GRAPH KERNELS

Karsten M. Borgwardt

kb@dbs.ifi.lmu.de

Lehrstuhl für Datenbanksysteme,
Ludwig-Maximilians-Universität München

Graphs are everywhere...

Graphs in Reality

- Graphs model objects and their relationships.
- Also referred to as *networks*.
- All common data structures can be modeled as graphs.

Graphs in Bioinformatics

- Molecular Biology studies relationship between molecular components.
- Graphs are ideal to model these:
  - Molecules
  - Protein-protein interaction networks
  - Metabolic networks

Central Questions

How similar are two graphs?

- Graph similarity is the central problem for all learning tasks such as clustering and classification on graphs.

Applications

- Function prediction for molecules, in particular proteins
- Comparison of protein-protein interaction networks

Challenges

- Subgraph isomorphism is NP-complete.
- Comparing graphs via isomorphism checking is thus prohibitively expensive!
- Graph kernels offer a faster, yet principled alternative.

From the beginning...

Definition of a Graph

- A $graph$ G is a set of nodes (or vertices) V and edges E, where $E \subset V^2$.
- An $attributed$ graph is a graph with labels on nodes and/or edges; we refer to labels as $attributes$.
- The $adjacency\ matrix$ A of G is defined as

$$[A]_{ij} = \begin{cases} 1 & \text{if } (v_i, v_j) \in E, \\ 0 & \text{otherwise} \end{cases},$$

  where v_i and v_j are nodes in G.
- A $walk$ w of length $k-1$ in a graph is a sequence of nodes $w = (v_1, v_2, \cdots, v_k)$ where $(v_{i-1}, v_i) \in E$ for $1 \leq i \leq k$.
- w is a $path$ if $v_i \neq v_j$ for $i \neq j$.

Graph Isomorphism

Graph isomorphism (cp Skiena, 1998)

- 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 number of nodes
- For large graphs with many nodes, and
- For large datasets of graphs
- this is an enormous problem

Wanted Polynomial-time similarity measure for graphs

Polynomial alternatives

Graph kernels

- Compare substructures of graphs that are computable in polynomial time
- Examples: walks, paths, cyclic patterns, trees

Criteria for a good graph kernel

- Expressive
- Efficient to compute
- Positive definite
- Applicable to wide range of graphs

Random Walks

Principle

- Compare walks in two input graphs (Kashima et al., 2003; Gärtner et al., 2003)
- Walks are sequences of nodes that allow repetitions of nodes

Important trick

- Walks of length k can be computed by taking the adjacency matrix A to the power of k
- $A^k(i,j) = c$ means that c walks of length k exist between vertex i and vertex j

Product Graph

How to find common walks in two graphs?

- Another trick: Use the Product Graph of G_1 and G_2

Definition

- $G_\times = (V_\times, E_\times)$, defined via

$$V_\times(G_1 \times G_2) = \{(v_1, w_1) \in V_1 \times V_2 : label(v_1) = label(w_1)\}$$

$$E_\times(G_1 \times G_2) = \{((v_1, w_1), (v_2, w_2)) \in V^2(G_1 \times G_2) : (v_1, v_2) \in E_1 \wedge (w_1, w_2) \in E_2 \wedge (label(v_1, v_2) = label(w_1, w_2))\}$$

Meaning

- Product graph consists of pairs of identically labeled nodes and edges from G_1 and G_2

Random walk kernel

The trick

- Common walks can now be computed from $A_\times^k$

Definition of random walk kernel

- $$k_\times(G_1, G_2) = \sum_{i,j=1}^{|V_\times|} [\sum_{n=0}^{\infty} \lambda^n A_\times^n]_{ij} = \mathbf{e}^\top (\mathbf{I} - \lambda A_\times)^{-1} \mathbf{e}$$

Meaning

- Random walk kernel counts all pairs of matching walks
- λ is decaying factor for the sum to converge

Runtime of random walk kernels

Notation

- given two graphs G_1 and G_2
- n is the number of nodes in G_1 and G_2

Computing product graph

- requires comparison of all pairs of edges in G_1 and G_2
- runtime $O(n^4)$

Powers of adjacency matrix

- matrix multiplication or inversion for n^2 * n^2 matrix
- runtime $O(n^6)$

Total runtime

- $O(n^6)$ - yet this can be sped up to $O(n^3)$! (Vishwanathan et al., 2006)

Fast kernel computation

Notation:

- Operator vec flattens a matrix and vec^{-1} reconstructs it.
- The Kronecker product of A and B is written as:

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

Product Graphs:

- Entries in the adjacency graph are 1 iff corresponding nodes are adjacent in both G_1 and G_2.
- The adjacency matrix of a product graph can be written as $A(G_1) \otimes A(G_2)$.

Sylvester Equations

Definition:

- Equations of the form

$$M = SMT + U$$

- The matrices S, T and U are given.
- We need to solve for M.

Properties:

- Also known as discrete-time Lyapunov equation.
- Typical solution is $O(n^3)$.
- We will show how to convert graph kernels to Sylvester Equations.

Sylvester to Random Walks

Gory Maths:

- Rewrite the Sylvester equation as

$$\mathrm{vec}(M) = \mathrm{vec}(SMT) + \mathrm{vec}(U)$$

- Use the well known identity

$$\mathrm{vec}(SMT) = (T^\top \otimes S)\,\mathrm{vec}(M),$$

  to rewrite

$$(\mathbf{I} - T^\top \otimes S)\,\mathrm{vec}(M) = \mathrm{vec}(U).$$

- Now we need to solve

$$\mathrm{vec}(M) = (\mathbf{I} - T^\top \otimes S)^{-1}\,\mathrm{vec}(U).$$

- Multiply both sides by $\mathrm{vec}(U)^\top$

$$\mathrm{vec}(U)^\top\,\mathrm{vec}(M) = \mathrm{vec}(U)^\top(\mathbf{I} - T^\top \otimes S)^{-1}\,\mathrm{vec}(U).$$

Recovering Graph Kernels

Gory Maths Contd . . .:

- In the equation

$$\mathrm{vec}(U)^\top \mathrm{vec}(M) = \mathrm{vec}(U)^\top (\mathbf{I} - T^\top \otimes S)^{-1} \mathrm{vec}(U).$$

substitute

$$\begin{aligned} U &= \mathbf{e}\,\mathbf{e}^\top \\ T &= \lambda A(G_1)^\top \\ S &= A(G_2) \end{aligned}$$

to get

$$\begin{aligned} \mathbf{e}^\top \mathrm{vec}(M) &= \mathbf{e}^\top (\mathbf{I} - \lambda A(G_1) \otimes A(G_2))^{-1}\,\mathbf{e} \\ &= \mathbf{e}^\top (\mathbf{I} - \lambda A_\times)^{-1}\,\mathbf{e}\,. \end{aligned}$$

This is exactly the random walk graph kernel!

Tottering

Artificially high similarity scores

- Walk kernels allow walks to visit same edges and nodes multiple times $\rightarrow$ artificially high similarity scores by repeated visiting of same two nodes

Additional node labels

- Mahe et al. (2004) add additional node labels to reduce number of matching nodes $\rightarrow$ improved classification accuracy

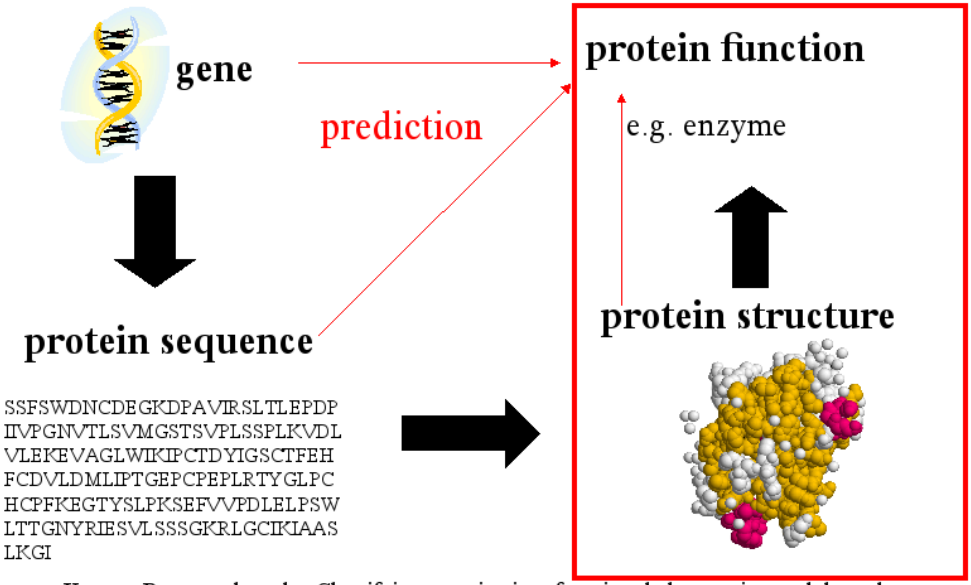
Forbidding cycles with 2 nodes

- Mahe et al. redefine walk kernel to forbid subcycles consisting of two nodes $\rightarrow$ no practical improvement

Graph kernel application

Protein function prediction

Molecular Information Flow



Karsten Borgwardt et al. - Classifying proteins into functional classes via graph kernels 3

Random walk kernel application

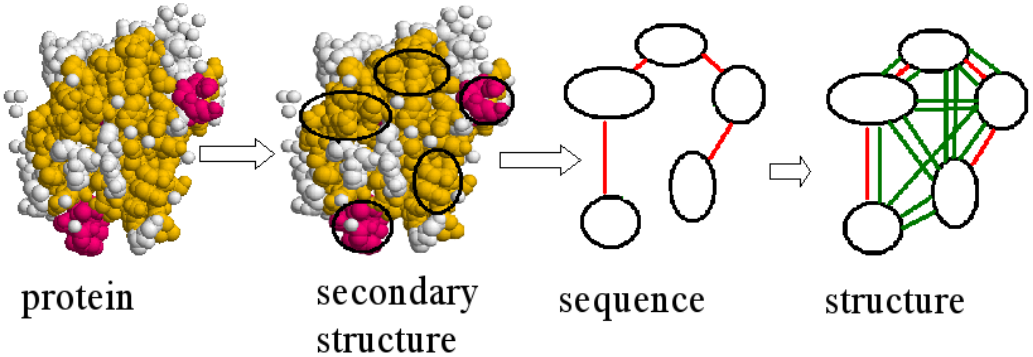
Protein function prediction (Borgwardt et al., 2005)

- Compare 3D structure of molecules modeled as graphs
- Then classify molecules into functional classes
- In other terms, predict function from structure

The task

- Given protein structures from PDB
- A functional classification scheme, e.g. BRENDA, which defines classes of proteins with similar function
- Build a SVM classifier to predict graph class membership of newly discovered proteins from their structure

Protein graph model

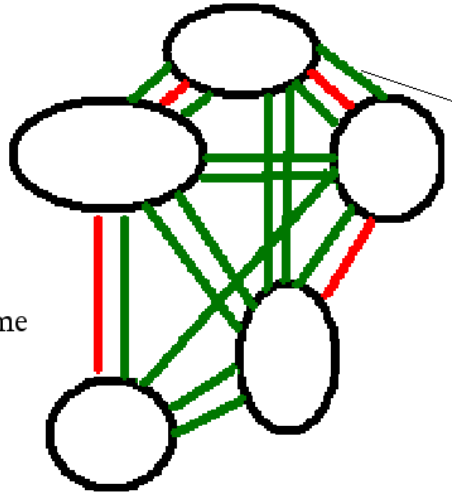


Karsten Borgwardt et al. - Classifying proteins into functional classes via graph kernels 10

Protein graph kernel

Protein graph model

Node attributes
- hydrophobicity
- polarity
- polarizability
- van der Waals volume
- length
- helix, sheet, loop



Edge attributes
- type (sequence, structure)
- length

Karsten Borgwardt et al. - Classifying proteins into functional classes via graph kernels 11

Evaluation: enzymes vs. non-enzymes

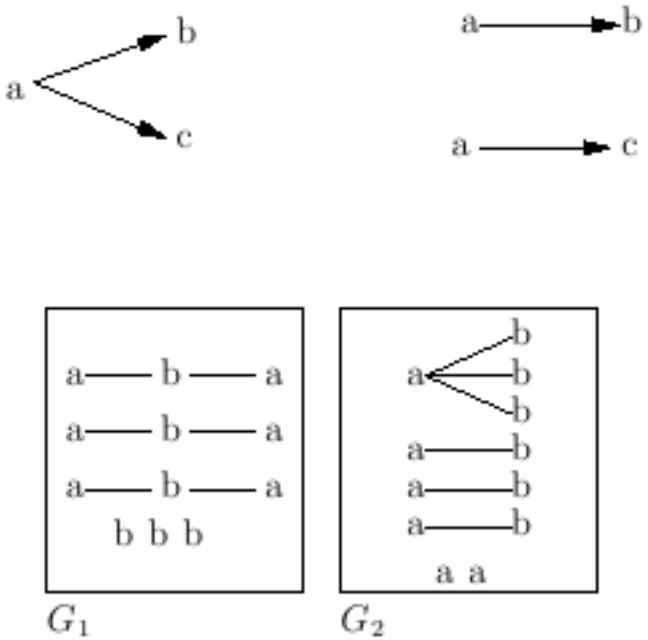
10-fold cross-validation on 1128 proteins from dataset by Dobson and Doig (2003); 59 % are enzymes.

Kernel type	accuracy	SD
Vector kernel	76,86	1,23
Optimized vector kernel	80,17	1,24
Graph kernel	77,30	1,20
Graph kernel without structure	72,33	5,32
Graph kernel with global info	84,04	3,33
DALI classifier	75,07	4,58

Karsten Borgwardt et al. - Classifying proteins into functional classes via graph kernels 17

Limitations of walks

Different graphs mapped to identical points in walks feature space (from Ramon and Gaertner, 2003)



Subtree kernel

Motivation

- Compare tree-like substructures of graphs
- May distinguish between substructures that walk kernel deems identical

Algorithmic principle

- for all pairs of nodes r from $\mathcal{V}_1(G_1)$ and s from $\mathcal{V}_2(G_2)$ and a predefined height h of subtrees:
- recursively compare neighbors (of neighbors) of r and s
- subtree kernel on graphs is sum of subtree kernels on nodes

Subtree kernel

Matching of neighborhoods

- $\delta^+(r)$ is the set of nodes adjacent to node r
- $M(r,s)$ is the set of all matchings from $\delta^+(r)$ to $\delta^+(s)$
-

$$\begin{aligned}&M(r,s) = \{R \subseteq \delta^+(r) \times \delta^+(s) | \\ &(\forall (a,b),(c,d) \in R : a = c \Longleftrightarrow b = d) \wedge \\ &(\forall (a,b) \in R : label(a) = label(b))\}\end{aligned}$$

Kernel computation on pairs of trees

- Then $k_h(r,s)$ can be computed as

$$k_h(r,s) = \lambda_r \lambda_s \sum_{R \in M(r,s)} \prod_{(r',s') \in R} k_{h-1}(r',s'),$$

- where λ_r and λ_s are positive scalars.

Subtree kernel

Subtree graph kernel

- The subtree graph kernel for fixed height h is

$$k_{tree,h}(G_1, G_2) = \sum_{r \in \mathcal{V}_1} \sum_{s \in \mathcal{V}_2} k_h(r, s).$$

- The subtree graph kernel for h approaching infinity:

$$k_{tree}(G_1, G_2) = \lim_{h \to \infty} k_{tree,h}(G_1, G_2),$$

  which will converge for suitable choice of λ_r and λ_s.
- both versions are positive definite
- large choice of h provides good approximation of k_{tree}.

Reminder: Tottering

Artificially high similarity scores

- Walk kernels allow walks to visit same edges and nodes multiple times $\rightarrow$ artificially high similarity scores by repeated visiting of same two nodes

Subtree kernels suffer from tottering as well!

Cycles instead of walks?

Idea

- Computing kernels based on cyclic and tree patterns (Horvath, Gärtner, Wrobel, 2005)
- Intersection kernel instead of kernel based on counts

Problems

- Computation of all general cycles is NP-hard
- Remedy: Consider graphs with up to k simple cycles only
- Problem: Cyclic pattern kernel can only be used on datasets fulfilling this constraint.

Depth-first search paths?

Idea

- Computing kernels based on paths of length up to d starting from a node r (Swamidass et al., ISMB 2005)
- These are determined by Depth-First Search (DFS)
- Once diverged, paths may not visit the same node
- Path counts are then combined into a kernel on graphs

Problems

- does only measure local similarity in structure, not global
- DFS paths exclude edges from graph comparison that are not on these paths

All-paths kernel?

Idea

- Idea: Determine all paths from two graphs
- Compare paths pairwise to yield kernel

Advantage

- No tottering

Problem

- All-Paths kernel is NP-hard to compute.

Proof

- If determining all paths were not NP-hard, then one could check whether a Hamilton path exists of length $n-1$.
- However, finding a Hamilton path is known to be NP-hard. Hence, determining all paths as well.

Alternatives?

Longest paths?

- Also NP-hard, same reason as for all paths.

Shortest Paths!

- computable in $O(n^3)$ by the classic Floyd-Warshall algorithm 'all-pairs shortest paths'

Shortest-paths kernel

Kernel computation (Borgwardt & Kriegel, 2005)

- Determine all shortest paths in two input graphs G_1 and G_2
- Compare all shortest distances in $SD(G_1)$ to all shortest distances in $SD(G_2)$
- Sum over kernels on all pairs of shortest distances gives shortest-path kernel

$$K_{shortest\,path}(G_1, G_2) = \sum_{s_1 \in SD(G_1)} \sum_{s_2 \in SD(G_2)} k(s_1, s_2)$$

Runtime shortest-paths kernel

Notation

- given two graphs G_1 and G_2
- n is the number of nodes in G_1 and G_2

Kernel computation

- Determine shortest paths, in G_1 and G_2 separately: $O(n^3)$
- Compare these pairwise: $O(n^4)$
- Hence: Total runtime complexity $O(n^4)$

Discussion

Advantages

- Compares meaningful features of graphs, namely shortest paths
- Positive definite
- No tottering
- Works on all graphs (using artificial edge length)
- Computable in $O(n^4) \rightarrow$ two magnitudes faster than the random walk kernel

Discussion

Disadvantages

- Does not exploit sparsity of graphs
- Leads to full matrix representations of graphs
- Ignores information represented by longer paths
- Most meaningful if edge labels represent some type of distance

Variants

equal-length shortest paths

- if two shortest paths contain a non-identical number of edges, count them as completely dissimilar

k shortest paths

- compare k shortest paths
- use algorithm by Yen 1971 for $k\ loopless\ shortest\ paths$
- Yen's runtime $O(kn(m + n * \log n))$
- runtime increases to $O(k * n^5)$

k shortest disjoint paths

- simpler approach: iteratively apply Dijkstra's algorithm and remove currently shortest path from graph
- compare k disjoint shortest paths
- worst case total runtime $O(k * n^4)$

Hot topics on graph kernels

Questions

- Are there principled approaches to speed up computation of graph kernels?
- Are there better polynomial algorithms to describe graph substructures?
- Can we employ graph kernels for different tasks in graph mining?

Applications in Bioinformatics

Current

- comparing structures of proteins
- comparing structures of RNA
- measuring similarity between metabolic networks
- measuring similarity between protein interaction networks
- measuring similarity between gene regulatory networks

Future

- detecting conserved paths in interspecies networks
- finding differences in individual or interspecies networks
- finding common motifs in biological networks