GRAPH MINING AND GRAPH KERNELS

Part II: Graph Kernels

Karsten Borgwardt^ and Xifeng Yan*
^University of Cambridge
*IBM T. J. Watson Research Center
Frequent Subgraph Mining and Graph Kernels

- Frequent Subgraph Mining seeks to find patterns in a dataset of graphs = pattern mining.
- Graph Kernels aim at computing similarity scores between graphs in a dataset = graph comparison
- Link: Patterns can be used as features for graph comparison (Deshpande et al., 2005)
Graph Comparison

Definition 1 (Graph Comparison Problem) Given two graphs $G$ and $G'$ from the space of graphs $\mathcal{G}$. The problem of graph comparison is to find a mapping

$$s : \mathcal{G} \times \mathcal{G} \rightarrow \mathbb{R}$$

such that $s(G,G')$ quantifies the similarity (or dissimilarity) of $G$ and $G'$. 
Applications of Graph Comparison

- Function prediction of chemical compounds
- Structural comparison and function prediction of protein structures
- Comparison of social networks
- Analysis of semantic structures in Natural Language Processing
- Comparison of UML diagrams
Graph Isomorphism

**Graph isomorphism**
- Find a mapping $f$ of the vertices of $G_1$ to the vertices of $G_2$ such that $G_1$ and $G_2$ are identical; i.e. $(x, y)$ is an edge of $G_1$ iff $(f(x), f(y))$ is an edge of $G_2$. Then $f$ is an isomorphism, and $G_1$ and $G_2$ are called isomorphic.
- No polynomial-time algorithm is known for graph isomorphism.
- Neither is it known to be NP-complete.

**Subgraph isomorphism**
- Subgraph isomorphism asks if there is a subset of edges and vertices of $G_1$ that is isomorphic to a smaller graph $G_2$.
- Subgraph isomorphism is NP-complete.
Subgraph Isomorphism

**NP-completeness**
- A decision problem C is NP-complete iff
- C is in NP
- C is NP-hard, i.e. every other problem in NP is reducible to it.

**Problems for the practitioner**
- Excessive runtime in worst case
- Runtime may grow exponentially with the number of nodes
- For larger graphs with many nodes and for large datasets of graphs, this is an enormous problem

**Wanted**
- Polynomial-time similarity measure for graphs
Graph Edit Distances

**Principle**
- Count operations that are necessary to transform $G_1$ into $G_2$
- Assign costs to different types of operations (edge/node insertion/deletion, modification of labels)

**Advantages**
- Captures partial similarities between graphs
- Allows for noise in the nodes, edges and their labels
- Flexible way of assigning costs to different operations

**Disadvantages**
- Contains subgraph isomorphism check as one intermediate step
- Choosing cost function for different operations is difficult
Topological Descriptors

**Principle**
- Map each graph to a feature vector
- Use distances and metrics on vectors for learning on graphs

**Advantages**
- Reuses known and efficient tools for feature vectors

**Disadvantages**
- Efficiency comes at a price: feature vector transformation leads to loss of topological information (or includes subgraph isomorphism as one step)
Polynomial Alternatives

**Graph kernels**
- Compare substructures of graphs that are computable in polynomial time.

**Criteria for a good graph kernel**
- Expressive
- Efficient to compute
- Positive definite
- Applicable to wide range of graphs
What is a Kernel?

- Map two objects $x$ and $x'$ via mapping $\phi$ into feature space $\mathcal{H}$.
- Measure their similarity in $\mathcal{H}$ as $\langle \phi(x), \phi(x') \rangle$.
- **Kernel Trick**: Compute inner product in $\mathcal{H}$ as kernel in input space $k(x, x') = \langle \phi(x), \phi(x') \rangle$.  

![Diagram](image_url)
What is a Graph Kernel?

**Instance of R-convolution kernels by Haussler (1999)**

- Kernels on pairs of graphs (not pairs of nodes, though this is a common use in the literature)
- Convolution kernels compare all pairs of decompositions of two structured objects; a new type of decomposition results in a new graph kernel.
- A graph kernel makes the whole family of kernel methods applicable to graphs.
Hardness Results on Graph Kernels

Link to graph isomorphism (Gaertner, Flach, Wrobel, COLT 2003)

- Let \( k(G, G') = \langle \phi(G), \phi(G') \rangle \) be a graph kernel.
- If \( \phi \) is injective, \( k \) is called a complete graph kernel.

**Proposition 1** Computing any complete graph kernel is at least as hard as deciding whether two graphs are isomorphic.

**Proof** As \( \phi \) is injective, 
\[
k(G, G') - 2k(G, G') + k(G', G') = \langle \phi(G) - \phi(G'), \phi(G) - \phi(G') \rangle = \| \phi(G) - \phi(G') \| = 0 \text{ if and only if } G \text{ is isomorphic to } G'.
\]
Random Walks

**Principle** (Kashima et al., ICML 2003, Gaertner et al., COLT 2003)
- Compare walks in two input graphs $G$ and $G'$
- Walks are sequences of nodes that allow repetitions of nodes

**Elegant computation**
- Walks of length $k$ can be computed by looking at the $k$-th power of the adjacency matrix
- Construct direct product graph of $G$ and $G'$
- Count walks in this product graph $G_x = (V_x, E_x)$
- Each walk in the product graph corresponds to one walk in $G$ and $G'$

$$k_x(G, G') = \sum_{i,j=1}^{\left| V_x \right|} \left[ \sum_{k=0}^{\infty} \lambda^k A_x^k \right]_{ij}$$
Random Walks – Direct Product Graph

Diagram showing the direct product of two graphs.
Setbacks of Random Walk Kernels

Disadvantages
- Runtime in $O(n^6)$
- Tottering
- 'Halting'

Potential solutions
- Fast computation of random walk graph kernels (Vishwanathan et al., NIPS 2006)
- Label enrichment and preventing tottering (Mahe et al., ICML 2004)
- Graph kernels based on shortest paths (B. and Kriegel, ICDM 2005)
Runtime

Direct computation: $O(n^6)$

$$k_x(G, G') = \sum_{i,j=1}^{\left| V_x \right|} \left[ \sum_{k=0}^{\infty} \lambda^k A_x^k \right]_{i,j} = e^T (1 - \lambda A_x)^{-1} e$$

Solution

- Cast computation of random walk kernel as Sylvester Equation (Vishwanathan et al., NIPS 2006)
- These can be solved in $O(n^3)$
Vec-Operator and Kronecker Products

Vec-Operator
- vec flattens an $n \times n$ matrix $A$ into an $n^2 \times 1$ vector $\text{vec}(A)$.
- It stacks the columns of the matrix on top of each other, from left to right.

Kronecker Product
- Product of two matrices $A$ and $B$
- Each element of $A$ is multiplied with the full matrix $B$:

$$A \otimes B := \begin{bmatrix}
A_{1,1}B & A_{1,2}B & \ldots & A_{1,n}B \\
\vdots & \vdots & \ddots & \vdots \\
A_{n,1}B & A_{n,2}B & \ldots & A_{n,m}B
\end{bmatrix}$$
Sylvester Equations

- Equations of the form
  \[ X = SXT + X_0 \]
- Given three \( n \times n \) matrices \( S, T, \) and \( X_0 \).
- We want to determine \( X \).
- Solvable in \( O(n^3) \).
- We show how to turn Sylvester equations into graph kernels.
From Sylvester Equations to Random Walk Kernels

- We rewrite the Sylvester equation as
  \[ \text{vec}(X) = \text{vec}(SXT) + \text{vec}(X_0) \]

- We exploit the well-known fact
  \[ \text{vec}(SXT) = (T^\top \otimes S) \text{vec}(X) \]
  to rewrite the above question as
  \[ (I - T^\top \otimes S) \text{vec}(X) = \text{vec}(X_0). \]

- Now we have to solve
  \[ \text{vec}(X) = (I - T^\top \otimes S)^{-1} \text{vec}(X_0). \]

- We multiply both sides by \( \text{vec}(X_0)^\top \)
  \[ \text{vec}(X_0)^\top \text{vec}(X) = \text{vec}(X_0)^\top (I - T^\top \otimes S)^{-1} \text{vec}(X_0). \]
From Sylvester Equations to Random Walk Kernels

- In

\[ \text{vec}(X_0) \top \text{vec}(X) = \text{vec}(X_0) \top (\mathbf{I} - T \top \otimes S)^{-1} \text{vec}(X_0) \]

we substitute

\[
\begin{align*}
X_0 &= e e^\top \\
T &= \lambda A(G)^\top \\
S &= A(G')
\end{align*}
\]

and obtain

\[
\begin{align*}
e \top \text{vec}(X) &= e \top (\mathbf{I} - \lambda A(G) \otimes A(G'))^{-1} e \\
&= e \top (\mathbf{I} - \lambda A_x)^{-1} e.
\end{align*}
\]
Further Speed-ups for Sparse Graphs

- **Vec-Trick**
  - Let $S$ and $T$ be sparse.
  - We can efficiently compute $(T^\top \otimes S) \text{vec } X$ for each $X$ as $\text{vec}(SXT)$.
  - How to exploit this fact?

- **Fix-Point Iteration (FP)**
  - Determine a fix point (Kashima et. al, 2003):
    $$\text{vec } X_{k+1} = e + (T^\top \otimes S) \text{vec } X_k$$

- **Conjugate Gradient (GC)**
  - Use conjugate gradient solver to compute $X$ in
    $$(I - T^\top \otimes S) \text{vec } X = e.$$  
  - Requires computation of $(T^\top \otimes S) \text{vec } X_k$ for the residuum $R$ in each step.
Impact on Runtime for Kernel Computation

![Bar chart showing runtime for different kernels across various datasets: MUTAG, PTC, Enzyme, Protein. The x-axis represents the dataset, and the y-axis represents the runtime in seconds on a logarithmic scale. Different colors denote different kernels: black for direct, red for Sylv., cyan for CG, and white for FP. The chart indicates the performance comparison between these kernels.]
Impact of vec-trick on runtime

Runtime in sec

# nodes

original
vec-trick
Tottering (Mahe et al., ICML 2004)

**Phenomenon of tottering**
- Walks allow for repetitions of nodes
- A walk can visit the same cycle of nodes all over again
- Kernel measures similarity in terms of common walks
- Hence a small structural similarity can cause a huge kernel value

![Diagram of tottering phenomenon](image)
Preventing Tottering

- Explicitly forbid tottering between 2 nodes, that is any walk \((v_1, \ldots, v_l)\) such that \(v_i = v_{i+2}\) for any \(i \in \{1, \ldots, l-2\}\).

- Special transformation of each of the input graphs \(G = (V, E)\) allows for this modification:
  - Create a new graph \(G_T\) with \(V_T = V \cup E\) and \(E_T = \{(v, (v, t))|v \in V, (v, t) \in E\} \cup \{((u, v), (v, t))|(u, v), (v, t) \in E, u \neq t\}\)
  - The node set of \(G_T\) is the set of vertices and edges of \(G\)
  - In \(G_T\), there are directed edges between each node from \(G\) and each adjacent edge, and between edges from \(G\) that share exactly one node (that is target node in one edge, and source node in the other)
Preventing Tottering

- Walks in $G_T$ correspond to walks in $G$, but it is not possible to totter between 2 nodes

Limitations

- Modification increases graph size from $O(n)$ to $O(n^2)$ with adverse effects on kernel computation runtime
- Experimental evidence does not show a uniform improvement of classification accuracy
Label Enrichment: Morgan Index (1965)

- Size of product graph affects runtime of kernel computation
- The more node labels, the smaller the product graph
- Trick: Introduce new artificial node labels
- Topological descriptors of nodes are natural extra labels
- For instance, the Morgan Index that counts k-th order neighbours of a node:

```
Original graph

2 2
2 2
2 2

1st order Morgan Index

2 2
2 2
5 5

2nd order Morgan Index

4 4
5 5
4 4
```
Replacing Walks by Paths

**Underlying idea**
- Paths do not suffer from tottering
- Define a graph kernel based on paths

**Setbacks**
- All paths are NP-hard to compute
- Longest paths are NP-hard to compute
- But shortest paths are computable in $O(n^3)$!

**Pitfall**
- Number of shortest paths in a graph may be exponential in the number of nodes (in pathological cases)

**Workaround**
- Shortest paths need not be unique, but shortest path distances are
- Define graph kernel based on shortest path distances
Shortest-Path Kernel on Graphs  

(B. and Kriegel, ICDM 2005)

- Compute all-pairs-shortest-paths for $G$ and $G'$ via Floyd-Warshall
- Define a kernel by comparing all pairs of shortest path lengths from $G$ and $G'$:

$$k(G, G') = \sum_{v_i, v_j \in G} \sum_{v'_k, v'_l \in G'} k_{\text{length}}(d(v_i, v_j), d(v'_k, v'_l))$$

- $d(v_i, v_j)$ is the length of the shortest path between node $v_i$ and $v_j$
- $k_{\text{length}}$ is a kernel that compares the lengths of two shortest paths, for instance,
  - a linear kernel $k(d(v_i, v_j), d(v'_k, v'_l)) = d(v_i, v_j) \times d(v'_k, v'_l)$, or
  - a delta kernel $k(d(v_i, v_j), d(v'_k, v'_l)) = \begin{cases} 1 & \text{if } d(v_i, v_j) = d(v'_k, v'_l) \\ 0 & \text{otherwise} \end{cases}$
Definition 1 (Wiener Index) Let $G = (V, E)$ be a graph. Then the Wiener Index $W(G)$ of $G$ is defined as

$$W(G) = \sum_{v_i \in G} \sum_{v_j \in G} d(v_i, v_j),$$

where $d(v_i, v_j)$ is defined as the length of the shortest path between nodes $v_i$ and $v_j$ from $G$. 

(Wiener, 1947)
Link to Wiener Index

- Compute the product of the Wiener Indices $W(G)$ and $W(G')$ as

$$W(G) \ast W(G') = \left( \sum_{v_i \in G} \sum_{v_j \in G} d(v_i, v_j) \right) \left( \sum_{v'_k \in G'} \sum_{v'_l \in G'} d(v'_k, v'_l) \right)$$

$$= \sum_{v_i \in G} \sum_{v_j \in G} \sum_{v'_k \in G'} \sum_{v'_l \in G'} d(v_i, v_j) d(v'_k, v'_l)$$

$$= \sum_{v_i, v_j \in G} \sum_{v'_k, v'_l \in G'} k_{linear}(d(v_i, v_j), d(v'_k, v'_l))$$

$$= k_{shortest \ path}(G, G')$$
Properties of Shortest-Path Kernel

**Advantages**

- No tottering, better accuracy on classification benchmarks
- Runtime is in $O(n^4)$ and includes
  - Computing all-pairs-shortest-paths for $G$ and for $G'$: $O(n^3)$
  - Comparing all pairs of shortest paths from $G$ and $G'$: $O(n^4)$
- Empirically faster than (fast) random walk kernels (probably due to graph size)

**Disadvantages**

- $O(n^4)$ too slow for large graphs
- Dense matrix representation for connected graphs, may lead to memory problems on large graphs
Optimal Assignment Kernel (Froehlich et al., ICML 2005)

- $G$ and $G'$ are graphs
- $\{x_1, \ldots, x_{|G|}\}$ are substructures of $G$, e.g. nodes
- $\{y_1, \ldots, y_{|G'|}\}$ are substructures of $G'$, e.g. nodes
- $k_1$ is a non-negative kernel comparing substructures
- $\pi$ is a permutation of the natural numbers $\{1, \ldots, \min(|G|, |G'|)\}$
- Then

$$k_A(G, G') := \begin{cases} \max_\pi \sum_{i=1}^{G} k_1(x_i, y_{\pi(i)}), & \text{if } |G'| \geq |G| \\ \max_\pi \sum_{j=1}^{G'} k_1(x_{\pi(j)}, y_j), & \text{otherwise} \end{cases}$$

is the optimal assignment kernel (Froehlich et al., ICML 2005)
- Not positive definite in general (Vert, 2008)
Weighted Decomposition Kernel  
(Menchetti et al., ICML 2005)

- $G = (V, E)$ and $G' = (V', E')$ are graphs
- Idea is to define two different types of substructures
- $s$ is a subgraph of $G$ called a selector, with associated kernel $\delta$
- $z = (z_1, ..., z_D)$ is a tuple of subgraphs of $G$ called the contexts of occurrence of $s$ in $x$, with associated kernel $\kappa$
- Then

$$k(G, G') := \sum_{(s, z) \in R^{-1}(G), (s', z') \in R^{-1}(G')} \delta(s, s') \sum_{d=1}^{D} \kappa(z_d, z_d')$$  \hspace{1cm} (1)

is the weighted decomposition kernel  
(Menchetti et al., ICML 2005)

- Example: $s$ can be a node and $z$ the neighbourhood of $s$ in $G$
Edit-Distance Kernel (Neuhaus and Bunke, 2006)

**Principle**
- Tries to combine the power of graph kernels and edit distances
- Random walk kernel that uses a modified product graph:
- It only contains pairs of nodes that were matched by a graph edit-distance beforehand

**Advantage**
- Edit-distance kernels outperform random walks and edit distances in their experimental evaluation

**Disadvantage**
- These edit-distance kernels are not positive definite in general
Subtree Kernel (Ramon and Gaertner, 2004)

**Principle**
- Compare subtree-like patterns in two graphs
- Subtree-like pattern is a subtree that allows for repetitions of nodes and edges (similar to walk versus path)
- For all pairs of nodes \( v \) from \( G \) and \( u \) from \( G' \):
  - Compare \( u \) and \( v \) via a kernel function
  - Recursively compare all sets of neighbours of \( u \) and \( v \) via a kernel function

**Advantages**
- Richer representation of graph structure than walk-based approach

**Disadvantages**
- Runtime grows exponentially with the recursion depth of the subtree-like patterns
Cyclic Pattern Kernel (Horvath et al., KDD 2004)

**Principle**
- Compare simple cycles in two graphs (paths where start node equals end node)
- Number of simple cycles is exponential in the number $n$ of vertices in worst case
- Define canonical string representation of each simple cycle, referred to as a cyclic pattern

**Advantages**
- Interesting alternative to walk-based kernels

**Disadvantages**
- Cyclic pattern kernel on general graphs is NP-hard to compute
- Restrict their attention to scenarios where the number of simple cycles in a graph dataset is bounded by a constant
Graphlet Kernel (B., Petri, et al., MLG 2007)

**Principle**
- Count subgraphs of limited size $k$ in $G$ and $G'$
- These subgraphs are referred to as graphlets (Przulj, Bioinformatics 2007)
- Define graph kernel that counts isomorphic graphlets in two graphs

**Runtime problems**
- Pairwise test of isomorphism is expensive
- Number of graphlets scales as $O(n^k)$

**Two solutions on unlabeled graphs**
- Precompute isomorphisms
- Sample graphlets

**Disadvantage**
- Same solutions not feasible on labeled graphs
Graphlet Kernel

1. clique
   111111
2. diamond
   111110
3. flower
   111100
4. star
   111000
5. square
   110011
6. line
   110010

7. triangle
   110100
8. 3-line
   110000
9. 2 separate edges
   100001
10. 1 edge
   100000
11. no edge
   000000
Recent Trends

**Combine graph kernels with graphical models (Bach, ICML 2008)**
- Presents a new kernel for 2D or 3D point clouds
- Compares local subsets of the point clouds
- Considers subsets based on subtrees and walks
- Uses a specific factorized form for the local kernels between subtrees.

**Combine graph kernels with group theory (Kondor and B., ICML 2008)**
- Represent graph as a function over the symmetric group
- Derive invariants for that function called the *skew spectrum*
- Use subset of these invariants that is computable in $O(n^3)$ as feature representation of the graph
Applications: Chemoinformatics (Ralaivola et al., 2005)

Graph kernels inspired by concepts from chemoinformatics
- Define three new kernels (Tanimoto, MinMax, Hybrid) for function prediction of chemical compounds
- Based on the idea of molecular fingerprints and
- Counting labeled paths of depth up to $d$ using depth-first search from each possible vertex

Properties
- Tailored for applications in chemical informatics,
- Exploit the small size and
- Low average degree of these molecular graphs.
Chemical Compound Classification (Wale et al, ICDM 2006)

New kernels and experimental comparison of existing techniques
- Define a kernel that considers graph fragments: Subgraphs with a maximum of 1 edges
- Fragment-based kernels outperform kernels using frequent subgraphs and walk-based kernels

Four choices in kernel design for chemical compounds
- Generation of patterns (learnt from dataset versus defined by expert)
- ‘Preciseness‘ of the patterns (whether subgraph features map to the same dimension in feature space)
- Complete coverage (whether the patterns occur in all of the instances of the dataset)
- Complexity of patterns (walks and cycles versus frequent subgraphs)
Applications: Protein Function Prediction (B. et al, ISMB 2005)

- Predict the function of a protein from its structure
- Model protein structure as graph
- Use graph kernels to measure structural similarity and SVM to predict functional class
- Reaches competitive results on benchmark datasets
Future Challenges for Graph Kernel Research

**Data level**
- Larger and more graph data
- More dynamic graph data

**Algorithmic level**
- Feature selection on graphs
- Scalability and efficiency
- Automatic choice of complexity of representation

**Interdisciplinary level**
- Link to graph mining, both current research and literature
- Applications in bioinformatics and the Internet
THANK YOU!

kmb51 @ cam.ac.uk
References

- Francis Bach: Graph kernels between point clouds. ICML 2008
- Karsten M. Borgwardt, Hans-Peter Kriegel: Shortest-Path Kernels on Graphs. ICDM 2005: 74-81
- Karsten M. Borgwardt, Tobias Petri, S. V. N. Vishwanathan, Hans-Peter Kriegel: An Efficient Sampling Scheme For Comparison of Large Graphs. MLG 2007
References

- Thomas Gärtner, Peter A. Flach, Stefan Wrobel: On Graph Kernels: Hardness Results and Efficient Alternatives. COLT 2003: 129-143
- Tamás Horváth, Thomas Gärtner, Stefan Wrobel: Cyclic pattern kernels for predictive graph mining. KDD 2004: 158-167
- Imre Risi Kondor, Karsten M. Borgwardt: The skew spectrum of graphs. ICML 2008
- Pierre Mahé, Nobuhisa Ueda, Tatsuya Akutsu, Jean-Luc Perret, Jean-Philippe Vert: Extensions of marginalized graph kernels. ICML 2004
References

- Sauro Menchetti, Fabrizio Costa, Paolo Frasconi: Weighted decomposition kernels. ICML 2005:585-592
- Michel Neuhaus, Horst Bunke: A Random Walk Kernel Derived from Graph Edit Distance. SSPR/SPR 2006: 191-199
- Jan Ramon, Thomas Gärtner: Expressivity versus Efficiency of Graph Kernels. First International Workshop on Mining Graphs, Trees and Sequences 2003
- Nikil Wale, George Karypis: Comparison of Descriptor Spaces for Chemical Compound Retrieval and Classification. ICDM 2006: 678-689