>
<td>58.82 (± 0.83) 25.79&quot;</td>
<td>&gt; 1 day</td>
</tr>
<tr>
<td>PM</td>
<td></td>
<td>86.67 (± 0.60) 3.52&quot;</td>
<td>40.33 (± 0.34) 21.50&quot;</td>
<td>72.91 (± 0.53) 5' 48&quot;</td>
<td>71.97 (± 0.15) 5' 50&quot;</td>
<td>60.22 (± 0.86) 9.58&quot;</td>
<td>77.78 (± 0.48) 4' 30&quot;</td>
</tr>
<tr>
<td>WL PM</td>
<td></td>
<td>87.77 (± 0.81) 9.20&quot;</td>
<td>55.55 (± 0.56) 1' 25&quot;</td>
<td>86.40 (± 0.20) 50' 35&quot;</td>
<td>85.34 (± 0.23) 50' 42&quot;</td>
<td>61.41 (± 0.81) 29.33&quot;</td>
<td>78.63 (± 0.26) 16' 15&quot;</td>
</tr>
</tbody>
</table>

**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 $\varphi$ kernel (Lo-$\varphi$), 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

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 non-empty $k$-core subgraph
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 $\delta_{\min}^*$ 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, $\ldots$, $\delta_{\min}^*$-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

\[ k(G, G') = k(C_0, C_0') \]
Degeneracy Framework for Graph Comparison - Example

\[ G \quad G' \]

\[ k \]
Degeneracy Framework for Graph Comparison - Example

\[ k_c(G, G') = k(C_0, C_0') \]
Degeneracy Framework for Graph Comparison - Example

\[ k_c(G, G') = k(C_0, C_0') + k(C_1, C_1') \]
Degeneracy Framework for Graph Comparison - Example

\[ k_c(G, G') = k(C_0, C'_0) + k(C_1, C'_1) + k(C_2, C'_2) \]
Degeneracy Framework for Graph Comparison - Example

\[ k_c(G, G') = k(C_0, C'_0) + k(C_1, C'_1) + k(C_2, C'_2) + k(C_3, C'_3) \]
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 $\delta_{\min} O_A$ time, where $O_A$ be the time complexity of algorithm $A$

The degeneracy of a graph is upper bounded by the largest eigenvalue of its adjacency matrix $\lambda_1$

In most real-world graphs, $\lambda_1 \ll n$, then $\delta_{\min} \ll n$, hence time complexity not prohibitive
$k$-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

<table>
<thead>
<tr>
<th>Dataset</th>
<th>MUTAG</th>
<th>ENZYMES</th>
<th>NCI1</th>
<th>PTC-MR</th>
<th>D&amp;D</th>
<th>IMDB BINARY</th>
<th>IMDB MULTI</th>
<th>REDDIT BINARY</th>
<th>REDDIT MULTI-5K</th>
<th>REDDIT MULTI-12K</th>
<th>COLLAB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max # vertices</td>
<td>28</td>
<td>126</td>
<td>111</td>
<td>111</td>
<td>109</td>
<td>620</td>
<td>5748</td>
<td>136</td>
<td>89</td>
<td>3782</td>
<td>3648</td>
</tr>
<tr>
<td>Min # vertices</td>
<td>10</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>2</td>
<td>4</td>
<td>30</td>
<td>12</td>
<td>7</td>
<td>6</td>
<td>22</td>
</tr>
<tr>
<td>Average # vertices</td>
<td>17.93</td>
<td>32.63</td>
<td>29.87</td>
<td>29.68</td>
<td>25.56</td>
<td>39.05</td>
<td>284.32</td>
<td>19.77</td>
<td>13.00</td>
<td>429.61</td>
<td>508.50</td>
</tr>
<tr>
<td>Max # edges</td>
<td>33</td>
<td>149</td>
<td>119</td>
<td>119</td>
<td>108</td>
<td>1049</td>
<td>14267</td>
<td>1249</td>
<td>1467</td>
<td>4071</td>
<td>4783</td>
</tr>
<tr>
<td>Min # edges</td>
<td>10</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>5</td>
<td>63</td>
<td>26</td>
<td>12</td>
<td>4</td>
<td>21</td>
</tr>
<tr>
<td>Average # edges</td>
<td>19.79</td>
<td>62.14</td>
<td>32.30</td>
<td>32.13</td>
<td>25.96</td>
<td>72.81</td>
<td>715.66</td>
<td>96.53</td>
<td>65.93</td>
<td>497.75</td>
<td>594.87</td>
</tr>
<tr>
<td># graphs</td>
<td>188</td>
<td>600</td>
<td>4110</td>
<td>4127</td>
<td>344</td>
<td>1113</td>
<td>1178</td>
<td>1000</td>
<td>1500</td>
<td>2000</td>
<td>4999</td>
</tr>
<tr>
<td># classes</td>
<td>2</td>
<td>6</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>5</td>
</tr>
</tbody>
</table>

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
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 embeds 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

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

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

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

- 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 $\delta^*$
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
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 $\mathcal{G}$
  - $\leftrightarrow$ There exists a Hilbert space $\mathcal{H}_1$ and a mapping $\phi_1$:
    \[ k_1(G, G') = \langle \phi_1(G), \phi_1(G') \rangle_{\mathcal{H}_1} \text{ for all } G, G' \in \mathcal{G} \]

- Define a kernel $k_2$ on feature space $\mathcal{H}_1$
  - $\leftrightarrow$ There exists a Hilbert space $\mathcal{H}_2$ and a mapping $\phi_2$ such that
    \[ k_2(\phi_1(G), \phi_1(G')) = \langle \phi_2(\phi_1(G)), \phi_2(\phi_1(G')) \rangle_{\mathcal{H}_2} \text{ for all } \phi_1(G), \phi_1(G') \in \mathcal{H}_1 \]

- Define a new kernel $k_3$ on feature space $\mathcal{H}_2$
  - $\leftrightarrow$ There exists a Hilbert space $\mathcal{H}_3$ and a mapping $\phi_3$ such that
    \[ k_3(\phi_2(\phi_1(G)), \phi_2(\phi_1(G')))) = \langle \phi_3(\phi_2(\phi_1(G))), \phi_3(\phi_2(\phi_1(G')))) \rangle_{\mathcal{H}_3} \text{ for all } \phi_2(\phi_1(G)), \phi_2(\phi_1(G')) \in \mathcal{H}_2 \]

      .......

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

Problem: Usually $x$ and $y$ not computed explicitly. How to apply Embedding 2?

↩ Use an implicit computation scheme
Proposed Instances of Kernels

The two kernels for vector data can be computed as:

1. **Polynomial kernel**: \( k_P(x, y) = (\langle x, y \rangle)^d = (k(x, y))^d, \quad d \in \mathbb{N} \)

2. **Gaussian kernel**: \( k_G(x, y) = \exp \left( - \frac{k(x, x) - 2k(x, y) + k(y, y)}{2\sigma^2} \right), \quad \sigma > 0 \)

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 → 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

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.

<table>
<thead>
<tr>
<th>Method</th>
<th>Dataset</th>
<th>MUTAG</th>
<th>ENZYMES</th>
<th>AIDS</th>
<th>NCI1</th>
<th>PTC5-MR</th>
<th>D&amp;D</th>
</tr>
</thead>
<tbody>
<tr>
<td>linear</td>
<td>BINARY</td>
<td>86.46 (± 1.47)</td>
<td>42.00 (± 0.96)</td>
<td>99.34 (± 0.07)</td>
<td>73.66 (± 0.36)</td>
<td>59.21 (± 1.87)</td>
<td>77.96 (± 0.58)</td>
</tr>
<tr>
<td>polynomial</td>
<td>IMDB BINARY</td>
<td>84.18 (± 1.89)</td>
<td>53.15 (± 1.03)</td>
<td>99.53 (± 0.03)</td>
<td><strong>77.89</strong> (± 0.25)</td>
<td>58.20 (± 2.47)</td>
<td><strong>80.95</strong> (± 0.53)</td>
</tr>
<tr>
<td>gaussian</td>
<td>IMDB MULTI</td>
<td>87.94 (± 1.02)</td>
<td><strong>50.52</strong> (± 1.34)</td>
<td><strong>99.64</strong> (± 0.01)</td>
<td><strong>77.80</strong> (± 0.36)</td>
<td>58.52 (± 1.95)</td>
<td><strong>80.29</strong> (± 0.42)</td>
</tr>
<tr>
<td>linear</td>
<td>REDDIT BINARY</td>
<td>80.30 (± 1.43)</td>
<td>50.65 (± 1.47)</td>
<td>98.05 (± 0.10)</td>
<td>84.68 (± 1.15)</td>
<td>61.62 (± 1.16)</td>
<td>79.25 (± 0.31)</td>
</tr>
<tr>
<td>polynomial</td>
<td>REDDIT MULTI 5K</td>
<td><strong>82.92</strong> (± 0.90)</td>
<td><strong>53.73</strong> (± 1.37)</td>
<td><strong>98.48</strong> (± 0.05)</td>
<td><strong>85.63</strong> (± 1.16)</td>
<td><strong>65.26</strong> (± 1.39)</td>
<td>79.58 (± 0.29)</td>
</tr>
<tr>
<td>gaussian</td>
<td>REDDIT MULTI 12K</td>
<td><strong>85.55</strong> (± 0.96)</td>
<td><strong>54.80</strong> (± 0.82)</td>
<td><strong>99.21</strong> (± 0.07)</td>
<td><strong>86.17</strong> (± 0.17)</td>
<td>59.81 (± 1.34)</td>
<td>77.91 (± 0.33)</td>
</tr>
<tr>
<td>linear</td>
<td>COLLAB</td>
<td>85.66 (± 1.18)</td>
<td>40.66 (± 0.82)</td>
<td>99.66 (± 0.02)</td>
<td>69.71 (± 0.73)</td>
<td>58.17 (± 1.86)</td>
<td>77.05 (± 0.96)</td>
</tr>
<tr>
<td>polynomial</td>
<td>IMDB BINARY</td>
<td>87.12 (± 1.51)</td>
<td><strong>48.43</strong> (± 1.08)</td>
<td>99.63 (± 0.03)</td>
<td><strong>77.35</strong> (± 0.32)</td>
<td><strong>61.19</strong> (± 1.43)</td>
<td><strong>78.74</strong> (± 0.59)</td>
</tr>
<tr>
<td>gaussian</td>
<td>IMDB MULTI</td>
<td><strong>87.71</strong> (± 1.39)</td>
<td><strong>48.32</strong> (± 0.88)</td>
<td>99.69 (± 0.03)</td>
<td><strong>76.84</strong> (± 0.26)</td>
<td><strong>61.47</strong> (± 1.97)</td>
<td>77.49 (± 0.58)</td>
</tr>
</tbody>
</table>
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

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

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
</table>

- However, BOW representation disregards word order!!!
- 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.
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.
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

Hence, document similarity problem → 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 \((d = 2)\)

\[ G \quad \rightarrow \quad C \]
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:

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

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

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

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

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

\[ d_1: \text{“barclays bank cut its base lending rate”} \]

\[ d_2: \text{“base rate of barclays bank dropped”} \]
Example

SP-transformation \( (d = 2) \)

\[ C_1 \]

\[ C_2 \]
\[ \sum_{v_1 \in V_1, v_2 \in V_2} k_{\text{node}}(v_1, v_2) = 4 \]
\[ \sum_{e_1 \in E_1, e_2 \in E_2} k^{(1)}_{\text{walk}}(e_1, e_2) = 1 + \frac{1}{2} = \frac{3}{2} \]
Example

\[ \text{norm} = 13.07 \]

\[ k(G_1, G_2) = \frac{4 + \frac{3}{2}}{13.07} = 0.42 \]
Run Time Complexity

Standard kernel computation:

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

Special case for $d = 1$:

$$k(G_1, G_2) = \sum M_1 \circ M_2 \over \|M_1\|_F \times \|M_2\|_F$$

$M_i = A_i + D_i$

$A_i$: adjacency matrix of the SP-transformed graph

$D_i$: diagonal matrix with diagonal entries set to 1 if corresponding term exists in corresponding document

$O(n + m)$ time in the worst case scenario
<table>
<thead>
<tr>
<th>Method</th>
<th>Dataset</th>
<th>WebKB</th>
<th>News</th>
<th>Subjectivity</th>
<th>Amazon</th>
<th>Polarity</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Acc</td>
<td>F1</td>
<td>Acc</td>
<td>F1</td>
<td></td>
</tr>
<tr>
<td>Dot product</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>n = 1</td>
<td>90.26</td>
<td>89.23</td>
<td>81.10</td>
<td>77.64</td>
<td>89.92</td>
</tr>
<tr>
<td></td>
<td>n = 2</td>
<td>90.47</td>
<td>89.50</td>
<td>80.91</td>
<td>77.32</td>
<td>91.01</td>
</tr>
<tr>
<td></td>
<td>n = 3</td>
<td>90.26</td>
<td>89.17</td>
<td>80.72</td>
<td>77.10</td>
<td>90.90</td>
</tr>
<tr>
<td></td>
<td>n = 4</td>
<td>89.40</td>
<td>88.13</td>
<td>80.31</td>
<td>76.51</td>
<td>90.39</td>
</tr>
<tr>
<td>Cosine</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>n = 1</td>
<td>92.48</td>
<td>91.88</td>
<td>81.17</td>
<td>77.66</td>
<td>90.03</td>
</tr>
<tr>
<td></td>
<td>n = 2</td>
<td>93.05</td>
<td>92.75</td>
<td>81.49</td>
<td>77.97</td>
<td>90.94</td>
</tr>
<tr>
<td></td>
<td>n = 3</td>
<td>92.98</td>
<td>92.59</td>
<td>80.97</td>
<td>77.38</td>
<td>90.99</td>
</tr>
<tr>
<td></td>
<td>n = 4</td>
<td>92.48</td>
<td>92.08</td>
<td>80.76</td>
<td>77.09</td>
<td>90.76</td>
</tr>
<tr>
<td>Tanimoto</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>n = 1</td>
<td>90.62</td>
<td>89.83</td>
<td>81.55</td>
<td>78.15</td>
<td>90.94</td>
</tr>
<tr>
<td></td>
<td>n = 2</td>
<td>90.40</td>
<td>89.45</td>
<td>80.75</td>
<td>77.00</td>
<td>90.61</td>
</tr>
<tr>
<td></td>
<td>n = 3</td>
<td>92.41</td>
<td>91.80</td>
<td>79.80</td>
<td>75.75</td>
<td>90.21</td>
</tr>
<tr>
<td></td>
<td>n = 4</td>
<td>91.76</td>
<td>90.84</td>
<td>78.99</td>
<td>74.83</td>
<td>89.53</td>
</tr>
<tr>
<td>DCNN</td>
<td></td>
<td>89.18</td>
<td>87.99</td>
<td>79.91</td>
<td>76.15</td>
<td>90.26</td>
</tr>
<tr>
<td>CNN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>static, rand</td>
<td>&gt; 1 day</td>
<td>77.57</td>
<td>73.37</td>
<td>87.16</td>
<td>87.15</td>
</tr>
<tr>
<td></td>
<td>non-static, rand</td>
<td>&gt; 1 day</td>
<td>81.13</td>
<td>77.49</td>
<td>89.61</td>
<td>89.60</td>
</tr>
<tr>
<td>SPGK</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>d = 1</td>
<td>93.27</td>
<td>92.78</td>
<td>81.04</td>
<td>77.49</td>
<td>91.48</td>
</tr>
<tr>
<td></td>
<td>d = 2</td>
<td>93.70</td>
<td>93.36</td>
<td>80.89</td>
<td>77.29</td>
<td>91.46</td>
</tr>
<tr>
<td></td>
<td>d = 3</td>
<td>92.91</td>
<td>92.33</td>
<td>80.78</td>
<td>77.03</td>
<td>91.37</td>
</tr>
<tr>
<td></td>
<td>d = 4</td>
<td>92.91</td>
<td>92.23</td>
<td>80.97</td>
<td>77.30</td>
<td>91.18</td>
</tr>
</tbody>
</table>
Story Link Detection

False Alarm probability (%)
Missed Detection probability (%)

Dot product
Cosine
Tanimoto coefficient
SPGK
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

<table>
<thead>
<tr>
<th>Kernels</th>
<th>DATASETS</th>
<th>MUTAG</th>
<th>ENZYMES</th>
<th>NCI1</th>
<th>PTC-MR</th>
</tr>
</thead>
<tbody>
<tr>
<td>VH</td>
<td></td>
<td>71.87 (± 1.83)</td>
<td>16.87 (± 1.50)</td>
<td>56.09 (± 0.35)</td>
<td>58.09 (± 0.62)</td>
</tr>
<tr>
<td>RW</td>
<td></td>
<td>82.24 (± 2.87)</td>
<td>12.90 (± 1.42)</td>
<td>TIMEOUT</td>
<td>51.26 (± 2.30)</td>
</tr>
<tr>
<td>SP</td>
<td></td>
<td>82.54 (± 1.00)</td>
<td>40.13 (± 1.34)</td>
<td>72.25 (± 0.28)</td>
<td>59.26 (± 2.34)</td>
</tr>
<tr>
<td>WL-VH</td>
<td></td>
<td>84.00 (± 1.25)</td>
<td>53.15 (± 1.22)</td>
<td>85.03 (± 0.20)</td>
<td>63.28 (± 1.34)</td>
</tr>
<tr>
<td>WL-SP</td>
<td></td>
<td>82.29 (± 1.03)</td>
<td>28.23 (± 1.00)</td>
<td>61.43 (± 0.32)</td>
<td>55.51 (± 1.68)</td>
</tr>
<tr>
<td>WL-PM</td>
<td></td>
<td>88.60 (± 0.95)</td>
<td>57.72 (± 0.84)</td>
<td>85.31 (± 0.42)</td>
<td>64.52 (± 1.36)</td>
</tr>
<tr>
<td>NH</td>
<td></td>
<td>87.74 (± 1.17)</td>
<td>43.43 (± 1.45)</td>
<td>74.81 (± 0.37)</td>
<td>60.50 (± 2.10)</td>
</tr>
<tr>
<td>NSPD-K</td>
<td></td>
<td>82.46 (± 1.55)</td>
<td>41.97 (± 1.66)</td>
<td>74.36 (± 0.31)</td>
<td>60.04 (± 1.15)</td>
</tr>
<tr>
<td>ODD-STh</td>
<td></td>
<td>79.01 (± 2.04)</td>
<td>31.87 (± 1.35)</td>
<td>75.03 (± 0.45)</td>
<td>59.08 (± 1.85)</td>
</tr>
<tr>
<td>PM</td>
<td></td>
<td>84.72 (± 1.67)</td>
<td>42.67 (± 1.78)</td>
<td>73.11 (± 0.49)</td>
<td>57.99 (± 2.45)</td>
</tr>
<tr>
<td>GH</td>
<td></td>
<td>82.11 (± 2.13)</td>
<td>36.47 (± 2.13)</td>
<td>71.36 (± 0.13)</td>
<td>55.64 (± 2.03)</td>
</tr>
<tr>
<td>SM</td>
<td></td>
<td>84.04 (± 1.55)</td>
<td>35.68 (± 0.80)</td>
<td>TIMEOUT</td>
<td>57.91 (± 1.73)</td>
</tr>
<tr>
<td>PK</td>
<td></td>
<td>77.23 (± 1.22)</td>
<td>44.48 (± 1.63)</td>
<td>82.12 (± 0.22)</td>
<td>59.30 (± 1.24)</td>
</tr>
<tr>
<td>ML</td>
<td></td>
<td>86.11 (± 1.60)</td>
<td>53.08 (± 1.53)</td>
<td>79.40 (± 0.47)</td>
<td>59.95 (± 1.71)</td>
</tr>
<tr>
<td>CORE-WL</td>
<td></td>
<td>85.90 (± 1.44)</td>
<td>52.37 (± 1.29)</td>
<td>85.12 (± 0.21)</td>
<td>63.03 (± 1.67)</td>
</tr>
<tr>
<td>CORE-SP</td>
<td></td>
<td>85.13 (± 2.46)</td>
<td>41.55 (± 1.66)</td>
<td>73.87 (± 0.19)</td>
<td>58.21 (± 1.87)</td>
</tr>
</tbody>
</table>

- 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 the kernel.

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

<table>
<thead>
<tr>
<th>Kernels</th>
<th>DATASETS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>MUTAG</td>
</tr>
<tr>
<td>VH</td>
<td>0.01s</td>
</tr>
<tr>
<td>RW</td>
<td>1M 46.86s</td>
</tr>
<tr>
<td>SP</td>
<td>0.92s</td>
</tr>
<tr>
<td>WL-VH</td>
<td>0.21s</td>
</tr>
<tr>
<td>WL-SP</td>
<td>7.02s</td>
</tr>
<tr>
<td>WL-PM</td>
<td>3M 42.07s</td>
</tr>
<tr>
<td>NH</td>
<td>0.40s</td>
</tr>
<tr>
<td>NSPDK</td>
<td>4.05s</td>
</tr>
<tr>
<td>ODD-STh</td>
<td>1.54s</td>
</tr>
<tr>
<td>PM</td>
<td>2.59s</td>
</tr>
<tr>
<td>GH</td>
<td>24.70s</td>
</tr>
<tr>
<td>SM</td>
<td>1M 57.25s</td>
</tr>
<tr>
<td>PK</td>
<td>0.48s</td>
</tr>
<tr>
<td>ML</td>
<td>10M 3.15s</td>
</tr>
<tr>
<td>CORE-WL</td>
<td>0.55s</td>
</tr>
<tr>
<td>CORE-SP</td>
<td>2.69s</td>
</tr>
</tbody>
</table>

- Average CPU running time for kernel matrix computation on the 7 classification datasets containing node-labeled graphs.
- “Avg. Rank” column illustrates average rank of each kernel. The lower the average rank, the lower the overall running time of the kernel.
**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 [sbmted ICML 2019]
- Learning Deep Set representations [submitted ICML 2019]
- 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/