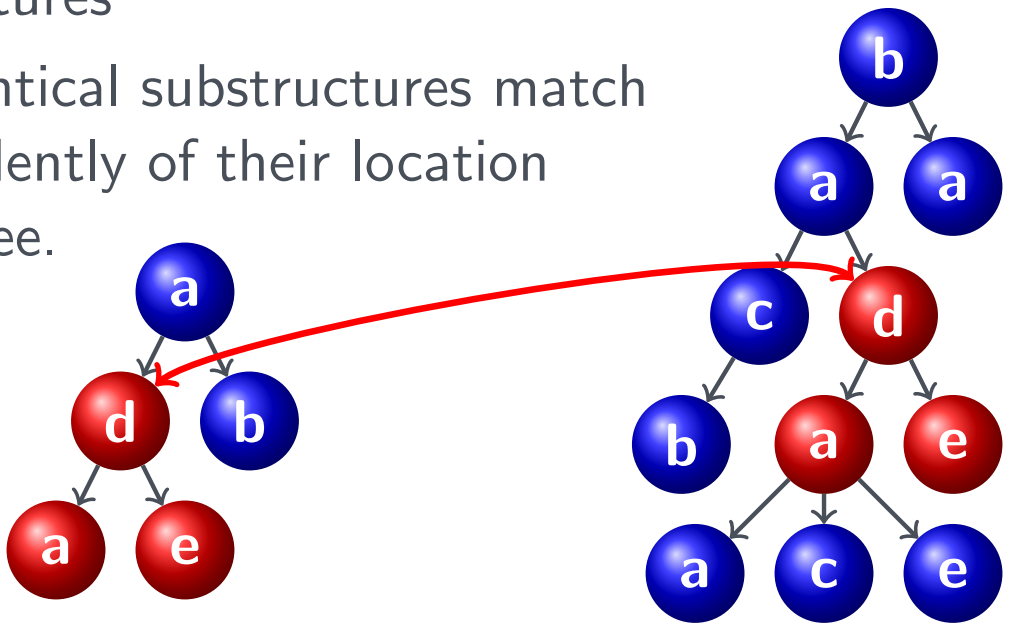
Loss of Structural Info in Tree Kernels

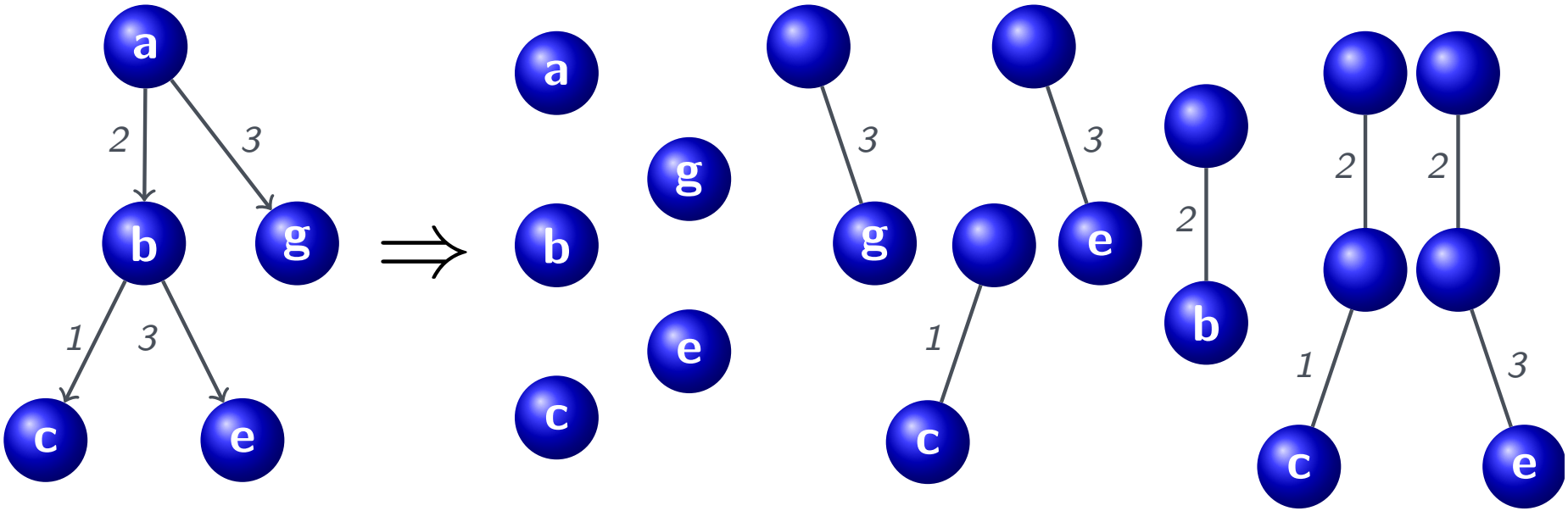
- Almost all tree kernels retain **only** structural properties of the substructures
- Two identical substructures match independently of their location in the tree.



- Routes π between pairs of nodes are considered. $\pi(v_1, v_2)$ is defined by the labels and the sequence of positions (with respect to the father) of the nodes comprising the path between v_1 and v_2.

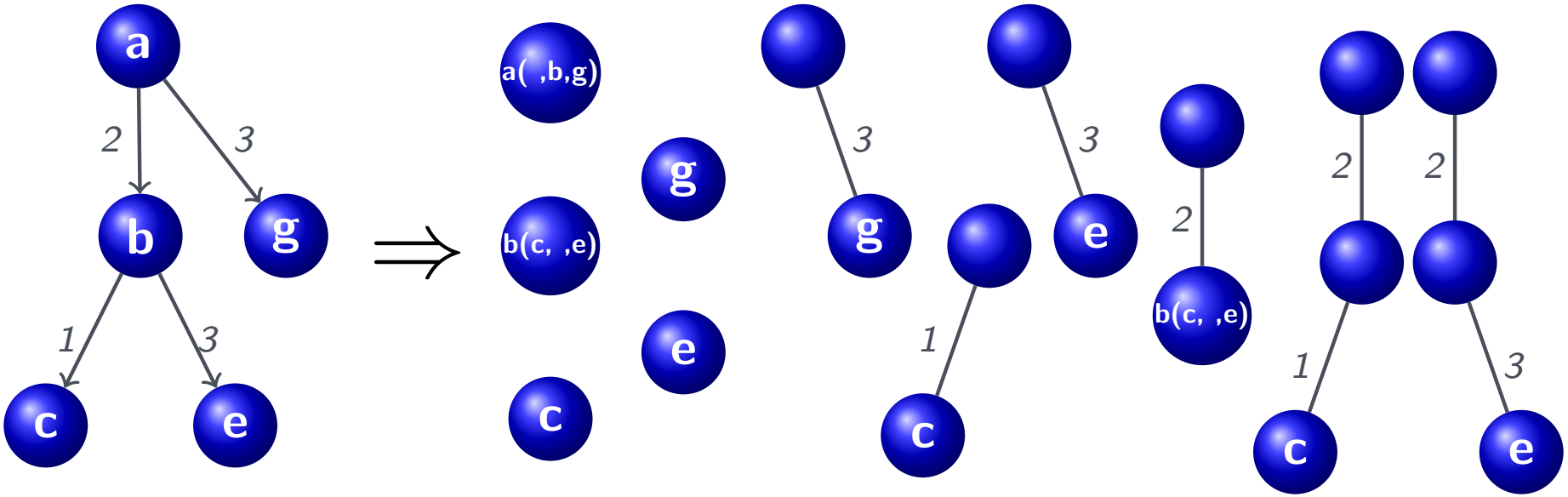
Route Kernel [19]: Example of Features

k_{label}: keep the label of the last node in the path.



Route Kernel [19]: Example of Features

k_{prod}: keep the production associated with the last node in the path.



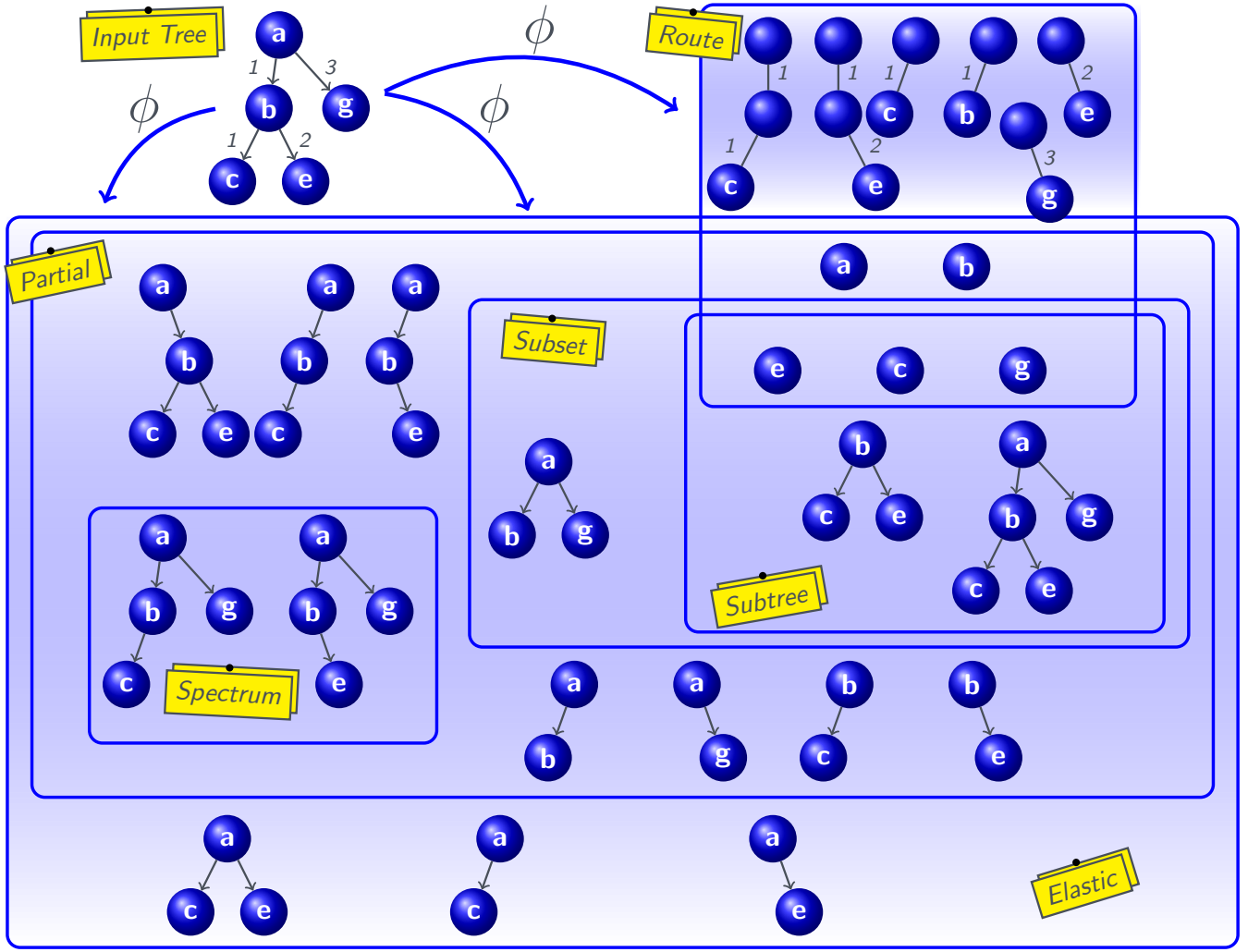
Route Kernel: Definition

- For each node, the routes between itself and all its ancestors up to the root are considered, thus a tree with m nodes is represented, at most, by $avgdepth(v_i) \cdot m$ features.
- An efficient algorithm for computing the route kernel is given: its computational complexity is $O(avgdepth(v_i) \cdot m + m \log m)$.
- Let $K_{route}(T_1, T_2)$ be the standard Route Kernel, an extended version matching also groups of routes can be easily defined as

$$K_{ext}(T_1, T_2) = (K_{route}(T_1, T_2))^d .$$

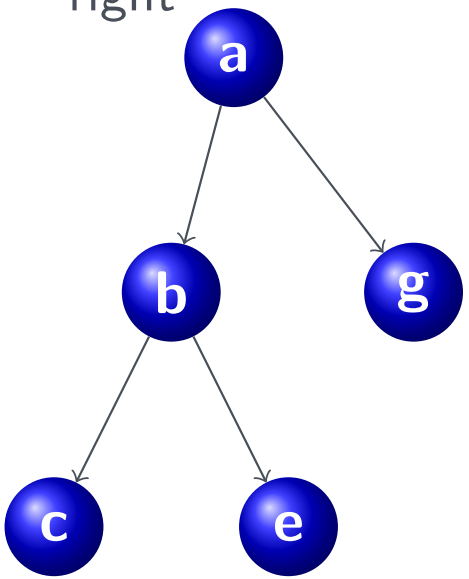
- For example if $d = 2$ then also pairs of routes are considered as features.

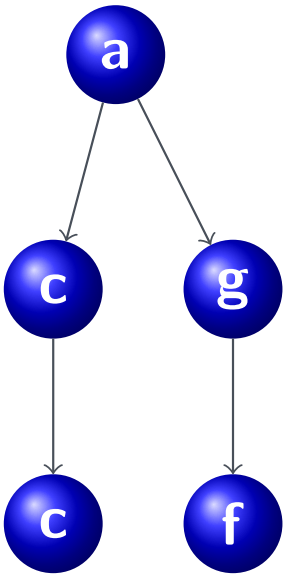
Relationships Among Feature Spaces



Tai Kernel [20]

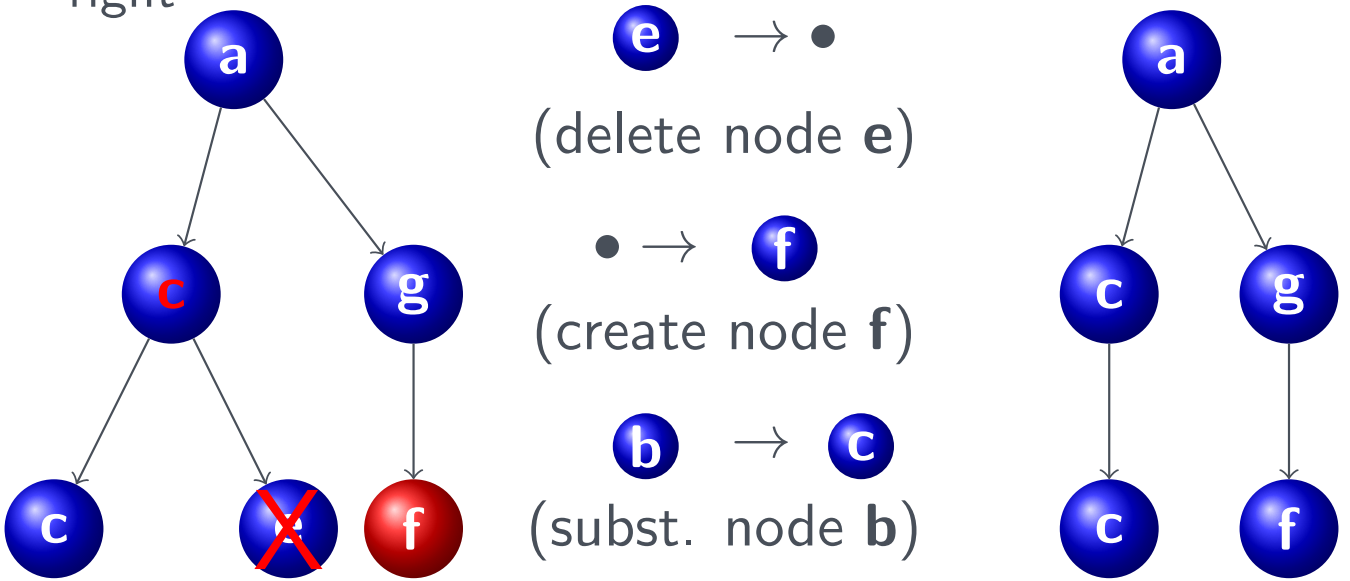
- Based on Tree Edit Distance
  - Associate a cost $\gamma()$ to $\gamma(v \rightarrow \bullet)$ (delete vertex), $\gamma(v_1 \rightarrow v_2)$ (substitute vertex) and $\gamma(\bullet \rightarrow v)$ (create vertex) operations
  - A script σ is a sequence of edit operations
  - Tai distance between T_1 and T_2 is the minimum over the costs of all scripts converting T_1 into T_2.
  - An example: convert the tree on the left to the one on the right





Tai Kernel [20]

- Based on Tree Edit Distance
  - Associate a cost $\gamma()$ to $\gamma(v \to \bullet)$ (delete vertex), $\gamma(v_1 \to v_2)$ (substitute vertex) and $\gamma(\bullet \to v)$ (create vertex) operations
  - A script σ is a sequence of edit operations
  - Tai distance between T_1 and T_2 is the minimum over the costs of all scripts converting T_1 into T_2.
  - An example: convert the tree on the left to the one on the right



Tai Kernel [20]

$$K(T_1, T_2) = \sum_{\sigma: T_1 \to T_2} e^{-\lambda\gamma(\sigma)} = \prod_{v_1 \in T_1} e^{-\gamma(v_1 \to \bullet)} \prod_{v_2 \in T_2} e^{-\gamma(\bullet \to v_2)} \cdot$$
$$\cdot \sum_{(\hat{x}_1, \hat{x}_2) \in \mathcal{M}^{\text{Tai}}_{T_1, T_2}} \prod_{i=1}^{|T_1|} \frac{e^{-\gamma(v_1^i \to v_2^i)}}{e^{-\gamma(v_1^i \to \bullet)} e^{-\gamma(\bullet \to v_2^i)}}$$

- The complexity of the kernel is $O(|T|^4)$.

Mapping Kernel Framework

The Convolution Kernel Framework

$$K(x_1, x_2) = \sum_{\hat{x}_1 \in \hat{\mathcal{X}}_{x_1}} \sum_{\hat{x}_2 \in \hat{\mathcal{X}}_{x_2}} k(\hat{x}_1, \hat{x}_2) = \sum_{(\hat{x}_1, \hat{x}_2) \in \hat{\mathcal{X}}_{T_1} \times \hat{\mathcal{X}}_{T_2}} k(\hat{x}_1, \hat{x}_2)$$

has been generalized by the Mapping Kernel Framework [21] [22]

$$K(x_1, x_2) = \sum_{(\hat{x}_1, \hat{x}_2) \in \mathcal{M}_{x_1, x_2}} k(\gamma_{x_1}(\hat{x}_1), \gamma_{x_2}(\hat{x}_2))$$

Mapping Kernel Framework

$$K(x_1, x_2) = \sum_{(\hat{x}_1, \hat{x}_2) \in \mathcal{M}_{x_1, x_2}} k(\gamma_{x_1}(\hat{x}_1), \gamma_{x_2}(\hat{x}_2))$$

The Mapping Kernel Framework extends the Convolution Kernel Framework in two aspects:

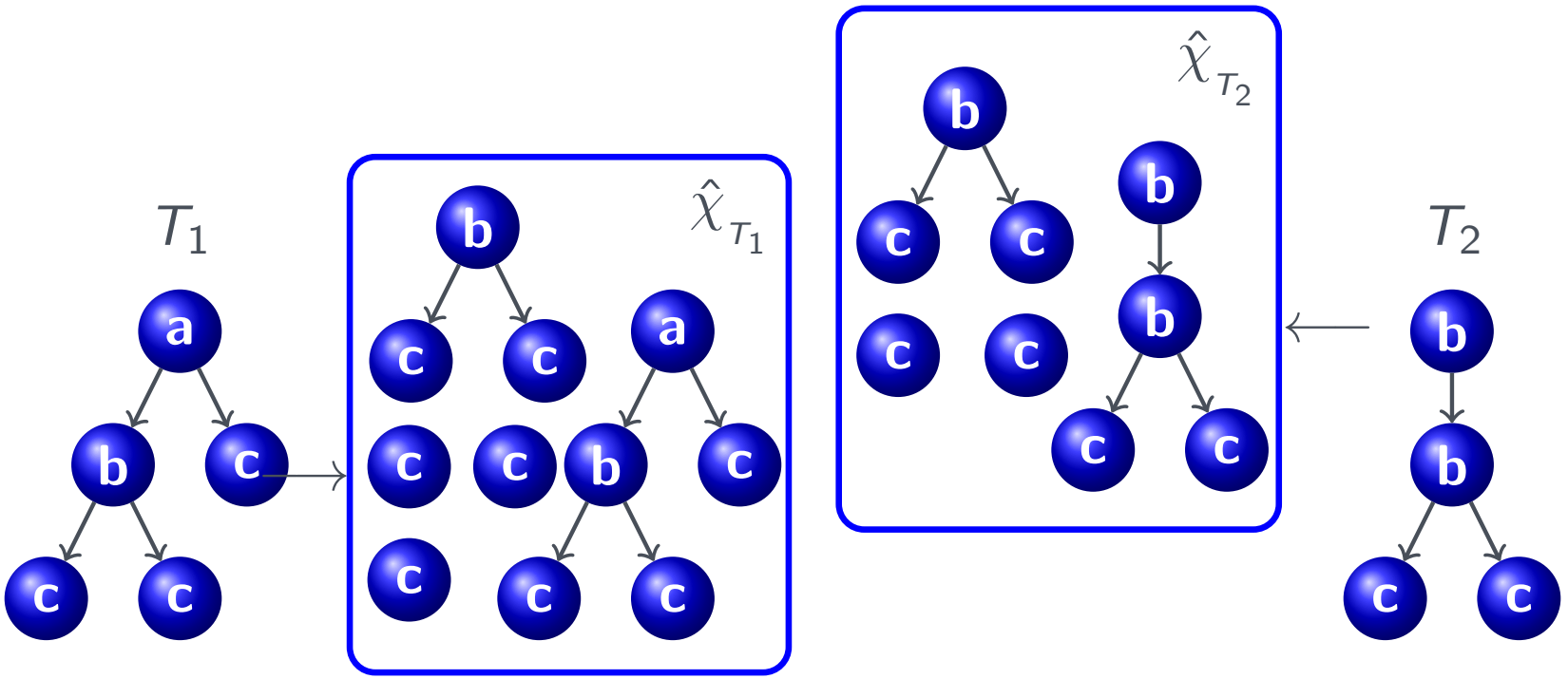
1. the pairs of substructures on which the local kernel has to be computed is restricted according to $\mathcal{M} \subseteq \hat{\chi}_{T_1} \times \hat{\chi}_{T_2}$
2. the functions $\gamma_x()$ allow $\hat{\chi}$ and the input space of the local kernel, $k()$, to not be identical.

Shin and Kuboyama [21] proved that, given a positive semidefinite kernel $k()$, $K()$ is positive semidefinite *if and only if* $\mathcal{M}$ is

- symmetric: $(\hat{x}_1, \hat{x}_2) \in \mathcal{M}_{x_1, x_2} \Leftrightarrow (\hat{x}_2, \hat{x}_1) \in \mathcal{M}_{x_2, x_1}$
- transitive: $(\hat{x}_1, \hat{x}_2) \in \mathcal{M}_{x_1, x_2} \quad \wedge \quad (\hat{x}_2, \hat{x}_3) \in \mathcal{M}_{x_2, x_3} \quad \Rightarrow \quad (\hat{x}_1, \hat{x}_3) \in \mathcal{M}_{x_1, x_3}$

Mapping Kernels: an Example

- Compute SST $C()$ function for all the subtrees at the same depth with respect to the root of the tree
  - $\hat{\chi}_T$ is the set of proper subtrees of T

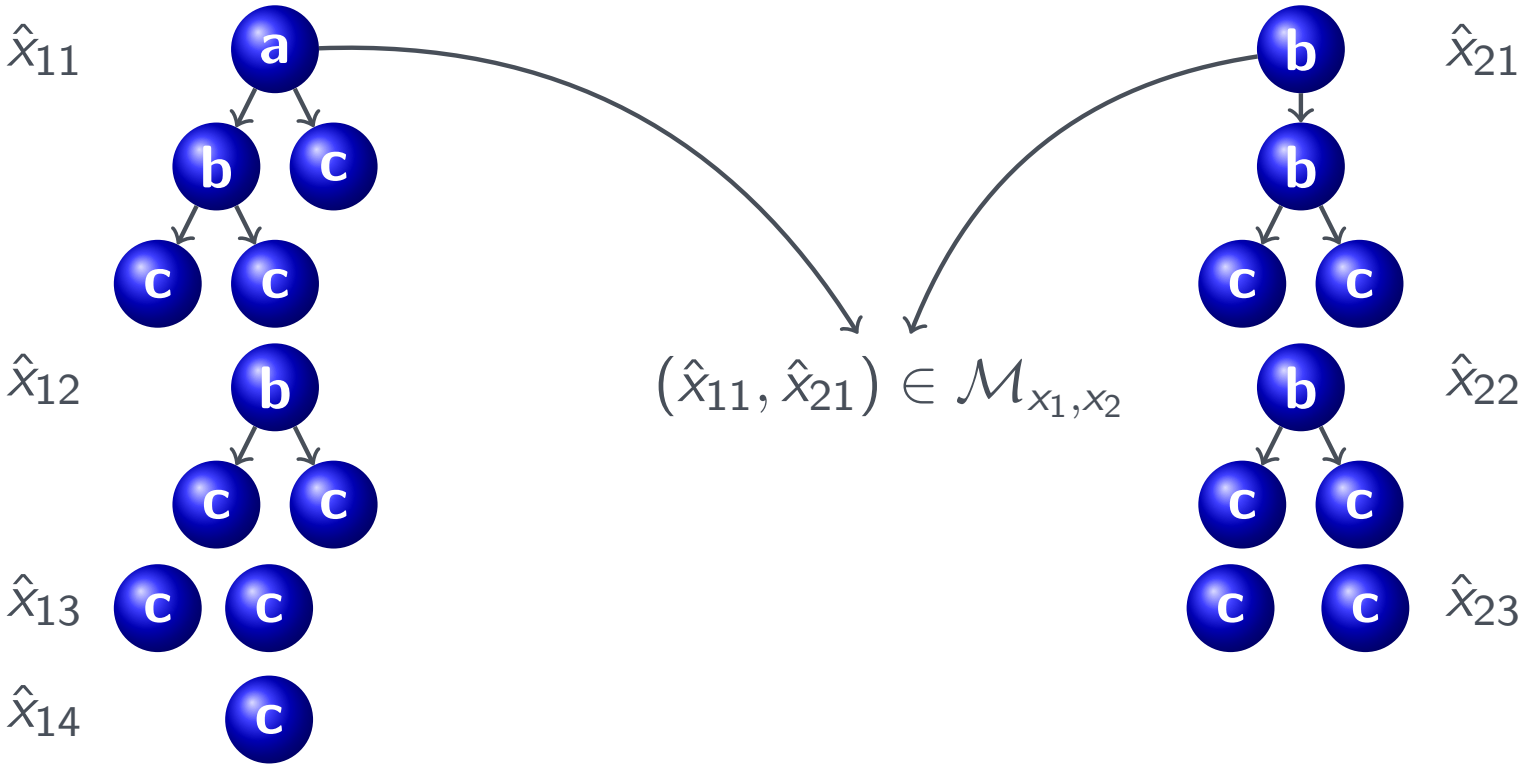


Mapping Kernels: an Example

- Compute SST $C()$ function for all the subtrees at the same depth with respect to the root of the tree
  - $\hat{\chi}_T$ is the set of proper subtrees of T
  - $(\hat{x}_1, \hat{x}_2) \in \mathcal{M}_{x_1,x_2} \Leftrightarrow depth(\hat{x}_1) = depth(\hat{x}_2)$

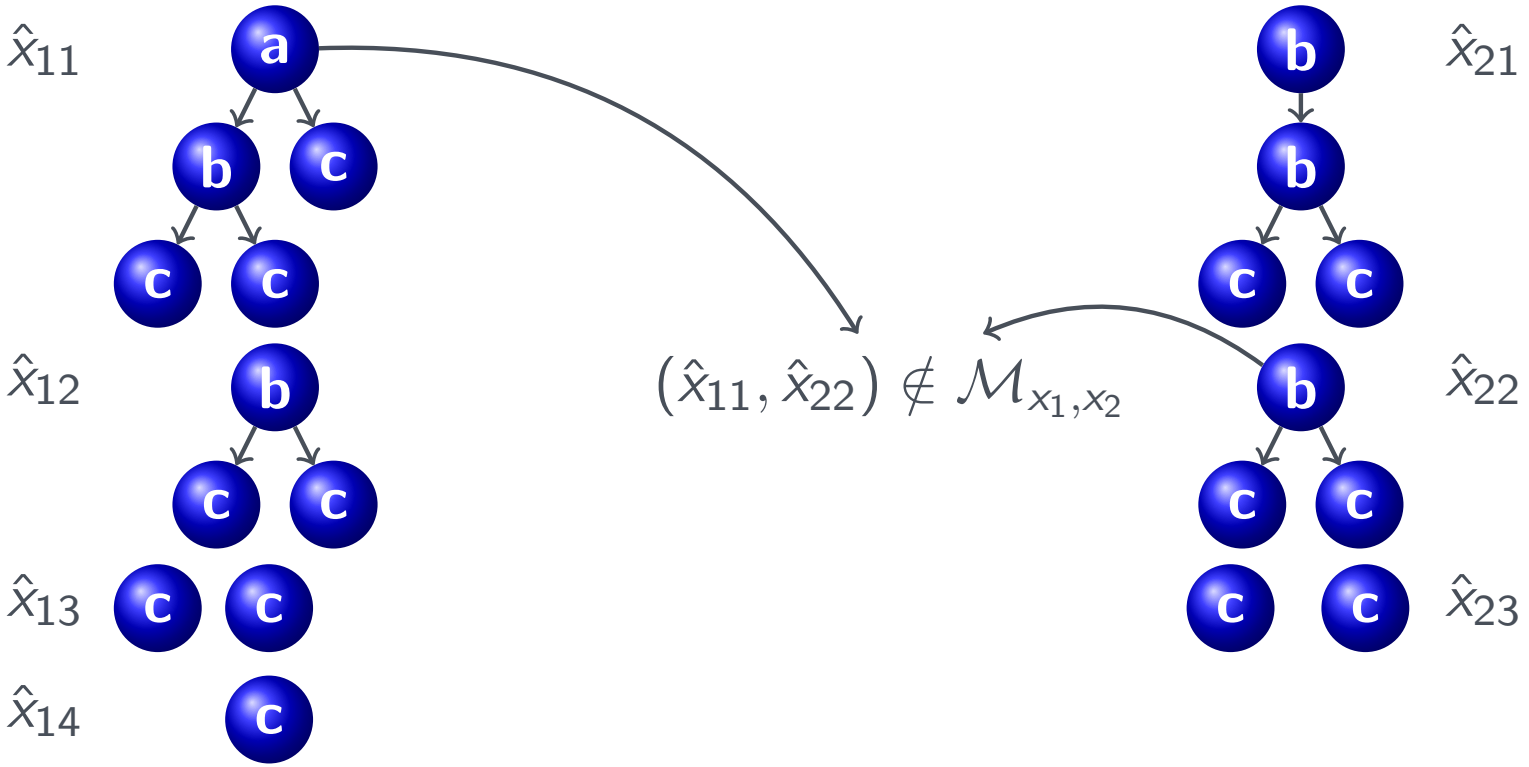
Mapping Kernels: an Example

- Compute SST $C()$ function for all the subtrees at the same depth with respect to the root of the tree
  - $\hat{\chi}_T$ is the set of proper subtrees of T
  - $(\hat{x}_1, \hat{x}_2) \in \mathcal{M}_{x_1,x_2} \Leftrightarrow depth(\hat{x}_1) = depth(\hat{x}_2)$



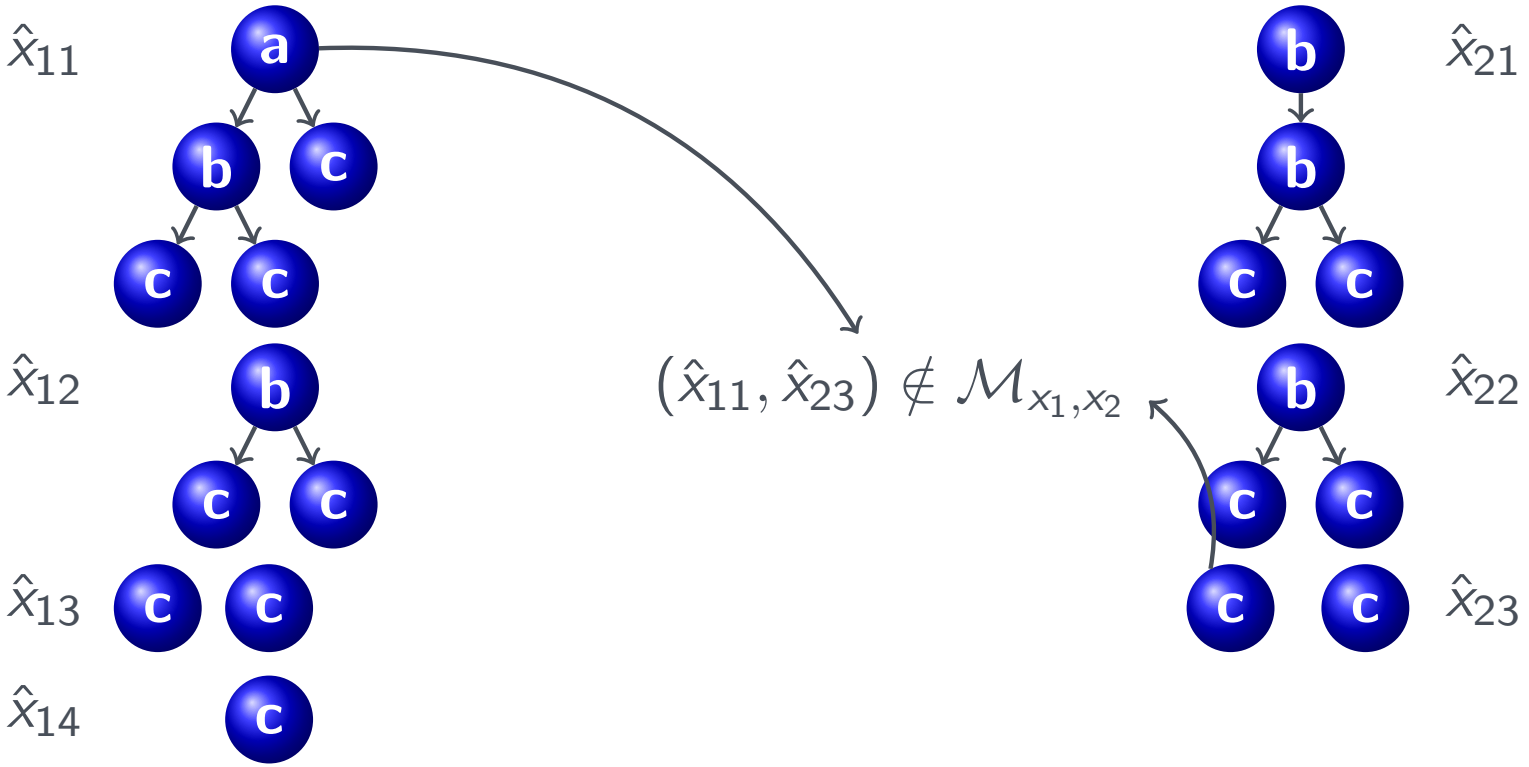
Mapping Kernels: an Example

- Compute SST $C()$ function for all the subtrees at the same depth with respect to the root of the tree
  - $\hat{\chi}_T$ is the set of proper subtrees of T
  - $(\hat{x}_1, \hat{x}_2) \in \mathcal{M}_{x_1,x_2} \Leftrightarrow depth(\hat{x}_1) = depth(\hat{x}_2)$



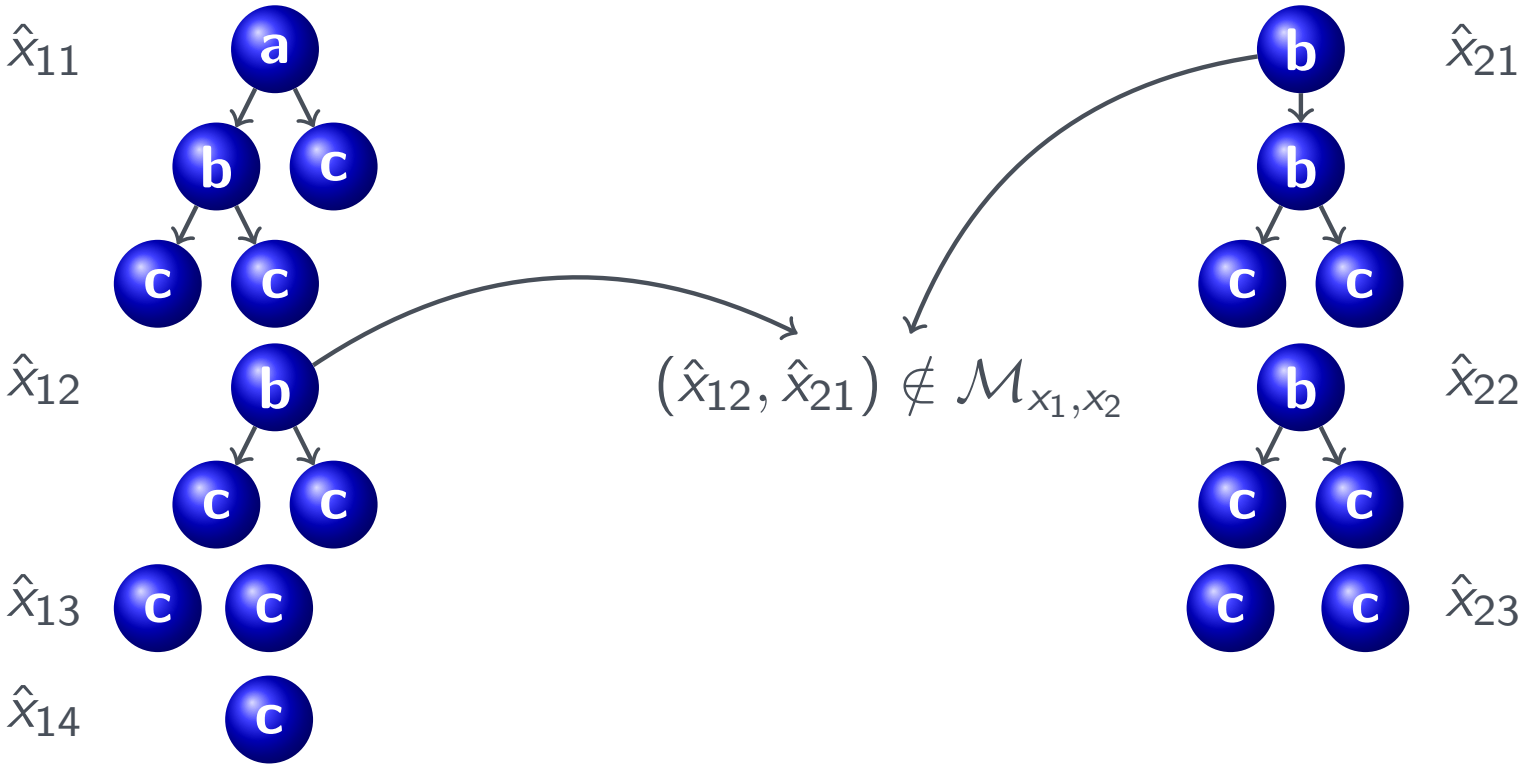
Mapping Kernels: an Example

- Compute SST $C()$ function for all the subtrees at the same depth with respect to the root of the tree
  - $\hat{\chi}_T$ is the set of proper subtrees of T
  - $(\hat{x}_1, \hat{x}_2) \in \mathcal{M}_{x_1,x_2} \Leftrightarrow depth(\hat{x}_1) = depth(\hat{x}_2)$



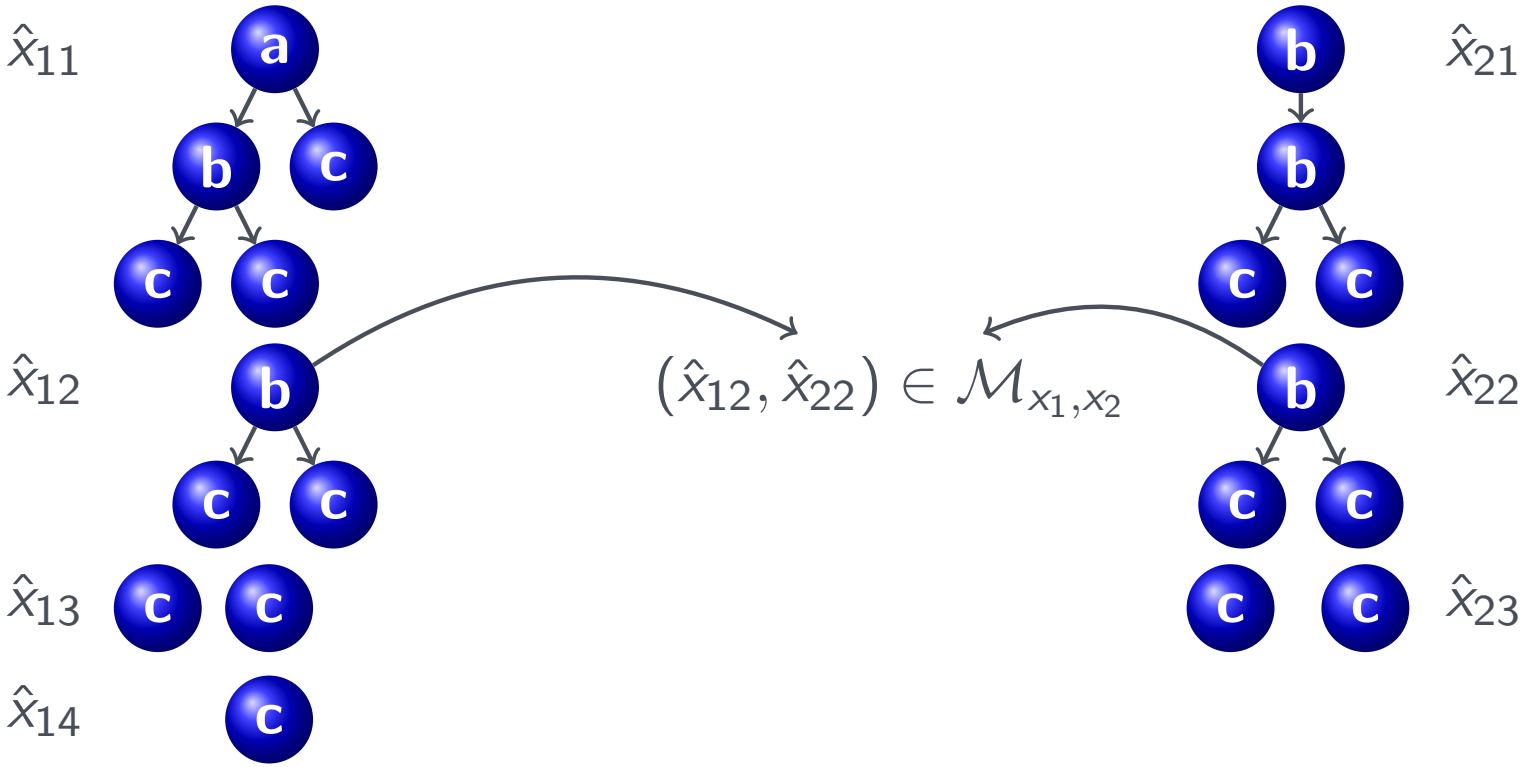
Mapping Kernels: an Example

- Compute SST $C()$ function for all the subtrees at the same depth with respect to the root of the tree
  - $\hat{\chi}_T$ is the set of proper subtrees of T
  - $(\hat{x}_1, \hat{x}_2) \in \mathcal{M}_{x_1,x_2} \Leftrightarrow depth(\hat{x}_1) = depth(\hat{x}_2)$



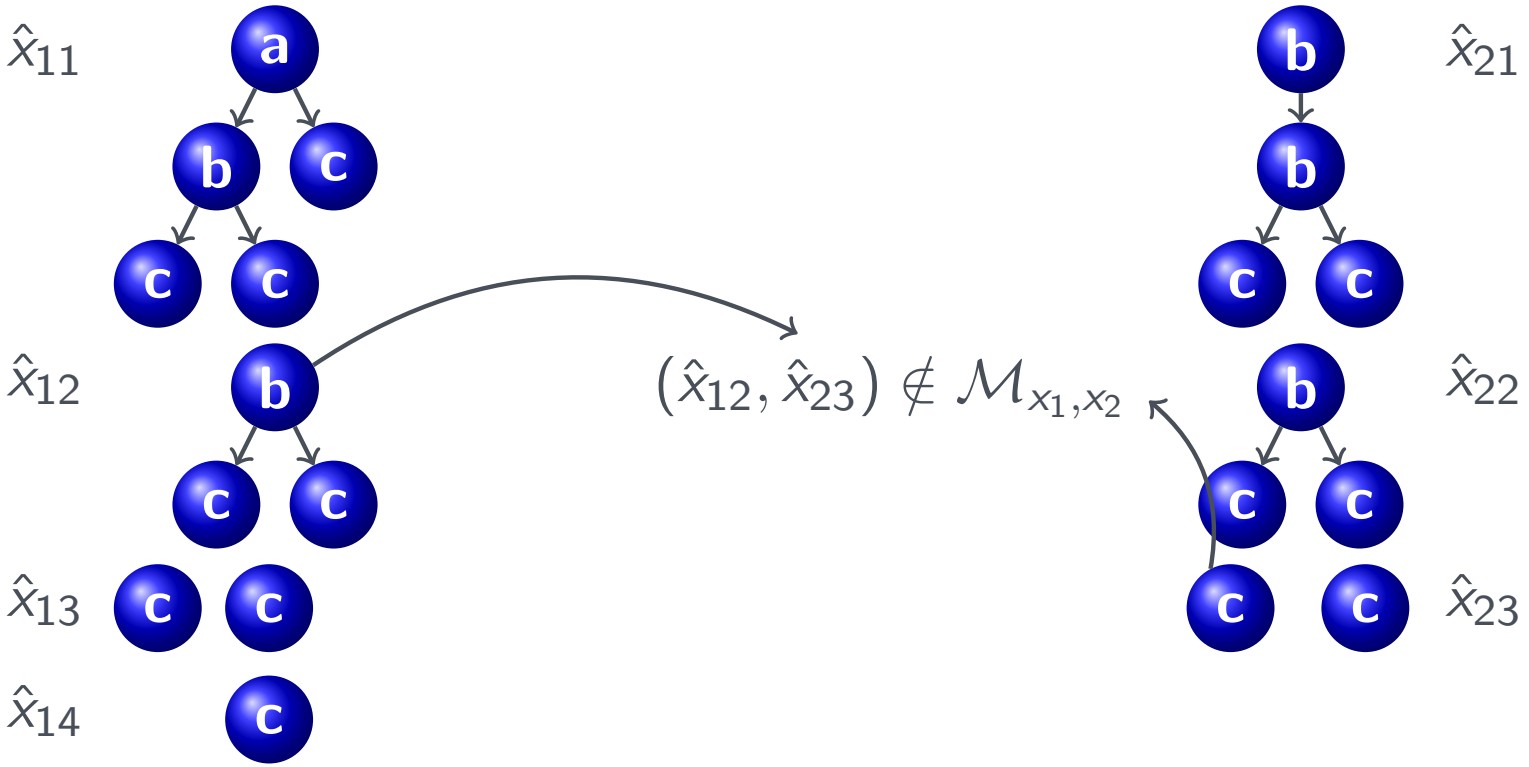
Mapping Kernels: an Example

- Compute SST $C()$ function for all the subtrees at the same depth with respect to the root of the tree
  - $\hat{\chi}_T$ is the set of proper subtrees of T
  - $(\hat{x}_1, \hat{x}_2) \in \mathcal{M}_{x_1,x_2} \Leftrightarrow depth(\hat{x}_1) = depth(\hat{x}_2)$



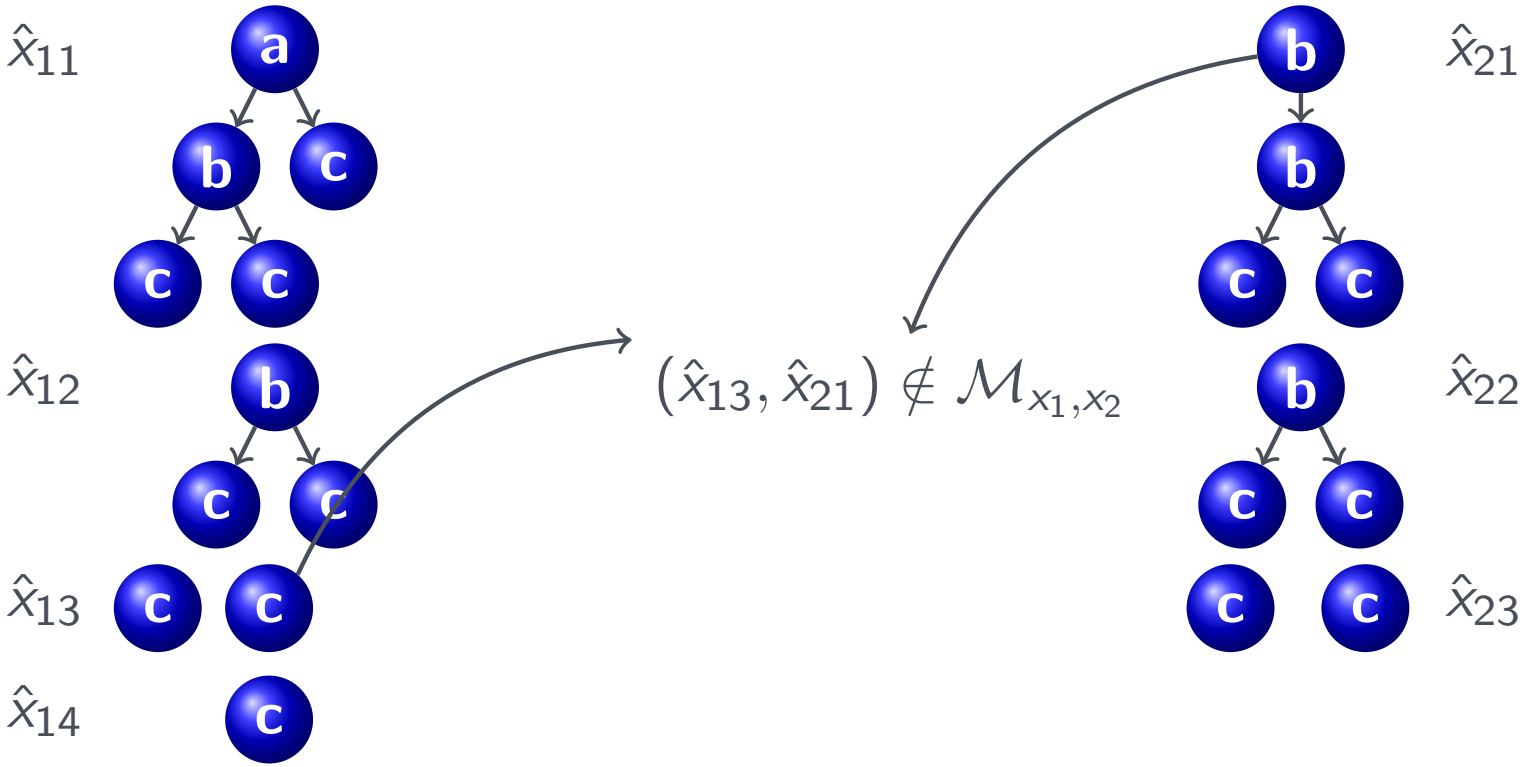
Mapping Kernels: an Example

- Compute SST $C()$ function for all the subtrees at the same depth with respect to the root of the tree
  - $\hat{\chi}_T$ is the set of proper subtrees of T
  - $(\hat{x}_1, \hat{x}_2) \in \mathcal{M}_{x_1,x_2} \Leftrightarrow depth(\hat{x}_1) = depth(\hat{x}_2)$



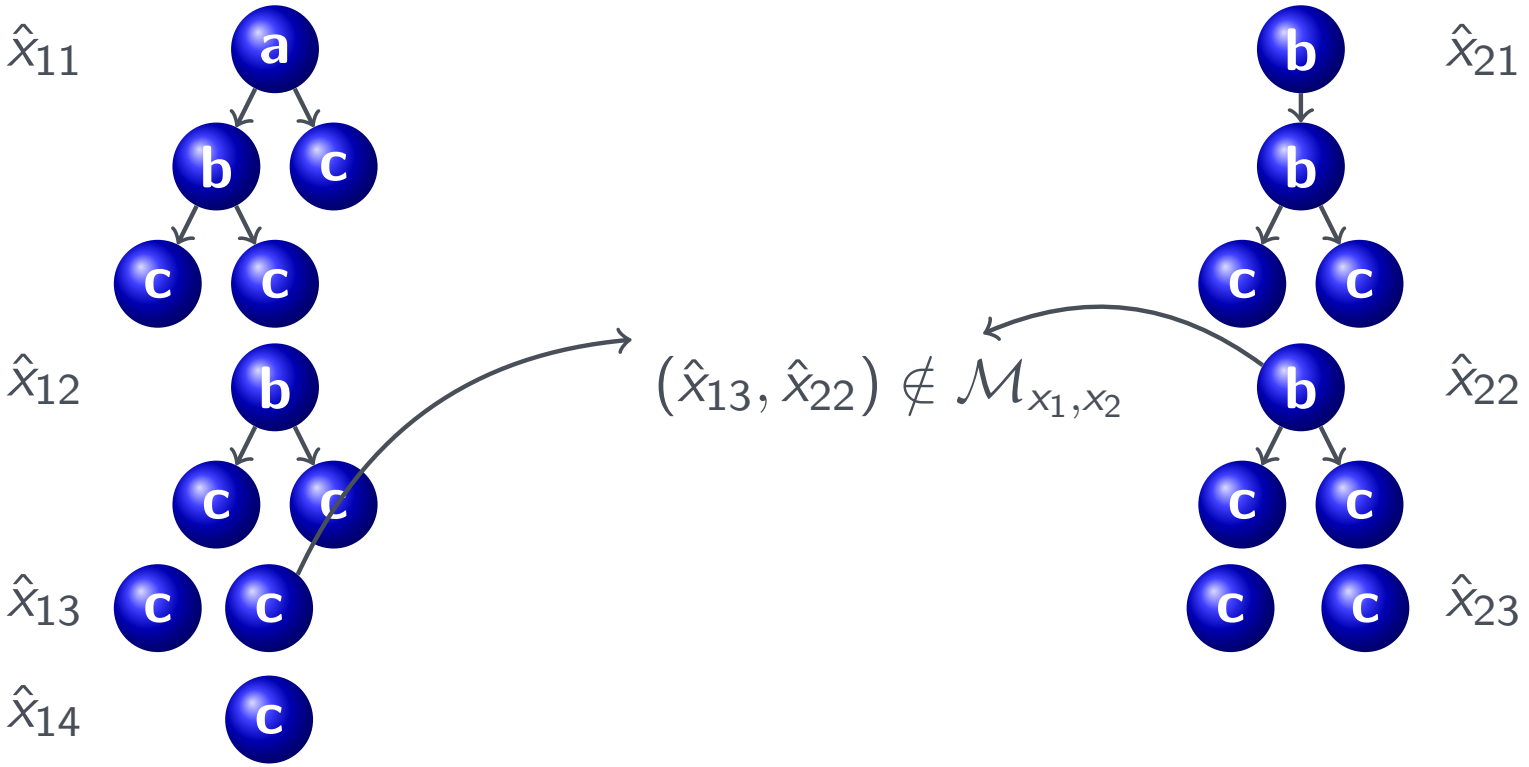
Mapping Kernels: an Example

- Compute SST $C()$ function for all the subtrees at the same depth with respect to the root of the tree
  - $\hat{\chi}_T$ is the set of proper subtrees of T
  - $(\hat{x}_1, \hat{x}_2) \in \mathcal{M}_{x_1,x_2} \Leftrightarrow depth(\hat{x}_1) = depth(\hat{x}_2)$



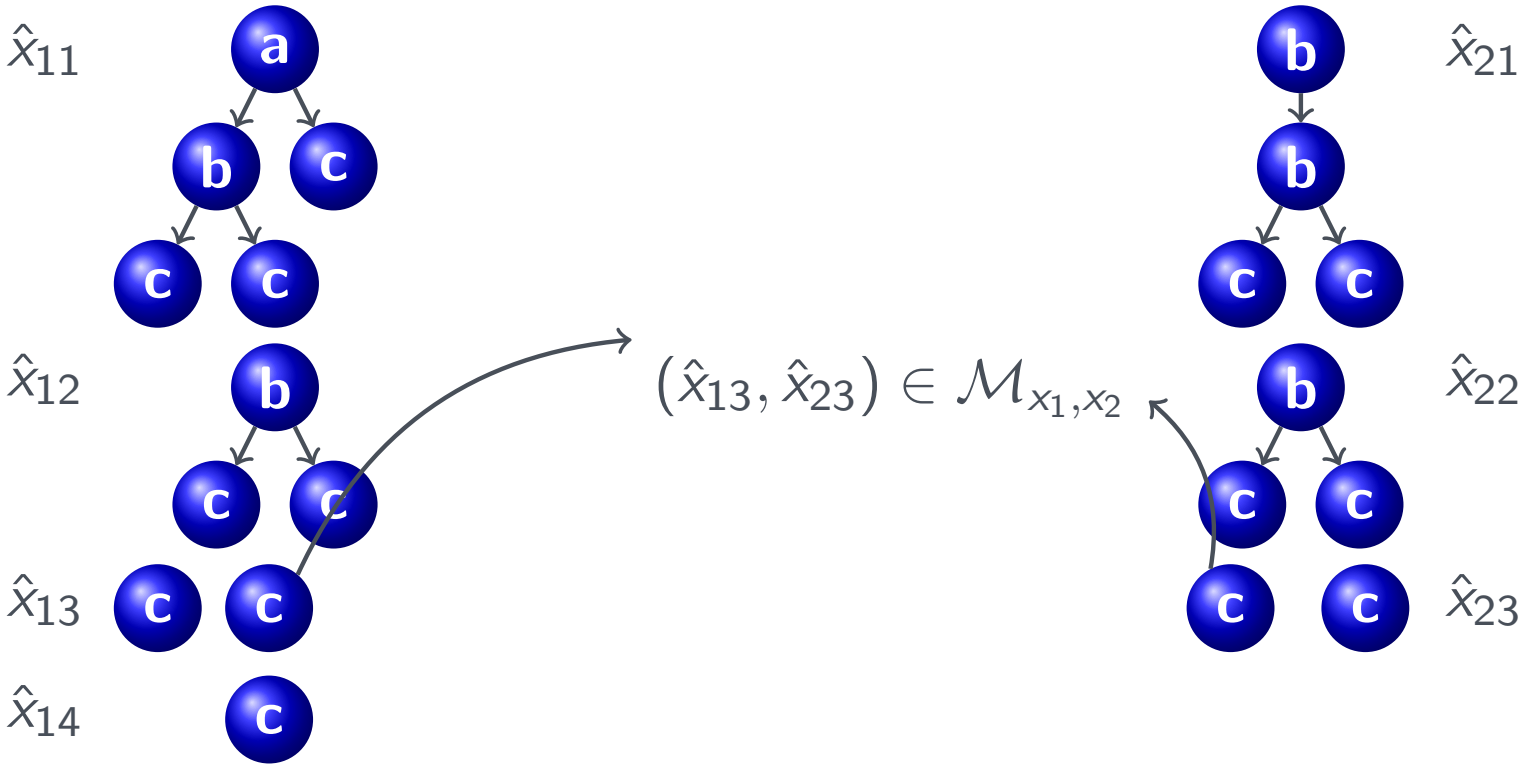
Mapping Kernels: an Example

- Compute SST $C()$ function for all the subtrees at the same depth with respect to the root of the tree
  - $\hat{\chi}_T$ is the set of proper subtrees of T
  - $(\hat{x}_1, \hat{x}_2) \in \mathcal{M}_{x_1,x_2} \Leftrightarrow depth(\hat{x}_1) = depth(\hat{x}_2)$



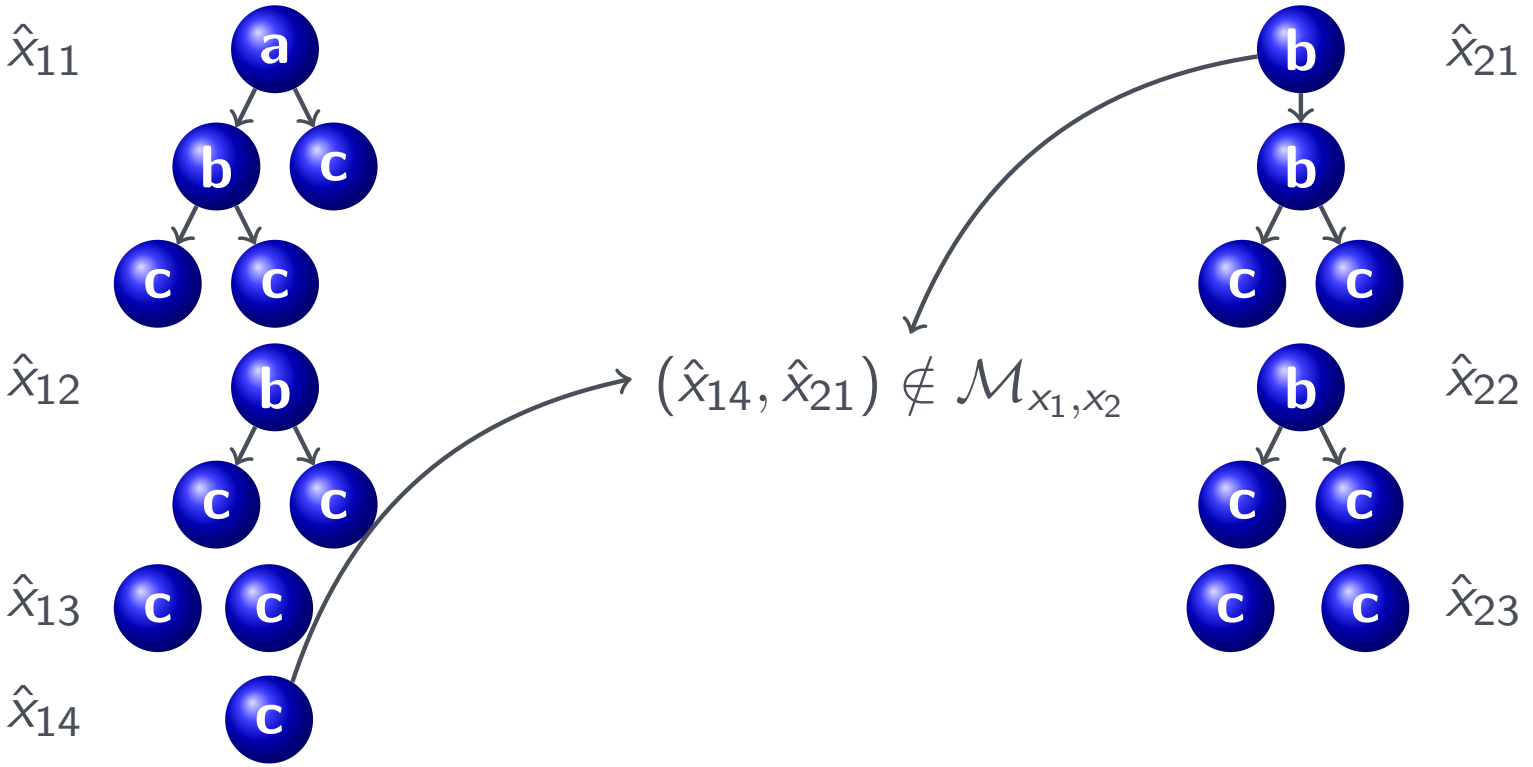
Mapping Kernels: an Example

- Compute SST $C()$ function for all the subtrees at the same depth with respect to the root of the tree
  - $\hat{\chi}_T$ is the set of proper subtrees of T
  - $(\hat{x}_1, \hat{x}_2) \in \mathcal{M}_{x_1,x_2} \Leftrightarrow depth(\hat{x}_1) = depth(\hat{x}_2)$



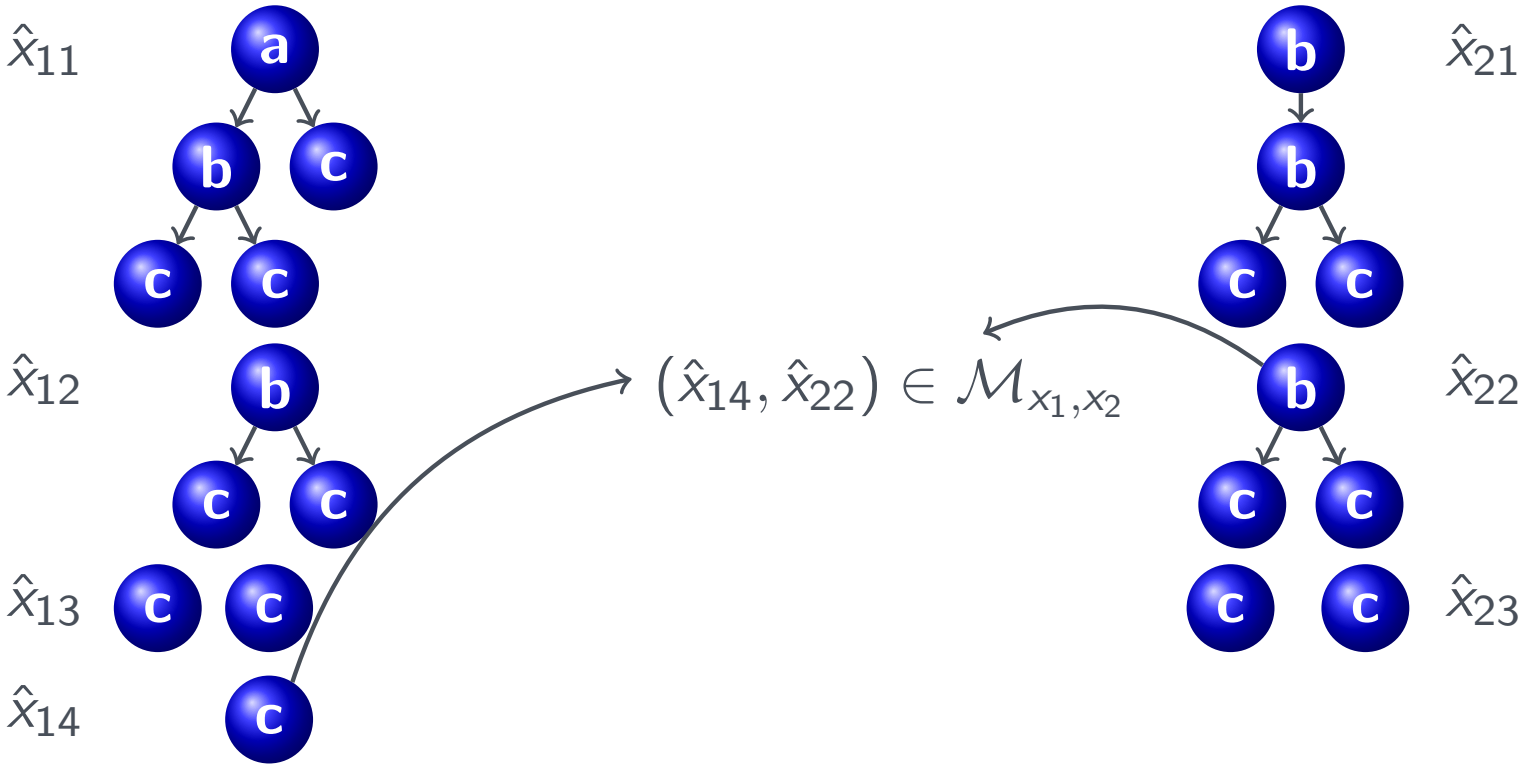
Mapping Kernels: an Example

- Compute SST $C()$ function for all the subtrees at the same depth with respect to the root of the tree
  - $\hat{\chi}_T$ is the set of proper subtrees of T
  - $(\hat{x}_1, \hat{x}_2) \in \mathcal{M}_{x_1,x_2} \Leftrightarrow depth(\hat{x}_1) = depth(\hat{x}_2)$



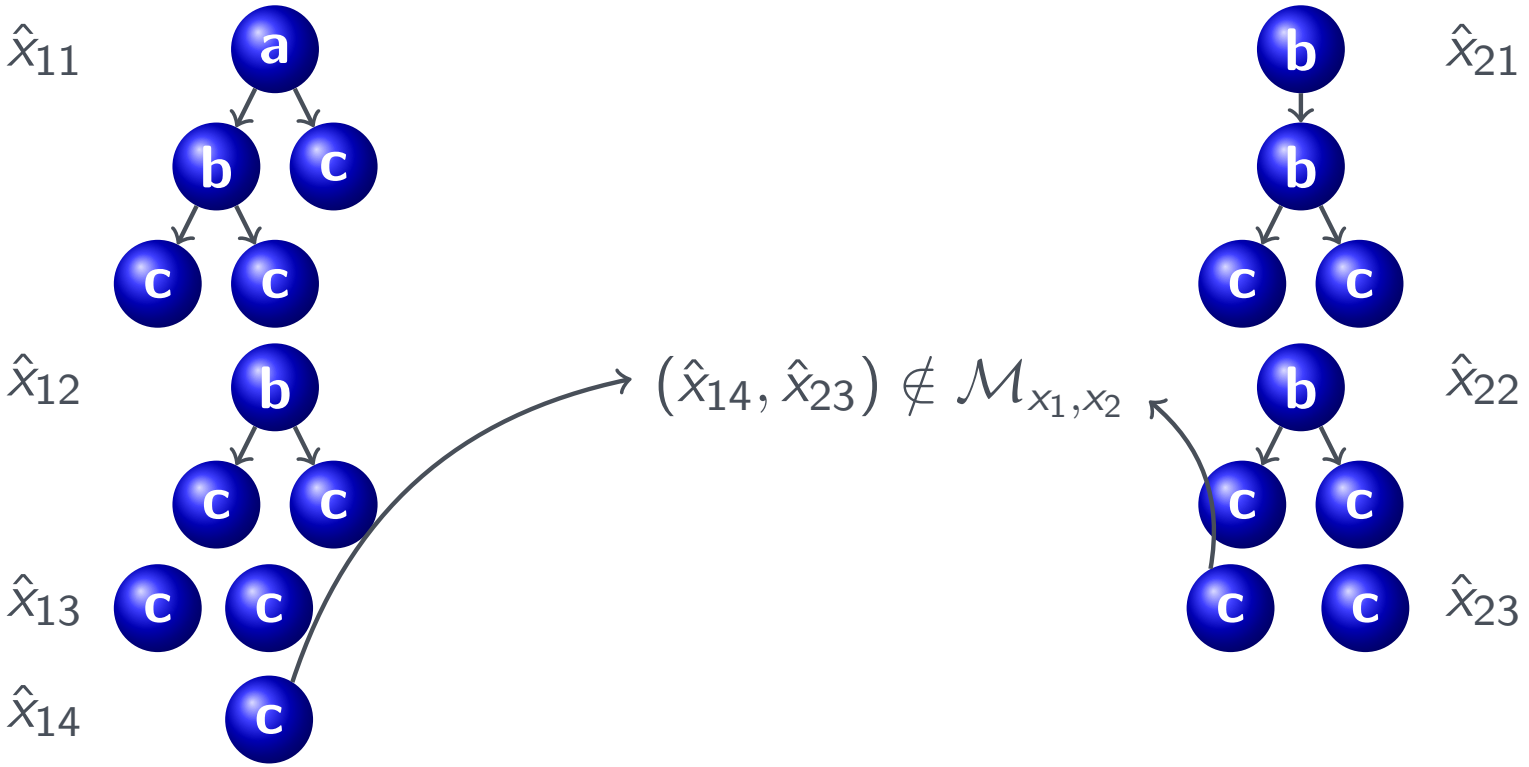
Mapping Kernels: an Example

- Compute SST $C()$ function for all the subtrees at the same depth with respect to the root of the tree
  - $\hat{\chi}_T$ is the set of proper subtrees of T
  - $(\hat{x}_1, \hat{x}_2) \in \mathcal{M}_{x_1,x_2} \Leftrightarrow depth(\hat{x}_1) = depth(\hat{x}_2)$



Mapping Kernels: an Example

- Compute SST $C()$ function for all the subtrees at the same depth with respect to the root of the tree
  - $\hat{\chi}_T$ is the set of proper subtrees of T
  - $(\hat{x}_1, \hat{x}_2) \in \mathcal{M}_{x_1,x_2} \Leftrightarrow depth(\hat{x}_1) = depth(\hat{x}_2)$



Mapping Kernels: an Example

- Compute SST $C()$ function for all the subtrees at the same depth with respect to the root of the tree
  - $\hat{\chi}_T$ is the set of proper subtrees of T
  - $(\hat{x}_1, \hat{x}_2) \in \mathcal{M}_{x_1,x_2} \Leftrightarrow depth(\hat{x}_1) = depth(\hat{x}_2)$
  - $\gamma_T(\hat{x}) = \hat{x}$ (identity functions)

$$K(T_1, T_2) = \sum_{\substack{\hat{x}_1 \in T_1, \hat{x}_2 \in T_2 \\ depth(\hat{x}_1) = depth(\hat{x}_2)}} C(\hat{x}_1, \hat{x}_2) =$$

$$= C_{SST}(\hat{x}_{11}, \hat{x}_{21}) + C_{SST}(\hat{x}_{21}, \hat{x}_{22}) + 2 * C_{SST}(\hat{x}_{13}, \hat{x}_{23}) +$$

$$+ C_{SST}(\hat{x}_{14}, \hat{x}_{22})$$

- $\mathcal{M}$ is transitive and thus $K(T_1, T_2)$ is positive semidefinite

Mapping Kernels: Discussion

- It is possible to define a new kernel by adding constraints to $\mathcal{M}$ such that symmetry and transitivity are preserved
- The positive semidefinitess of a kernel can be proved easily if it can be represented inside the Mapping Kernel Framework; indeed it reduces to prove symmetry and transitivity of $\mathcal{M}$.
- The framework is fairly general: Shin and Kuboyama [21] showed that 18 out of 19 kernels in literature can be represented inside the framework

Non Adaptive Feature Spaces: Discussion

For all kernels presented so far the type of features was fixed in advance and then their frequency in the input structures is measured

- interpretability of the results (in terms of features)
- if the type of features relevant for the problem at hand are not known the kernel is ineffective.
  - Non adaptive kernels deal with this problem by defining very large feature spaces or allowing some sort of soft matching.
  - In both cases, the resulting kernel may be sparse.

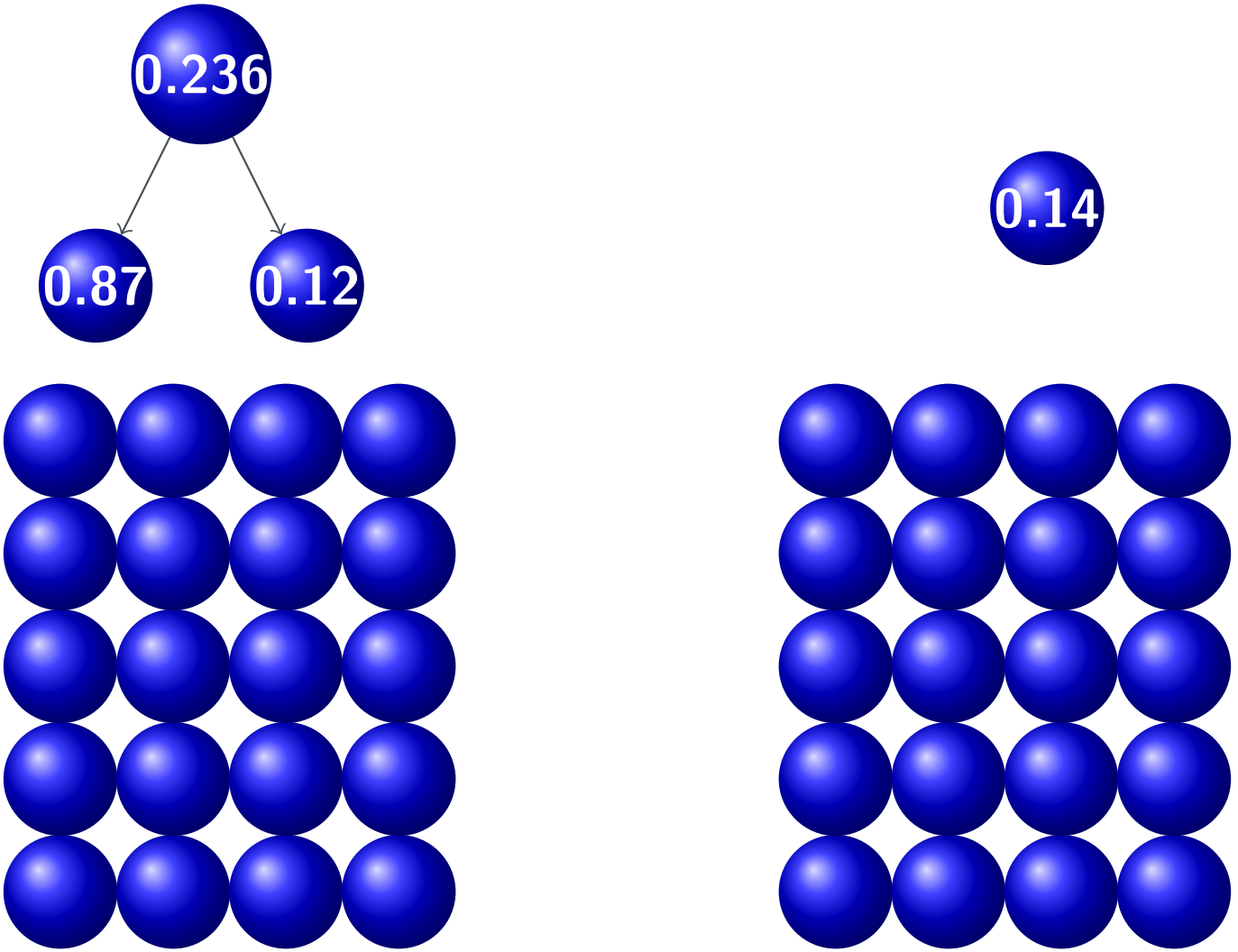
Expressive and Non Sparse Kernels

- Defining very large feature spaces may have the drawback that the associated kernel may be sparse
  - For example SST is sparse when node labels are chosen from a real valued domain.
- Solution: map the data into a vectorial space and define a kernel on it. If the mapping preserves all interesting properties of the data, the kernel can be both efficient and expressive.
  - Examples of mappings: PCA, SOM
  - The Tree Fisher Kernel [23]: it assumes that the data is generated from a parametric probability distribution dependent from a set of parameters of the model. The kernel captures the differences in the generative process of a pairs of objects.

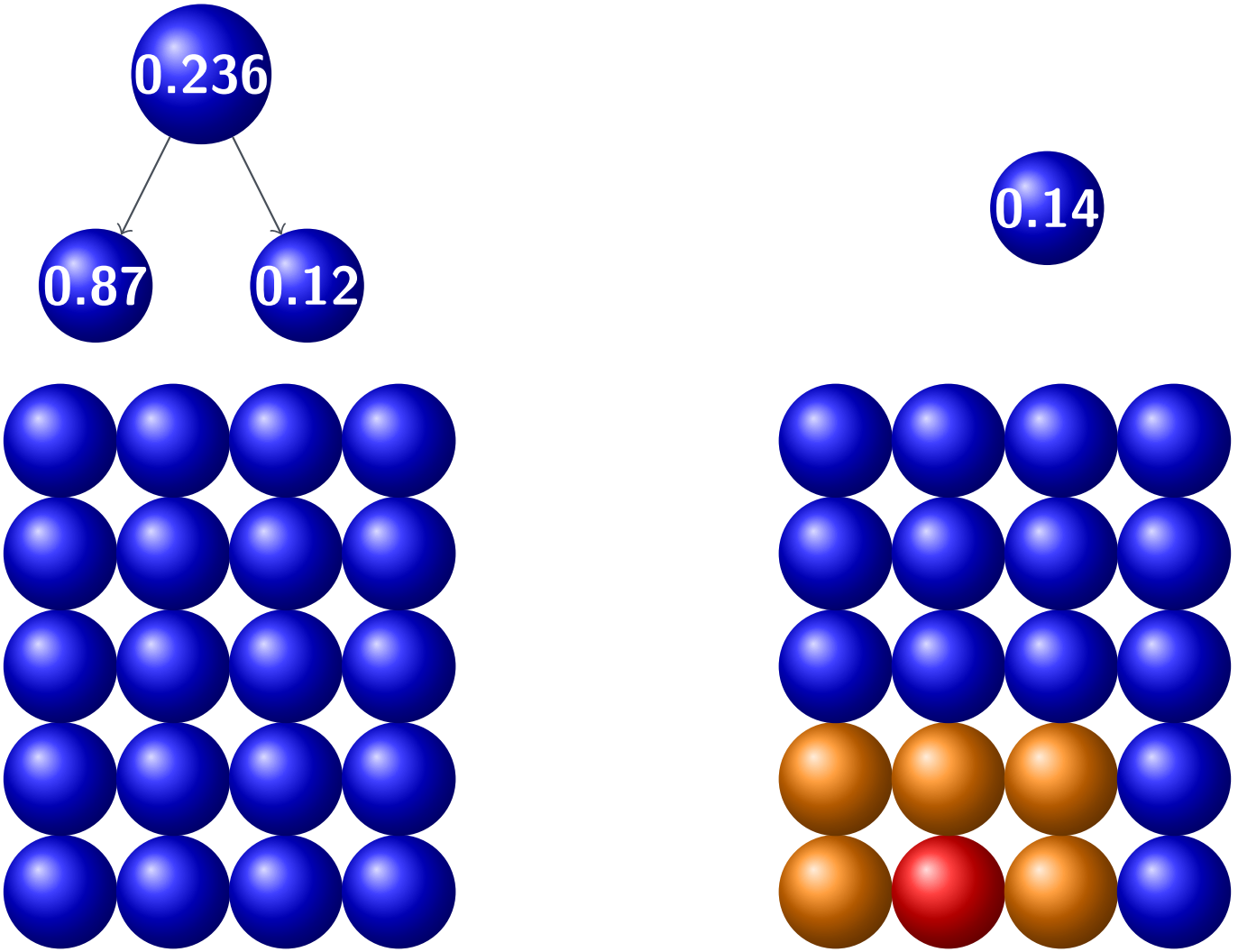
SOMSD

- The SOMSD is composed by a, usually two dimensional, lattice of points.
- Each point has associated a prototype vector which represent the enconding for a tree substructure.
- During learning the prototypes change in order to represent the distribution of the data.
- After learning, a tree structure can be represented by the coordinates of the most similar prototype.
- SOMSD has the property that similar structures tend to be represented nearby in the map.
- Tree nodes that originally wouldn't match may be represented similarly by the SOMSD and thus match at some degree.

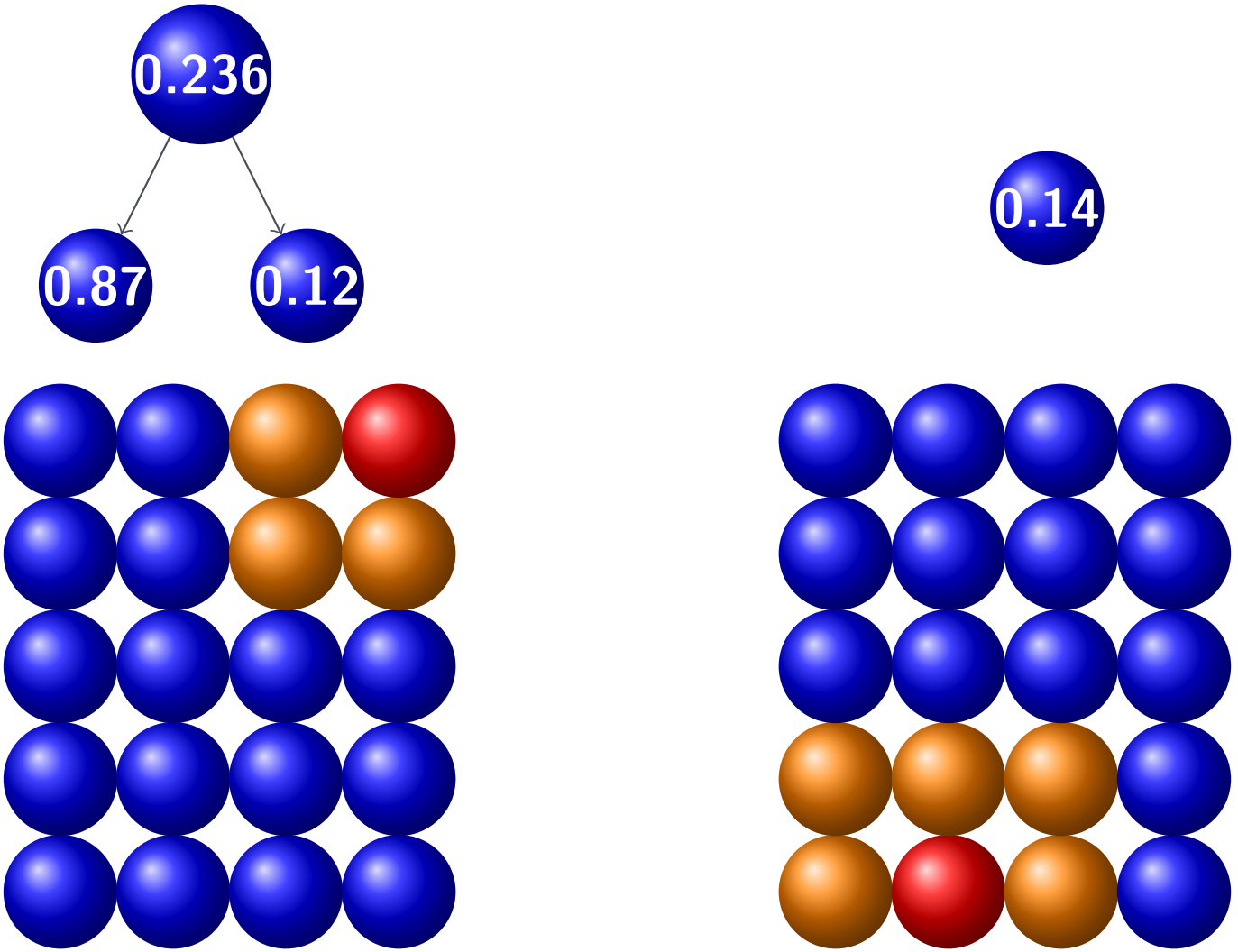
Activation Mask Kernel [24]: an Example



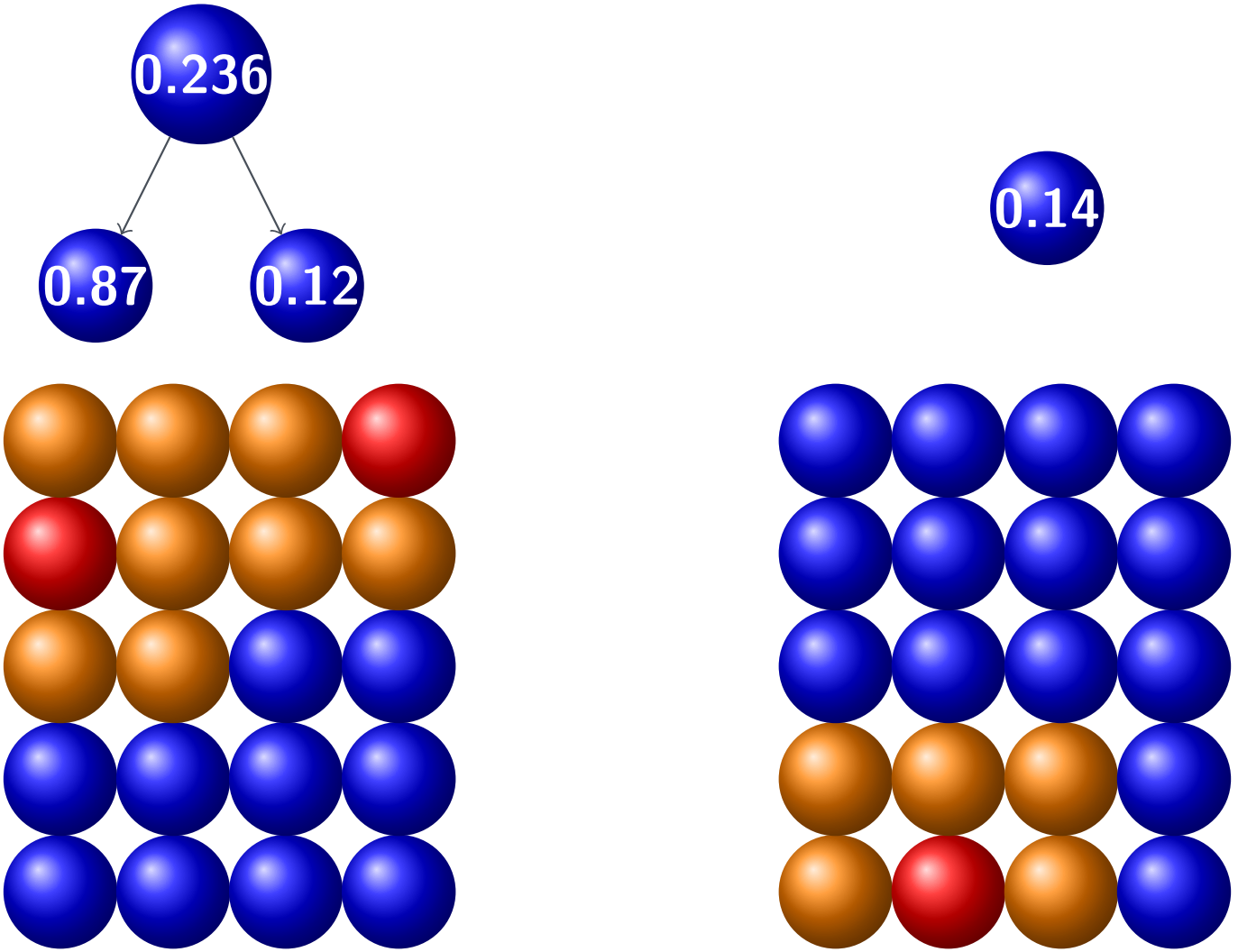
Activation Mask Kernel [24]: an Example



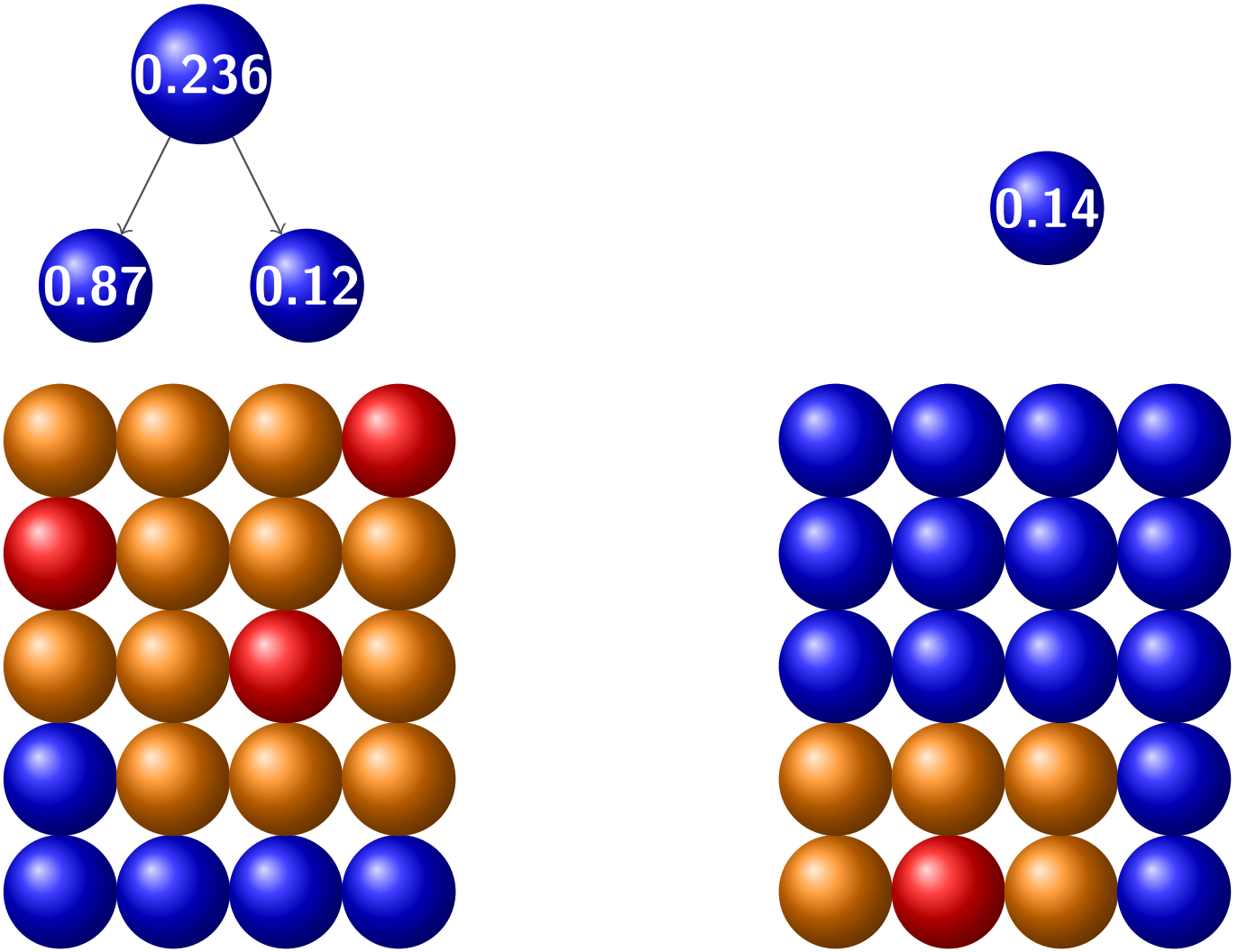
Activation Mask Kernel [24]: an Example



Activation Mask Kernel [24]: an Example



Activation Mask Kernel [24]: an Example



Activation Mask Kernel (AM)

- Given two matrices M_1 and M_2 with as many elements as the SOMSD map, and for which $M(i,j)$ has the same value as the activation mask for a tree T at position (i,j), the AM kernel is given by:

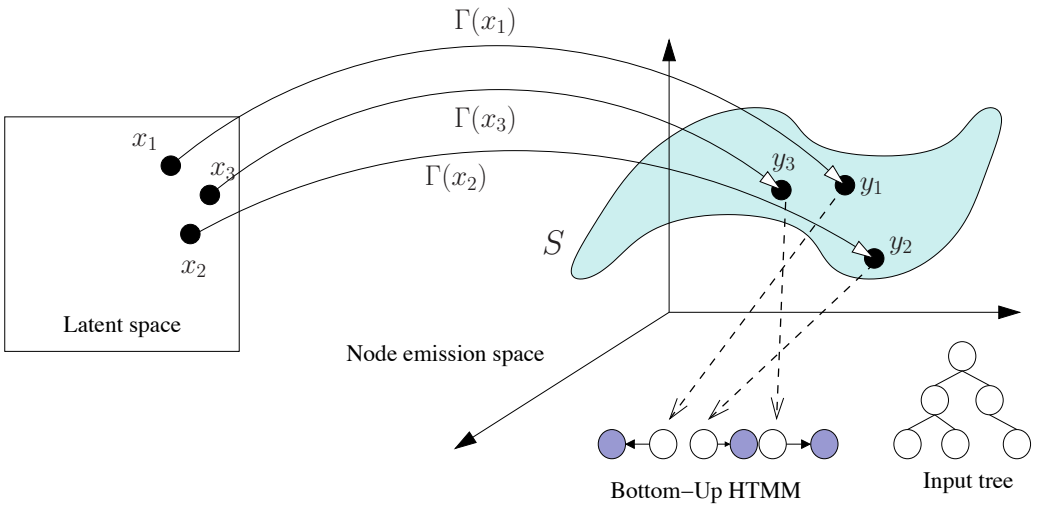
$$K_\epsilon(T_1, T_2) = \sum_{i,j} M_1(i,j) \cdot M_2(i,j)$$

- K_ϵ is positive semidefinite.

Compositional GTM for Trees (GTM-SD)

Bacciu et al. [25] introduced GTM-SD, a probabilistic version of SOM-SD

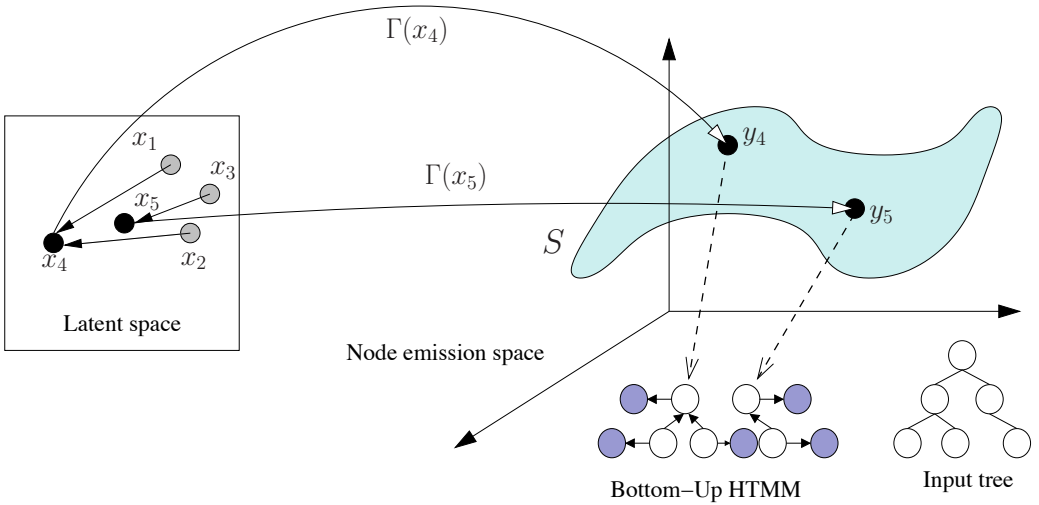
- A compositional approach based on BHTMM
  - A tree $\mathbf{t}_n$ is not an atomic i.i.d entity
  - It is a collection of constrained observations
- Constraints are
  - Determined by structural children-to-parent relationships
  - Modeled by L-order Markovian dependencies



Compositional GTM for Trees (GTM-SD)

Bacciu et al. [25] introduced GTM-SD, a probabilistic version of SOM-SD

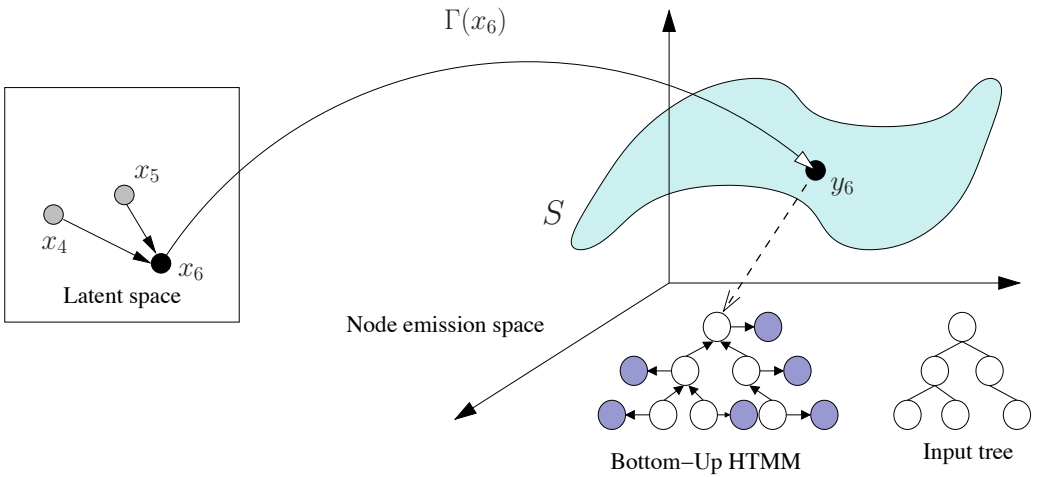
- A compositional approach based on BHTMM
  - A tree $\mathbf{t}_n$ is not an atomic i.i.d entity
  - It is a collection of constrained observations
- Constraints are
  - Determined by structural children-to-parent relationships
  - Modeled by L-order Markovian dependencies



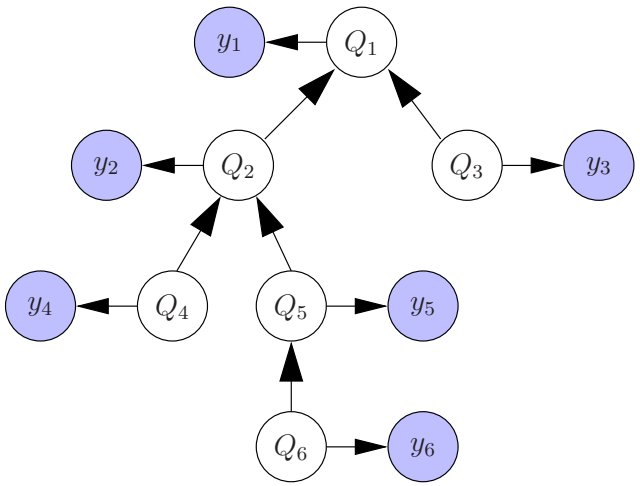
Compositional GTM for Trees (GTM-SD)

Bacciu et al. [25] introduced GTM-SD, a probabilistic version of SOM-SD

- A compositional approach based on BHTMM
  - A tree $\mathbf{t}_n$ is not an atomic i.i.d entity
  - It is a collection of constrained observations
- Constraints are
  - Determined by structural children-to-parent relationships
  - Modeled by L-order Markovian dependencies



Bottom-up Hidden Tree Markov Model



$u \rightarrow$ node index
$Q_u \rightarrow$ hidden state R.V.
$y_u \rightarrow$ observation (label)
$ch_l(u) \rightarrow$ l-th child of u

- Label emission governed by $P(y_u|Q_u)$
  Parent-children relationships are modeled by children-to-parent state transitions

$$P(Q_u|Q_{ch_1(u)}, \ldots, Q_{ch_L(u)})$$

- Compositional $\rightarrow$ *simpler* structures are processed first

Combinatorial Problem

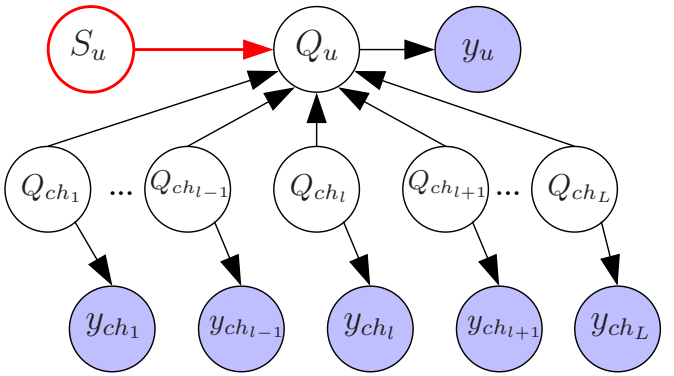
State transition distribution is $O(C^{L+1})$ for a C-dimensional hidden state space

Switching Parents BHTMM

Key Idea

Approximate the joint state transition distribution as a mixture of pairwise transition matrices

- Introduce a child selector variable for each parent u
- Switching Parent $S_u \in \{1, \ldots, L\}$
- $P(S_u = l)$ measures the influence of the l-th child on the state transition to Q_u



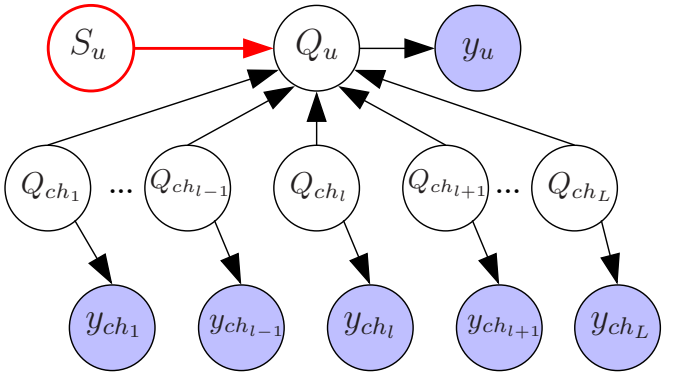
$$P(Q_u|Q_{ch_1(u)}, \ldots, Q_{ch_L(u)}) = \sum_{l=1}^{L} P(S_u = l) P(Q_u|Q_{ch_l(u)})$$

Switching Parents BHTMM

Key Idea

Approximate the joint state transition distribution as a mixture of pairwise transition matrices

- Introduce a child selector variable for each parent u
- Switching Parent $S_u \in \{1, \ldots, L\}$
- $P(S_u = l)$ measures the influence of the l-th child on the state transition to Q_u

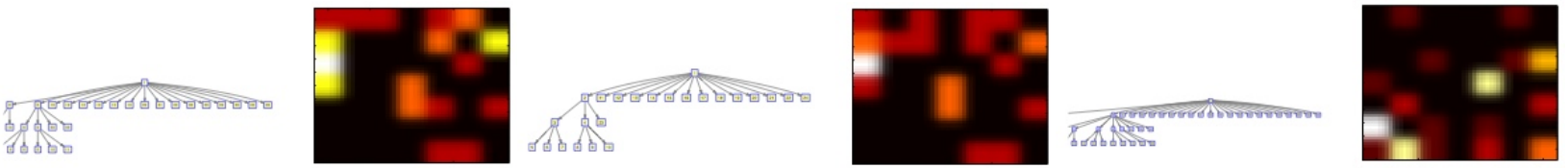


Parameters space reduces to $O(LC^2 + L)$

Tree Projection in GTM-SD

- Visualization of a tree $\mathbf{t}$ is based on projecting its root onto the lattice by using its hidden state assignment Q_1
  - Mean projection $\longrightarrow X_{mean}(\mathbf{t}) = \sum_{i=1}^{C} P(Q_1 = x_i|\mathbf{t})x_i$
  - Mode projection $\longrightarrow X_{mode}(\mathbf{t}) = \arg\max_{x_i} P(Q_1 = x_i|\mathbf{t})$
- Distribution $P(Q_1 = x_i|\mathbf{t})$ is obtained as a by-product of EM
  - Alternatively, reversed Viterbi inference can be used

Examples of tree projections



AM Kernel based on GTM-SD

Given

$$T_\epsilon(x_u, x_{u'}) = \begin{cases} \epsilon - d(x_u, x_{u'}), & \text{if } d(x_u, x_{u'}) \leq \epsilon \\ 0, & \text{otherwise} \end{cases}$$

where

- x_u and $x_{u'}$ are the posterior mode projections of subtrees $\mathbf{t}^1_u$ and $\mathbf{t}^2_{u'}$ from trees $\mathbf{t}^1$ and $\mathbf{t}^2$, respectively
- $d(x_u, x_{u'})$ is the Euclidean distance
- ϵ determines a neighborhood for the points on the map

The μGTM-SD Activation Mask kernel is defined as

$$K(\mathbf{t}^1, \mathbf{t}^2) = \sum_{u \in \mathcal{U}_1} \sum_{u' \in \mathcal{U}_2} T_\epsilon(x_u, x_{u'})$$

Experiments on INEX 2005

INEX 2005: XML data, 11-class problem, training set 4824 documents, test set 4811 documents.

Kernel Type	Valid Error %	Test Error %
ST	13.15	11.27
SST	12.79	11.21
Polynomial SST	12.09	10.67
Partial Tree	2.96	**2.96**
Route, k_{label}	3.10	3.06
Route, k_{prod}	3.31	3.52

Experiments on INEX 2006

INEX 2006: XML data, 18-class problem, training set 6053 documents, test set 6054 documents.

Kernel Type	Valid Error %	Test Error %
ST	68.32	67.98
SST	56.55	59.56
Polynomial SST	55.55	59.88
Partial Tree	57.83	58.87
Route, k_{label}	58.72	59.94
Route, k_{prod}	55.55	**58.09**

Experiments on PROPBANK

PROPBANK: annotated texts from Penn Tree Bank corpus, 2-class problem, training set 75314 examples, test set 40495 examples.

Kernel Type	Valid Error %	Test Error %
ST	5.43	5.71
SST	4.65	4.62
Polynomial SST	4.65	4.62
Partial Tree	4.71	**4.55**
Route, k_{label}	5.07	4.90
Route, k_{prod}	4.92	4.76

Experiments on LOGML

LOGML: logs of web user sessions, 2-class problem, 3 sets of data of 8074, 7409 and 7628 examples.

Kernel Type	Cross Validation Error %
ST	16.72
SST	16.84
Polynomial SST	16.82
Partial Tree	16.40
Route, k_{label}	**16.20**
Route, k_{prod}	16.79

Kernels for Trees: Discussion

- Non adaptive feature spaces
  - interpretability of the results,
  - if the features are not appropriate for the task, a novel kernel has to be defined
- Adaptive feature spaces
  - kernels tend to be more expressive since features are built adaptively for the given dataset
  - may not be easy to analyse the results in terms of the most relevant features.

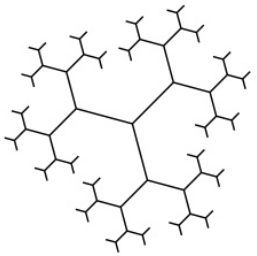
KERNEL FUNCTIONS FOR GRAPHS

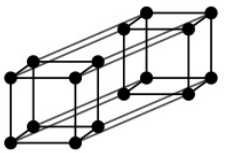
Kernels for Graphs

WARNING!

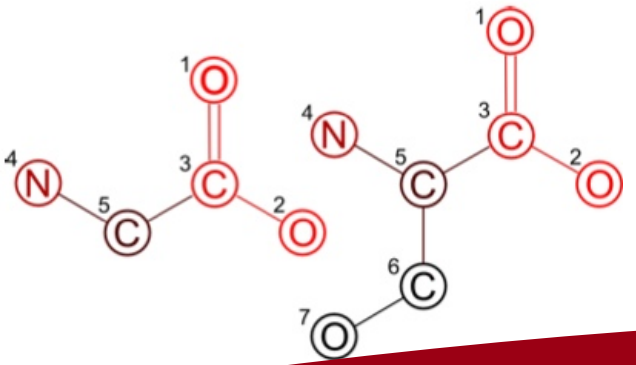
The term graph kernel is used with multiple meanings:

- Kernel between nodes of a graph: e.g. Diffusion Kernel [27]

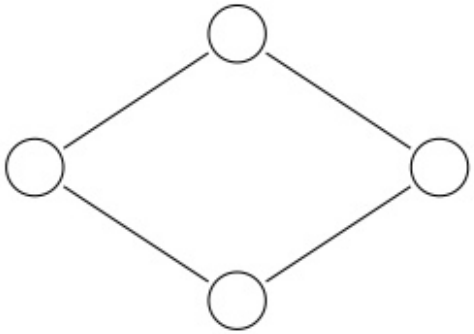




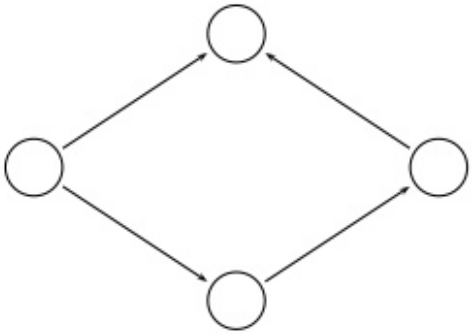
- Kernels between pairs of graphs (we focus on this case)



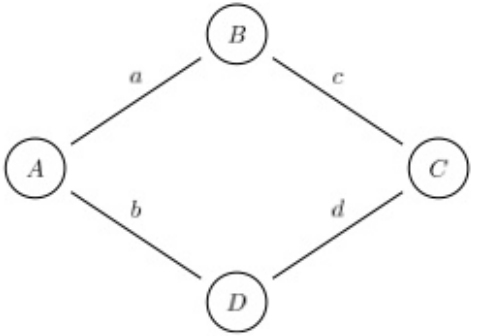
Graphs



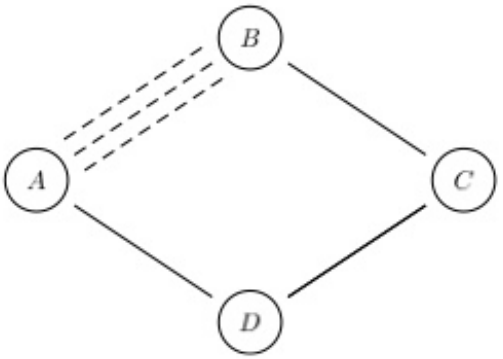
undirected graph



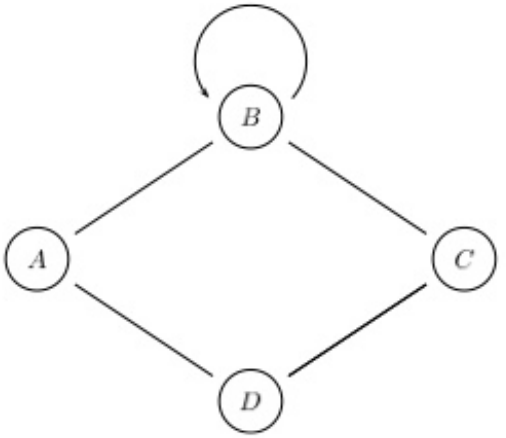
directed graph



undirected labeled graph



graph with multiple edges

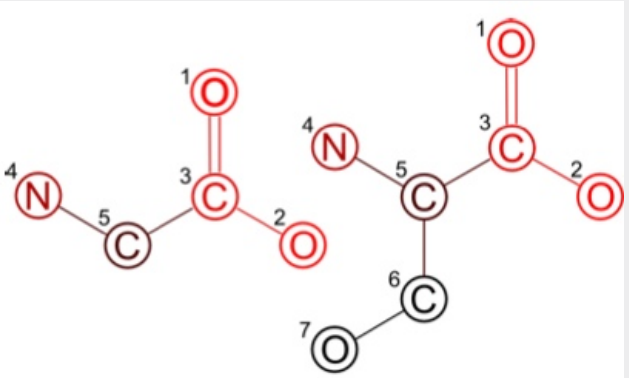


graph with self-loop

Comparison between Graphs

The basic idea underpinning graph kernels is to quantify graph similarity

How much similar (or dissimilar) are these two graphs ?



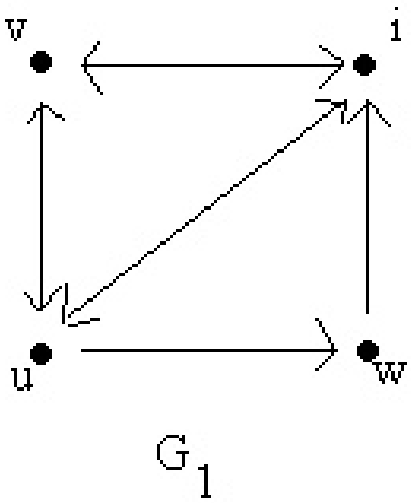
How to measure similarity between graphs ?

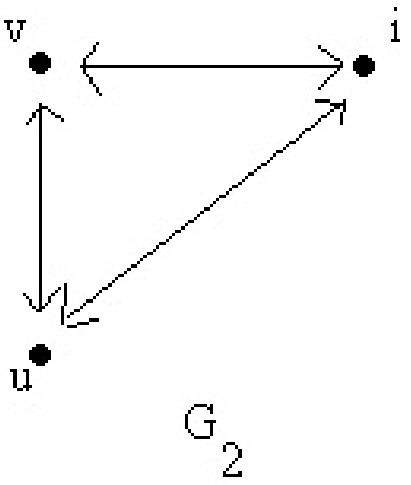
For example, by counting how many subgraphs they share . . .

Basic Definitions on Graphs

Definition: Subgraph

A subgraph $G_2 = (V_2, E_2)$ of $G_1 = (V_1, E_1)$ is a graph for which $V_2 \subseteq V_1$, $E_2 = E_1 \cap (V_2 \times V_2)$.

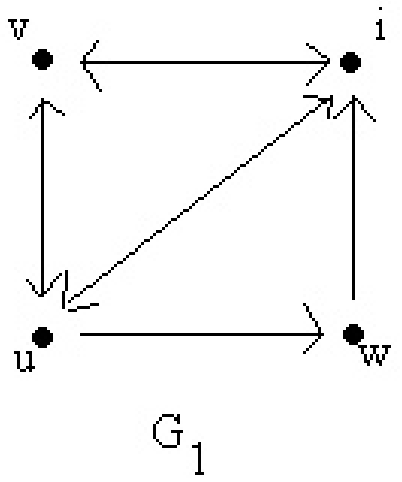




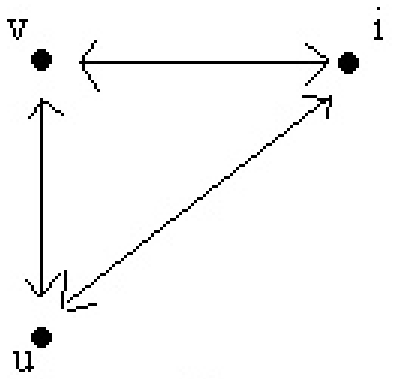
Basic Definitions on Graphs

Definition: Isomorphism

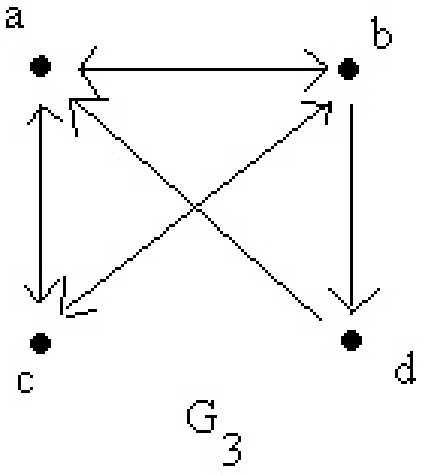
A graph $G_1 = (V_1, E_1)$ is isomorphic to $G_2 = (V_2, E_2)$ if there exists a mapping $f : V_1 \times V_2$ such that
$\forall (v_1, v_1') \in E_1 \Leftrightarrow (f(v_1), f(v_1')) \in E_2$



G_1



G_2



G_3

G_1 and G_3 are isomorphic: $f(i) = a,\ f(u) = b,\ f(v) = c,\ f(w) = d$

Graph Comparison and Isomorphism

Two problems

- Same graph can be represented in different ways ...
  (any graph comparison algorithm has to consider that!)
- How to recognize that a given graph G_2 is a subgraph of G_1 ?

- It is not known whether Graph Isomorphism is NP-Complete
  (no polynomial-time algorithm is known, neither is it known to be NP-Complete)
- Subgraph Isomorphism is NP-Complete !!
  ... runtime may grow exponentially with the number of nodes

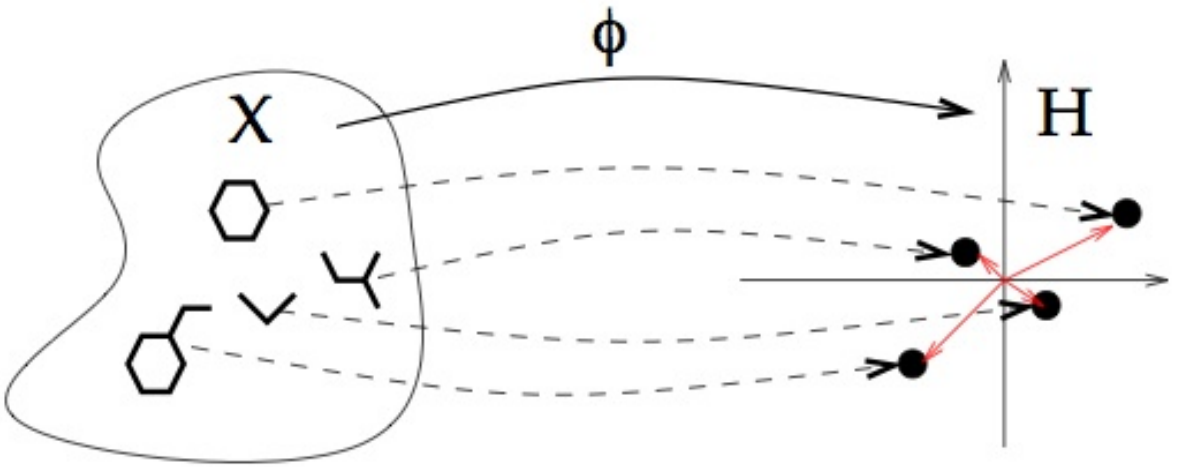
Kernels for Graphs

Definition

A graph kernel between two graphs G_1 and G_2 is defined as

$$K(G_1, G_2) = \phi(G_1)^\top \phi(G_2),$$

where $\phi : X \mapsto H$, X is the graph domain, and H is a Hilbert space



Kernels for Graphs: A Theoretical Result

Definition: Complete Kernel

A kernel is complete if it separates non-isomorphic graphs, i.e. $\phi(G_1) \neq \phi(G_2)$ if G_1 and G_2 are not isomorphic

Theorem

Computing any complete graph kernel is at least as hard as deciding whether two graphs are isomorphic [26]

- Defining kernels for graphs is a more challenging task than for trees: application of the convolution framework may be infeasible given the fact that graph isomorphism has to be checked for every substructure

Some negative results

Graph enumeration

Let $enum(G) \mapsto \mathbb{N}$ be a function that enumerates all $G \in X$, such that $enum(G) = enum(G')$ *iff* G is isomorphic to G'

Subgraph Kernel

Given $\tilde{G} \in X$, let consider a ϕ such that

$$\phi_i(\tilde{G}) = |\{\tilde{G}_{sg} \text{ is a subgraph of } \tilde{G} \text{ s.t. } enum(\tilde{G}_{sg}) = i\}|.$$

Given $\lambda_i \geq 0$, the subgraph kernel is defined as

$$K_{subgraph}(G_1, G_2) = \sum_i \lambda_i \phi_i(G_1)\phi_i(G_2)$$

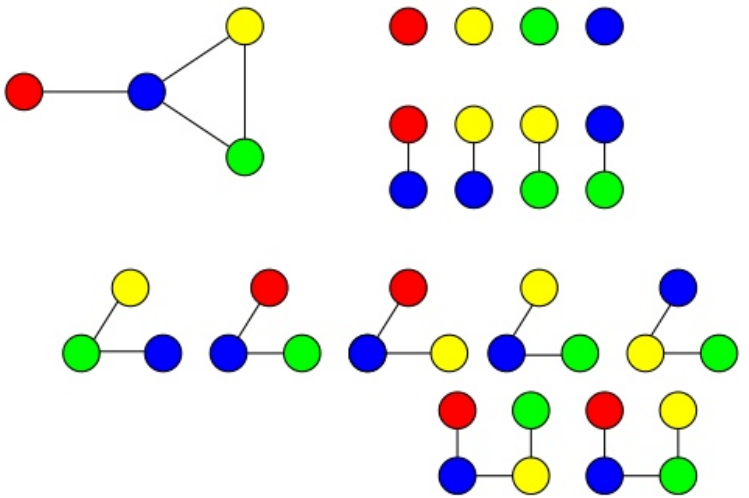
Some negative results

Theorem

Computing the subgraph kernel is NP-Hard [27]

Hummm... this is not good!

Let's try to consider only *linear subgraphs* (i.e. paths)



Some negative results

Path Kernel

Given $\tilde{G} \in X$, let consider a ϕ^{path} such that

$$\phi_i^{path}(\tilde{G}) = |\{\tilde{G}_{sg} \text{ is a } \textit{linear} \text{ subgraph of } \tilde{G} \text{ s.t. } \textit{enum}(\tilde{G}_{sg}) = i\}|.$$

Given $\lambda_i \geq 0$, the path kernel is defined as

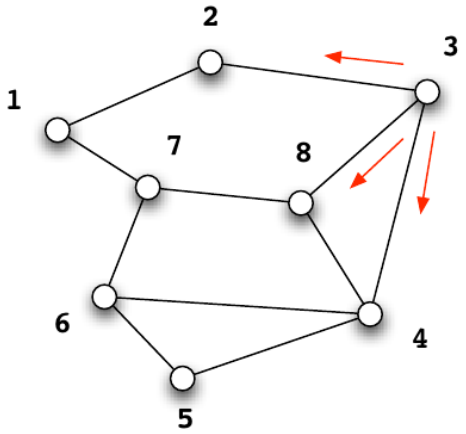
$$K_{path}(G_1, G_2) = \sum_i \lambda_i \phi_i^{path}(G_1)\phi_i^{path}(G_2)$$

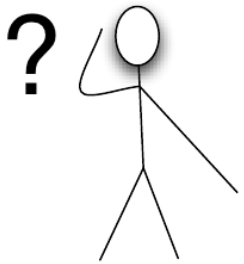
Theorem

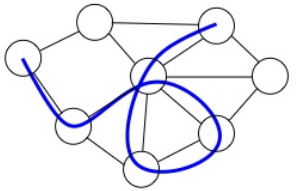
Computing the path kernel is NP-Hard [27]

Still not good! Let's try with something different... random walks!

Random walk kernels







What is a random walk?

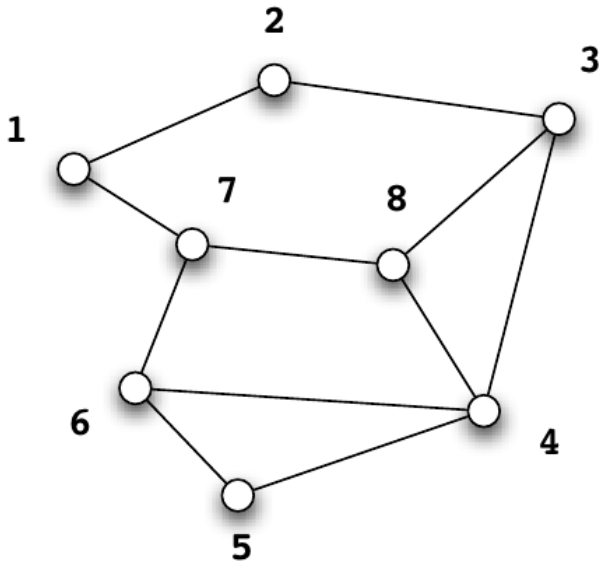
- from a vertex i randomly jump to a vertex in the neighborhood with a certain probability
- or stop the walk with a fixed probability

Random walk kernel:

- feature space: all possible walks
- active features: all walks that can be generated from the graph

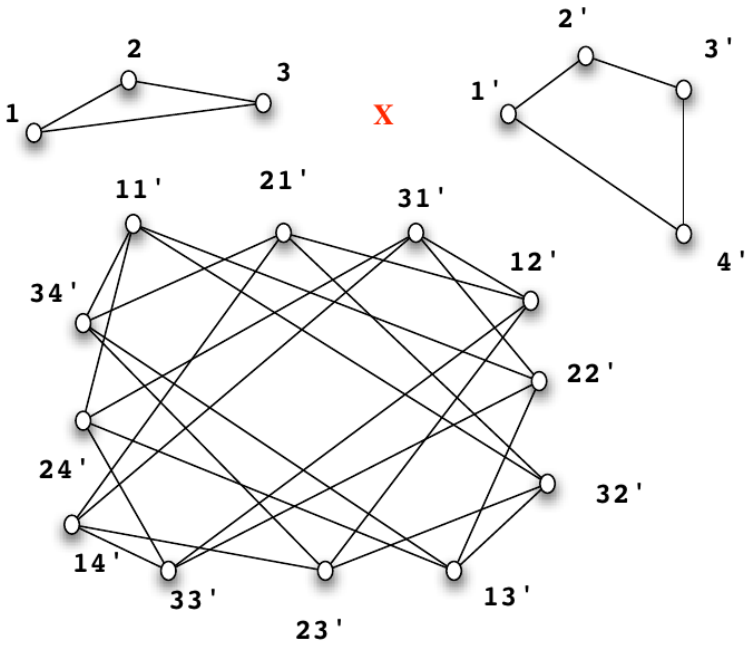
Random walk kernels

Walks can be computed as exponentiation of adjacency matrix: entries of A^2 are all the walks of length 2.



$$A^2 = \begin{bmatrix} 2 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 2 & 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 3 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 4 & 1 & 1 & 2 & 1 \\ 0 & 0 & 1 & 1 & 2 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 & 3 & 0 & 2 \\ 0 & 1 & 1 & 2 & 1 & 0 & 3 & 0 \\ 1 & 1 & 1 & 1 & 1 & 2 & 0 & 3 \end{bmatrix}$$

Random walk kernels



Direct product graph:

- Computing random walks on this graph, corresponds to calculate the number of common walks
- decaying factor for practical computability

$$V_\times = \{(x, y) : x \in V(G_1) \wedge y \in V(G_2) \wedge L_{G_1}(x) = L_{G_2}(y)\}$$
$$E_\times = \{((x, y), (x', y')) \in V_\times \times V_\times :$$
$$(x, y) \in E(G_1) \wedge (y, y') \in E(G_2) \wedge \{L_{G_1}(x, y) = L_{G_2}(y, y')\}$$

Random walk kernels - Drawbacks

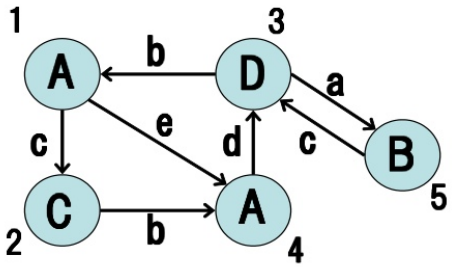
$$K_\times(G_1, G_2) = \sum_{x,y=0}^{|V_\times|} \left[\sum_{k=0}^{\infty} \lambda_k A_\times^k \right]_{x,y}$$

- The computational complexity of $K_\times$ is $O(|G|^6)$,
  - very high, in [28] it is reduced to $O(|G|^3)$ via *Sylvester* equations
  - Mahe et al. [29] proposed to reduce the size of the product graph by adding local information to vertex labels
- Halting problem: the series in the formula above may require a very small value of λ to converge, i.e only nodes or, at least single edges, are considered, graph structure is discarded

Marginalized Random Walk Kernel

Kashima et al. [30] introduced a probabilistic version for random walk kernels for graphs with labels on vertices and edges

- Based on counting labeled paths (random variables) obtained by random walks



Example: $(A, e, A, d, D, a, B, c, D)$

- The kernel is defined as the inner product of the count vectors averaged over all possible label paths, which is regarded as a special case of marginalized kernels [31]

Marginalized Random Walk Kernel

Generation of a random walk

1. select the starting vertex $v_1 \in G$ according to $p_s(v_1)$ defined over all vertices
2. at the i-th step, the vertex v_i is extracted according to the transition probability $p_t(v_i|v_{i-1})$ to go in the vertex v_i starting from v_{i-1}, or the random walk stops with a probability $p_q(v_{i-1})$

The probability of a walk w to be generated (leading to labeled path h_w) is:

$$p(w|G) = p_s(v_1) \left(\prod_{i=2}^{l} p_t(v_i|v_{i-1}) \right) p_q(v_l)$$

Marginalized Random Walk Kernel

The probability for the labeled path h to be generated is the sum over all the probabilities of the walks w that generates the same labeled path $h_w = h$:

$$p(h|G) = \sum_w \delta(h, h_w) \left\{ p_s(v_1) \prod_{i=2}^{l} p_t(v_i|v_{i-1}) p_q(v_l) \right\}$$

where δ is Kronecker's delta.

Marginalized Random Walk Kernel

It's now possible to define a kernel k_z for labeled paths, using a positive definite kernel for vertices k_v and one for edges k_e:

$$k_z(h, h') = k_v(h_1, h'_1) \prod_{i=2}^{l} k_e(h_{2i-2}, h'_{2i-2}) k_v(h_{2i-1}, h'_{2i-1})$$

Finally, the graph kernel is defined as the expected value of k_z over every possible labeled path pair h and h' from two graphs G_1 and G_2:

$$K(G_1, G_2) = \sum_{h} \sum_{h'} k_z(h, h') p(h|G_1) p(h'|G_2)$$

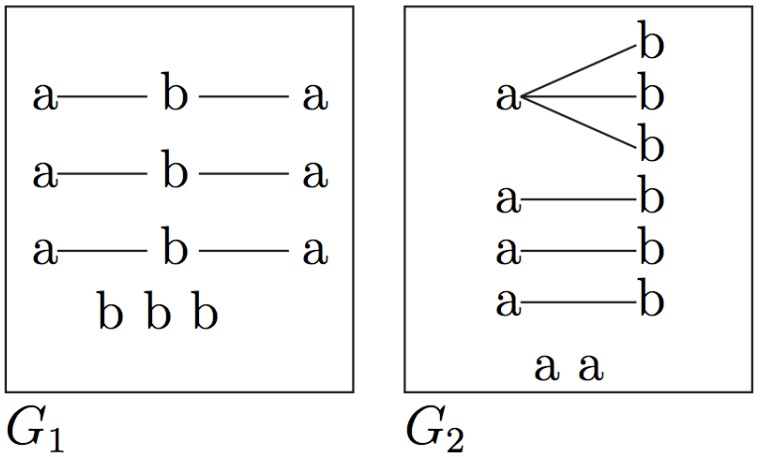
Marginalized Random Walk Kernel

Computational issues

- Direct computation of the kernel is infeasible because labeled paths may be infinite
- It is possible to rearrange equations so to obtain a recursive formulation
- Final result of the kernel obtained by solving a linear equation with a $|G_1||G_2| \times |G_1||G_2|$ coefficient matrix
- Conditions for convergence (i.e., linear equations have a solution) are given

Random walks - Drawbacks

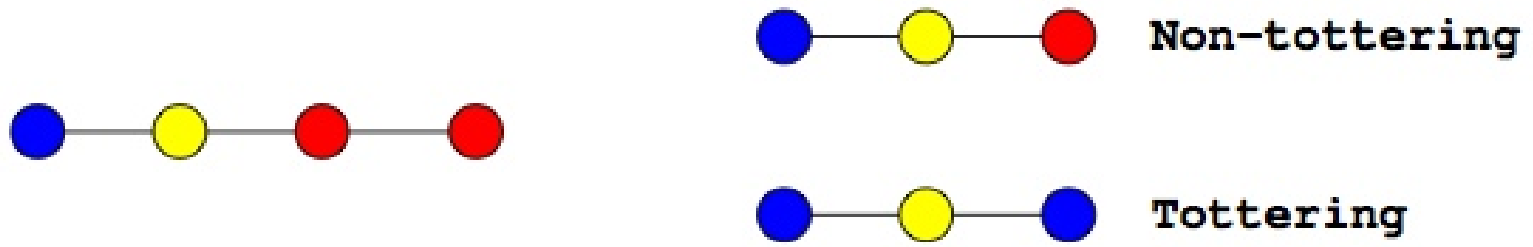
- not so expressive: easy to find identically mapped graphs



G_1 G_2

- Tottering problem: Since, in a walk, a vertex can be visited multiple times, small sets of connected vertices repeatedly visited may have a too high influence on the final value of the kernel.

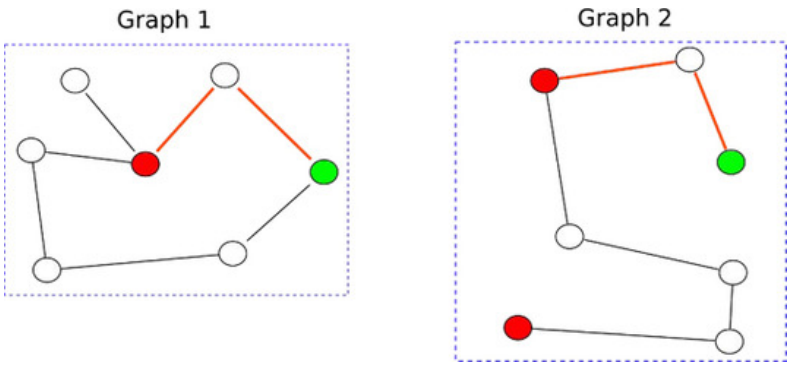
Tottering



- Tottering can be partly avoided by modifying the graphs in order to avoid visiting the same edge twice consecutively [29]
- Borgwardt et al. [32] proposed graph kernels based on shortest path between pairs of vertices

Shortest-Path Kernel

Shortest-paths for pairs of nodes in G_1 and G_2:



- Since the shortest path may not be unique, kernel focuses on the length of the shortest paths, i.e. given $d(v_i, v_j)$ the length of the shortest path between v_i and v_j:

$$K(G_1, G_2) = \sum_{v_i, v_j \in G_1} \sum_{u_i, u_j \in G_2} k_{length}(d(v_i, v_j), d(u_i, u_j))$$

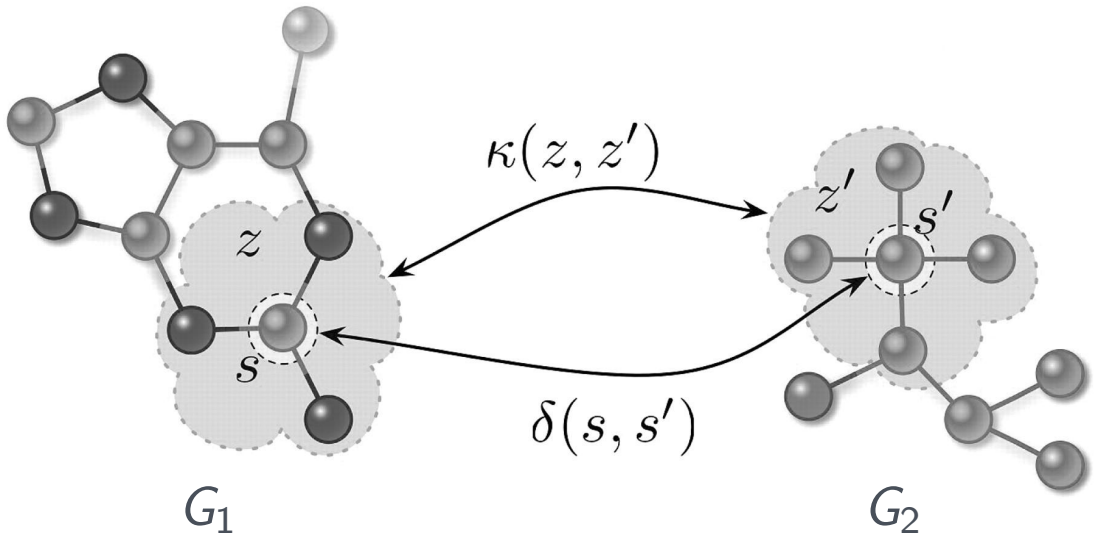
- Compute all-pairs-shortest-paths for G_1 and G_2 via Floyd-Warshall

Kernels Comparing Complex Substructures

Menchetti et al. [33] proposed the Weighted Decomposition Kernel, which extracts and compares all set of structures s (the *selector*) and their *context* z from the graph:

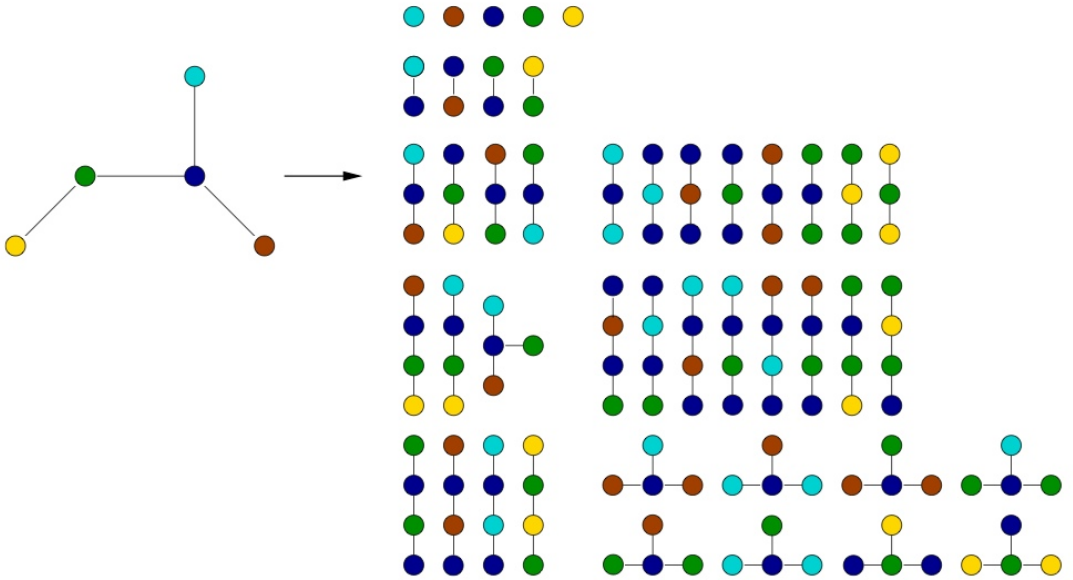
$$K(G_1, G_2) = \sum_{(s,z)\in R^{-1}(G_1)} \sum_{(s',z')\in R^{-1}(G_2)} \delta(s, s')\kappa(z, z')$$

- For example a structure can be a vertex and the context its set of neighbours:



Kernels Comparing Complex Substructures

Mahe and Vert [34] proposed the Tree-pattern Graph Kernel (revisitation of a proposal by Ramon and Gärtner), which compares subtree-like substructures in two graphs



- Subtrees are formed by selecting a subset of connected vertices from a graph (the same vertex may be selected more than once)

Kernels Comparing Complex Substructures

Tree-pattern Graph Kernel

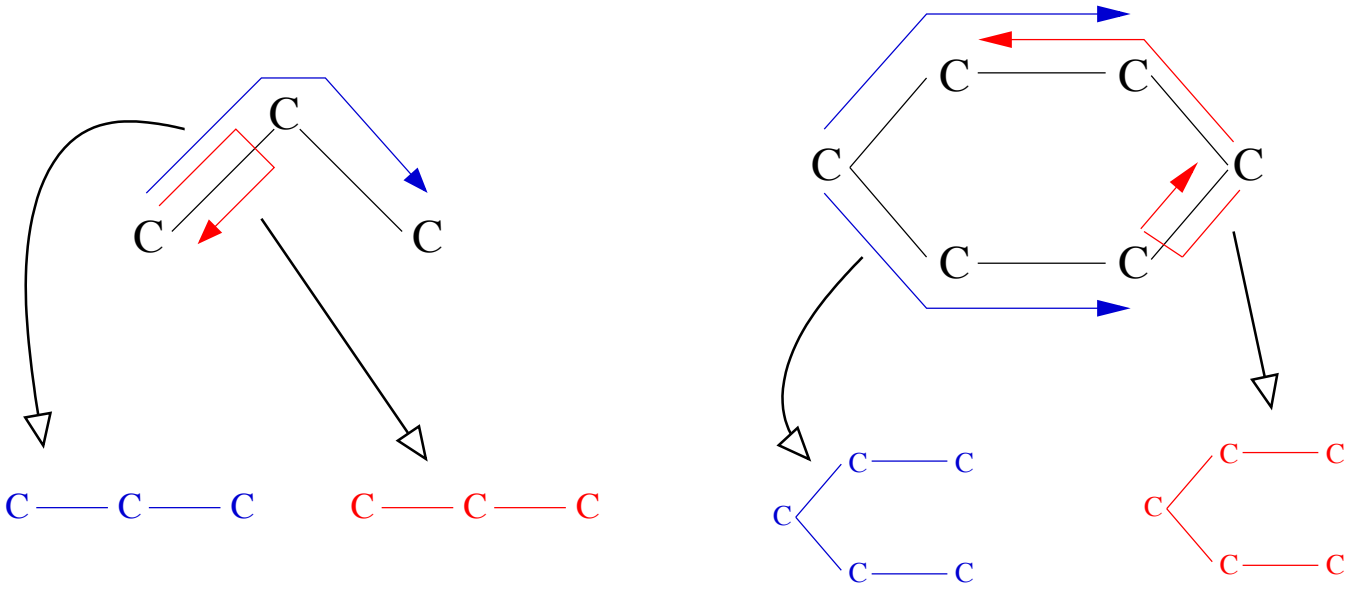
- Like the walk kernel, amounts to compute the (weighted) number of subtrees in the product graph
- Recursion: if $\mathcal{T}(v, n)$ denotes the weighted number of subtrees of depth n rooted at the vertex v, then:

$$\mathcal{T}(v, n+1) = \sum_{R \subset \mathcal{N}(v)} \prod_{v' \in R} \lambda_t(v, v') \mathcal{T}(v', n)$$

  where $\mathcal{N}(v)$ is the set of neighbors of v
- The complexity of the kernel is $O(|G_1||G_2|h\rho^{2\rho})$, where h is the depth of the tree patterns

Kernels Comparing Complex Substructures

Problem: Tree-pattern Tottering

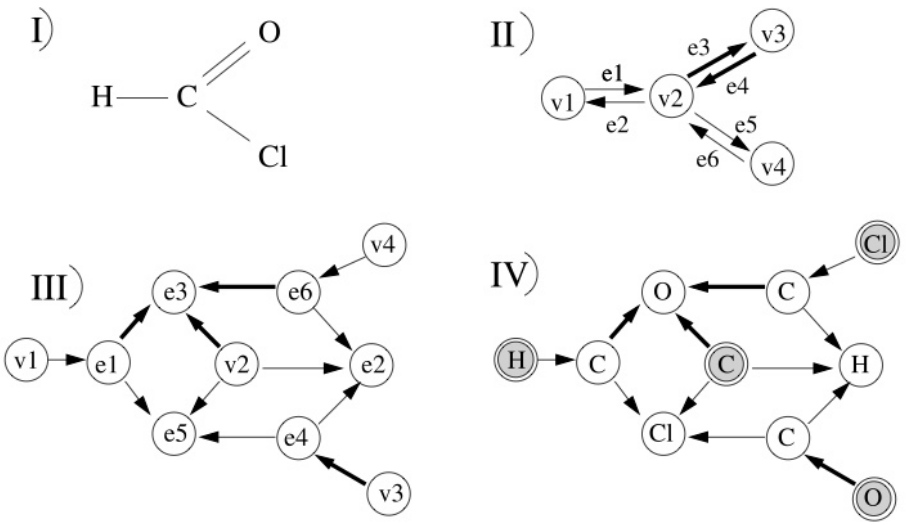


- The ratio of the number of tottering tree-patterns over the number of non-tottering tree-patterns quickly increases with the depth h of the trees

Kernels Comparing Complex Substructures

Tree-pattern Graph Kernel

- The non-tottering subtree kernel can be obtained by applying a *non-tottering graph transformation preprocessing*:



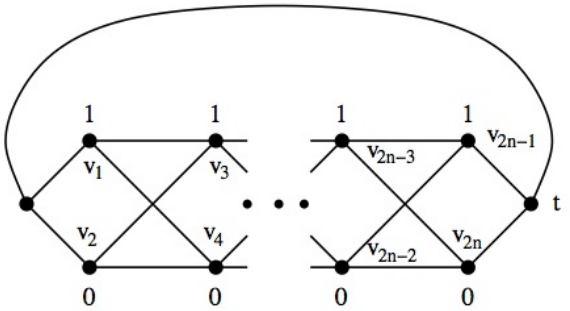
(I) The original molecule
(II) The corresponding graph G
(III) The transformed graph
(IV) The labels on the transformed graph

Different widths stand for different edges labels, and gray nodes are the nodes belonging to G

Kernels Comparing Complex Substructures

Horvath et al. [35] introduced the Cyclic Pattern Kernel

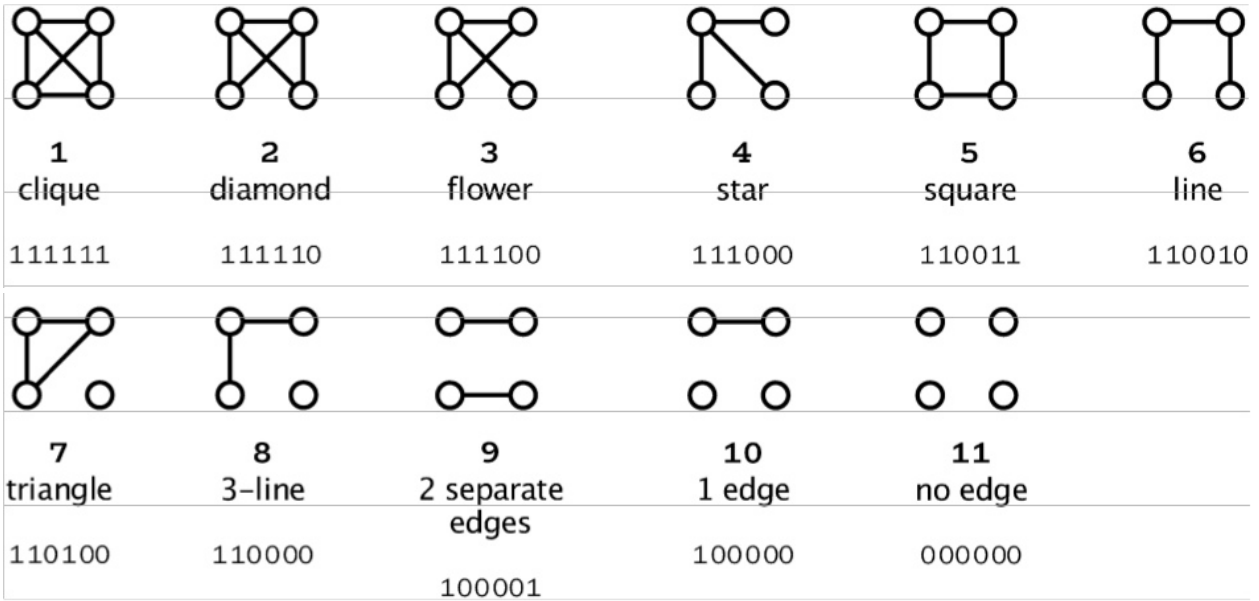
- It compares simple cycles in two graphs
- Needs to define a canonical string representation of each simple cycle, referred to as a cyclic pattern
- Problem: in the worst case the number of simple cycles is exponential in the number n of vertices



  - so, the kernel is NP-Hard to compute on general graphs !
- useful in scenarios where the number of simple cycles in a graph dataset is bounded by a constant

Kernels Comparing Complex Substructures

Borgwardt et al. [36] proposed the Graphlet Kernel, which counts the number of isomorphic common graphlets, i.e. subgraphs of limited size less or equal to k, in G_1 and G_2, e.g. graphlets with $k = 4$:



Kernels Comparing Complex Substructures

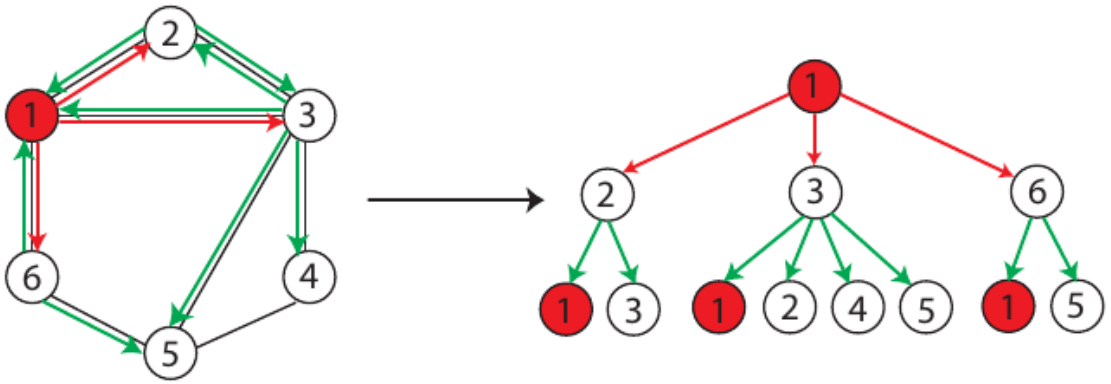
Graphlet Kernel

- Problems: Pairwise test of isomorphism is expensive and the number of graphlets scales as $O(n^k)$
- Possible solutions on unlabeled graphs: precompute isomorphism, employ sampling schemes on graphlets or restrict to graphlets with bounded out-degree to reduce the runtime
- The algorithm proposed for bounded out-degree graphlets has a complexity of $O(|G|\rho^k - 1)$.

Kernels Comparing Complex Substructures

Shervashidze and Borgwardt [37] introduced the Weisfeiler-Lehman subtree kernel, subsequently generalized in [38]

- based on the Weisfeiler-Lehman test of isomorphism on graphs
- exploits subtree patterns (tree walk visits)



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

Kernels Comparing Complex Substructures

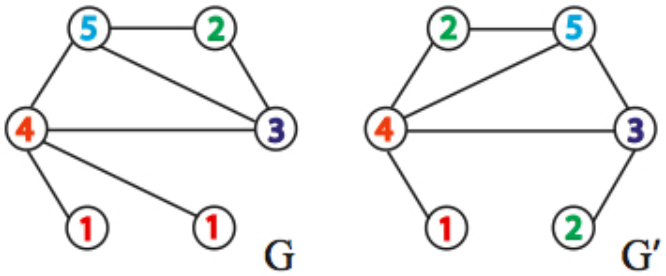
Weisfeiler-Lehman subtree kernel performs the following three steps h times for G_1 and G_2:

1. Sorting: represent each node v as a sorted label list L_v of its neighbours
2. Compression: compress each list into a hash value $hash(L_v)$
3. Relabeling: relabel v with $hash(L_v)$ as its new node label

Subsequently, the kernel is computed by counting the number of hard matching between the labels contained in the lists generated for G_1 and G_2

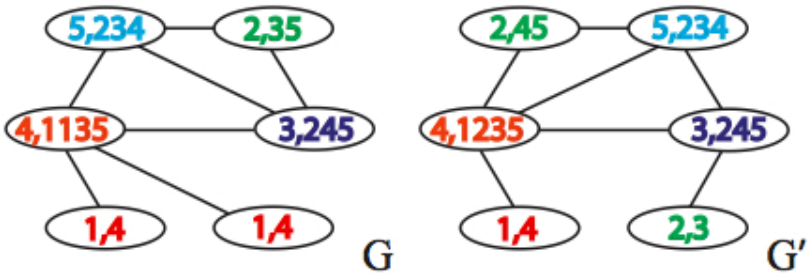
Weisfeiler-Lehman ST kernel - example

Given labeled graphs G and G′



a

1st iteration
Result of steps 1 and 2: multiset-label determination and sorting



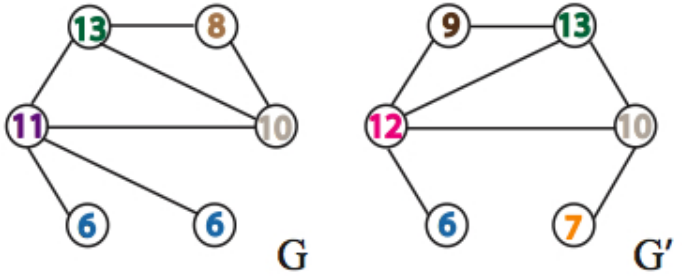
b

1st iteration
Result of step 3: label compression

1,4	⟶	6	3,245	⟶	10
2,3	⟶	7	4,1135	⟶	11
2,35	⟶	8	4,1235	⟶	12
2,45	⟶	9	5,234	⟶	13

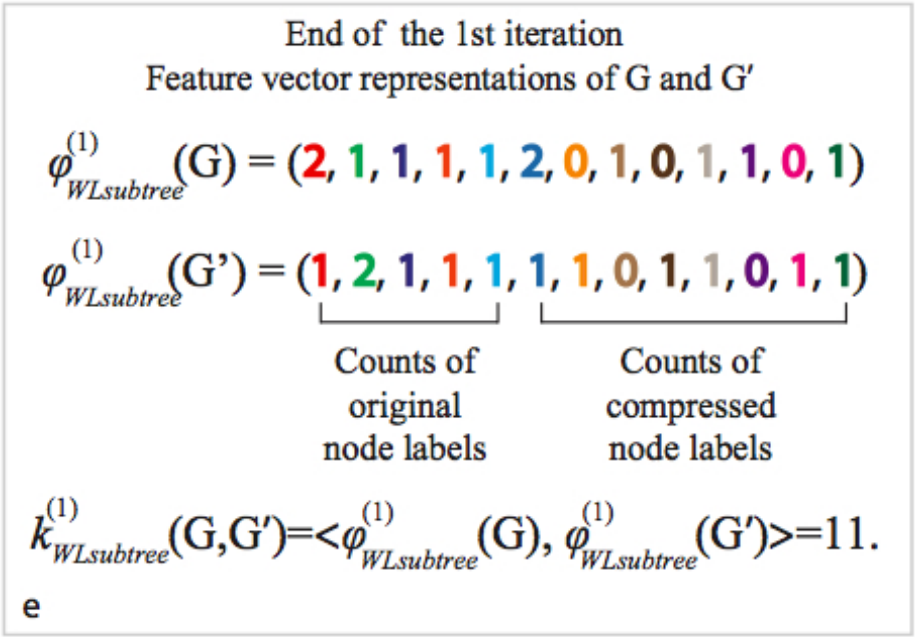
c

1st iteration
Result of step 4: relabeling



d

Weisfeiler-Lehman ST kernel - example



- Quite efficient: $O(hm)$ on 2 graphs, $O(Nhm + N2hn)$ on N graphs
- Expressivity: better than walk kernels, but at most hn active features for each graph

Weisfeiler-Lehman ST kernel

Features

- feature space: all subtree-patterns
- active features: all subtree-patterns up to heigth h that appear in a graph

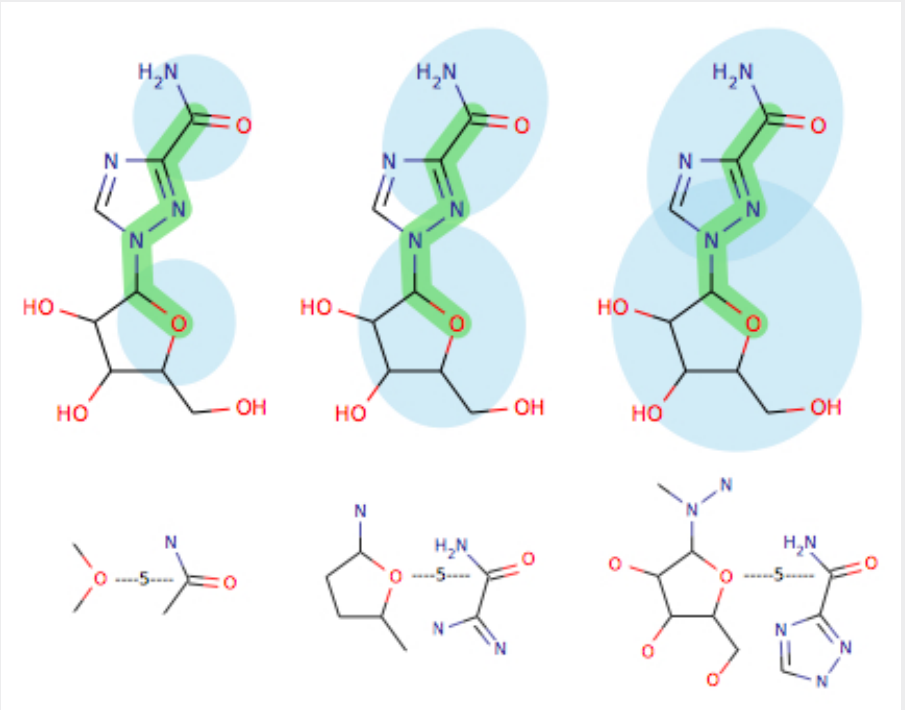
Considerations:

- only hard match between subtree-patterns
- good performance on some chemical datasets
- some more expressive extensions proposed, but with an higher computational complexity

A Weisfeiler-Lehman Kernel Extension

Costa and De Grave [39] proposed the Neighborhood Subgraph Pairwise Distance Kernel

Key idea



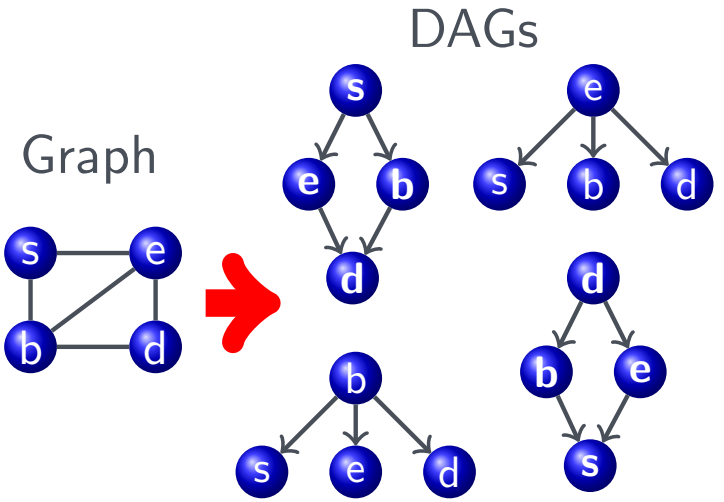
The kernel counts the number of identical pairs of neighboring graphs of radius r at distance d between two graphs

Each vertex is relabeled with a string that encodes the vertex distance from all other labeled vertices (plus the distance from the root vertex).
The graph encoding is obtained as the sorted edge list, where each edge is annotated with the endpoints new labels.
Resorts to hashing function to map the graph encoding string to a 32-bit integer.

Ordered Decompositional DAG Kernels

Da San Martino et al. [40] introduced a framework for graph kernels based on Directed Acyclic Graphs (DAGs):

- Decompose the graph in a multi set of DAGs: one tree-visit for each vertex $\Rightarrow$ same representation for isomorphic graphs



Edges between vertices at the same level of visit are removed

ODD Kernels

- The kernels for graphs are defined summing up the contribution of a local kernel for DAGs, over all pairs of DAGs in the multisets
- Kernel for DAGs obtained extending tree kernels to ordered DAGs

$$K_{K_T}(G_1, G_2) = \sum_{\substack{D_1 \in ODD(G_1) \\ D_2 \in ODD(G_2)}} K_{DAG}(D_1, D_2)$$

- Ordered DAGs (ODD) obtained through the recursive definition of the following alphanumeric string $S(v)$ for each vertex v:

$$S(v) = L(v) \cdot outdeg(v) \cdot \left(\prod_{i=1}^{outdeg(v)} S(ch_i[v]) \right)$$

ODD Kernels

Kernels for DAGs

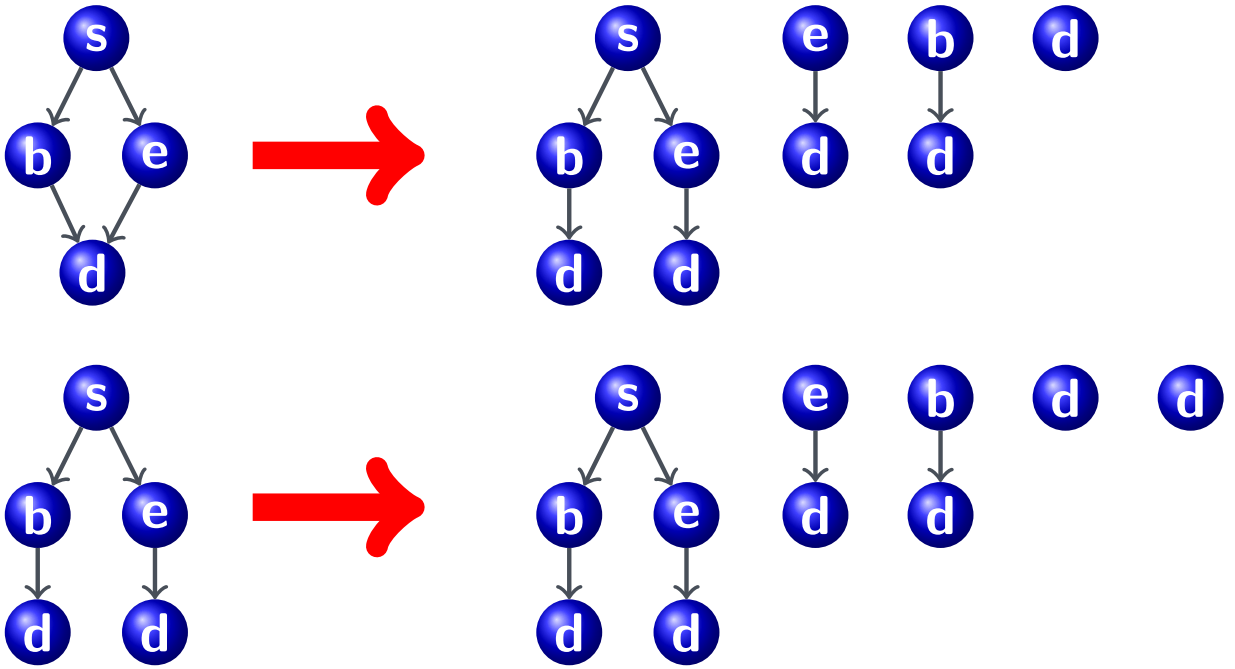
- The basic idea is to use tree-visits to project subdags to a tree space and then apply tree kernels on the visits
- If we consider tree kernels such as ST, SST, and PT, we have

$$K_{DAG}(D_1, D_2) = \sum_{\substack{v_1 \in V_{D_1} \\ v_2 \in V_{D_2}}} C(root(T(v_1)), root(T(v_2))),$$

where $C(\cdot, \cdot)$ is any of the local kernels defined for ST, SST, PT, and $T(v)$ is the tree-visit starting from v

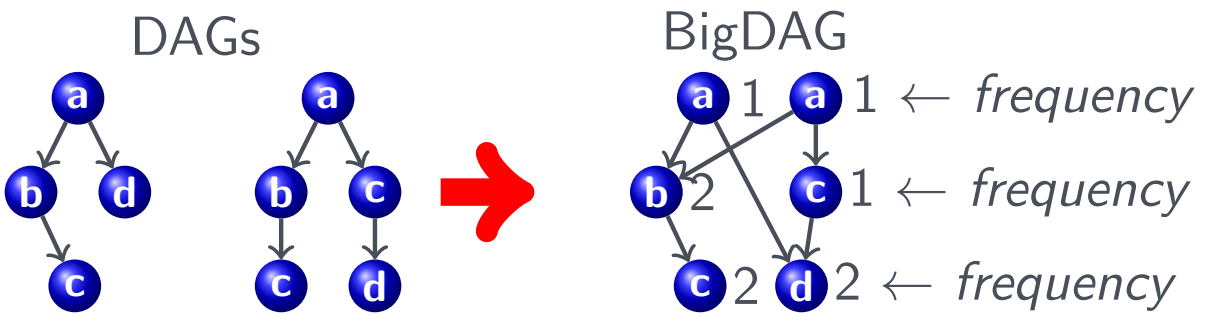
ODD Kernels

The feature space of K_{DAG} is equivalent to the feature space of the used tree kernel, however the mapping (i.e., ϕ) is not the same, e.g. for the ST kernel:



ODD Kernels

- Efficiency improved by defining a compact lossless representation for the multiset of DAGs (BigDAG) $\Rightarrow$ kernel between BigDAGs

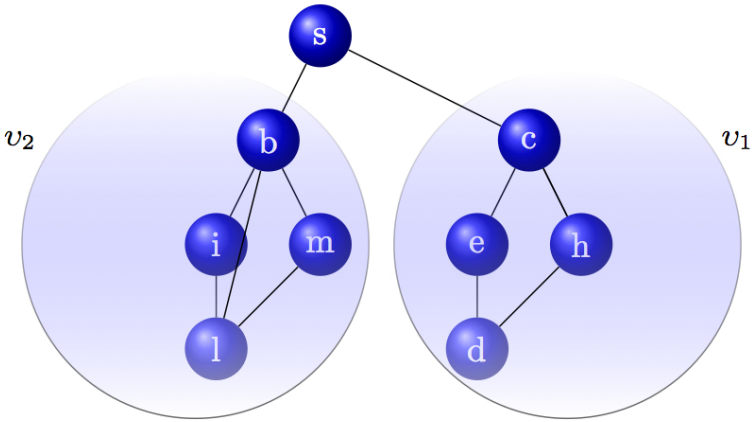


$$K_{BigDAG}(G_1, G_2) = \sum_{\substack{u_1 \in V(BigDAG(G_1)) \\ u_2 \in V(BigDAG(G_2))}} f_{u_1} f_{u_2} C(u_1, u_2),$$

where f_u is the frequency of the ordered DAG rooted at u

ODD Kernels

How many nodes for the BigDAG ?

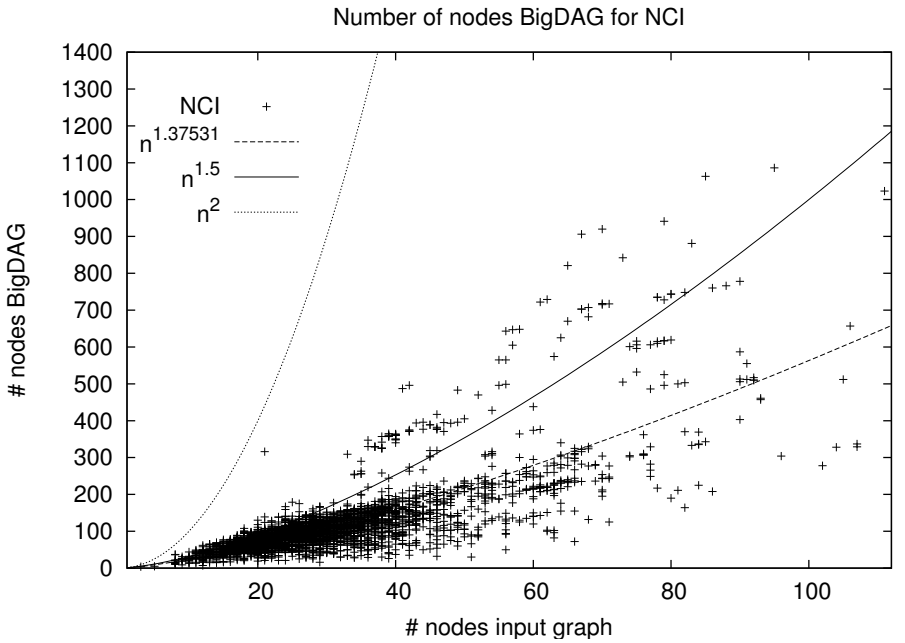


Bound on number of nodes:

$$|BigDAG(G)| \leq \sum_{i=1}^{q} (p_{v_i} + n_{v_i}^2)$$

where $v_i \in Polytree$ with degree p_{v_i}, containing n_{v_i} nodes of G

Embed the graph G in a Polytree where each node is a strongly connected component of G

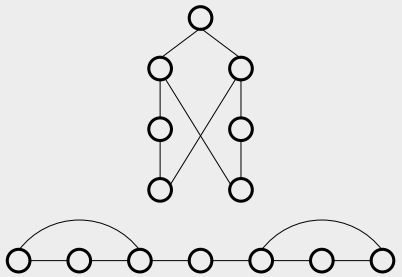


ODD Kernels

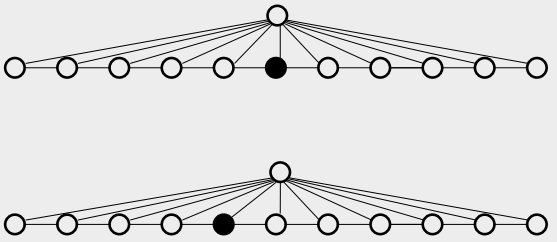
Complexity:

- $O(|ODD(G_1)||ODD(G_2)| \cdot Q(n))$, where $Q(n)$ is K_T complexity
- if $K_T \equiv K_{ST} \rightarrow O(n^3 \log n)$
- but limiting the depth of the DAGs: $K_{ST_{limited}} \rightarrow O(n \log n)$ (using hash tables to store BigDAGs)

Expressivity (ODD kernels can deal with self-loops by modifying node labels)



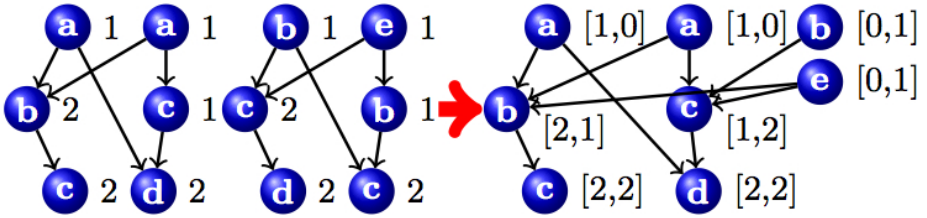
Graphs that cannot be discriminated by Fast Subtree kernel ODD kernel can!



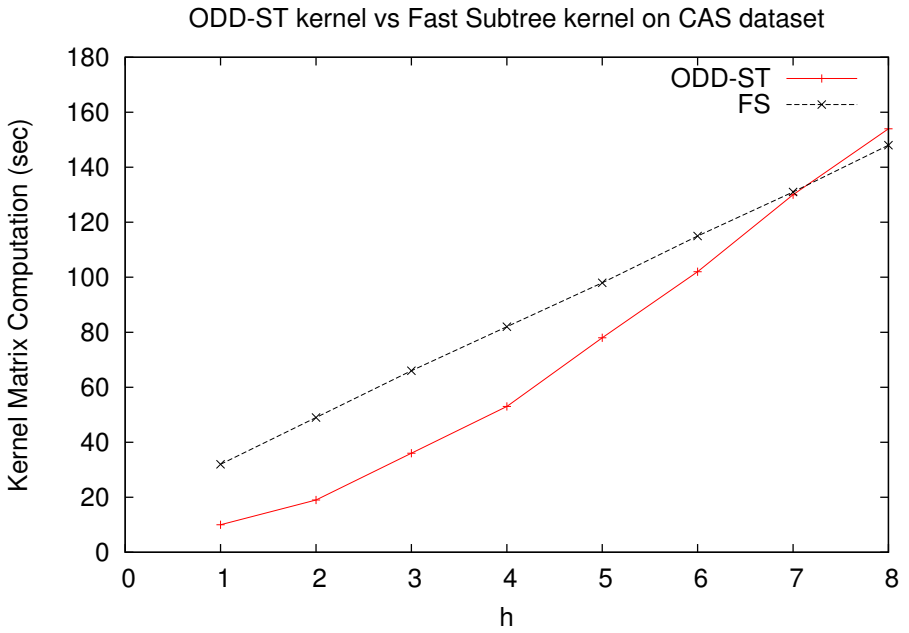
Neither Fast Subtree kernel, nor ODD kernel can discriminate these two graphs!

ODD Kernels

Efficient kernel matrix computation by using the BigDAG of the BigDAGs:



Actual time to compute the ODD-ST kernel matrix for dataset CAS (h = DAG depth)

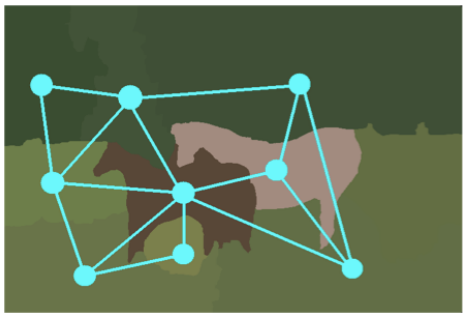


Some other Graph Kernels

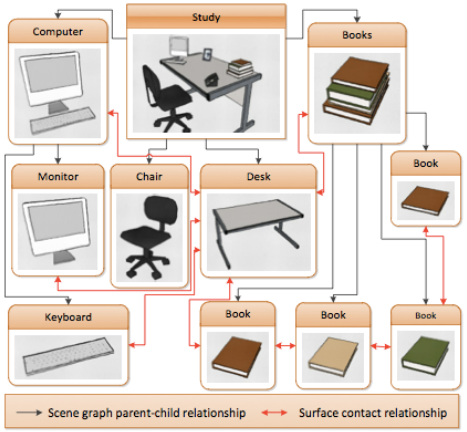
- Kernels for Chemoinformatics [41]
  - Three new kernels (Tanimoto, MinMax, Hybrid) based on the idea of molecular fingerprints and counting labeled paths of depth up to d using depth-first search from each possible vertex
- Graph Kernels between Point Clouds [42]
  - Extensions of graph kernels for point clouds, which allow one to use kernel methods for such objects as shapes, line drawings, or any three-dimensional point clouds
  - Kernels between covariance matrices and their factorizations on probabilistic graphical models
- The skew spectrum of graphs [43]
  - Based on mapping the adjacency matrix of any (weighted, directed, unlabeled) graph to a function on the symmetric group and computing bispectral invariants

Graph Kernels Applied to Computer Vision

Image Classification with Segmentation Graph Kernels [44]



Inexact graph matching based on kernels for object retrieval in image databases [45]



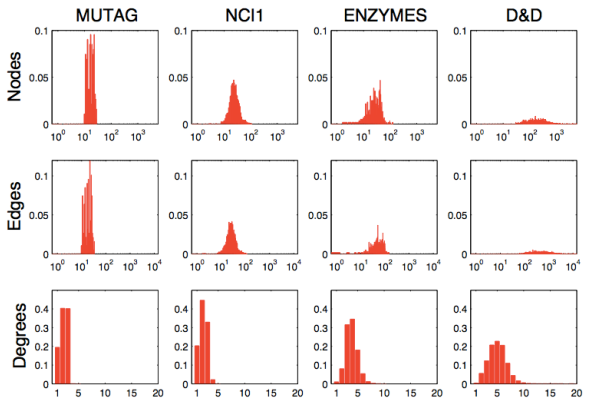
Characterizing Structural Relationships in Scenes Using Graph Kernels [46]

Graph Kernels which are not Definite Positive in General

- Optimal Assignment Kernel [47]
  - based on the idea of computing an optimal assignment from the atoms of one molecule to those of another one
  - includes information on neighborhood, membership to a certain structural element and other characteristics for each atom
- Edit-Distance Kernel [48]
  - Tries to combine the power of graph kernels and edit distances

Some experimental results from [38]

Example of computation times for some chemical datasets



Data Set	MUTAG	NCI1	NCI109	ENZYMES	D & D
Maximum # nodes	28	111	111	126	5748
Average # nodes	17.93	29.87	29.68	32.63	284.32
# labels	7	37	38	3	82
Number of graphs	188	4110	4127	600	1178
WL subtree	6”	7’20”	7’21”	20”	11’0”
WL edge	3”	1’5”	58”	11”	3 days
WL shortest path	2”	2’20”	2’23”	1’3”	484 days
Ramon & Gärtner	40’6”	81 days	81 days	38 days	103 days
p-random walk	4’42”	5 days	5 days	10’	4 days
Random walk	12”	9 days	9 days	12’19”	48 days
Graphlet count	3”	1’27”	1’27”	25”	30’21”
Shortest path	2”	4’38”	4’39”	5”	23h 17’2”

Some experimental results from [38]

10-fold cross-validation accuracy (binary classification)

Method/Data Set	MUTAG	NCI1	NCI109	ENZYMES	D & D
WL subtree	82.05 (±0.36)	82.19 (± 0.18)	82.46 (±0.24)	52.22 (±1.26)	79.78 (±0.36)
WL edge	81.06 (±1.95)	84.37 (±0.30)	84.49 (±0.20)	53.17 (±2.04)	77.95 (±0.70)
WL shortest path	83.78 (±1.46)	84.55 (±0.36)	83.53 (±0.30)	59.05 (±1.05)	79.43 (±0.55)
Ramon & Gärtner	85.72 (±0.49)	61.86 (±0.27)	61.67 (±0.21)	13.35 (±0.87)	57.27 (±0.07)
p-random walk	79.19 (±1.09)	58.66 (±0.28)	58.36 (±0.94)	27.67 (±0.95)	66.64 (±0.83)
Random walk	80.72 (±0.38)	64.34 (±0.27)	63.51 (± 0.18)	21.68 (±0.94)	71.70 (±0.47)
Graphlet count	75.61 (±0.49)	66.00 (±0.07)	66.59 (±0.08)	32.70 (±1.20)	78.59 (±0.12)
Shortest path	87.28 (±0.55)	73.47 (±0.11)	73.07 (±0.11)	41.68 (±1.79)	78.45 (±0.26)

WL = Weisfeiler-Lehman kernels

Other experimental results from [40]

Average accuracy results (when available) for Gaston, MOLFEA, Correlated Pattern Mining,
Marginalized Graph Kernel, SVM with Frequent Mining, gBoost, the Fast Subtree and the
ODD-ST kernels

Dataset	graphs	pos(%)	avg atoms	avg edges
Mutag	188	66.48	45.1	47.1
CAS	4337	55.36	29.9	30.9
CPDB	684	49.85	14.1	14.6
AIDS	1503	28.07	58.9	61.4
NCI1	4110	50.04	29.87	32.3

Kernel	Mutag		CAS		CPDB		AIDS		NCI1	
	Acc	Rank	Acc	Rank	Acc	Rank	Acc	Rank	Acc	Rank
Gaston	-	-	0.79	5	-	-	-	-	-	-
MOLFEA	-	-	-	-	-	-	0.785	5	-	-
CPM	-	-	0.801	4	0.760	6	0.832	2	-	-
MGK	0.808	4	0.771	7	0.765	4	0.762	7	-	-
freqSVM	0.808	4	0.773	6	0.778	3	0.782	6	-	-
gBoost	0.852	3	0.825	2	0.788	2	0.802	3	0.708 (μ=0.01)	3
FS	0.893 (h=1, c=10000)	1	0.816 (h=3, c=0.1)	3	0.763 (h=1, c=1)	5	0.791 (h=9, c=0.01)	4	0.863 (h=8, c=0.01)	1
$ODD-ST_h$	0.878 (h=2, λ=0.5, c=1000)	2	0.842 (h=4, λ=1.2, c=100)	1	0.804 (h=8, λ=1.2, c=10)	1	0.835 (h=5, λ=1.8, c=10)	1	0.853 (h=5, λ=1.4, c=100)	2

Data and Software Resources

Kernel for Trees

- Datasets
  - the INEX 2005 and INEX 2006 datasets can be found here: http://www.math.unipd.it/~dasan/tutorialwcci12.htm
  - LOGML: http://www.cs.rpi.edu/~zaki/software/logml
- Software
  - Tree Kernels in SVMLight, the ST, SST, PT (and others): http://disi.unitn.it/moschitti/Tree-Kernel.htm
  - Route Kernel: http://www.math.unipd.it/~dasan/routekernel.htm

Data and Software Resources

Kernel for Graphs

- Datasets
  - NCI datasets (and more): http://pubchem.ncbi.nlm.nih.gov/
  - CAS: http://cheminformatics.org/datasets/bursi
  - MUTAG: http://cdb.ics.uci.edu/cgibin/LearningDatasetsWeb.psp
  - AIDS: [49]
- Software
  - Fast neighborhood subgraph pairwise distance kernel: http://dtai.cs.kuleuven.be/ml/systems
  - Fast Subtree Kernel, Shortest Path Kernel (as well as some datasets): http://mlcb.is.tuebingen.mpg.de/Mitarbeiter/Nino/WL

Conclusions

- Kernel Methods are a viable solution to the design of learning algorithms for structured data
- The design of kernel functions for structured data poses specific challenges due to the high dimensionality of the data
  - Expressivity (sparsity) and computational complexity are the critical factors that should be taken into account
  - The Mapping Kernel provide an easy way to demonstrate positive semidefiniteness. Available kernels can be easily extended to suit a current task
  - Most of the available kernels are not able to deal with sparse domains.
  - The complexity is an issue not only during in learning phase but also during classification of novel examples

Future Directions: Tree kernels

- Expressivity
  - The Mapping Kernel is able to express a great number of kernels and allows to prove results valid for all its instances
  - Typically features are based on subtree like structures; Route kernels showed that other types of information can be effective!
  - Adaptive kernels may not be competitive on non-sparse domains (both from accuracy and complexity point of view). Approaches able to control how the data is represented would be interesting
- Complexity
  - More and more data is available in structured form; having to deal with huge datasets means that even constant factors in the complexity may be significant

Future Directions: Graph kernels

- More and more data is available in structured form; even single graphs can be of high dimensionality
- Thus graph kernels are moving towards the development of fast algorithms for computing the kernel functions
  - Among these, many kernels are computed through the use of hash functions (using an explicit feature space)
  - The use of explicit features spaces allow to apply existing techniques and develop novel ones for feature selection

Bibliography

[1] A. Moschitti, J. Chu-carroll, S. Patwardhan, J. Fan, and G. Riccardi. "Using syntactic and semantic structural kernels for classifying definition questions in jeopardy!". In Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing, pages 712724, Edinburgh, Scotland, UK., July 2011.

[2] D.E. Goldberg. "Genetic Algorithms in Search, Optimization and Machine Learning". Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 1989.

[3] J. Kennedy, R. Eberhart. "Particle Swarm Optimization". Proceedings of IEEE International Conference on Neural Networks. IV. pp. 19421948, 1995.

[4] P. Frasconi, M. Gori, A. Sperduti. "A general framework for adaptive processing of data structures". IEEE Transactions on Neural Networks 9(5): 768-786, 1998

[5] B. Hammer, A. Micheli, M. Strickert, and A. Sperduti. "A general framework for unsupervised processing of structured data". Neurocomputing, 57(5):3335, 2004.

[6] D. Kimura, T. Kuboyama, T. Shibuya, H. Kashima. "A Subpath Kernel for Rooted Unordered Trees". Proceedings of the 15th Pacific-Asia conference on Advances in knowledge discovery and data mining, pagg. 62-74, 2011.

[7] T. Kuboyama, K. Hirata, H. Kashima, e K.F. Aoki-Kinoshita, "A Spectrum Tree Kernel," Information and Media Technologies, vol. 2, 2007, pagg. 292-299.

[8] D. Haussler, "Convolution Kernels on Discrete Structures", 1999.

Bibliography

[9] S.V.N. Vishwanathan e A.J. Smola, "Fast kernels for string and tree matching", 2003.

[10] M. Collins e N. Duffy. "A New ranking algorithms for parsing and tagging: kernels over discrete structures, and the voted perceptron". Proceedings of the 40th Annual Meeting on Association for Computational Linguistics, Philadelphia, Pennsylvania: Association for Computational Linguistics, 2002, pagg. 263-270.

[11] A. Moschitti. "Making tree kernels practical for natural language learning". Proceedings of the Eleventh International Conference on European Association for Computational Linguistics, Trento, 2006.

[12] J. Suzuki, H. Isozaki. "Sequence and tree kernels with statistical feature mining". Advances in Neural Information Processing Systems, Vancouver: MIT Press, 2006.

[13] K. Rieck, T. Krueger, U. Brefeld, K. Müller. "Approximate kernels for trees". Journal of Machine Learning Research, 11(Feb):555580, 2010.

[14] N. Cristianini, J. Kandola, A. Elisseeff, J. Shawe-Taylor. "On kernel-target alignment". In Advances in Neural Information Processing Systems 14, volume 14, pages 367373, 2002.

[15] A. Moschitti. "Efficient Convolution Kernels for Dependency and Constituent Syntactic Trees'.' Machine Learning: ECML 2006, 2006, pagg. 318-329.

Bibliography

[16] H. Kashima e T. Koyanagi. "Kernels for Semi-Structured Data". Proceedings of the Nineteenth International Conference on Machine Learning, Morgan Kaufmann Publishers Inc., 2002, pagg. 291-298.

[17] M. Zhang, W. Che, A. Aw, C.L. Tan, G. Zhou, T. Liu, e S. Li. "A Grammar-driven Convolution Tree Kernel for Semantic Role Classification". Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics, Prague, Czech Republic: Association for Computational Linguistics, 2007, pagg. 200-207.

[18] S. Bloehdorn e A. Moschitti. "Structure and semantics for expressive text kernels". Proceedings of the sixteenth ACM conference on Conference on information and knowledge management, Lisbon, Portugal: ACM, 2007, pagg. 861-864.

[19] F. Aiolli, G. Da San Martino, A. Sperduti. "Route kernels for trees". Proceedings of International Conference on Machine Learning, June, 14 - 18, 2009, Montreal, Canada.

[20] T. Kuboyama, K. Shin, H. Kashima. "Flexible tree kernels based on counting the number of tree mappings". Proceedings of Machine Learning with Graphs, 2006

[21] K. Shin, T. Kuboyama. "A generalization of Hausslers convolution kernel - Mapping kernel and its application to tree kernels". Journal of Computer Science and Technology 25(5): 10401054 Sept. 2010

Bibliography

[22] K. Shin. "Mapping kernels defined over countably infinite mapping systems and their application". Journal of Machine Learning Research 20, pagg. 367-382, 2011.

[23] L. Nicotra, A. Micheli, A. Starita, "Fisher kernel for tree structured data". Neural Networks, 2004. Proceedings. 2004 IEEE International Joint Conference on, 2004, pagg. 1917-1922 vol.3.

[24] F. Aiolli, G. Da San Martino, A. Sperduti, M. Hagenbuchner, "Kernelized" Self Organizing Maps for Structured Data" Proceedings of the 2007 ESANN Conference, April 24 - 27, 2007, Bruges, Belgium.

[25] D. Bacciu, A. Micheli, A. Sperduti "Adaptive tree kernel by multinomial generative topographic mapping". The 2011 International Joint Conference on Neural Networks (IJCNN), pagg. 1651 - 1658, 2011.

[26] R.I. Kondor, J. Lafferty. "Diffusion kernels on graphs and other discrete structures". 2002.

[27] T. Gärtner, P. Flach, S. Wrobel. "On graph kernels: Hardness results and efficient alternatives". Lecture notes in computer science, 2003, pagg. 129-143.

[28] S.V.N. Vishwanathan, K.M. Borgwardt, N.N. Schraudolph. "Fast Computation of Graph Kernels".Neural information processing systems, pagg. 1449-1456, 2006.

Bibliography

[29] P. Mahe, N. Ueda, T. Akutsu, J. Perret, e J. Vert. "Extensions of marginalized graph kernels" Proceedings of the twenty-first international conference on Machine learning, Banff, Alberta, Canada: ACM, 2004, pag. 70.

[30] H. Kashima, K. Tsuda, A. Inokuchi. "Marginalized Kernels Between Labeled Graphs". ICML 2003, pagg. 321-328.

[31] K. Tsuda, T. Khin, K. Asai. "Marginalized Kernels for Biological Sequences". Bioinformatics 18, pagg. 268-275, 2002.

[32] K.M. Borgwardt, H. Kriegel. "Shortest-Path Kernels on Graphs". Proceedings of the Fifth IEEE International Conference on Data Mining, IEEE Computer Society, pagg. 74-81, 2005.

[33] S. Menchetti, F. Costa, P. Frasconi. "Weighted decomposition kernels". Proceedings of the 22nd international conference on Machine learning, Bonn, Germany: ACM, pagg. 585-592, 2005.

[34] P. Mahe, J. Vert. "Graph kernels based on tree patterns for molecules". Machine Learning, vol. 75, Apr. 2009, pagg. 3-35.

[35] T. Horvath, T. Gärtner, S. Wrobel. "Cyclic pattern kernels for predictive graph mining". Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining, Seattle, WA, USA: ACM, pagg. 158-167, 2004.

Bibliography

[36] K.M. Borgwardt, T. Petri, S.V.N. Vishwanathan, H. Kriegel. "An Efficient Sampling Scheme for Comparison of Large Graphs". Firenze: 2007.

[37] N. Shervashidze, K. M. Borgwardt. "Fast subtree kernels on graphs". In Y. Bengio, D. Schuurmans, J. Lafferty, C. K. I. Williams, and A. Culotta, editors, Proceedings of the Conference on Advances in Neural Information Processing Systems, pages 1660-1668, 2009.

[38] N. Shervashidze, P. Schweitzer, E. Jan van Leeuwen, K. Mehlhorn, K. M. Borgwardt. "Weisfeiler-Lehman kernels on graphs". Journal of Machine Learning Research (JMLR) 12(Sep):2539-2561, 2011.

[39] F. Costa, K. De Grave. "Fast neighborhood subgraph pairwise distance kernel". In Proceedings of the International Conference on Machine Learning, pages 255-262, 2010.

[40] G. Da San Martino, N. Navarin, A. Sperduti, "A Tree-Based Kernel for Graphs". To appear in Proceedings of Siam International Conference on Data Mining, 2012.

[41] L. Ralaivola, S. J. Swamidass, H. Saigo, P. Baldi. "Graph kernels for chemical informatics". Neural Networks 18(8): 1093-1110, 2005.

[42] F. R. Bach. "Graph kernels between point clouds". ICML 2008:25-32.

Bibliography

[43] R. I. Kondor, K. M. Borgwardt. "The skew spectrum of graphs". ICML 2008:496-503.

[44] Z. Harchaoui, F. Bach. "Image Classification with Segmentation Graph Kernels". CVPR 2007.

[45] J. Lebrun, P. H. Gosselin, S. Philipp-Foliguet. "Inexact graph matching based on kernels for object retrieval in image databases". Image Vision Comput. 29(11): 716-729, 2011.

[46] M. Fisher, M. Savva, P. Hanrahan. "Characterizing structural relationships in scenes using graph kernels". ACM Trans. Graph. 30(4): 34, 2011.

[47] M. Neuhaus, H. Bunke. "Edit distance-based kernel functions for structural pattern classification". Pattern Recognition 39(10): 1852-1863, 2006.

[48] H. Fröhlich, J. K. Wegner, F. Sieker, A. Zell. "Optimal assignment kernels for attributed molecular graphs". ICML 2005: 225-232.

[49] O. S. Weislow, R. Kiser, D. L. Fine, J. Bader, R. H. Shoemaker, M. R. Boyd, "New soluble-formazan assay for hiv-1 cytopathic effects: application to high-flux screening of synthetic and natural products for aids-antiviral activity." Journal of the National Cancer Institute, vol. 81, no. 8, pp. 57786, 1989.