Instructor: Sael Lee

CS549 Spring – Computational Biology

LECTURE 20: GRAPH KERNELS

Resources:

- Shervashidze, N., et al. (2011). Weisfeiler-Lehman Graph Kernels. *Journal of Machine Learning Research*, *12*, 2539–2561.
- "Graph Mining and Graph Kernels" *K. Borgwardt and X. Yan* KDD2008 Tutorial
- Vishwanathan, S. V. N., et al. (2010). Graph Kernels. *Journal of Machine Learning Research*, *11*, 1201–1242.
- "Graph kernels and chemoinformatics" Jean-Philippe Vert. Slides from Gbr'2007

FREQUENT SUBGRAPH MINING

Frequent Subgraph Mining seeks to find patterns in a dataset of graphs

Given

- ✓ a set $D = \{G_1, G_2, \dots, G_N\}$ of **graphs**
- ✓ a minimum frequency $0 \leq \theta_{min} \leq 1$

Find the set of **frequent subgraphs**, i.e.

$$F(\theta_{min}) = \{H \mid |\{i\colon H \text{ subgraph of } G_i\}| \geq N\theta_{min}\}$$

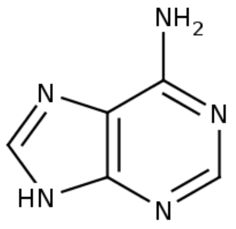
- The frequency of subgraph H is called the **support** of H

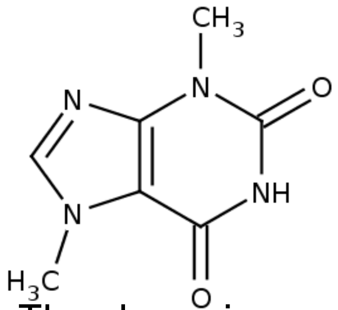
$$supp(H) = |\{i : H \text{ subgraph of } G_i\}|$$

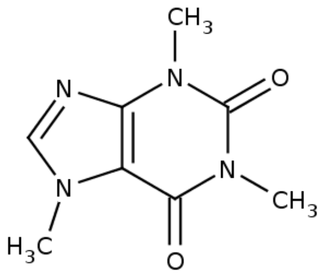
- θ_{min} is called the **minimimum support**
- Often focus on **connected** subgraphs

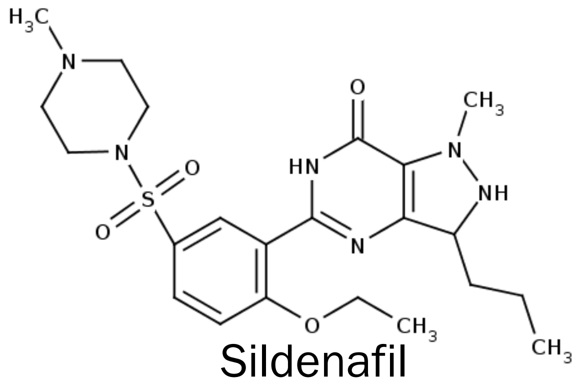
EX. APPLICATION

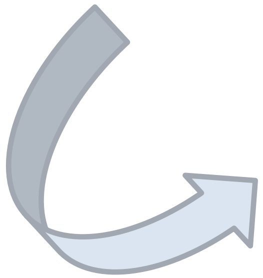
Finding **moieties** in chemical compounds


Adenine

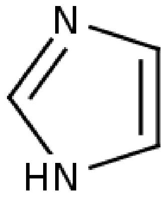

Theobromine

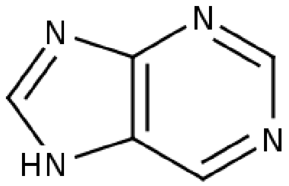

Caffeine


Sildenafil



Frequent subgraphs


Imidazole


Purine

GRAPH COMPARISON

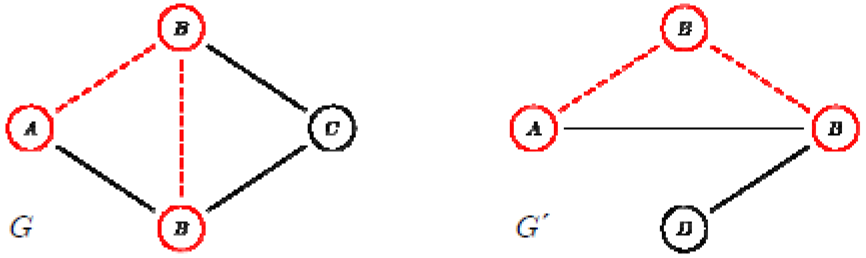
Graph Kernels aim at computing similarity scores between graphs in a dataset

Definition 1 **(Graph Comparison Problem)**

Given two graphs G and G′ from the space of graphs G. The problem of graph comparison is to find a mapping

$$s : G \times G' \rightarrow R$$

such that s(G,G′) quantifies the similarity (or dissimilarity) of G and G′.



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

G_1 and G_2 are **isomorphic** if there exists a subgraph isomorphism from G_1 to G_2 and from G_2 to G_1

- Subgraph isomorphism is NP-complete

We want polynomial-time similarity measure for graphs

MEASURING GRAPH SIMILARITY 1

Graph Edit Distances

Principle

- Count operations that are necessary to transform G1 into G2
- 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 (NP-complete) as one intermediate step
- Choosing cost function for different operations is difficult

MEASURING GRAPH SIMILARITY 2

Topological Descriptors

Principle

- Map each graph to a feature vector (ex> finger printing methods)
- Use distances and metrics on vectors for learning on graphs

Advantages

- Reuses known and efficient tools for feature vectors

Disadvantages

- Most feature vector transformation leads to loss of topological information
- Or includes subgraph isomorphism as one step

feature vectors (chemical **fingerprints)**

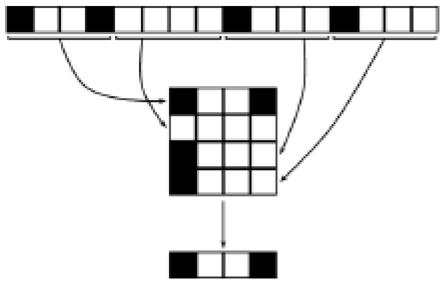
$$\phi(A) = (\phi_s(A))_{s \text{ substructure}}$$

where

$$\phi_s(A) = \begin{cases} 1 & \text{if } s \text{ occurs in } A \\ 0 & \text{otherwise} \end{cases}$$

0	1	0	0	0	1	1	0	1	0	0	0	...

Modulo Compression (lossy)



Elias-Gamma Monotone Encoding (lossless)

[Baldi et al., 2007]

- index $j \rightarrow \lfloor log(j) \rfloor$ 0 bits + binary encoding of j
- $j_i < j_{i+1}$: $\lfloor log(j_{i+1}) \rfloor \rightarrow \lfloor log(j_i) - log(j_{i+1}) \rfloor$
- average compressed size $= 1,800$ bits

MEASURING GRAPH SIMILARITY 3:

Graph Kernels: Kernels on pairs of graphs

Principle

- Let $\phi(\mathrm{x})$ be a vector representation of the graph x
- The kernel between two graphs is defined by:

$$K(x, x') = \phi(x)^T \phi(x')$$

- To solve convex optimization with kernels, kernels needs to be
  - Symmetric, that is, $k(x, x') = k(x', x)$, and
  - Positive semi-definite (p.s.d.)
- Comparing nodes in a graph involves constructing a kernel between nodes
- Comparing graphs involves constructing a kernel between graphs.

Advantages

- Similarity of two graphs are inferred through kernel function

Disadvantages

- Defining a kernel that captures the semantics inherent in the graph structure and is reasonably efficient to evaluate is the key challenge.

BRIEF HISTORY OF GRAPH KERNELS

- The idea of **constructing kernels *on* graphs** (i.e., between the nodes of a single graph) was first proposed by Kondor and Lafferty (2002), and extended by Smola and Kondor (2003).

- Idea of **kernels *between* graphs** were proposed by G¨artner et al. (2003) and later extended by Borgwardt et al. (2005).

- Idea of **marginalized kernels** (Tsuda et al., 2002) was extended to graphs by Kashima et al. (2003, 2004), then further refined by Mah´e et al. (2004).

GRAPH KERNELS TERMINOLOGY

- A **graph** *G* as a triplet (V, E, l), where *V* is the set of vertices, *E* is the set of undirected edges, and $l : V \rightarrow \Sigma$ is a function that assigns labels from an alphabet Σ to nodes in the graph.
- The **neighborhood** *N* (*v*) of a node *v* is the set of nodes to which *v* is connected by an edge, that is $N(v) = \{v' | (v, v') \in E\}$.

For simplicity, we assume that every graph has ***n*** nodes, ***m*** edges, and a maximum degree of ***d***. The **size of *G*** is defined as the cardinality of *V*.

- A **path** is a walk that consists of distinct nodes only.
- A **walk** is a sequence of nodes in a graph, in which consecutive nodes are connected by an edge. walk extends the notion of path by allowing nodes to be equal
- A ***(rooted) subtree*** is a subgraph of a graph, which has no cycles, but a designated root node.
- The **height of a subtree** is the maximum distance between the root and any other node in the subtree.

GRAPH KERNELS TERMINOLOGY CONT.

Complete graph kernels

A graph **kernel is complete** if it separates non-isomorphic graphs, i.e.:

$$\forall G_1, G_2 \in X, d_K(G_1, G_2) = 0 \Rightarrow G1 \cong G2 .$$

Equivalently, $\phi(G_1) \neq \phi(G_1)$ if G_1 and G_2 are not isomorphic.

- If a graph kernel is not complete, then there is cannot cover all possible functions over X: the kernel is not expressive enough.
- On the other hand, kernel computation must be tractable, i.e., no more than polynomial (with small degree) for practical applications.
- Can we define tractable and expressive graph kernels?

Computing any complete graph kernel is at least as hard as the graph isomorphism problem. (Gärtner et al., 2003)

SUBGRAPH KERNEL

Let $\lambda(G)_{G\in X}$ a set or **nonnegative** real-valued weights

For any graph G ∈ X, let

$$\forall H \in X, \qquad \phi_H(G) = |\text{G}' \text{ is a subgraph of G : G}' \cong \text{H}$$

The **subgraph kernel** between any two graphs G_1 and G_2 ∈ X is defined by:

$$K_{subgraph}(G_1, G_2) = \sum_{H\in X} \lambda_H \, \phi_H(G_1)\phi_H(G_2)$$

NOTE: Computing the subgraph kernel is NP-hard. (Gärtner et al., 2003)

GRAPH KERNELS TERMINOLOGY CONT.

subtree patterns (also called ***tree-walks***, Bach, 2008) can have nodes that are equal .

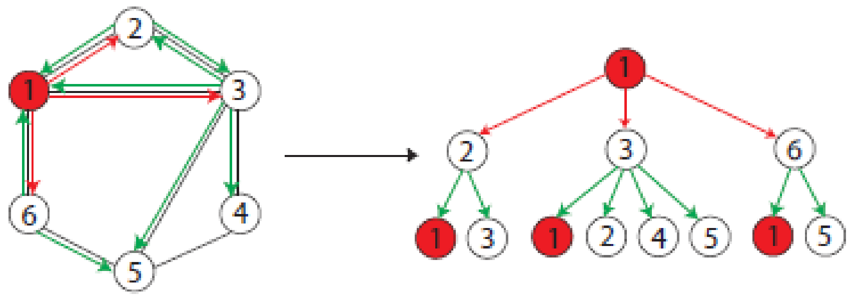


Figure 1: A subtree pattern of height 2 rooted at the node 1. Note the repetitions of nodes in the unfolded subtree pattern on the right.

Note that all **subtree kernels** compare **subtree** ***patterns*** in two graphs, not (strict) subtrees.

PATH KERNEL

A **path** of a graph (V,E) is sequence of **distinct vertices**

$v_1, \ldots, v_n \in V$ $(i \neq j \Rightarrow v_i \neq v_j)$ such that $(v_i, v_{i+1}) \in E$ for i = 1, . . . , n − 1.

Equivalently the paths are the **linear subgraphs.**

The **path kernel** is the subgraph kernel restricted to paths, i.e.,

$$K_{path}(G_1, G_2) = \sum_{H \in P} \lambda_H \, \phi_H(G_1)\phi_H(G_2)$$

where P ⊂ X is the set of path graphs.

NOTE: Computing the path kernel is NP-hard. (Gärtner et al., 2003)

EXPRESSIVENESS VS COMPLEXITY TRADE-OFF

- It is **intractable** to compute **complete graph kernels.**
- It is **intractable** to compute the **subgraph kernels**.
- Restricting subgraphs to be linear does not help:
  - it is **intractable** to compute the **path kernel**.
- One approach to define polynomial time computable graph kernels is to have the feature space be made up of graphs **homomorphic to subgraphs**, e.g., to consider walks instead of paths.

THREE CLASSES OF GRAPH KERNELS

- Graph kernels based on walks and paths
  - Compute the number of matching pairs of random walks (resp. paths) in two graphs
  - **Random walk kernel** are generated by direct **product graph** of two graphs
  - Walks (Kashima et al., 2003; G¨artner et al., 2003)
  - Paths (Borgwardt and Kriegel, 2005),
- Graph kernels based on limited-size subgraphs
  - Kernels based on **graphlets**, that represent graphs as counts of all types (or certain type of) of subgraphs of size k ∈{3,4,5}.
  - (Horv´ath et al., 2004; Shervashidze et al., 2009),
- Graph kernels based on subtree patterns
  - Subtree kernels iteratively compares all matchings between neighbors of two nodes v from G and v' from G'. In other words, for all pairs of nodes v from G and v' from G', it counts all pairs of matching substructures in subtree patterns rooted at v and v'.
  - (Ramon and G¨artner, 2003; Mah´e and Vert, 2009)

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

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 $G_\times = (V_\times, E_\times)$
- Each walk in the product graph corresponds to one walk in G and G'

$$k_\times(G, G') = \sum_{i,j=1}^{|V_\times|} \Big[\sum_{k=0}^{\infty} \lambda^k A_\times^k\Big]_{ij}$$

Runtime in $O(n^6)$

Some proposed speed up:

- 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(Kriegel, ICDM 2005)

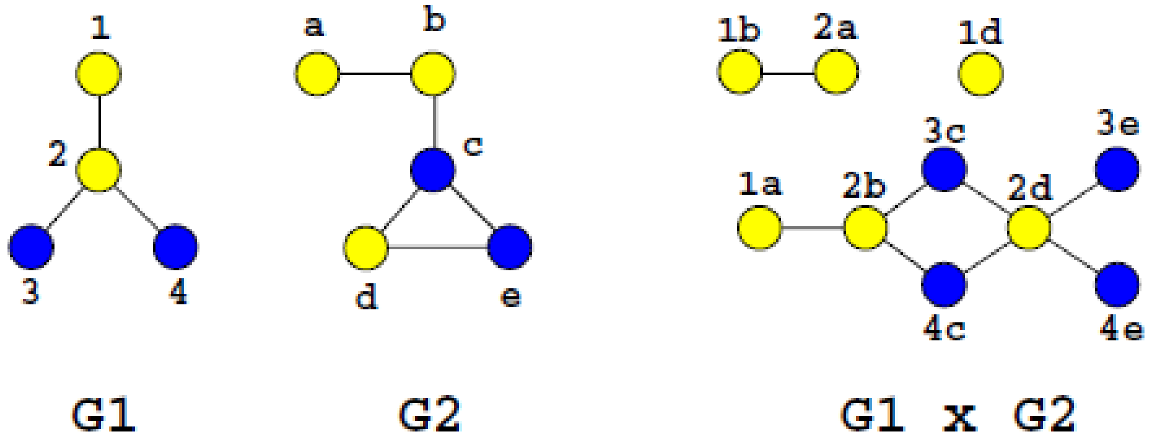
PRODUCT GRAPH

Let $G1 = (V1, E1)$ and $G2 = (V2, E2)$ be two graphs with labeled vertices. The $product\ graph\ G = G1 \times G2$ is the graph G = (V,E) with:

$$V = \{(v_1, v_2) \in V_1 \times V_2 : v_1 \text{ and } v_2 \text{ have the same label}\},$$

$$E = \{((v_1, v_2), (v_1', v_2')) \in V\text{x}V : (v_1, v_1') \in E_1\ and (v_2, v_2') \in E_2\}$$

.

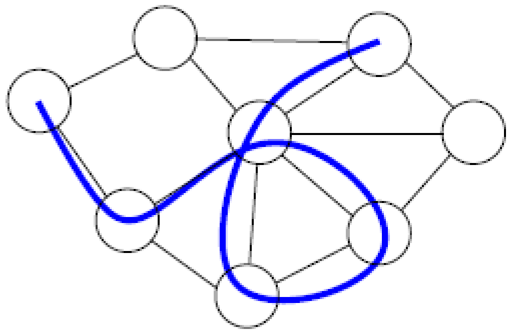


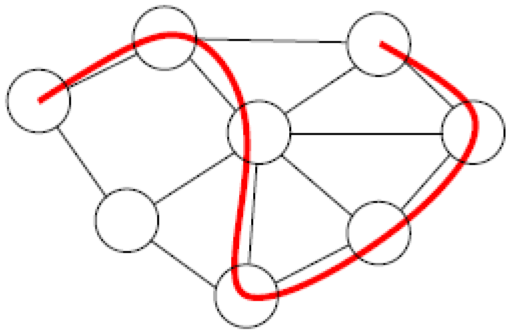
- Product graph consists of pairs of identically labeled nodes and edges from G1 and G2

WALKS

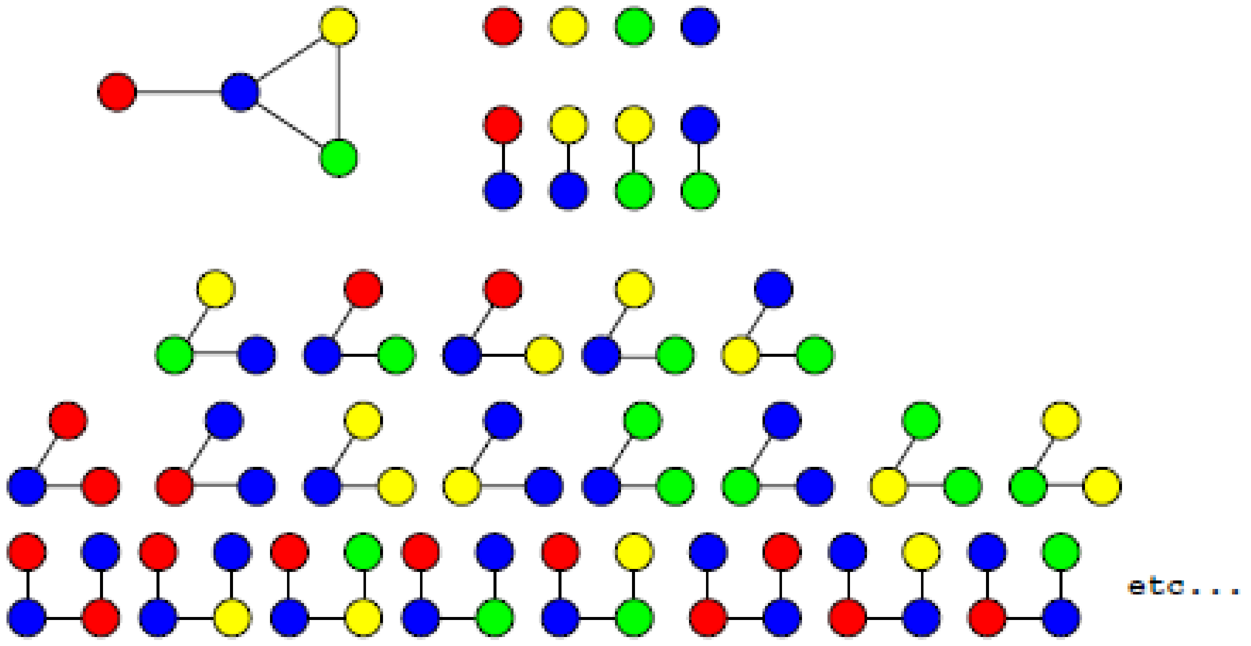
A **walk** of a graph (V,E) is sequence of $v_1, \dots, v_n \in V$ such that $(vi, vi+1) \in E$ for $i = 1, \dots, n-1$.

We note $\boldsymbol{W_n(G)}$ the set of walks with n vertices of the graph G, and $\boldsymbol{W(G)}$ the set of all walks.


walks


Paths

WALK KERNELS



WALK KERNEL

- Let $\boldsymbol{S_n}$ denote the set of all possible **label sequences** of walks of length n (including vertices and edges labels), and $S = \cup_{n\geq 1} S_n$.
- For any graph X let **a weight** $\lambda_G(w)$ be associated to each walk $w \in W(G)$.
- Let the feature vector $\phi(G) = (\phi_s(G))_{s\in S}$ be defined by:

$$\phi_s(G) = \sum_{w\in W(G)} \lambda_G(w)\mathbf{1}\,(s \text{ is the label sequence of } w)\,.$$

- A **walk kernel** is a graph kernel defined by:

$$K_{walk}(G_1, G_2) = \sum_{s\in S} \phi_S(G_1)\phi_S(G_2)$$

- 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

WALK KERNEL EXAMPLES

- The **nth-order walk kernel** is the walk kernel with $\lambda_G(w)$ = 1 if the length of w is n, 0 otherwise. It compares two graphs through their common walks of length n.
- The **random walk kernel** is obtained with $\lambda_G(w) = P_G(w)$, where P_G is a Markov random walk on G. In that case we have:

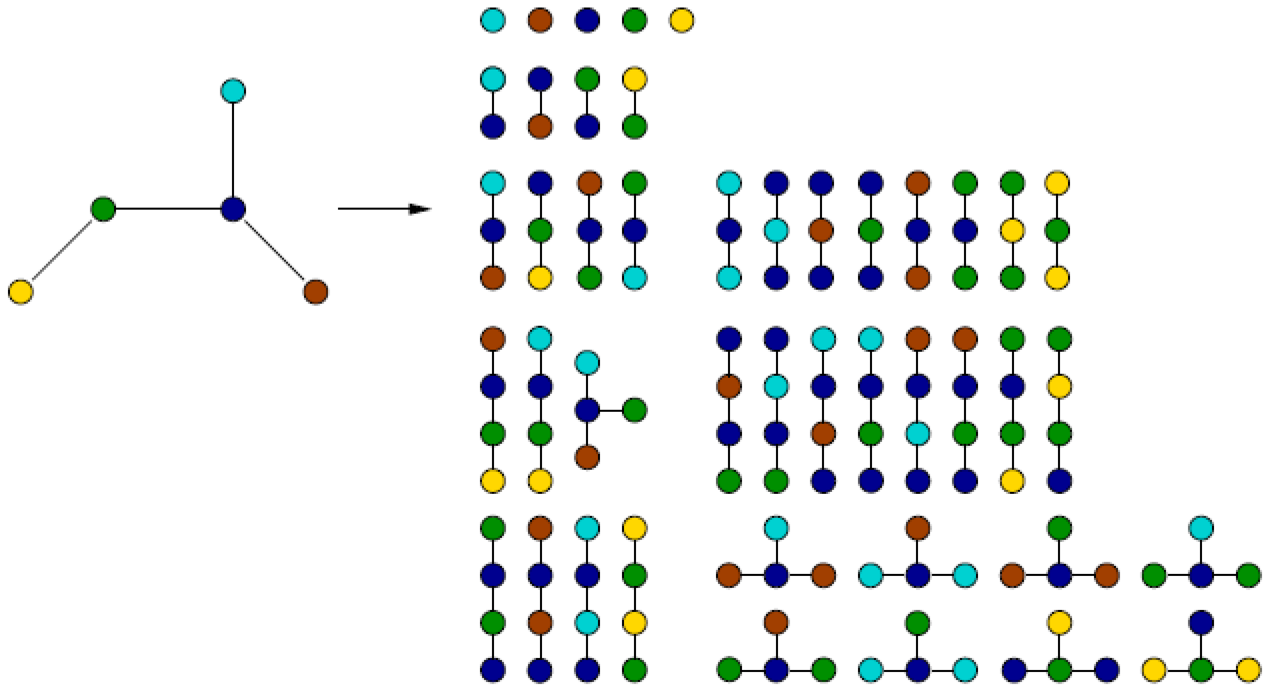
  $$K(G_1, G_2) = P(label(W_1) = label(W_2)),$$

  where W_1 and W_2 are two independent random walks on G_1 and G_2, respectively (Kashima et al., 2003).
- The **geometric walk kernel** is obtained (when it converges) with $\lambda_G(w) = \beta^{length(w)}$, for β > 0. In that case the feature space is of **infinite dimension** (Gärtner et al., 2003).

These three kernels (nth-order, random and geometric walk kernels) can be computed efficiently in **polynomial time.**

SUBTREE KERNELS

Like the walk kernel, amounts to compute the (weighted) number of subtrees in the product graph.



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 V1(G1) and s from V2(G2) 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

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

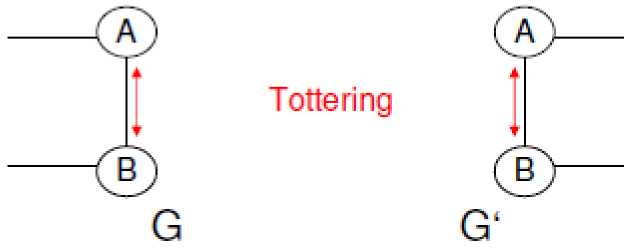
- Shortest paths need not be unique, but shortest path distances are
- Define graph kernel based on shortest path distances

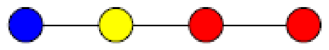
TOTTERING

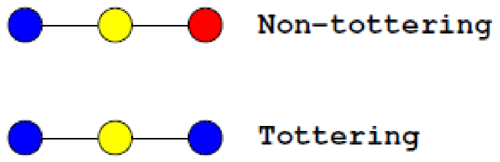
Tottering (Mahe et al., ICML 2004)

A **tottering walk** is a walk $w = v_1 \dots v_n$ with $v_i = v_i + 2$ for some i.

- 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
- Focusing on non-tottering walks is a way to get closer to the path kernel (e.g., equivalent on trees).

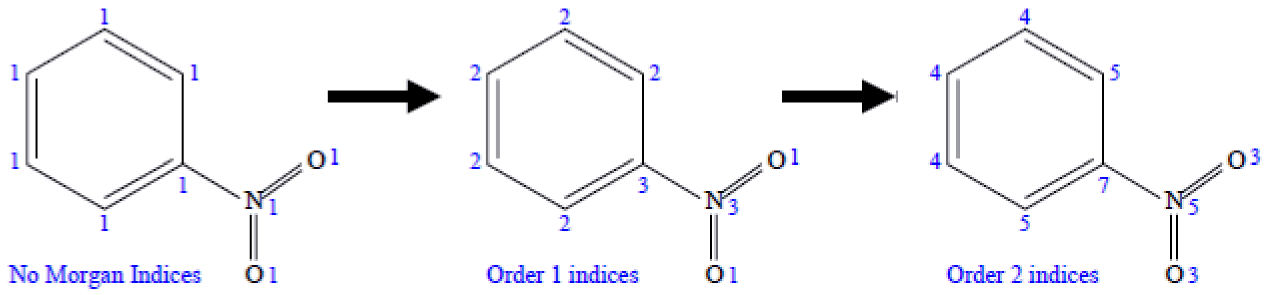






LABEL ENRICHMENT: MORGAN INDEX (1965)

- 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:


No Morgan Indices
Order 1 indices
Order 2 indices