Instructor: Sael Lee

CS549 Spring – Computational Biology

Random Walk Kernels and Other 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

Graph Comparison

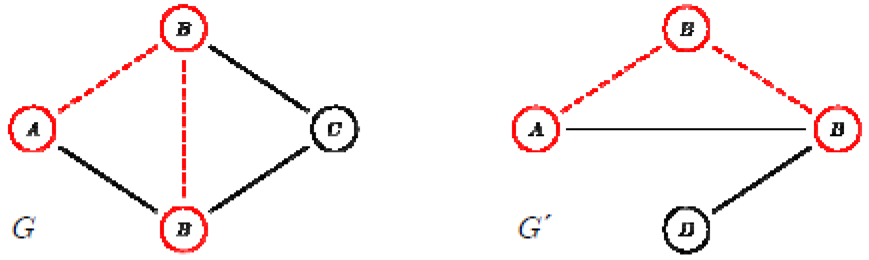
Graph Kernels aim at computing similarity scores between graphs in a dataset

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 Kernels Measuring Graph Similarity

Principle

- Let ϕ(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).

What is a Graph Kernel?

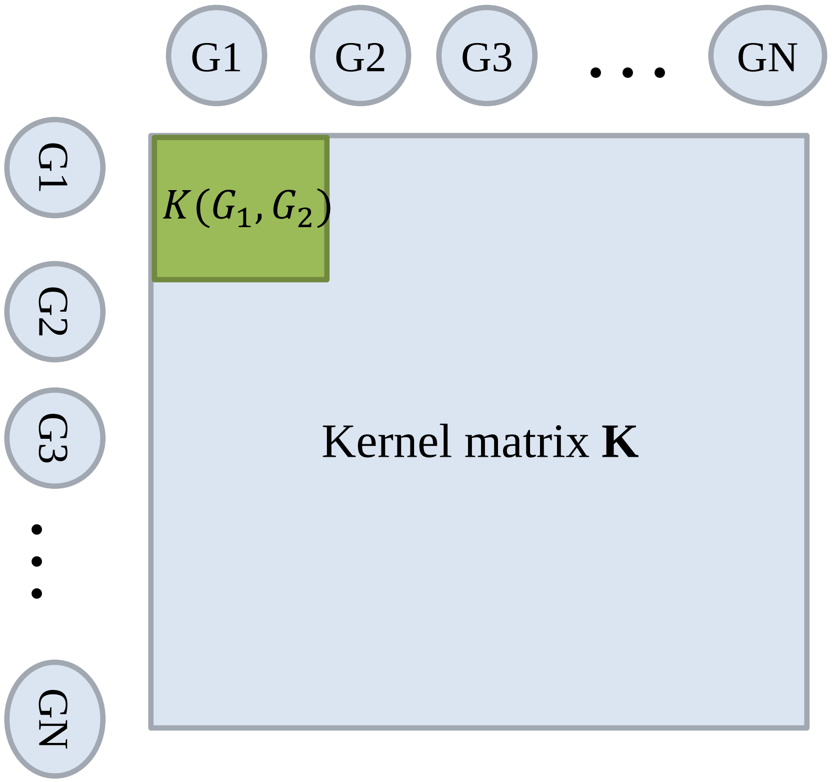
Graph kernels are Instance of **R-convolution** kernels by Haussler (1999)

***R*-convolution** is a generic way of defining kernels on discrete compound objects by <u>comparing all pairs of decompositions thereof</u>.
Therefore, a new type of decomposition of a graph results in a new graph kernel.

A graph kernel makes the whole family of kernel methods applicable to graphs

<u>Generation of complete decompositions of graph is as hard as subgraph isomorphism !!</u>

Graph Kernels



How to define a **valid kernel** function $K(G_j, G_j)$, between two graphs G_j and G_j.

- $K(G_j, G_j)$ should provide relationship (similarity / dissimilarity / correlation etc.) measure for between two graphs.
- $K(G_j, G_j)$ should be able to be applied in kernel based machine learning methods such that it provide optimal classification / clustering performance.

We will look at graph kernels that states similarity between kernels.

Graph 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*.

Graph Terminology cont.

- 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.

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 it 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 Kernel Terminology cont.

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

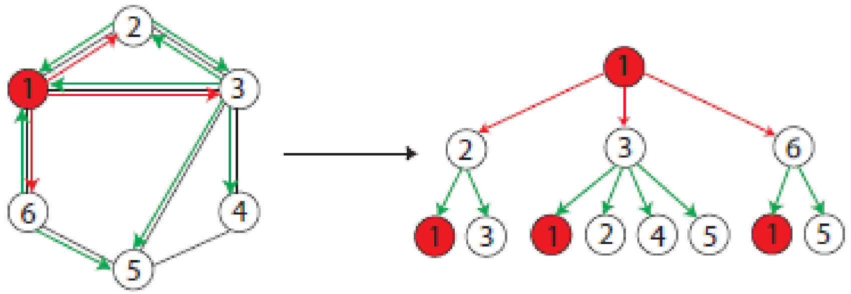


Figure 1: A subtree pattern of height 2 rooted at the node 1. Note the repetitions of nodes in the unfolded subtree pattern on the right.

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 <u>consider walks instead of paths.</u>

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),

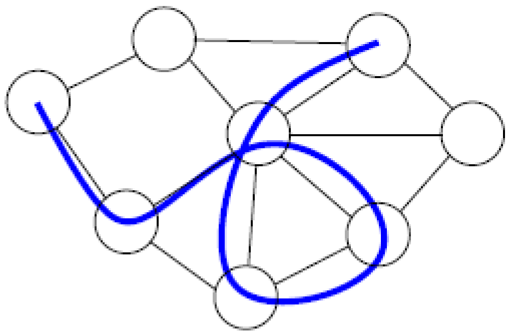
Three classes of graph kernels cont.

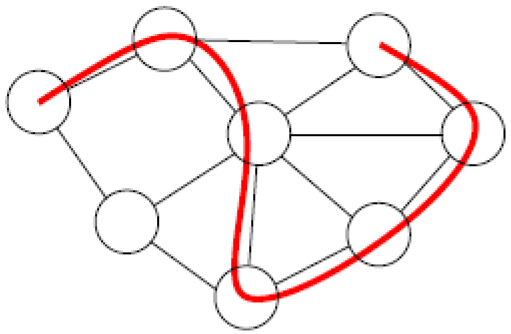
- Graph kernels based on subtree patterns
  - Subtree kernels iteratively compares all matchings between neighbors of two nodes v from G and v’ from G’. In other words, for all pairs of nodes v from G and v’ from G’, it counts all pairs of matching substructures in subtree patterns rooted at v and v’.
  - (Ramon and G¨artner, 2003; Mah´e and Vert, 2009)

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 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:

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

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:

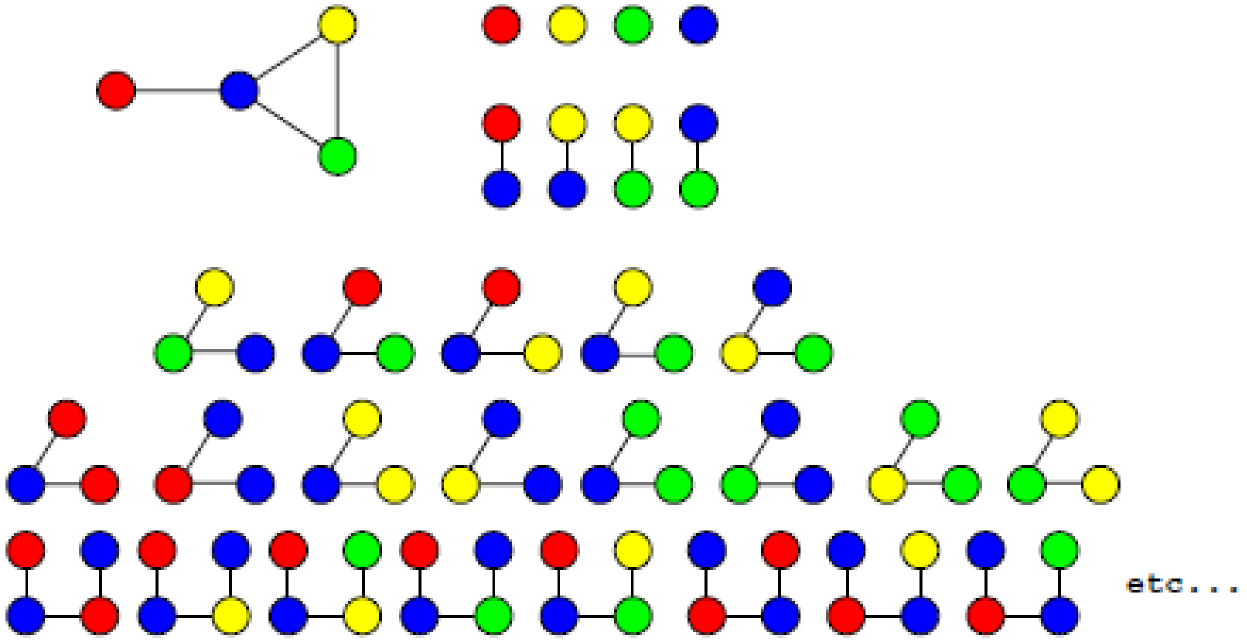
$$K(G_1, G_2) = P(label(W_1) = label(W_2)),$$

  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 $\beta > 0$. In that case the feature space is of **infinite dimension** (Gärtner et al., 2003).

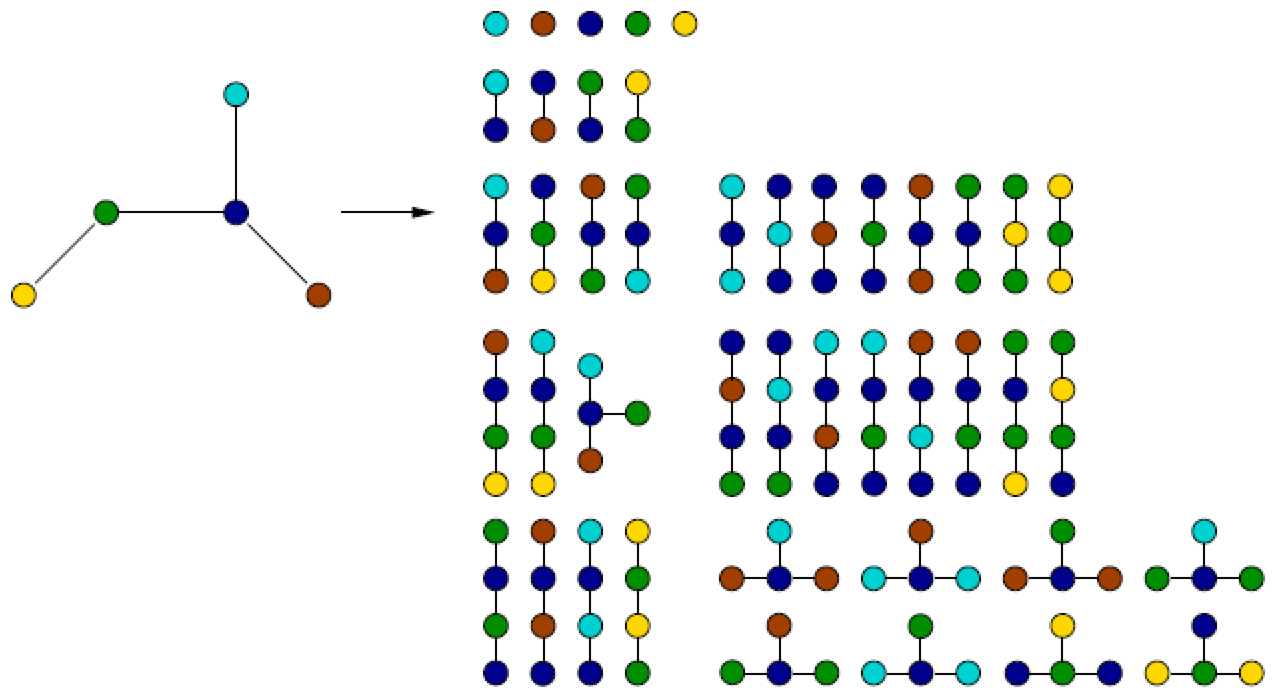
These three kernels (nth-order, random and geometric walk kernels) can be computed efficiently in **polynomial time.**

Walk Kernel Example



Subtree Kernels

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



Subtree Kernels

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

Marginalized Kernels Between Labeled Graphs

(Kashima et al., ICML 2003)

Marginalized Kernels

- Assume **hidden variables h** (ex> walk of a graph) and make use of the probability distribution of **visible variables x, x'** (structured data ex> Graph) and hidden variables

Marginalized Kernels: Expectation of the joint kernel over all possible values of *h* and *h'*

posterior probability

$$K(\boldsymbol{x},\boldsymbol{x}') = \sum_{h}\sum_{h'} K_z(\boldsymbol{z},\boldsymbol{z}')\boldsymbol{p}(\boldsymbol{h}|\boldsymbol{x})p(\boldsymbol{h}|\boldsymbol{x}')$$

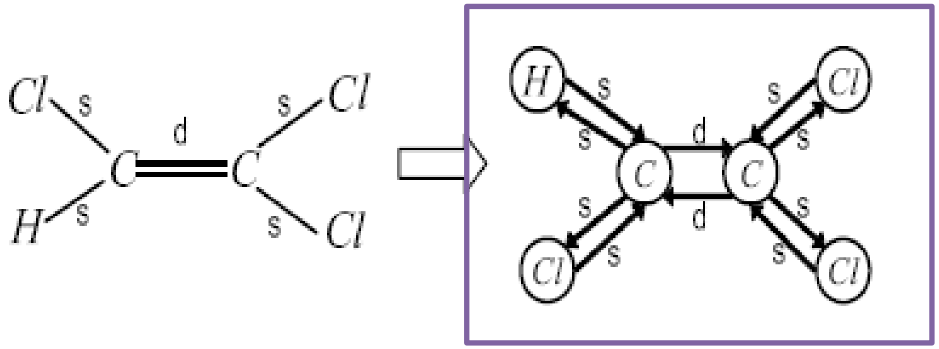
joint kernel & *z* = [*x*;*h*]

posterior probability *p*(*h*|*x*) can be interpreted as a **feature extractor** that extracts informative features for classification from *x*

Note: Undirected Graph to Directed Graph

- A **graph** G =(V, E, l),
  - V is the set of vertices,
  - $E \subset (V \times V)$ is the set of undirected edges (Changed to directed for random walk), and
  - $l : V, E \rightarrow \Sigma$ is a function that assigns labels from an alphabet Σ to nodes in the graph.

Changing undirected graph to directed graph



G =(V, E, l),

- 's' and 'd' denote single and double bonds, respectively.
- Kernel assumes a directed graph, undirected edges are replaced by directed edges

First Order Markov Random Walks on Graphs

Hidden variable: Random Walks on Graphs

$$K(\mathbf{x},\mathbf{x}') = \underbrace{\sum_{\mathbf{h}}\sum_{\mathbf{h}'}} K_z(\mathbf{z},\mathbf{z}')p(\mathbf{h}|\mathbf{x})p(\mathbf{h}'|\mathbf{x}').$$

- Hidden variable $\boldsymbol{h} = (h_1, \ldots, h_l)$ associated with graph G is a sequence of natural numbers from 1 to |G|.

|G| : number of vertices

- *h* is generated by a random walk

uniform distribution can be used for uninformative prior

1-st step) h_1 is sampled from the **prior probability distribution** $p_s(\boldsymbol{h})$.

i-th step) h_i sampled subject to the **transition probability** $p_t(h_i|h_{i-1})$ and with **walk termination probability** $p_q(h_{i-1})$:

$$\sum_{j=1}^{|G|} p_t(j|i) + p_q(i) = 1.$$

- Posterior probability for the walk ***h* : *p(h|G)***

$$p(h|G) = p_s(h_1)\prod_{i=2}^{\ell} p_t(h_i|h_{i-1})p_q(h_\ell),$$ *where* l is the length of *h*

- traversed labels are listed: $v_{h_1} e_{h_1 h_2} v_{h_2} e_{h_2 h_3} v_{h_3} \cdots$

Define Joint Kernel

$$K(\boldsymbol{x},\boldsymbol{x}') = \sum_{\boldsymbol{h}}\sum_{\boldsymbol{h}'} \boxed{K_z(\boldsymbol{z},\boldsymbol{z}')}\, p(\boldsymbol{h}|\boldsymbol{x})p(\boldsymbol{h}'|\boldsymbol{x}').$$

Define vertex kernel & edge kernel

Assume that two kernel functions are readily defined:

- *K*(v, v') : *Kernel* between vertex labels
- *K*(e, e'): *Kernel* between edge labels,

Example of the vertex label kernels

Dirac kernel: For Discrete labels

$$K(v,v') = \delta(v = v'),$$

Gaussian kernel: For Real value labels

$$K(v,v') = \exp(-\| v - v' \|^2 / 2\sigma^2)$$

Constrain both kernels to be nonnegative

$$K(v,v') \geq 0;\ K(e,e') \geq 0$$

Joint Kernel

$$K_z(z,z') = \begin{cases} 0 & (\ell \neq \ell') \\ K(v_{h_1}, v'_{h'_1}) \prod_{i=2}^{\ell} K(e_{h_{i-1}h_i}, e'_{h'_{i-1}h'_i}) \times \\ \qquad K(v_{h_i}, v'_{h'_i}) & (\ell = \ell') \end{cases}$$

where $z = (G, h)$.

Computing Joint Kernel

$$
\begin{aligned}
&K(G,G') \\
&= \sum_{\ell=1}^{\infty} \sum_{\boldsymbol{h}} \sum_{\boldsymbol{h}'} p_s(h_1) \prod_{i=2}^{\ell} p_t(h_i|h_{i-1}) p_q(h_l) \times \\
&\qquad p'_s(h'_1) \prod_{j=2}^{\ell} p'_t(h'_j|h'_{j-1}) p'_q(h'_\ell) \times \\
&\quad K(v_{h_1}, v'_{h'_1}) \prod_{k=2}^{\ell} K(e_{h_{k-1}h_k}, e'_{h'_{k-1},h'_k}) K(v_{h_k}, v'_{h'_k}),
\end{aligned}
$$

Where $\sum_{\boldsymbol{h}} := \sum_{h_1=1}^{|G|} \cdots \sum_{h_\ell=1}^{|G|}$

The straightforward enumeration is **impossible**, because l spans from 1 to infinity.

Computing Joint Kernel cont.

$$\begin{aligned} K(G,G') &= \sum_{h_1,h_1'} s(h_1,h_1') \lim_{L\to\infty} \sum_{\ell=1}^{L} r_\ell(h_1,h_1') \\ &= \sum_{h_1,h_1'} s(h_1,h_1') \lim_{L\to\infty} R_L(h_1,h_1'), \end{aligned}$$

$$\begin{aligned} & r_\ell(h_1,h_1') \\ & := \left(\sum_{h_2,h_2'} t(h_2,h_2',h_1,h_1') \left(\sum_{h_3,h_3'} t(h_3,h_3',h_2,h_2') \times \right.\right. \\ & \left.\left. \left(\cdots \left(\sum_{h_\ell,h_\ell'} t(h_\ell,h_\ell',h_{\ell-1},h_{\ell-1}') q(h_\ell,h_\ell')\right)\right)\cdots\right), \end{aligned}$$

$$\left\{ \begin{aligned} & \ell \geq 2 \\ s(h_1,h_1') &:= p_s(h_1)p_s'(h_1')K(v_{h_1},v'_{h_1'}) \\ t(h_i,h_i',h_{i-1},h_{i-1}') &:= p_t(h_i|h_{i-1})p_t'(h_i'|h_{i-1}') \times \\ & \quad K(v_{h_i},v'_{h_i'})K(e_{h_{i-1}h_i},e_{h_{i-1}'h_i'}) \end{aligned} \right.$$

$$\left\{ \begin{aligned} q(h_\ell,h_\ell') &:= p_q(h_\ell)p_q'(h_\ell') \\ r_1(h_1,h_1') &:= q(h_1,h_1'). \\ R_L(h_1,h_1') &:= \sum_{\ell=1}^{L} r_\ell(h_1,h_1'). \end{aligned} \right.$$

Computing Joint Kernel cont.

Restate this problem in **recursive form**

$$r_\ell(h_1,h_1') := \left(\sum_{h_2,h_2'} t(h_2,h_2',h_1,h_1')\left(\sum_{h_3,h_3'} t(h_3,h_3',h_2,h_2')\times \left(\cdots\left(\sum_{h_\ell,h_\ell'} t(h_\ell,h_\ell',h_{\ell-1},h_{\ell-1}')q(h_\ell,h_\ell')\right)\right)\cdots\right)\right)$$

$$r_1(h_1,h_1') := q(h_1,h_1')$$

$$R_L(h_1,h_1') := \sum_{\ell=1}^{L} r_\ell(h_1,h_1').$$

$$r_k(h_1,h_1') = \sum_{i,j} t(i,j,h_1,h_1')r_{k-1}(i,j).$$

$$\begin{aligned} R_L(h_1,h_1') &= r_1(h_1,h_1') + \sum_{k=2}^{T} r_k(h_1,h_1') \\ &= r_1(h_1,h_1') + \sum_{k=2}^{T}\sum_{i,j} t(i,j,h_1,h_1')r_{k-1}(i,j) \\ &= r_1(h_1,h_1') + \sum_{i,j} t(i,j,h_1,h_1')R_{L-1}(i,j). \quad (\end{aligned}$$

Equilibrium equation:

$$R_\infty(h_1,h_1') = r_1(h_1,h_1') + \sum_{i,j} t(i,j,h_1,h_1')R_\infty(i,j)$$

Computing Joint Kernel cont.

computation of the marginalized kernel finally comes down to iteratively solving for

$$R_L(h_1,h_1') = r_1(h_1,h_1') + \sum_{k=2}^{T} r_k(h_1,h_1')$$

$$r_k(h_1,h_1') = \sum_{i,j} t(i,j,h_1,h_1') r_{k-1}(i,j).$$

$$= r_1(h_1,h_1') + \sum_{k=2}^{T}\sum_{i,j} t(i,j,h_1,h_1') r_{k-1}(i,j)$$

$$= r_1(h_1,h_1') + \sum_{i,j} t(i,j,h_1,h_1') R_{L-1}(i,j). \;($$

until convergence starting from

$$R_1(h_1,h_1') = r_1(h_1,h_1') \;:=\; q(h_1,h_1')$$

$$q(h_\ell,h_\ell') \;:=\; p_q(h_\ell)p_q'(h_\ell')$$

Proof of convergence in Section 3.4 of Kashima et al., 2003

and substituting the solutions into

$$K(G,G') \;=\; \sum_{h_1,h_1'} s(h_1,h_1') \lim_{L\to\infty} \sum_{\ell=1}^{L} r_\ell(h_1,h_1')$$

$$\;=\; \sum_{h_1,h_1'} s(h_1,h_1') \lim_{L\to\infty} R_L(h_1,h_1'),$$

$$s(h_1,h_1') \;:=\; p_s(h_1)p_s'(h_1')K(v_{h_1},v_{h_1'}')$$

Extension to Marginalized Graph Kernel

(Mahé et al. ICML 2004)

Model: Marginalized Graph Kernel with **Dirac** joint kernel

Approaches:

1 Size of product graph affects runtime of kernel computation

- The **more node labels, the smaller the product graph**
- Trick: Introduce new artificial node labels

Iterative Label Enrichment:
Morgan Index (1965)

2 Focusing on non-tottering walks is a way to get closer to the path kernel

Reduce Tottering effect by
Using **2nd Order Markov Random Walk** instead of 1st order

Simplified Marginalized Graph Kernel

K: Marginalized graph kernel

$$K(\boldsymbol{x},\boldsymbol{x}') = \sum_{\boldsymbol{h}} \sum_{\boldsymbol{h}'} K_z(\boldsymbol{z},\boldsymbol{z}') p(\boldsymbol{h}|\boldsymbol{x}) p(\boldsymbol{h}'|\boldsymbol{x}').$$

$$K_z(\boldsymbol{z},\boldsymbol{z}') = \begin{cases} 0 & (\ell \neq \ell') \\ K(v_{h_1}, v'_{h'_1}) \prod_{i=2}^{\ell} K(e_{h_{i-1}h_i}, e'_{h'_{i-1}h'_i}) \times \\ \qquad K(v_{h_i}, v'_{h'_i}) & (\ell = \ell') \end{cases}$$

where $\boldsymbol{z} = (G, \boldsymbol{h})$.

Simplified by
1) not using edge kernel defined
2) Using Dirac vertex kernel

$$K(G,G') = \sum_{(\boldsymbol{h},\boldsymbol{h}') \in V^* \times V'^*} p(\boldsymbol{h}|G) p'(\boldsymbol{h}'|G') K_L(l(\boldsymbol{h}), l(\boldsymbol{h}'))$$

K_L: Dirac kernel between labeled sequence

$$p(v_1 \ldots v_n) = p_s(v_1) \prod_{i=2}^{n} p_t(v_i | v_{i-1}).$$

$$K_L(l,l') = \begin{cases} 1 & \text{if } l = l' \\ 0 & \text{otherwise} \end{cases}$$

$$\begin{cases} p_s(v) = p_0(v) p_q(v), \\ p_t(u|v) = \frac{1-p_q(v)}{p_q(v)} p_a(u|v) p_q(u). \end{cases}$$

Simplified Marginalized Graph Kernel in Matrix

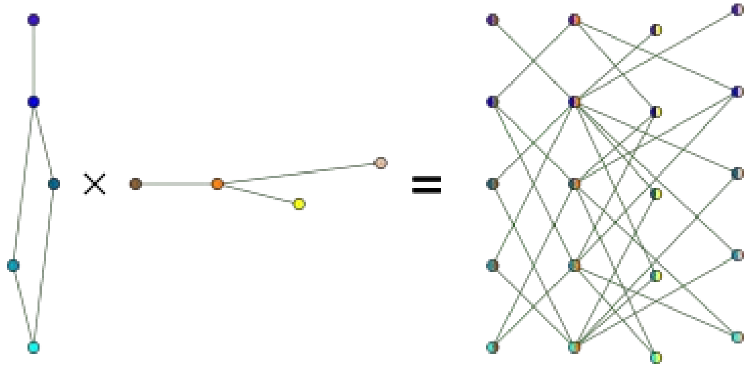
two labeled graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$

Tensor product graph is defined as labeled graph $G_p = (V_p, \mathrm{E}_p)$ with $V_p \subset V_1 \times V_2$ are pairs of vertices with identical labels

$$(v_1, v_2) \in V_p \text{ iff } l(v_1) = l(v_2)$$

and edges connecting the vertices

$$(u_1, u_2)\text{and}(v_1, v_2) \text{ iff } (u_i, v_i) \in E_p, for\ \ i = 1,2, \ldots l$$



$$\mathbf{A} \otimes \mathbf{B} = \begin{bmatrix}
a_{11}b_{11} & a_{11}b_{12} & \cdots & a_{11}b_{1q} & \cdots & \cdots & a_{1n}b_{11} & a_{1n}b_{12} & \cdots & a_{1n}b_{1q} \\
a_{11}b_{21} & a_{11}b_{22} & \cdots & a_{11}b_{2q} & \cdots & \cdots & a_{1n}b_{21} & a_{1n}b_{22} & \cdots & a_{1n}b_{2q} \\
\vdots & \vdots & \ddots & \vdots & & & \vdots & \vdots & \ddots & \vdots \\
a_{11}b_{p1} & a_{11}b_{p2} & \cdots & a_{11}b_{pq} & \cdots & \cdots & a_{1n}b_{p1} & a_{1n}b_{p2} & \cdots & a_{1n}b_{pq} \\
\vdots & \vdots & & \vdots & \ddots & & \vdots & \vdots & & \vdots \\
\vdots & \vdots & & \vdots & & \ddots & \vdots & \vdots & & \vdots \\
a_{m1}b_{11} & a_{m1}b_{12} & \cdots & a_{m1}b_{1q} & \cdots & \cdots & a_{mn}b_{11} & a_{mn}b_{12} & \cdots & a_{mn}b_{1q} \\
a_{m1}b_{21} & a_{m1}b_{22} & \cdots & a_{m1}b_{2q} & \cdots & \cdots & a_{mn}b_{21} & a_{mn}b_{22} & \cdots & a_{mn}b_{2q} \\
\vdots & \vdots & \ddots & \vdots & & & \vdots & \vdots & \ddots & \vdots \\
a_{m1}b_{p1} & a_{m1}b_{p2} & \cdots & a_{m1}b_{pq} & \cdots & \cdots & a_{mn}b_{p1} & a_{mn}b_{p2} & \cdots & a_{mn}b_{pq}
\end{bmatrix}.$$

G H G × H

tensor product of agency matrix G × H

Fig: http://en.wikipedia.org/wiki/Tensor_product_of_graphs

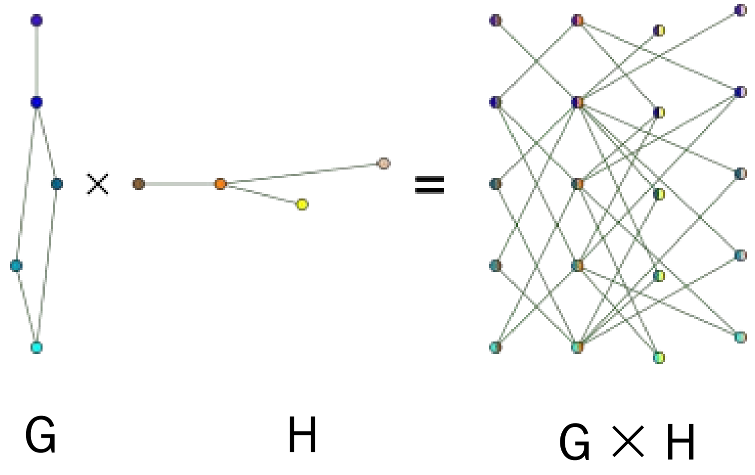
Simplified Marginalized Graph Kernel in Matrix

A function π on the set of walks(paths) $H(G_p)$

$$\pi\left((u_1,v_1)(u_2,v_2)\ldots(u_n,v_n)\right) = \pi_s(u_1,v_1)\prod_{i=2}^{n}\pi_t\left((u_i,v_i)|(u_{i-1},v_{i-1})\right),$$

with

$$\begin{cases}\pi_s(u_1,u_2) = p_s^{(1)}(u_1)p_s^{(2)}(u_2),\\ \pi_t((v_1,v_2)|(u_1,u_2)) = p_t^{(1)}(v_1|u_1)p_t^{(2)}(v_2|u_2),\end{cases}$$



π_t

$$\mathbf{A}\otimes\mathbf{B} = \begin{bmatrix} a_{11}b_{11} & a_{11}b_{12} & \cdots & a_{11}b_{1q} & \cdots & \cdots & a_{1n}b_{11} & a_{1n}b_{12} & \cdots & a_{1n}b_{1q} \\ a_{11}b_{21} & a_{11}b_{22} & \cdots & a_{11}b_{2q} & \cdots & \cdots & a_{1n}b_{21} & a_{1n}b_{22} & \cdots & a_{1n}b_{2q} \\ \vdots & \vdots & \ddots & \vdots & & & \vdots & \vdots & \ddots & \vdots \\ a_{11}b_{p1} & a_{11}b_{p2} & \cdots & a_{11}b_{pq} & \cdots & \cdots & a_{1n}b_{p1} & a_{1n}b_{p2} & \cdots & a_{1n}b_{pq} \\ \vdots & \vdots & & \vdots & \ddots & & \vdots & \vdots & & \vdots \\ \vdots & \vdots & & \vdots & & \ddots & \vdots & \vdots & & \vdots \\ a_{m1}b_{11} & a_{m1}b_{12} & \cdots & a_{m1}b_{1q} & \cdots & \cdots & a_{mn}b_{11} & a_{mn}b_{12} & \cdots & a_{mn}b_{1q} \\ a_{m1}b_{21} & a_{m1}b_{22} & \cdots & a_{m1}b_{2q} & \cdots & \cdots & a_{mn}b_{21} & a_{mn}b_{22} & \cdots & a_{mn}b_{2q} \\ \vdots & \vdots & \ddots & \vdots & & & \vdots & \vdots & \ddots & \vdots \\ a_{m1}b_{p1} & a_{m1}b_{p2} & \cdots & a_{m1}b_{pq} & \cdots & \cdots & a_{mn}b_{p1} & a_{mn}b_{p2} & \cdots & a_{mn}b_{pq} \end{bmatrix}.$$

Simplified Marginalized Graph Kernel in Matrix cont.

$$K(G_1, G_2) = \sum_{(\boldsymbol{h_1}, h_2) \in V_1{}^* \times V_2{}^*} p_1(\boldsymbol{h_1}|G_1) p_2(\boldsymbol{h_2}|G_1) K_L(l(\boldsymbol{h_1}), l(\boldsymbol{h_2}))$$

$$K(G_1, G_2) = \sum_{h \in H(\mathcal{G})} \pi(h).$$

$$\sum_{h \in H(\mathcal{G}), |h|=n} \pi(h) = \pi_s^\top \Pi_t^n \mathbf{1},$$

$$K(G_1, G_2) = \sum_{n=1}^{\infty} \left(\sum_{h \in H(\mathcal{G}), |h|=n} \pi(h) \right) = \pi_s^\top (I - \Pi_t)^{-1} \mathbf{1}.$$

Label Enrichment with Morgan Index (1965)

Problems:

- The computation of graph kernels is time-consuming.
- Need to increase the relevance of the features used to compare graphs.

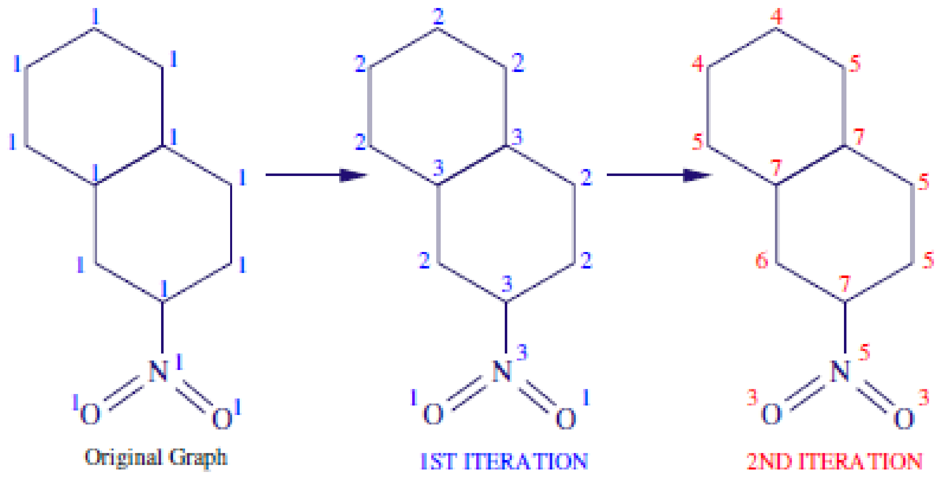
Expected outcome:

- The computation of graph kernels is time-consuming.
- Need to increase the relevance of the features used to compare graphs.

Label Enrichment with Morgan Index cont.

Enrichment with vertex connectivity properties

→ **extended connectivity descriptor :**



Algorithm :

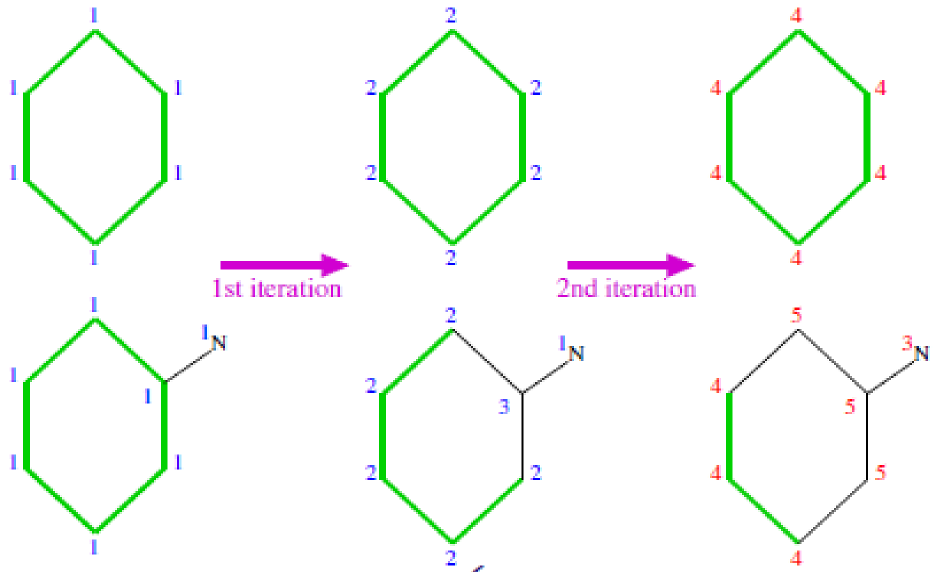
- $M_0(v) = 1, \; \forall v$
- $M_t(v) = \sum_{\text{neig}(v)} M_{t-1}(u)$
- $\Rightarrow$ New label : $l_t(v) = l(v) \circ M_t(v)$

Label Enrichment with Morgan Index cont.

M_n: vector of labels in graph

Given adjacency matrix A and setting $M_0 = \mathbf{1}$

$M_{n+1} = (A + I)M_n$

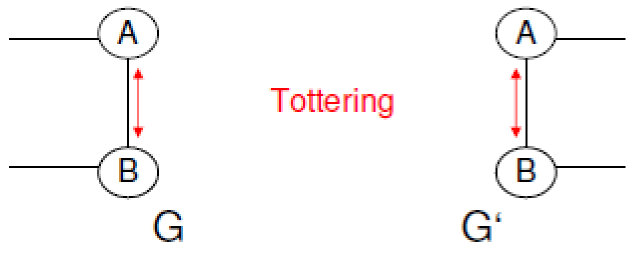


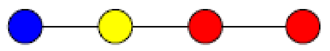
$\Rightarrow$ Family of kernels K_n : $\begin{cases} \text{decreased kernel complexity } (\mathcal{O}(|G_1 \times G_2|^3)) \\ \text{(potential) increased kernel expressivity} \end{cases}$

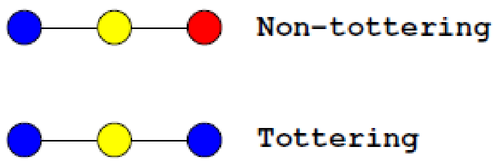
Preventing Tottering

A **tottering walk** is a walk $w = v_1 \ldots v_n$ 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).

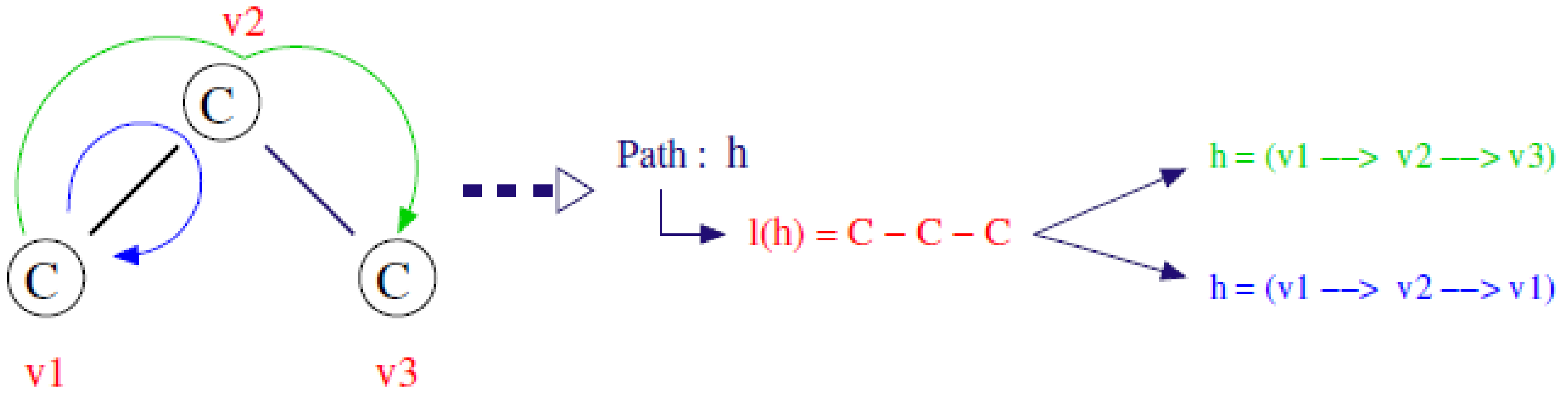






Preventing Tottering Cont.

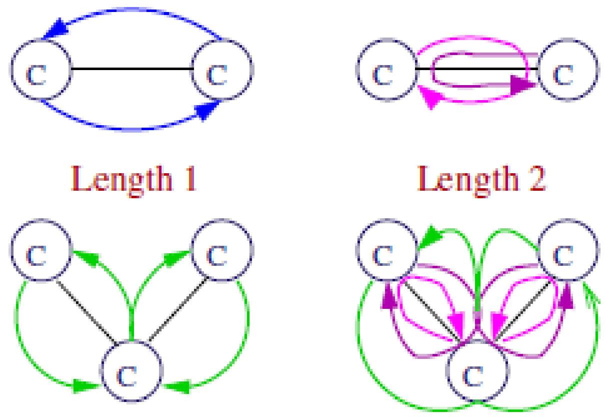
- Tottering path : $h = (v_1, \dots, v_n)$, $\exists\, i : v_{i+2} = v_i$.



$\Rightarrow$ preventing totters $\Leftrightarrow$ filtering blue path.

Preventing Tottering Cont.

- Motivation:

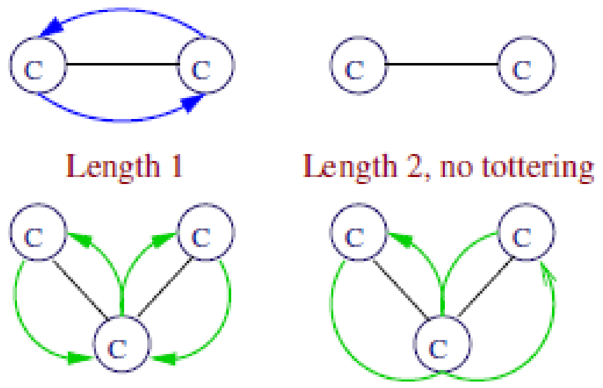


⋆ Every path of G_2 can be matched to a tottering path of G_1

⋆ $\Rightarrow$ Compounds are considered as identical

Preventing Tottering Cont.

- Motivation:



⋆ Only "real" chemical path are matched

⋆ ⇒ Compounds are now seen as different

- Solution : increase the order of the random walk model :
  $\Rightarrow p_G(h) = p_s(v_1)p_t(v_2|v_1)\prod_{i=3}^{n} p_t(v_i|v_{i-2}, v_{i-1})$

2nd order Markov Random Walk

$$p_G(h) = p_s(v_1)p_t(v_2|v_1)\prod_{i=3}^{n} p_t(v_i|v_{i-2}, v_{i-1})$$

$$\begin{cases} p_s(v) = p_0(v)p_q^{(0)}(v), \\ p_t(u|v) = \frac{1-p_q^{(0)}(v)}{p_q^{(0)}(v)} p_a(u|v)p_q(u), \\ p_t(u|w,v) = \frac{1-p_q(v)}{p_q(v)} p_a(u|w,v)p_q(u). \end{cases}$$

The function is still a valid kernel but the implementation described for the first order Markov random walk cannot be directly used anymore.

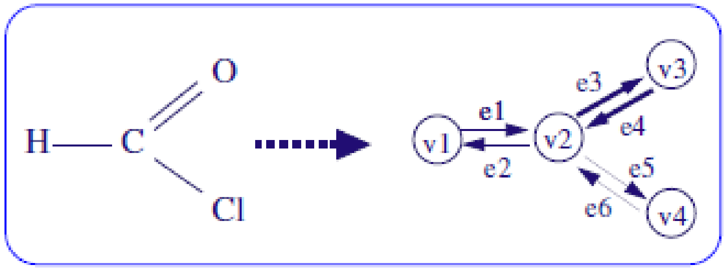
=> Instead of explicitly working with 2nd Order Markov Random walk, transform the original graph G to G' such that G' contains the look ahead information.

Graph Transformation Cont.

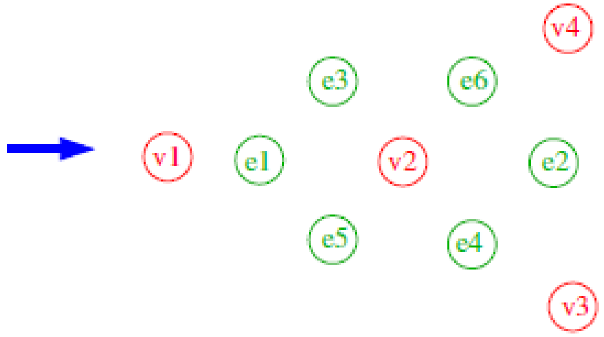
* Don't confuse G' used in the last notation for compared Graph

Transformation : $G = (V, E, l) \Rightarrow G' = (V', E', l')$ where :

- $V' = V \cup E$
- $E' = \{(v, (v, t)) \,|\, v \in V, (v, t) \in E\}$
  $\cup \{((u, v), (v, t)) \,|\, (u, v), (v, t) \in E, u \neq t\}$



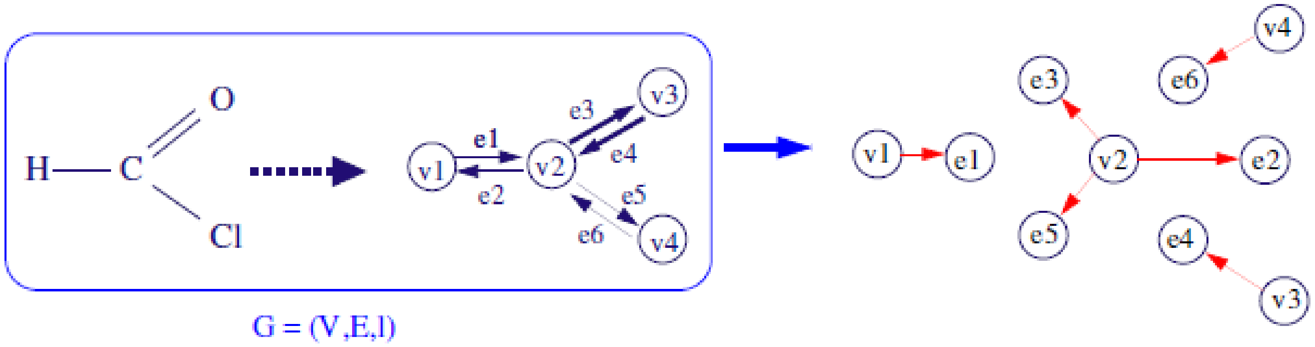
G = (V,E,l)



Graph Transformation Cont.

Transformation : $G = (V, E, l) \Rightarrow G' = (V', E', l')$ where :

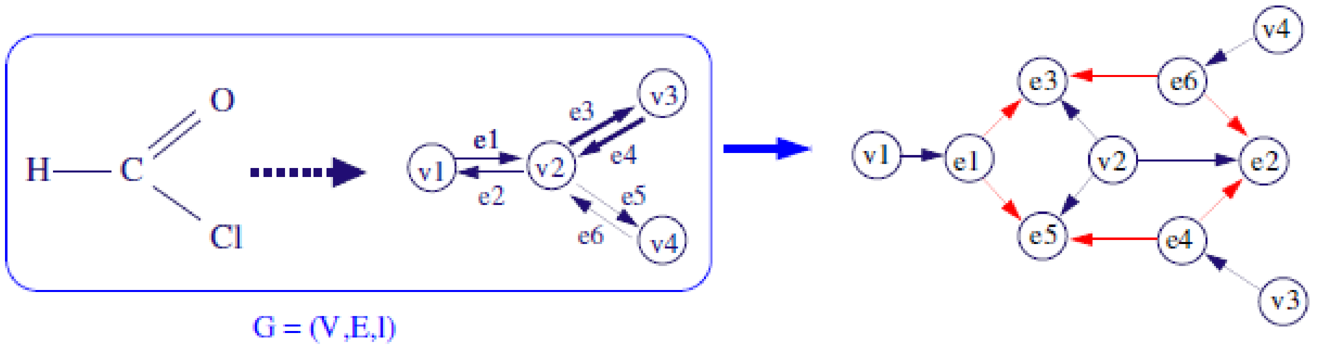
- $V' = V \cup E$
- $E' = \{(v, (v,t)) \mid v \in V, (v,t) \in E\}$
  $\cup \{((u,v), (v,t)) \mid (u,v), (v,t) \in E, u \neq t\}$



Graph Transformation Cont.

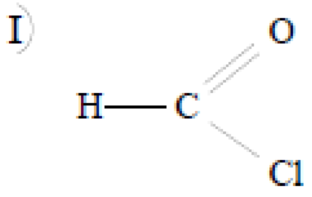
Transformation : $G = (V, E, l) \Rightarrow G' = (V', E', l')$ where :

- $V' = V \cup E$
- $E' = \{(v, (v,t)) \mid v \in V, (v,t) \in E\}$
  $\cup \; \{((u,v),(v,t)) \mid (u,v),(v,t) \in E, u \neq t\}$

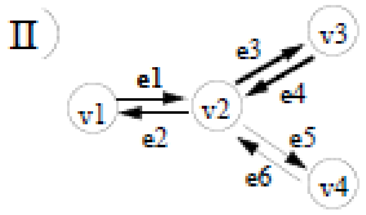


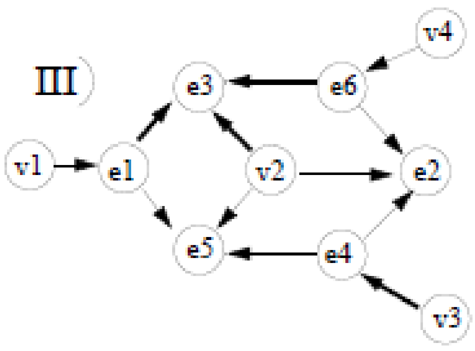
Graph Transformation Cont.

Original Graph

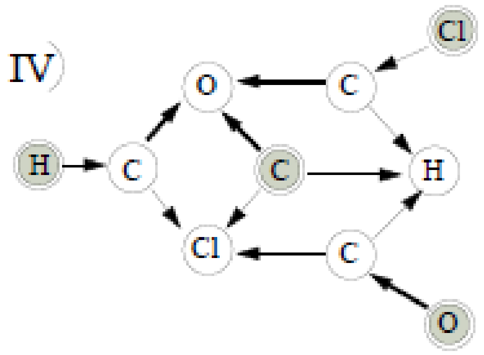


Corresponding directed graph G = (V,E,I)





Transformed Graph



Labels in the transformed graph

Modified Kernel Computation cont.

- Consider : $\begin{cases} H_0(G) = \{\text{Non tottering paths of G}\} \\ H_1(G') = \{\text{Paths of } G' \text{ starting from a node } v \in V\ \} \end{cases}$

- Theorem: p' factorizes as

$$p'(h') = p'_s(v'_1) \prod_{i=2}^{n} p'_t(v'_i | v'_{i-1})$$

  - ⋆ $p'_s(v') = p_s(v')$
  - ⋆ $p'_t(u'|v') = \begin{cases} p_t(u|v') \text{ if } v' \in V \text{ and } u' = (v', u) \in E \\ p_t(u|v, w) \text{ if } v' = (v, w) \text{ and } u' = (w, u) \in E \end{cases}$

- Corollary :

$$\left.\begin{array}{l} \text{- graph transformation} \\ \text{- original graph kernel} \end{array}\right\} \Rightarrow \text{tottering paths removed}$$

Modified Kernel Computation cont.

- Consider : $\begin{cases} H_0(G) = \{\text{Non tottering paths of G}\} \\ H_1(G') = \{\text{Paths of } G' \text{ starting from a node } v \in V\ \} \end{cases}$

- The mapping $f : H_0(G) \to H_1(G')$ defined by

$$h = (v_1, ..., v_n) \mapsto h' = (v'_1, ..., v'_n) \text{ such that } \begin{cases} v'_1 = v_1 \\ v'_i = (v_{i-1}, v_i) \end{cases}$$

establishes a bijection between $H_0(G)$ and $H_1(G')$

one-to-one correspondence

- Let p' be the image of p_G by f:

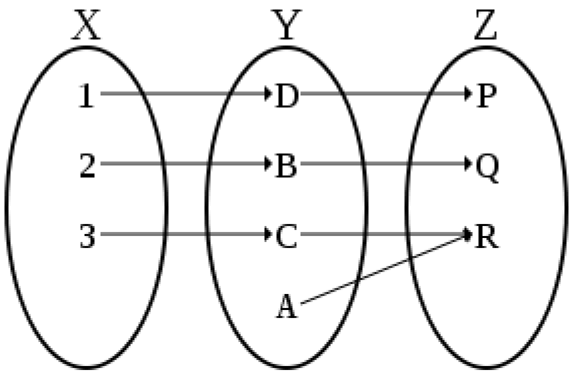
$$\forall h' \in H_1(G'), \quad p'(h') := p_G\left(f^{-1}(h')\right)$$

Review Bijection

http://en.wikipedia.org/wiki/Bijection

- **Bijection** (or bijective function or one-to-one correspondence) is a function giving an exact pairing of the elements of two sets.

- **Bijective function** *f*: *X* → *Y* is a **one to one** and **onto** mapping of a set *X* to a set *Y*.



A bijection composed of an injection (left) and a surjection (right).

Review Bijection cont.

Theorem 1. f is a **Bijective function** between $H_0(G)$ and $H_1(G')$, and for any path $\mathbf{h} \in H_0(G)$ we have

$$f: H_0(G) \to H_1(G')$$

$$\begin{cases} l(\boldsymbol{h}|G) = l'(f(\boldsymbol{h})|G') \\ p(\boldsymbol{h}|G) = p'(f(\boldsymbol{h})|G') \end{cases}$$

Corollary 1. For any two graphs G_1 and G_2, the marginalized graph kernel can be expressed in terms of the transformed graphs G'_1 and G'_2 by:

$$K(G_1, G_2) = \sum_{(h'_1, h'_2) \in (\Sigma'_1)^* \times (\Sigma'_2)^*} p'_1(h'_1) p'_2(h'_2)\, K_L(l'_1(h'_1) l'_2(h'_2))$$