Graph kernels and applications in chemoinformatics

Jean-Philippe Vert
Jean-Philippe.Vert@ensmp.fr

Center for Computational Biology
Ecole des Mines de Paris, ParisTech

International Workshop on Graph-based Representations in Pattern Recognition (Gbr'2007), Alicante, Spain, June 13, 2007.

Outline

Outline

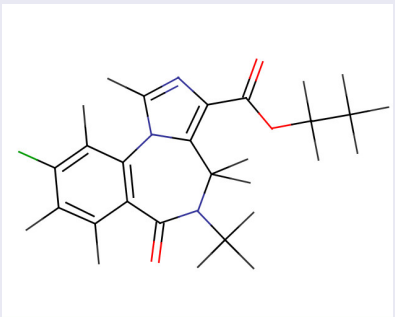
Ligand-Based Virtual Screening

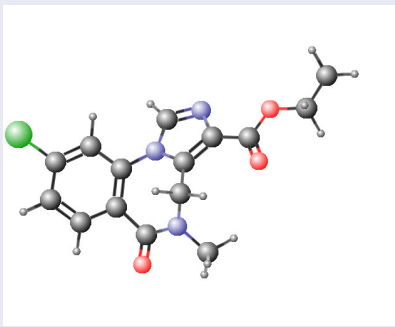
Objective

Build models to predict biochemical properties of small molecules from their structures.

Structures

$C_{15}H_{14}ClN_3O_3$

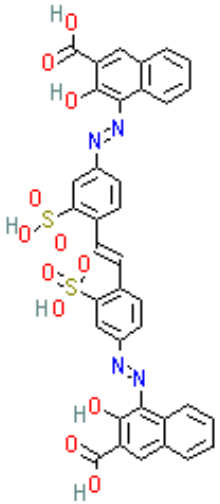




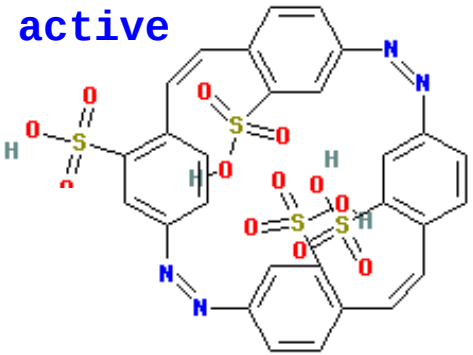
Properties

- binding to a therapeutic target,
- pharmacokinetics (ADME),
- toxicity...

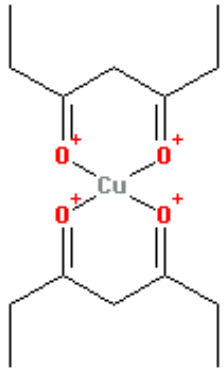
Example

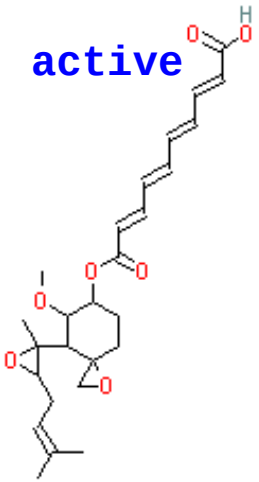


active

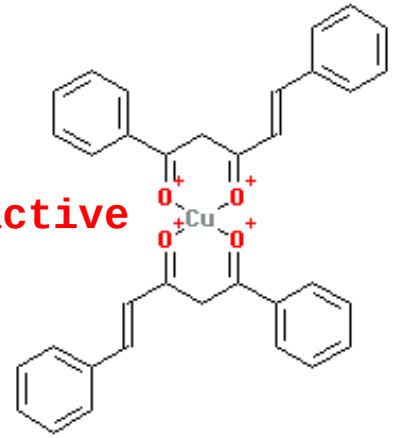


inactive

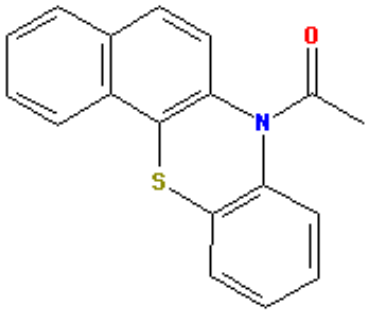




active



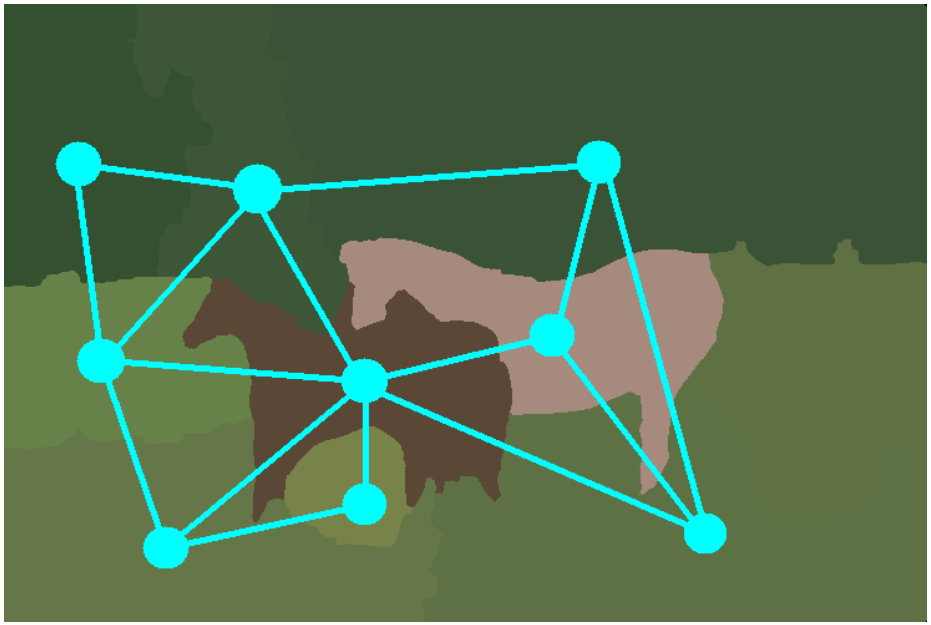
inactive



inactive

NCI AIDS screen results (from http://cactus.nci.nih.gov).

Image retrieval and classification



From Harchaoui and Bach (2007).

Formalization

The problem

- Given a set of training instances $(x_1, y_1), \ldots, (x_n, y_n)$, where x_i's are graphs and y_i's are continuous or discrete variables of interest,
- Estimate a function

$$y = f(x)$$

where x is any graph to be labeled.

- This is a classical regression or pattern recognition problem over the set of graphs.

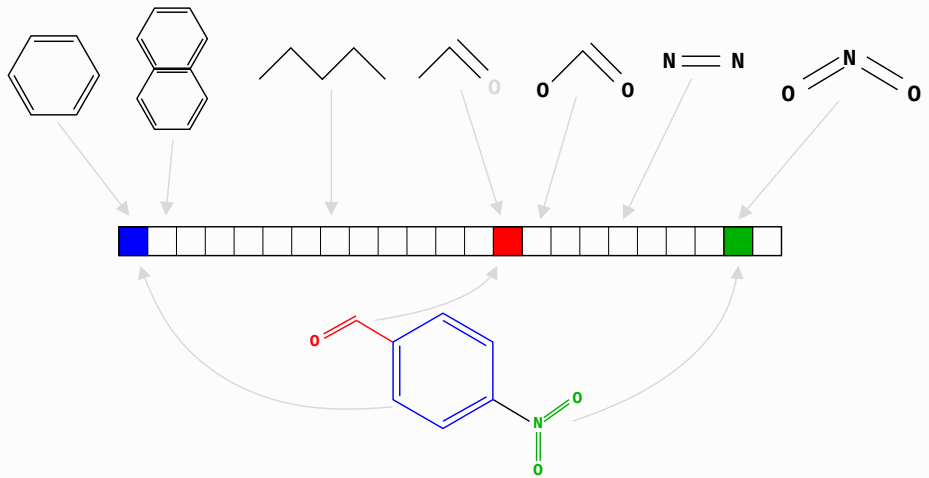
Classical approaches

Classical approaches

1. Map each molecule to a vector of fixed dimension.
2. Apply an algorithm for regression or pattern recognition over vectors.

Example: 2D structural keys in chemoinformatics

A vector indexed by a limited set of informative stuctures



+ NN, PLS, decision tree, ...

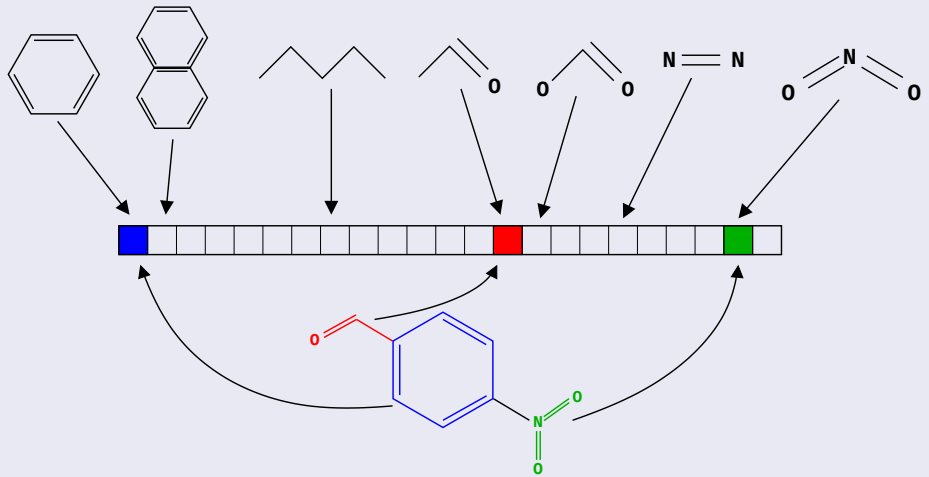
Classical approaches

Classical approaches

1. Map each molecule to a vector of fixed dimension.
2. Apply an algorithm for regression or pattern recognition over vectors.

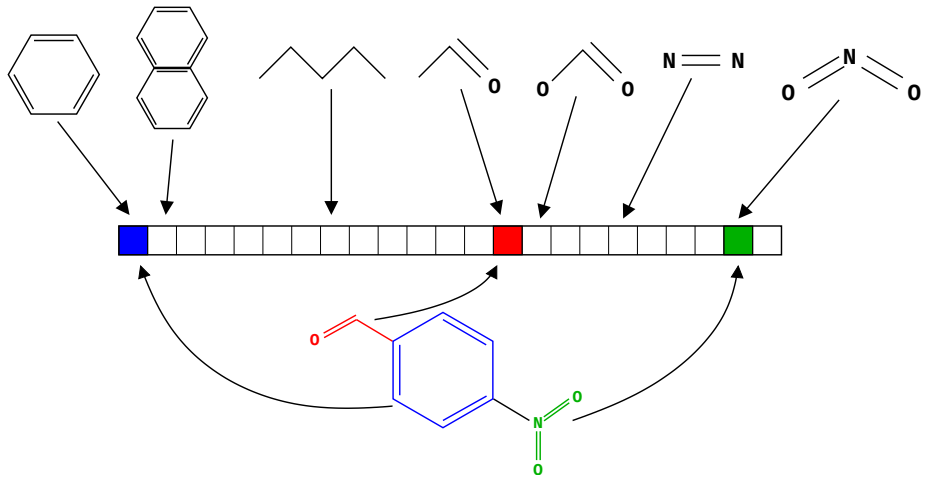
Example: 2D structural keys in chemoinformatics

A vector indexed by a limited set of informative stuctures



+ NN, PLS, decision tree, ...

Difficulties



- Expressiveness of the features (which features are relevant?)
- Large dimension of the vector representation (memory storage, speed, statistical issues)

The kernel trick

Kernel

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

$$K(x, x') = \Phi(x)^\top \Phi(x') \,.$$

The trick

- Many linear algorithms for regression or pattern recognition can be expressed only in terms of inner products between vectors
- Computing the kernel is often more efficient than computing $\Phi(x)$, especially in high or infinite dimensions!

The kernel trick

Kernel

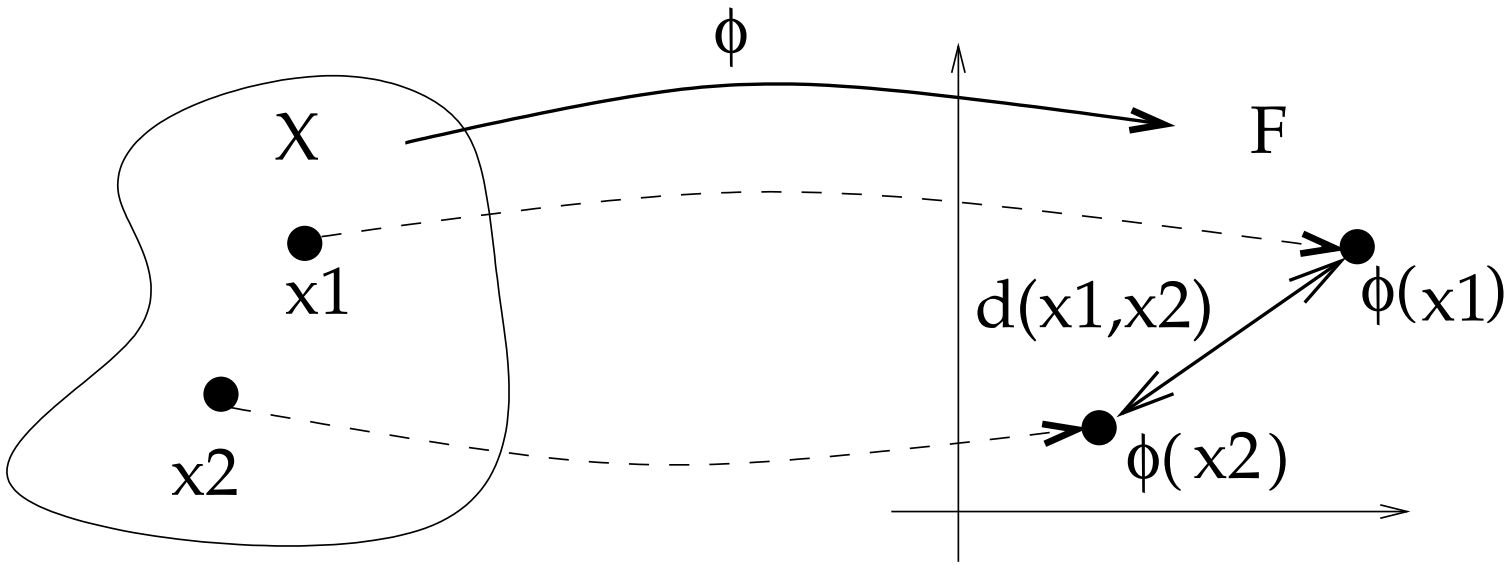
- Let $\Phi(x)$ be a vector representation of the graph x
- The kernel between two graphs is defined by:

$$K(x, x') = \Phi(x)^\top \Phi(x') \,.$$

The trick

- Many linear algorithms for regression or pattern recognition can be expressed only in terms of inner products between vectors
- Computing the kernel is often more efficient than computing $\Phi(x)$, especially in high or infinite dimensions!

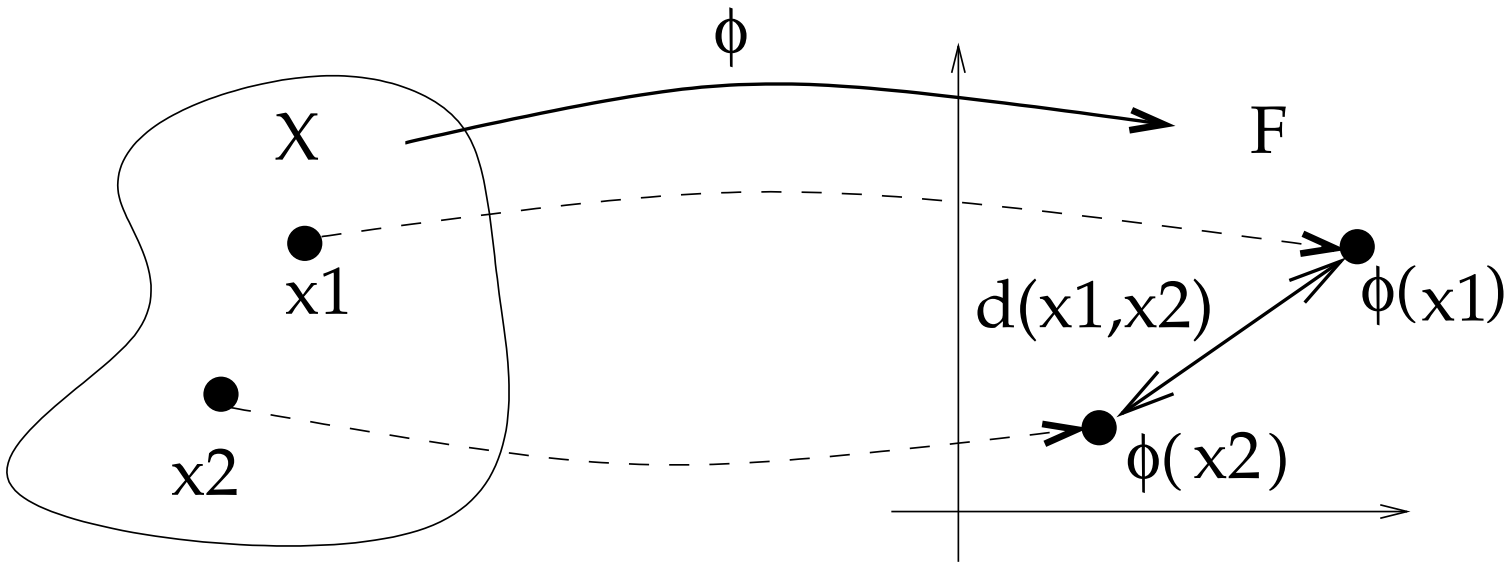
Kernel trick example: computing distances in the feature space



$$
\begin{aligned}
d_K(\mathbf{x}_1, \mathbf{x}_2)^2 &= \| \Phi(\mathbf{x}_1) - \Phi(\mathbf{x}_2) \|_{\mathcal{H}}^2 \\
&= \langle \Phi(\mathbf{x}_1) - \Phi(\mathbf{x}_2), \Phi(\mathbf{x}_1) - \Phi(\mathbf{x}_2) \rangle_{\mathcal{H}} \\
&= \langle \Phi(\mathbf{x}_1), \Phi(\mathbf{x}_1) \rangle_{\mathcal{H}} + \langle \Phi(\mathbf{x}_2), \Phi(\mathbf{x}_2) \rangle_{\mathcal{H}} - 2 \langle \Phi(\mathbf{x}_1), \Phi(\mathbf{x}_2) \rangle_{\mathcal{H}}
\end{aligned}
$$

$$
d_K(\mathbf{x}_1, \mathbf{x}_2)^2 = K(\mathbf{x}_1, \mathbf{x}_1) + K(\mathbf{x}_2, \mathbf{x}_2) - 2K(\mathbf{x}_1, \mathbf{x}_2)
$$

Kernel trick example: computing distances in the feature space



$$
\begin{aligned}
d_K(\mathbf{x}_1, \mathbf{x}_2)^2 &= \| \Phi(\mathbf{x}_1) - \Phi(\mathbf{x}_2) \|_{\mathcal{H}}^2 \\
&= \langle \Phi(\mathbf{x}_1) - \Phi(\mathbf{x}_2), \Phi(\mathbf{x}_1) - \Phi(\mathbf{x}_2) \rangle_{\mathcal{H}} \\
&= \langle \Phi(\mathbf{x}_1), \Phi(\mathbf{x}_1) \rangle_{\mathcal{H}} + \langle \Phi(\mathbf{x}_2), \Phi(\mathbf{x}_2) \rangle_{\mathcal{H}} - 2 \langle \Phi(\mathbf{x}_1), \Phi(\mathbf{x}_2) \rangle_{\mathcal{H}}
\end{aligned}
$$

$$
d_K(\mathbf{x}_1, \mathbf{x}_2)^2 = K(\mathbf{x}_1, \mathbf{x}_1) + K(\mathbf{x}_2, \mathbf{x}_2) - 2K(\mathbf{x}_1, \mathbf{x}_2)
$$

Positive Definite (p.d.) Kernels

Definition

A positive definite (p.d.) kernel on a set $\mathcal{X}$ is a function $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ symmetric:

$$\forall \left(\mathbf{x}, \mathbf{x}'\right) \in \mathcal{X}^2, \quad K\left(\mathbf{x}, \mathbf{x}'\right) = K\left(\mathbf{x}', \mathbf{x}\right),$$

and which satisfies, for all $N \in \mathbb{N}$, $(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N) \in \mathcal{X}^N$ et $(a_1, a_2, \ldots, a_N) \in \mathbb{R}^N$:

$$\sum_{i=1}^{N} \sum_{j=1}^{N} a_i a_j K\left(\mathbf{x}_i, \mathbf{x}_j\right) \geq 0\,.$$

P.d. kernels are inner products

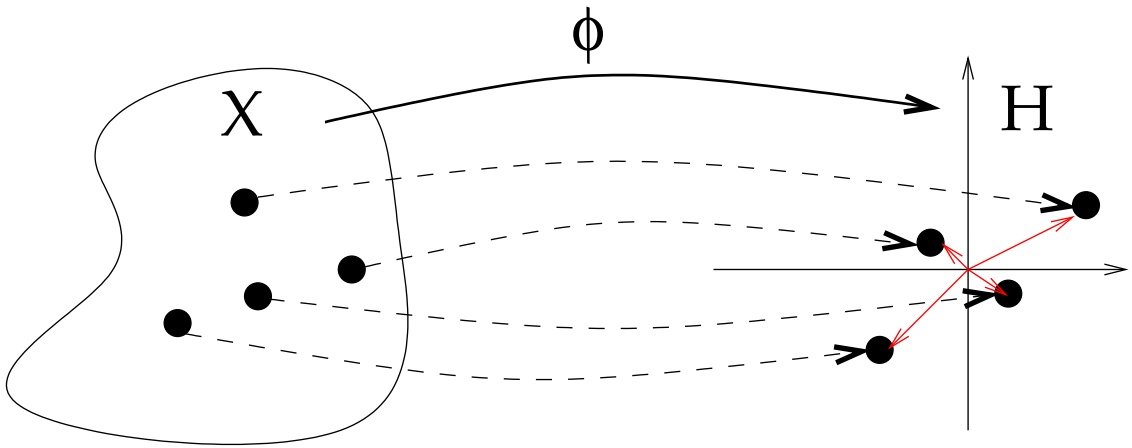
Theorem (Aronszajn, 1950)

K is a p.d. kernel on the set $\mathcal{X}$ if and only if there exists a Hilbert space $\mathcal{H}$ and a mapping

$$\Phi : \mathcal{X} \mapsto \mathcal{H} \ ,$$

such that, for any $\mathbf{x}, \mathbf{x}'$ in $\mathcal{X}$:

$$K\left(\mathbf{x}, \mathbf{x}'\right) = \left\langle \Phi\left(\mathbf{x}\right), \Phi\left(\mathbf{x}'\right) \right\rangle_{\mathcal{H}} \ .$$

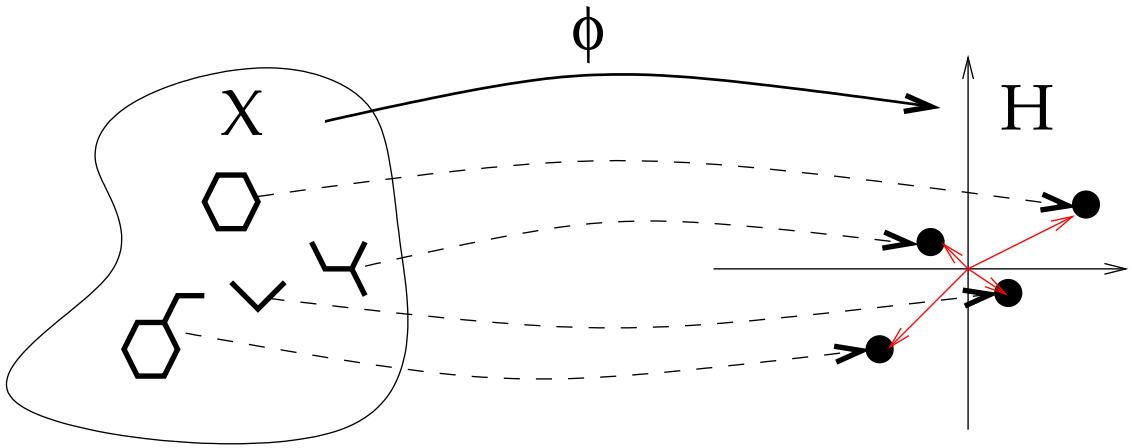


Graph kernels

Definition

- A graph kernel $K(x, x')$ is a p.d. kernel over the set of (labeled) graphs.
- It is equivalent to an embedding $\Phi : \mathcal{X} \mapsto \mathcal{H}$ of the set of graphs to a Hilbert space through the relation:

$$K(x, x') = \Phi(x)^\top \Phi(x') .$$



Summary

The problem

- Regression and pattern recognition over labeled graphs
- Classical vector representation is both statistically and computationally challenging

The kernel approach

By defining a graph kernel we work implicitly in large (potentially infinite!) dimensions:

- Allows to consider a large number of potentially important features.
- No need to store explicitly the vectors (no problem of memory storage or hash clashes) thanks to the kernel trick
- Use of regularized statistical algorithm (SVM, kernel PLS, kernel perceptron...)to handle the statistical problem of large dimension

Summary

The problem

- Regression and pattern recognition over labeled graphs
- Classical vector representation is both statistically and computationally challenging

The kernel approach

By defining a graph kernel we work implicitly in large (potentially infinite!) dimensions:

- Allows to consider a large number of potentially important features.
- No need to store explicitly the vectors (no problem of memory storage or hash clashes) thanks to the kernel trick
- Use of regularized statistical algorithm (SVM, kernel PLS, kernel perceptron...)to handle the statistical problem of large dimension

Outline

Expressiveness vs Complexity

Definition: Complete graph kernels

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

$$\forall G_1, G_2 \in \mathcal{X}, \quad d_K(G_1, G_2) = 0 \implies G_1 \simeq G_2 \,.$$

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

Expressiveness vs Complexity trade-off

- If a graph kernel is not complete, then there is no hope to learn all possible functions over $\mathcal{X}$: 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?

Expressiveness vs Complexity

Definition: Complete graph kernels

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

$$\forall G_1, G_2 \in \mathcal{X}, \quad d_K(G_1, G_2) = 0 \implies G_1 \simeq G_2 \,.$$

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

Expressiveness vs Complexity trade-off

- If a graph kernel is not complete, then there is no hope to learn all possible functions over $\mathcal{X}$: 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?

Complexity of complete kernels

Proposition (Gärtner et al., 2003)

Computing any complete graph kernel is at least as hard as the graph isomorphism problem.

Proof

- For any kernel K the complexity of computing d_K is the same as the complexity of computing K, because:

$$d_K(G_1, G_2)^2 = K(G_1, G_1) + K(G_2, G_2) - 2K(G_1, G_2)\,.$$

- If K is a complete graph kernel, then computing d_K solves the graph isomorphism problem ($d_K(G_1, G_2) = 0$ iff $G_1 \simeq G_2$). □

Complexity of complete kernels

Proposition (Gärtner et al., 2003)

Computing any complete graph kernel is at least as hard as the graph isomorphism problem.

Proof

- For any kernel K the complexity of computing d_K is the same as the complexity of computing K, because:

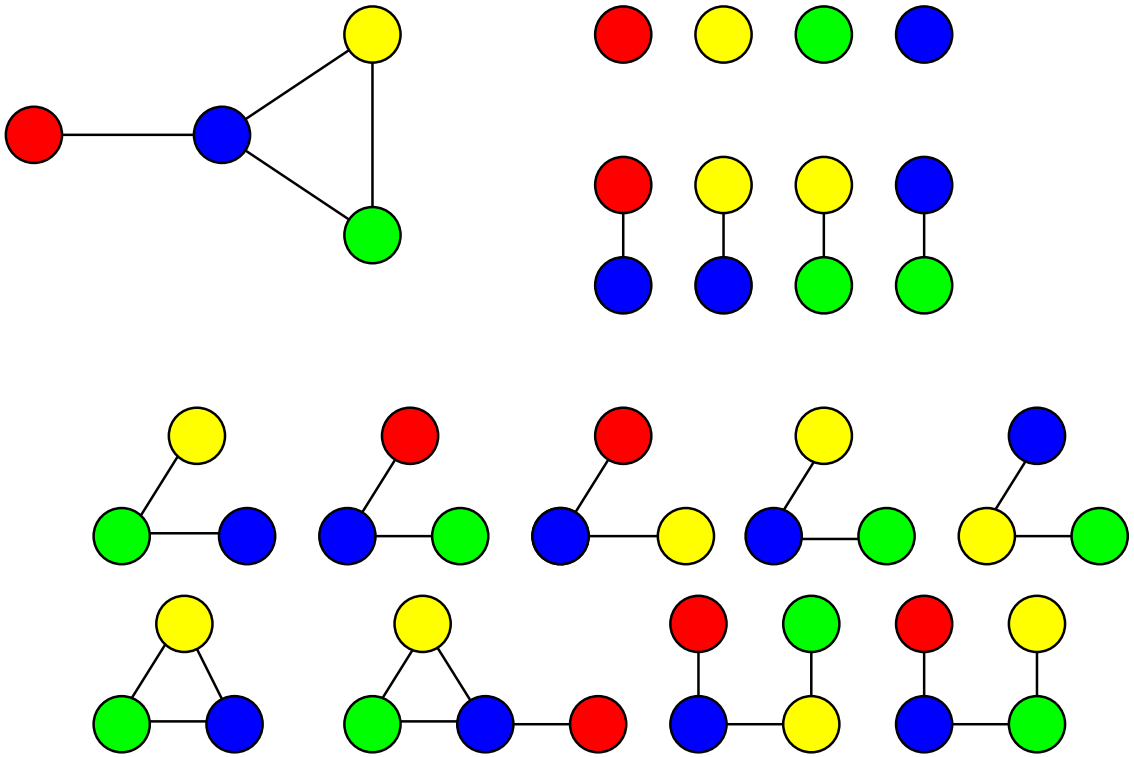
$$d_K(G_1, G_2)^2 = K(G_1, G_1) + K(G_2, G_2) - 2K(G_1, G_2)\,.$$

- If K is a complete graph kernel, then computing d_K solves the graph isomorphism problem ($d_K(G_1, G_2) = 0$ iff $G_1 \simeq G_2$). $\square$

Subgraphs

Definition

A subgraph of a graph (V, E) is a connected graph (V', E') with $V' \subset V$ and $E' \subset E$.



Subgraph kernel

Definition

- Let $(\lambda_G)_{G \in \mathcal{X}}$ a set or nonnegative real-valued weights
- For any graph $G \in \mathcal{X}$, let

$$\forall H \in \mathcal{X}, \quad \Phi_H(G) = \left| \left\{ G' \text{ is a subgraph of } G \,:\, G' \simeq H \right\} \right| .$$

- The subgraph kernel between any two graphs G_1 and $G_2 \in \mathcal{X}$ is defined by:

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

Subgraph kernel complexity

Proposition (Gärtner et al., 2003)

Computing the subgraph kernel is NP-hard.

Proof (1/2)

- Let P_n be the path graph with n vertices.
- Subgraphs of P_n are path graphs:

$$\Phi(P_n) = n e_{P_1} + (n-1) e_{P_2} + \ldots + e_{P_n} .$$

- The vectors $\Phi(P_1), \ldots, \Phi(P_n)$ are linearly independent, therefore:

$$e_{P_n} = \sum_{i=1}^{n} \alpha_i \Phi(P_i) ,$$

where the coefficients α_i can be found in polynomial time (solving a $n \times n$ triangular system).

Subgraph kernel complexity

Proposition (Gärtner et al., 2003)

Computing the subgraph kernel is NP-hard.

Proof (1/2)

- Let P_n be the path graph with n vertices.
- Subgraphs of P_n are path graphs:

$$\Phi(P_n) = n e_{P_1} + (n-1) e_{P_2} + \ldots + e_{P_n} .$$

- The vectors $\Phi(P_1), \ldots, \Phi(P_n)$ are linearly independent, therefore:

$$e_{P_n} = \sum_{i=1}^{n} \alpha_i \Phi(P_i) ,$$

where the coefficients α_i can be found in polynomial time (solving a $n \times n$ triangular system).

Subgraph kernel complexity

Proposition (Gärtner et al., 2003)

Computing the subgraph kernel is NP-hard.

Proof (2/2)

- If G is a graph with n vertices, then it has a path that visits each node exactly once (Hamiltonian path) if and only if $\Phi(G)^\top e_n > 0$, i.e.,

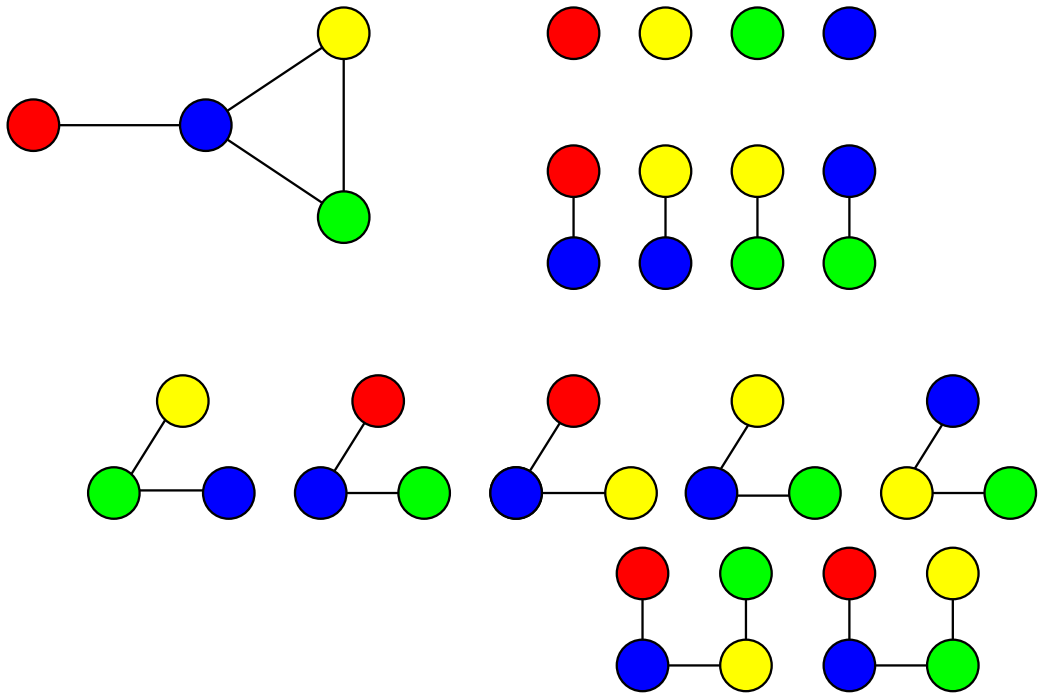
$$\Phi(G)^\top \left(\sum_{i=1}^{n} \alpha_i \Phi(P_i) \right) = \sum_{i=1}^{n} \alpha_i K_{subgraph}(G, P_i) > 0 \,.$$

- The decision problem whether a graph has a Hamiltonian path is NP-complete. $\square$

Paths

Definition

- A path of a graph (V, E) is sequence of distinct vertices $v_1, \ldots, v_n \in V$ $(i \neq j \implies v_i \neq v_j)$ such that $(v_i, v_{i+1}) \in E$ for $i = 1, \ldots, n-1$.
- Equivalently the paths are the linear subgraphs.



Path kernel

Definition

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

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

where $\mathcal{P} \subset \mathcal{X}$ is the set of path graphs.

Proposition (Gärtner et al., 2003)

Computing the path kernel is NP-hard.

Proof

Same as the subgraph kernel. □

Path kernel

Definition

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

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

where $\mathcal{P} \subset \mathcal{X}$ is the set of path graphs.

Proposition (Gärtner et al., 2003)

Computing the path kernel is NP-hard.

Proof

Same as the subgraph kernel. □

Path kernel

Definition

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

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

where $\mathcal{P} \subset \mathcal{X}$ is the set of path graphs.

Proposition (Gärtner et al., 2003)

Computing the path kernel is NP-hard.

Proof

Same as the subgraph kernel. $\square$

Summary

Expressiveness vs Complexity trade-off

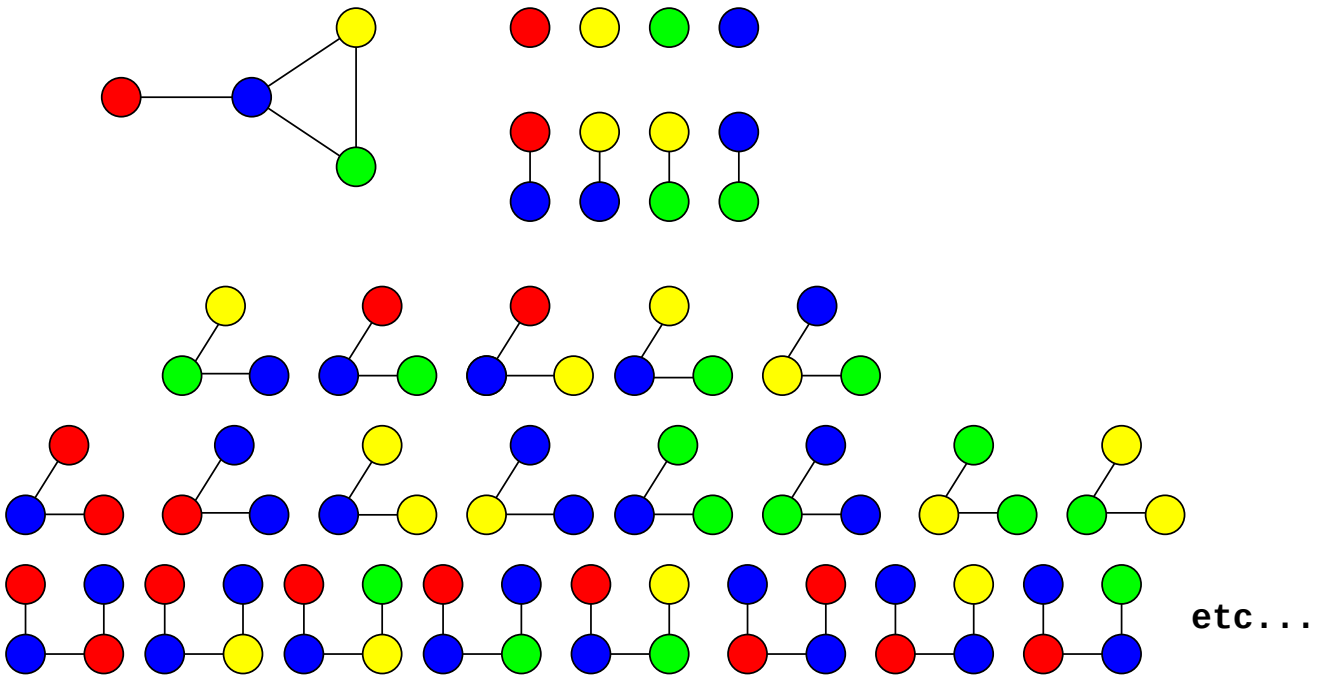
- It is intractable to compute complete graph kernels.
- It is intractable to compute the subgraph kernels.
- Restricting subgraphs to be linear does not help: it is also intractable to compute the path kernel.
- One approach to define polynomial time computable graph kernels is to have the feature space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

Outline

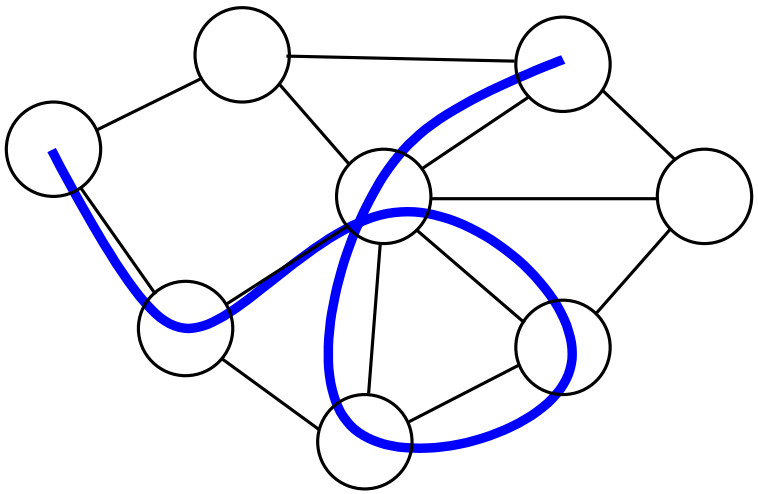
Walks

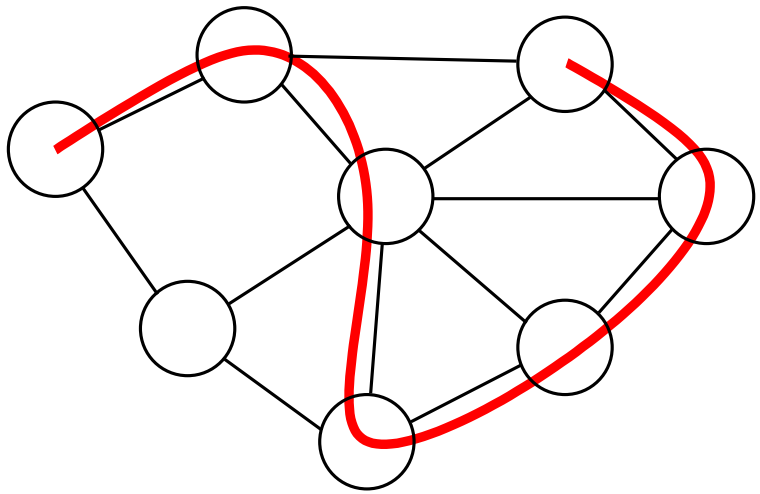
Definition

- A walk of a graph (V, E) is sequence of $v_1, \ldots, v_n \in V$ such that $(v_i, v_{i+1}) \in E$ for $i = 1, \ldots, n-1$.
- We note $\mathcal{W}_n(G)$ the set of walks with n vertices of the graph G, and $\mathcal{W}(G)$ the set of all walks.



Paths and walks





Walk kernel

Definition

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

$$\Phi_s(G) = \sum_{w \in \mathcal{W}(G)} \lambda_G(w) \mathbf{1}\left(s \text{ is the label sequence of } w\right) .$$

- A walk kernel is a graph kernel defined by:

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

Walk kernel

Definition

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

$$\Phi_s(G) = \sum_{w \in \mathcal{W}(G)} \lambda_G(w) \mathbf{1}\left(s \text{ is the label sequence of } w\right) .$$

- A walk kernel is a graph kernel defined by:

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

Walk kernel examples

Examples

- The *n*th-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 independant 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).

Walk kernel examples

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 independant 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).

Walk kernel examples

Examples

- The *n*th-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 independant 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).

Computation of walk kernels

Proposition

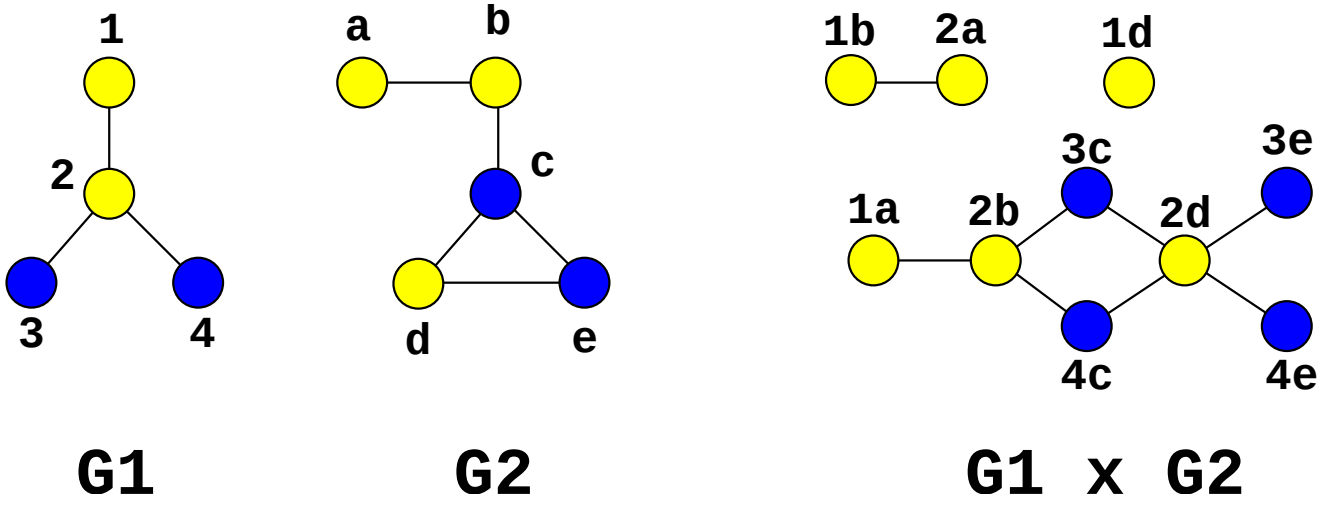
These three kernels (*n*th-order, random and geometric walk kernels) can be computed efficiently in polynomial time.

Product graph

Definition

Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be two graphs with labeled vertices. The product graph $G = G_1 \times G_2$ is the graph $G = (V, E)$ with:

1. $V = \{(v_1, v_2) \in V_1 \times V_2 \,:\, v_1 \text{ and } v_2 \text{ have the same label}\}$,
2. $E = \left\{\left((v_1, v_2), (v_1', v_2')\right) \in V \times V \,:\, (v_1, v_1') \in E_1 \text{ and } (v_2, v_2') \in E_2\right\}$.



Walk kernel and product graph

Lemma

There is a bijection between:

1. The pairs of walks $w_1 \in \mathcal{W}_n(G_1)$ and $w_2 \in \mathcal{W}_n(G_2)$ with the same label sequences,
2. The walks on the product graph $w \in \mathcal{W}_n(G_1 \times G_2)$.

Corollary

$$
\begin{aligned}
K_{walk}(G_1, G_2) &= \sum_{s \in \mathcal{S}} \Phi_s(G_1) \Phi_s(G_2) \\
&= \sum_{(w_1, w_2) \in \mathcal{W}(G_1) \times \mathcal{W}(G_1)} \lambda_{G_1}(w_1) \lambda_{G_2}(w_2) \mathbf{1}(l(w_1) = l(w_2)) \\
&= \sum_{w \in \mathcal{W}(G_1 \times G_2)} \lambda_{G_1 \times G_2}(w) \,.
\end{aligned}
$$

Walk kernel and product graph

Lemma

There is a bijection between:

1. The pairs of walks $w_1 \in \mathcal{W}_n(G_1)$ and $w_2 \in \mathcal{W}_n(G_2)$ with the same label sequences,
2. The walks on the product graph $w \in \mathcal{W}_n(G_1 \times G_2)$.

Corollary

$$
\begin{aligned}
K_{walk}(G_1, G_2) &= \sum_{s \in \mathcal{S}} \Phi_s(G_1) \Phi_s(G_2) \\
&= \sum_{(w_1, w_2) \in \mathcal{W}(G_1) \times \mathcal{W}(G_1)} \lambda_{G_1}(w_1) \lambda_{G_2}(w_2) \mathbf{1}(l(w_1) = l(w_2)) \\
&= \sum_{w \in \mathcal{W}(G_1 \times G_2)} \lambda_{G_1 \times G_2}(w) \,.
\end{aligned}
$$

Computation of the nth-order walk kernel

- For the nth-order walk kernel we have $\lambda_{G_1 \times G_2}(w) = 1$ if the length of w is n, 0 otherwise.
- Therefore:

$$K_{nth-order}(G_1, G_2) = \sum_{w \in \mathcal{W}_n(G_1 \times G_2)} 1 \,.$$

- Let A be the adjacency matrix of $G_1 \times G_2$. Then we get:

$$K_{nth-order}(G_1, G_2) = \sum_{i,j} [A^n]_{i,j} = \mathbf{1}^\top A^n \mathbf{1} \,.$$

- Computation in $O(n|G_1||G_2|d_1 d_2)$, where d_i is the maximum degree of G_i.

Computation of random and geometric walk kernels

- In both cases $\lambda_G(w)$ for a walk $w = v_1 \dots v_n$ can be decomposed as:

$$\lambda_G(v_1 \dots v_n) = \lambda^i(v_1) \prod_{i=2}^{n} \lambda^t(v_{i-1}, v_i) \,.$$

- Let Λ_i be the vector of $\lambda^i(v)$ and Λ_t be the matrix of $\lambda^t(v, v')$:

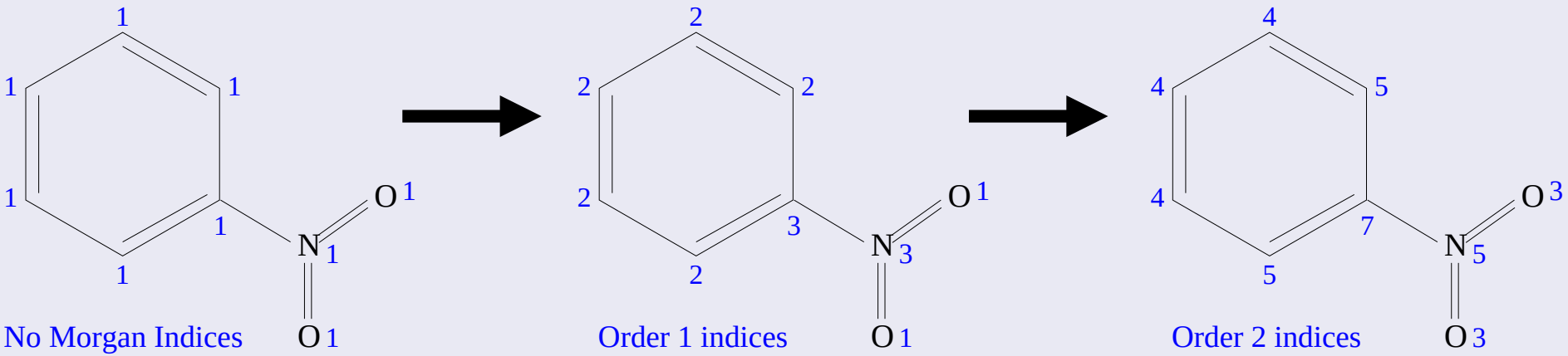
$$\begin{aligned} K_{walk}(G_1, G_2) &= \sum_{n=1}^{\infty} \sum_{w \in \mathcal{W}_n(G_1 \times G_2)} \lambda^i(v_1) \prod_{i=2}^{n} \lambda^t(v_{i-1}, v_i) \\ &= \sum_{n=0}^{\infty} \Lambda_i \Lambda_t^n \mathbf{1} \\ &= \Lambda_i \left(I - \Lambda_t\right)^{-1} \mathbf{1} \end{aligned}$$

- Computation in $O(|G_1|^3 |G_2|^3)$

Outline

Extensions 1: label enrichment

Atom relabebling with the Morgan index

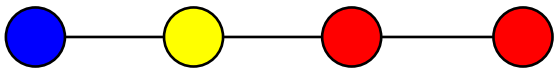


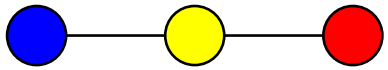
- Compromise between fingerprints and structural keys features.
- Other relabeling schemes are possible (graph coloring).
- Faster computation with more labels (less matches implies a smaller product graph).

Extension 2: Non-tottering walk kernel

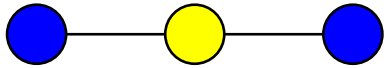
Tottering walks

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





Non-tottering

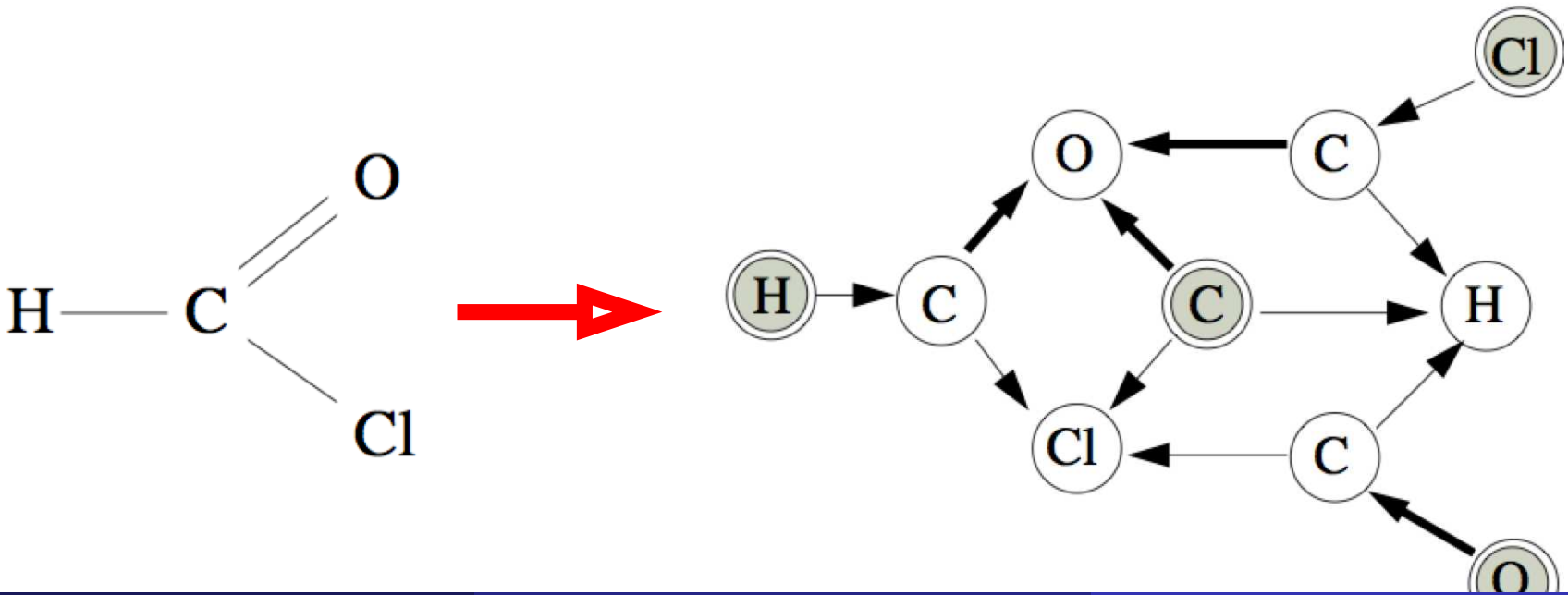


Tottering

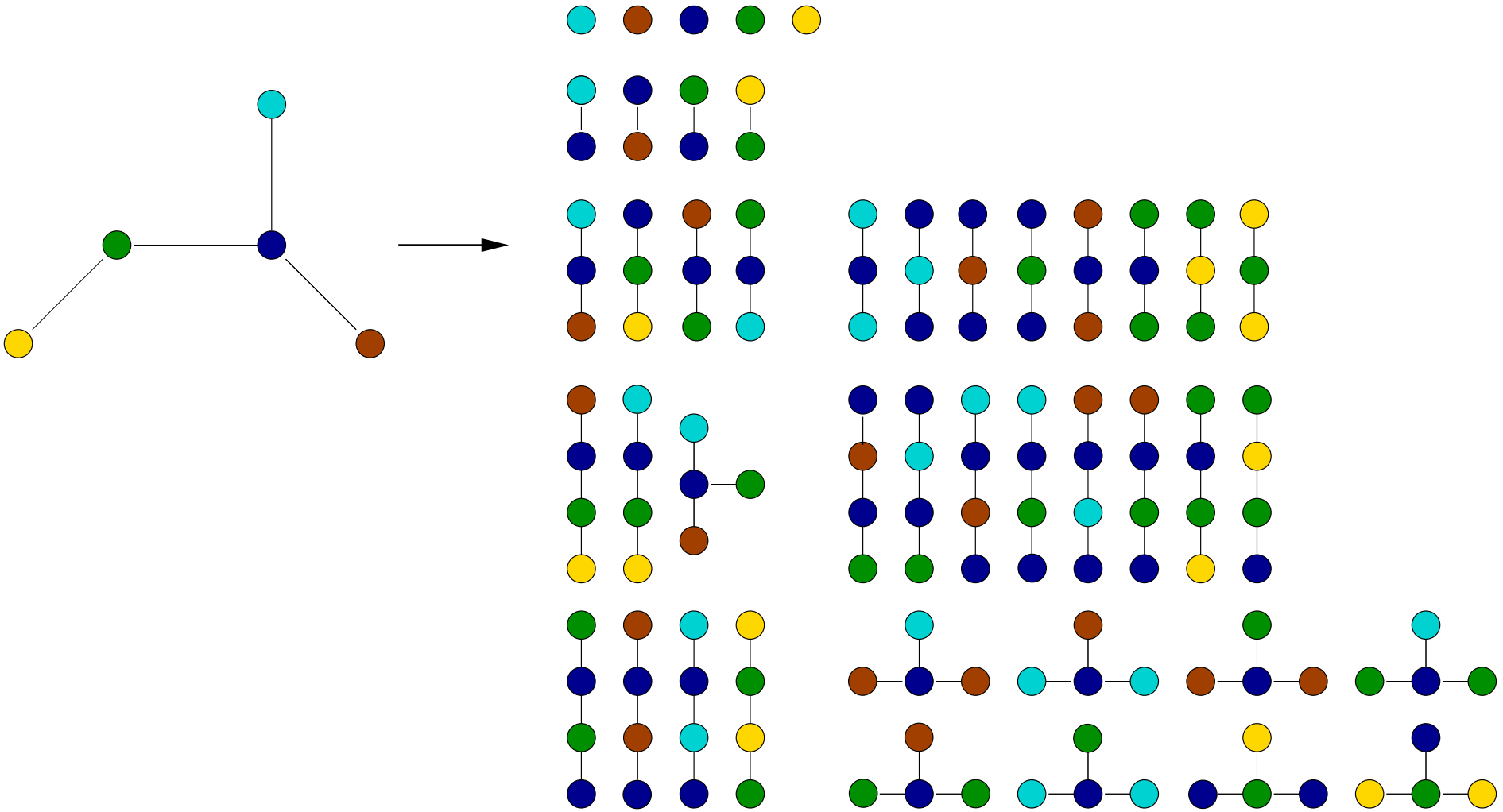
- Tottering walks seem irrelevant for many applications
- Focusing on non-tottering walks is a way to get closer to the path kernel (e.g., equivalent on trees).

Computation of the non-tottering walk kernel (Mahé et al., 2005)

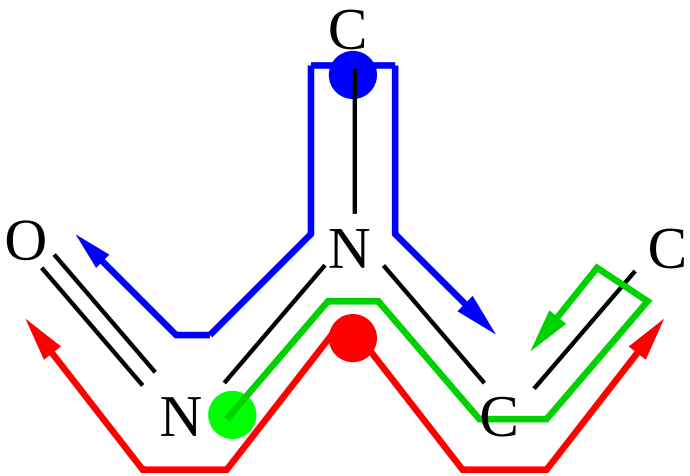
- Second-order Markov random walk to prevent tottering walks
- Written as a first-order Markov random walk on an augmented graph
- Normal walk kernel on the augmented graph (which is always a directed graph).

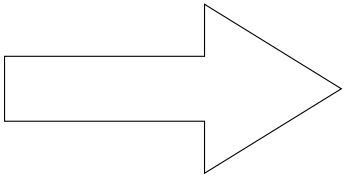


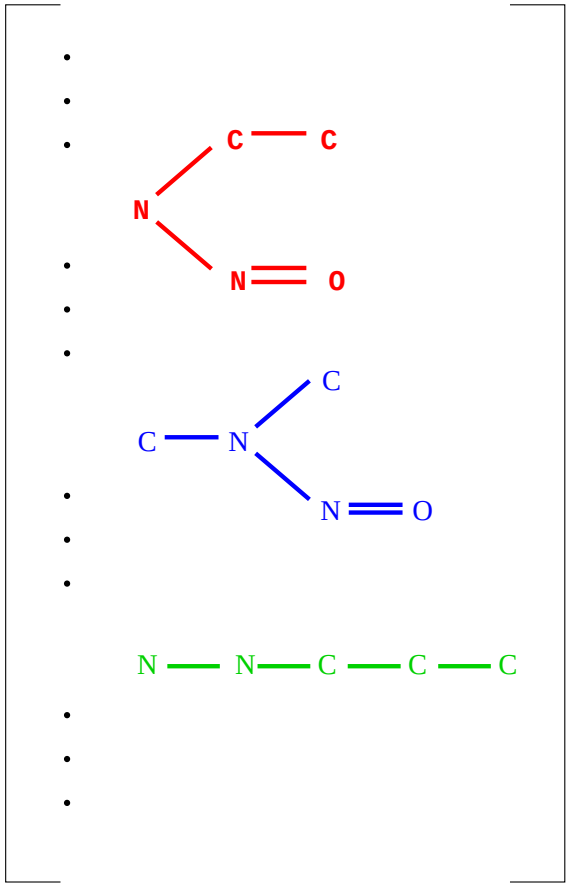
Extension 2: Subtree kernels



Example: Tree-like fragments of molecules







Computation of the subtree 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.
- Can be combined with the non-tottering graph transformation as preprocessing to obtain the non-tottering subtree kernel.

Outline

Chemoinformatics (Mahé et al., 2004)

MUTAG dataset

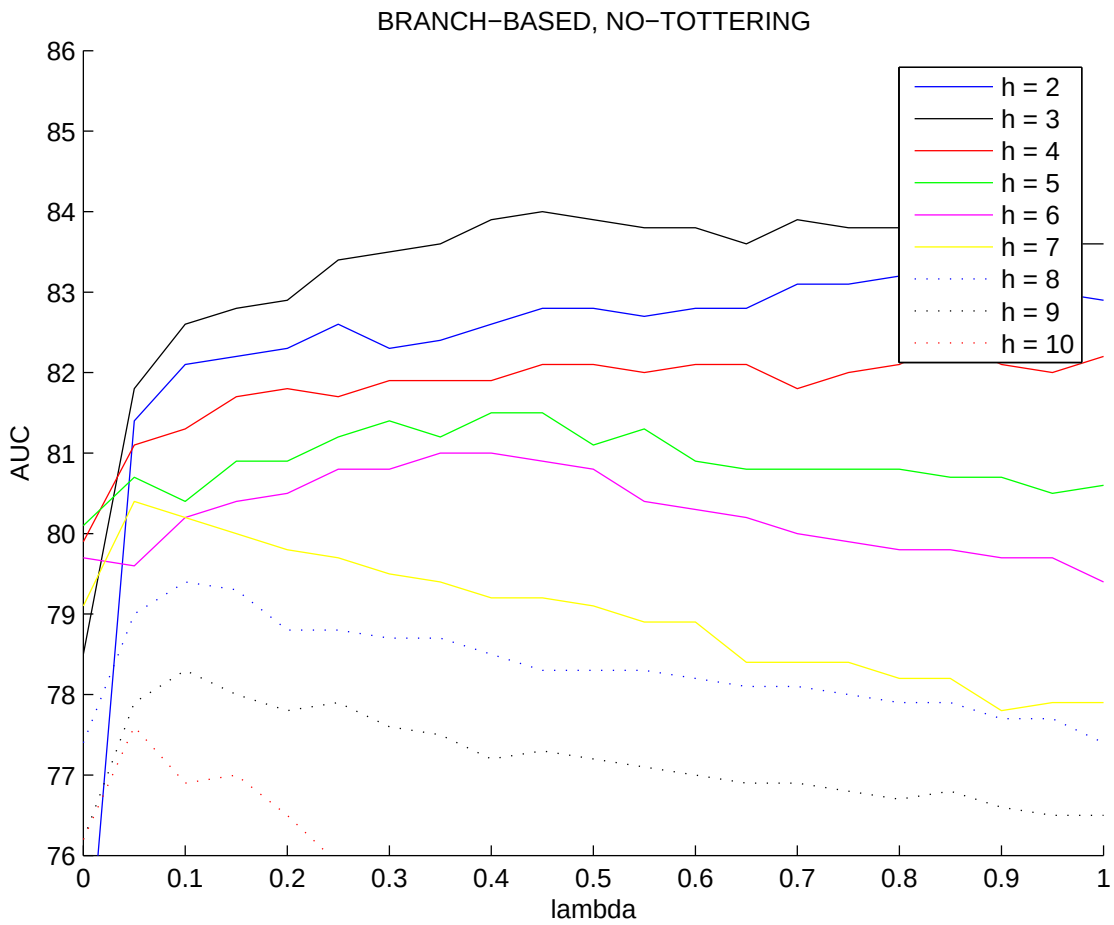
- aromatic/hetero-aromatic compounds
- high mutagenic activity /no mutagenic activity, assayed in *Salmonella typhimurium*.
- 188 compouunds: 125 + / 63 -

Results

10-fold cross-validation accuracy

Method	Accuracy
Progol1	81.4%
2D kernel	91.2%

Subtree kernels

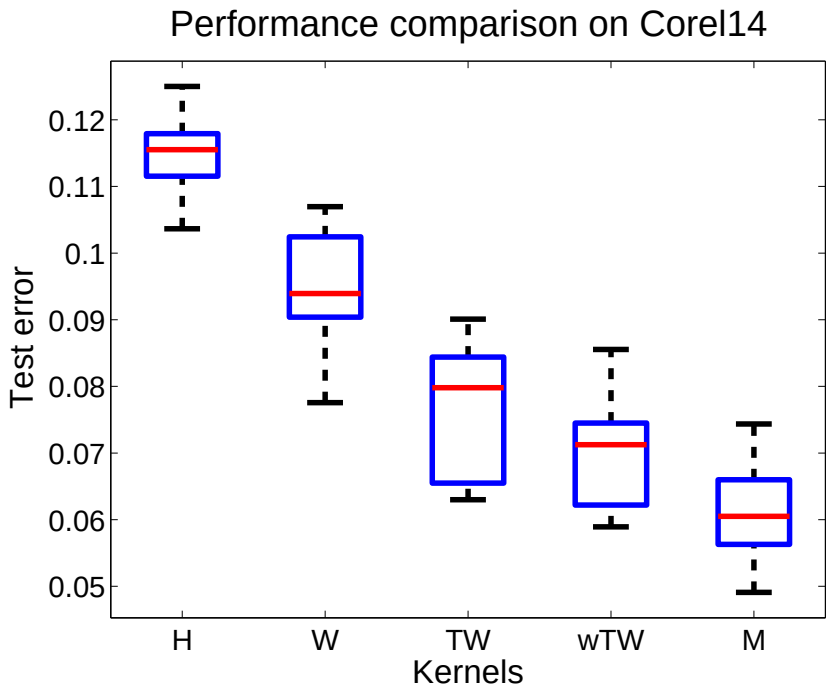


AUC as a function of the branching factors for different tree depths (from Mahé et al., 2007).

Image classification (Harchaoui and Bach, 2007)

COREL14 dataset

- 1400 natural images in 14 classes
- Compare kernel between histograms (H), walk kernel (W), subtree kernel (TW), weighted subtree kernel (wTW), and a combination (M).



Outline

Conclusion

What we saw

- Extension of machine learning algorithms to graph data through the definition of positive definite kernels for graphs
- The 2D kernel for molecule extends classical fingerprint-based approches. It solves the problem of bit clashes, allows infinite fingerprints and various extensions.
- Increasingly used in real-world applications.

Open question

- How to design / choose / learn a kernel for a given application in practice?
- How to improve scalability of kernel methods + graph kernels to large datasets?

References

- Kashima, H., Tsuda, K., and Inokuchi, A. *Marginalized kernels between labeled graphs*. Proceedings of the 20th ICML, 2003, pp. 321-328.
- T. Gärtner, P. Flach, and S. Wrobel. *On graph kernels: hardness results and efficient alternatives*. Proceedings of COLT, p.129–143, Springer, 2003.
- J. Ramon and T. Gärtner. *Expressivity versus Efficiency of Graph Kernels*. First International Workshop on Mining Graphs, Trees and Sequences, 2003.
- P. Mahé, N. Ueda, T. Akutsu, J.-L. Perret, and J.-P. Vert. *Graph kernels for molecular structure-activity relationship analysis with SVM*. J. Chem. Inf. Model., 45(4):939-951, 2005.
- P. Mahé and J.-P. Vert. *Graph kernels based on tree patterns for molecules*. Technical report HAL:ccsd-00095488, 2006.
- P. Mahé. *Kernel design for virtual screening of small molecules with support vector machines*. PhD thesis, Ecole des Mines de Paris, 2006.
- Z. Harchaoui and F. Bach. *Image classification with segmentation graph kernels.* IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR), 2007.
- Open-source kernels for chemoinformatics: `http://chemcpp.sourceforge.net/`