Mach Learn (2009) 75: 3–35
DOI 10.1007/s10994-008-5086-2

Graph kernels based on tree patterns for molecules

Pierre Mahé · Jean-Philippe Vert

Received: 9 March 2007 / Revised: 28 July 2008 / Accepted: 4 September 2008 /
Published online: 4 October 2008
The Author(s) 2008. This article is published with open access at Springerlink.com

Abstract Motivated by chemical applications, we revisit and extend a family of positive definite kernels for graphs based on the detection of common subtrees, initially proposed by Ramon and Gärtner (Proceedings of the first international workshop on mining graphs, trees and sequences, pp. 65–74, 2003). We propose new kernels with a parameter to control the complexity of the subtrees used as features to represent the graphs. This parameter allows to smoothly interpolate between classical graph kernels based on the count of common walks, on the one hand, and kernels that emphasize the detection of large common subtrees, on the other hand. We also propose two modular extensions to this formulation. The first extension increases the number of subtrees that define the feature space, and the second one removes noisy features from the graph representations. We validate experimentally these new kernels on problems of toxicity and anti-cancer activity prediction for small molecules with support vector machines.

Keywords Graph kernels · Support vector machines · Chemoinformatics

1 Introduction

There is an increasing need for algorithms to analyze and classify graph data, motivated in particular by various applications in chemoinformatics and bioinformatics. A prominent example in chemoinformatics, which motivates this work, is the generic problem

Guest Editors: Thomas Gärtner, Gemma C. Garriga.

P. Mahé · J.-P. Vert (✉)
Centre for Computational Biology, Ecole des Mines de Paris—ParisTech, 35 rue Saint Honoré, 77305 Fontainebleau, France
e-mail: Jean-Philippe.Vert@mines-paristech.fr

P. Mahé
e-mail: pierre.mahe@ensmp.fr

J.-P. Vert
Institut Curie, 75248 Paris, France

J.-P. Vert
INSERM, U900, 75248 Paris, France

of predicting various properties of small molecules, such as toxicity or anti-cancer activity, given their *molecular graph*, that is, the graph representing the covalent bonds between atoms (Leach and Gillet 2003). Classification of graphs is often associated with the problem of graph mining, which consists in detecting interesting patterns occurring in the graphs, and using them as features to build predictive models (King et al. 1996; Inokuchi et al. 2003; Helma et al. 2004; Deshpande et al. 2005). As an alternative to this approach, kernel methods associated with graph kernels have recently emerged as a promising approach for classification of graph data. Kernel methods such as support vector machines (SVM) operate implicitly in a possibly high-dimensional Hilbert space of features, in the sense that no explicit computation of the image of the input data in the feature space is required. Instead, only the inner product between the images of any two input data points, called the *kernel*, is required (Schölkopf and Smola 2002; Shawe-Taylor and Cristianini 2004). Applying kernel methods to graph data therefore requires the definition of a kernel between graphs, thereafter simply referred to as *graph kernel*. Choosing a graph kernel implicitly amounts to defining a set of features to represent the graphs and an inner product in the space of features.

Graph kernels were pioneered by Kashima et al. (2004) and Gärtner et al. (2003), who showed how to map graphs to an infinite-dimensional feature space indexed by linear subgraphs, and compute an inner product in that space. The resulting graph kernels compare two graphs through their common walks, weighted by a function of their lengths (Gärtner et al. 2003) or by their probability under a random walk model on the graphs (Kashima et al. 2004). While this representation might appear restrictive, these kernels led to promising empirical results, often comparing to state-of-the-art approaches in the fields of chemoinformatics (Mahé et al. 2005; Ralaivola et al. 2005) and bioinformatics (Borgwardt et al. 2005; Karklin et al. 2005).

Nevertheless, Ramon and Gärtner (2003) highlighted the limited expressiveness of graph kernels based on linear features, showing in particular that many different graphs can be mapped to the same point in the corresponding feature space. Figure 1 illustrates this issue on a simple example. On the other hand, they also showed that computing a complete graph kernel, that is, a kernel mapping non-isomorphic graphs to distinct points in the feature space, is at least as hard as the graph isomorphism problem. This suggests that the expressiveness of graph kernels must be traded for their computational complexity. Different approaches have been proposed to refine the features used in walk-based graph kernels. Horváth et al. (2004) introduced kernels based on the detection of common cyclic patterns, Menchetti et al. (2005) derived a kernel from the comparison of neighborhoods of atoms, and Ramon and Gärtner (2003) introduced a kernel comparing graphs on the basis of their common subtrees. This latter representation looks promising in particular in chemoinformatics, because physicochemical properties of atoms are known to be related to their topological environment that could be well captured by subtrees. However, the relationship between the new subtree-based kernel and previous walk-based kernels was not analyzed in details, and the relevance of the new kernel was not tested empirically.

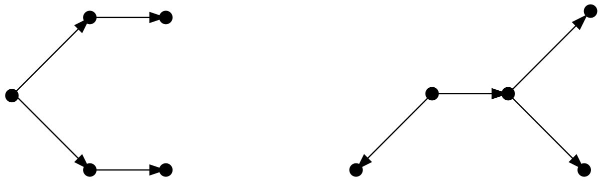


Fig. 1 Two graphs having the same walk content, namely $\bullet$: $\times 5$; $\bullet\!\to\!\bullet$: $\times 4$ and $\bullet\!\to\!\bullet\!\to\!\bullet$: $\times 2$, and consequently mapped to the same point of the feature space corresponding a kernel based on the count of walks (Gärtner et al. 2003)

Our motivation in this paper is to study in detail, both theoretically and empirically, the relevance of subtree features for graph kernels, and in particular to assess the benefits they bring compared to walk-based graph kernels. For that purpose we first revisit the formulation introduced by Ramon and Gärtner (2003) and propose two new kernels with an explicit description of their feature spaces and corresponding inner products. We introduce a parameter in the formulations that allows to gradually increase the complexity of the subtrees used as features to represent the graphs, the notion of complexity depending on the formulation. By decreasing the parameter we recover classical walk-based kernels, and by increasing it, we can empirically observe in detail the effect of increasing the number and the complexity of the tree features used to represent the graphs. Both formulations can be efficiently computed by dynamic programming, in the spirit of the kernel proposed by Ramon and Gärtner (2003). When the size of allowed subtrees is increased, however, we observe that the practical use of this kernel is limited by the explosion in the number of subtrees occurring in the graphs. In a second step, we therefore introduce two extensions to the initial formulation of the kernels that allow, on the one hand, to extend and generalize their associated feature space, and on the other hand, to remove noisy features that correspond to unwanted subtrees. The different kernels are compared experimentally on two small binary classification tasks consisting in discriminating toxic from non-toxic molecules with an SVM, as well as on one large-scale experiment to predict the anti-cancer properties of molecules on 60 cancer cell lines. In all cases we show that graph kernels based on tree patterns can significantly outperform graph kernels based on walks.

Although our main motivations are in chemical applications, we adopt the general framework of graph kernels in this paper, because the kernels introduced may find different applications in domains where data have a natural graph structure, such as bioinformatics, natural language processing or image processing. We assume that the reader is familiar with kernel functions and SVMs, and refer him to Schölkopf and Smola (2002), Shawe-Taylor and Cristianini (2004) and references therein for a background on the subject. The remaining of the paper is organized as follows. Notations and definitions related to graphs and trees are introduced in Sect. 2, followed in Sect. 3 by the definition of a general class of kernels based on the detection of common subtrees. The next section (Sect. 4) revisits the framework introduced in Ramon and Gärtner (2003), from which two particular graph kernels are derived and further extended in Sect. 5. The kernels are validated experimentally in Sect. 6, and we give concluding remarks in Sect. 7.

2 Notations and definitions

In this section we introduce notations and general definitions related to graphs and trees.

2.1 Labeled directed graphs

A *labeled graph* $G = (\mathcal{V}_G, \mathcal{E}_G)$ is defined by a finite set of *vertices* $\mathcal{V}_G$, a set of *edges* $\mathcal{E}_G \subset \mathcal{V}_G \times \mathcal{V}_G$, and a labeling function $l : \mathcal{V}_G \cup \mathcal{E}_G \to \mathcal{A}$ which assigns a *label* $l(x)$ taken from an alphabet $\mathcal{A}$ to any vertex or edge x. We let $|\mathcal{V}_G|$ be the number of vertices of G, $|\mathcal{E}_G|$ be its number of edges, and we assume below that a set of labels $\mathcal{A}$ common to all graphs has been fixed. In *directed* graphs, edges are oriented and to each vertex $u \in \mathcal{V}_G$ corresponds a set of *incoming neighbors* $\delta^-(u) = \{v \in \mathcal{V}_G : (v, u) \in \mathcal{E}_G\}$ and *outgoing neighbors* $\delta^+(u) = \{v \in \mathcal{V}_G : (u, v) \in \mathcal{E}_G\}$. We let $d^-(u) = |\delta^-(u)|$ be the *in-degree* of the vertex u, and $d^+(u) = |\delta^+(u)|$ be its *out-degree*. A *walk* of length n in the graph $G = (\mathcal{V}_G, \mathcal{E}_G)$ is a succession of

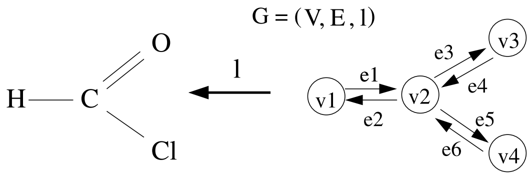


Fig. 2 A chemical compound seen as a labeled graph

$n+1$ vertices $(v_0,\dots,v_n)\in\mathcal{V}_G^{n+1}$, such that $(v_i,v_{i+1})\in\mathcal{E}_G$ for $i=0,\dots,n-1$. A *path*[1] is a walk $(v_0,\dots,v_n)$ with the additional condition that $i\neq j \iff v_i\neq v_j$. Finally, a graph is said to be *connected* if there is a walk between any pair of vertices when the orientation of edges is dropped.

For applications in chemistry considered below, we associate a labeled directed graph $G=(\mathcal{V}_G,\mathcal{E}_G)$ to the planar structure of a molecule. To do so, we let the set of vertices $\mathcal{V}_G$ correspond to the set of atoms of the molecule, the set of edges $\mathcal{E}_G$ to its covalent bonds, and label these graph elements according to an alphabet $\mathcal{A}$ consisting of the different types of atoms and bonds. Note that since graphs are directed, a pair of edges of opposite direction is introduced for each covalent bond of the molecule. Figure 2 shows a chemical compound seen as a labeled directed graph.

2.2 Trees

A *rooted tree* t is a directed connected acyclic graph in which all vertices have in-degree one, except one that has in-degree zero. The trees considered in the following will always be rooted, and we will simply use the term “tree” to denote a rooted tree. The node with in-degree zero is called the *root* $r(t)$ of the tree. Nodes with out-degree zero are called *leaf* nodes, others are called *internal* nodes. Trees are naturally oriented, edges being directed from the root to the leaves. The outgoing neighbors of an internal node are called its *children*, and the unique incoming neighbor of a node (apart from the root) is called its *parent*. If two nodes have the same parent, their are said to be *siblings*. The *size* $|t|$ of the tree t is its number of nodes: $|t|=|\mathcal{V}_t|$. The *depth* of a node corresponds to the number of edges connecting it to the root plus one,[2] and the depth of the tree is the maximum depth of its nodes. Finally, we introduce a couple of definitions that will be useful in the following.

Definition 1 (Balanced tree) A *perfectly depth-balanced tree* of order h is a tree where the depth of each leaf node is h. Perfectly depth-balanced trees are also called *balanced trees* below.

Definition 2 (Branching cardinality) We define the *branching cardinality* of the tree t, noted $\mathrm{branch}(t)$, as its number of leaf nodes minus one. More formally, for the tree $t=(\mathcal{V}_t,\mathcal{E}_t)$ with $\mathcal{V}_t=(v_1,\dots,v_{|t|})$, $\mathrm{branch}(t)$ is given by

$$\mathrm{branch}(t)=\sum_{i=1}^{|t|}\mathbf{1}(d^+(v_i)=0)-1,$$

where $\mathbf{1}(.)$ is a binary function equal to one if its argument is true, and zero otherwise.

[1] What we call a path is sometimes called a *simple path*.

[2] Note that the depth of the root node is one.

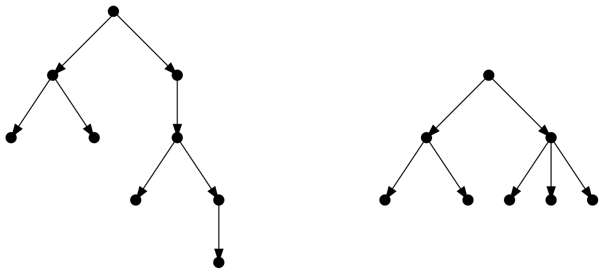


Fig. 3 *Left*: a tree t_1 of depth 5 with $|t_1| = 9$ and $\text{branch}(t_1) = 3$. *Right*: a balanced tree t_2 of order 3 with $|t_2| = 8$ and $\text{branch}(t_2) = 4$. *Top* nodes are root nodes, *bottom* nodes are leaf nodes

This terminology stems from the observation that this quantity also corresponds to the sum, over the non-leaf nodes of the tree, of how many extra children they have compared to linear graph with only one child per non-leaf node. It therefore measures how many extra branchings there are compared to a linear tree, which has branching cardinality 0. These definitions are illustrated in Fig. 3.

The remaining of the paper introduces kernel functions between labeled directed graphs based on the detection in the graphs of patterns corresponding to labeled trees. To lighten notations, we simply refer below to labeled directed graphs and labeled trees as graphs and trees.

3 The tree-pattern graph kernel

This section introduces a general class of graph kernel based on the detection, in the graphs, of patterns corresponding to particular tree structures. We start by defining precisely this notion of tree-pattern.

Definition 3 (Tree-pattern) Let a graph $G = (\mathcal{V}_G, \mathcal{E}_G)$ and a tree $t = (\mathcal{V}_t, \mathcal{E}_t)$, with $\mathcal{V}_t = (n_1, \ldots, n_{|t|})$. A $|t|$-tuple of vertices $(v_1, \ldots, v_{|t|}) \in \mathcal{V}_G^{|t|}$ is a *tree-pattern* of G with respect to t, which we denote by $(v_1, \ldots, v_{|t|}) = \text{pattern}(t)$, if and only if the following holds:

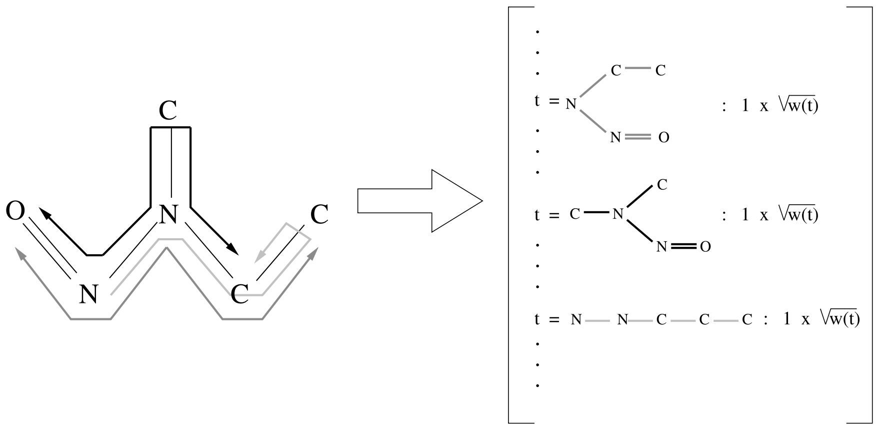
$$\begin{cases} \forall i \in [1, |t|], \quad l(v_i) = l(n_i), \\ \forall (n_i, n_j) \in \mathcal{E}_t, \quad (v_i, v_j) \in \mathcal{E}_G \wedge l\big((v_i, v_j)\big) = l\big((n_i, n_j)\big), \\ \forall (n_i, n_j), (n_i, n_k) \in \mathcal{E}_t, \quad j \neq k \quad \Longleftrightarrow \quad v_j \neq v_k. \end{cases}$$

In other words a tree-pattern is a combination of graph vertices that can be arranged in a particular tree structure, according to the labels and the connectivity properties of the graph. Note from this definition that vertices of the graph are allowed to appear several times in a tree-pattern, under the condition that siblings nodes of the corresponding tree are associated to distinct vertices of the graphs. We now introduce a functional to count occurrences of these patterns.

Definition 4 (Tree-pattern counting function) A *tree-pattern counting function* returning the number of times a tree-pattern occurs in a graph is defined for the tree t and the graph $G = (\mathcal{V}_G, \mathcal{E}_G)$, $\mathcal{V}_G = (v_1, \ldots, v_{|\mathcal{V}_G|})$, as

$$\psi_t(G) = \Big|\big\{(\alpha_1, \ldots, \alpha_{|t|}) \in [1, |\mathcal{V}_G|]^{|t|} : (v_{\alpha_1}, \ldots, v_{\alpha_{|t|}}) = \text{pattern}(t)\big\}\Big|.$$

A restriction of ψ_t to patterns rooted in a specified vertex v is given by



C
O
N
C
N
C
t = N — C — C, N = O : 1 x √w(t)
t = C — N — C, N = O : 1 x √w(t)
t = N — N — C — C — C : 1 x √w(t)

Fig. 4 A molecular compound G (*left*) and its feature space representation $\phi(G)$ (*right*). Note that the *top* and *bottom* trees are balanced. Note moreover that the *bottom* tree consists of a set of linearly connected atoms, which is known as *molecular fragment* in chemoinformatics. Note finally that the same C atom appears in the 3rd and 5th positions in the tree-pattern corresponding to the *bottom* tree

$$\psi_t^{(v)}(G) = \Big|\Big\{(\alpha_1, \ldots, \alpha_{|t|}) \in [1, |\mathcal{V}_G|]^{|t|} : \\ (v_{\alpha_1}, \ldots, v_{\alpha_{|t|}}) = \text{pattern}(t) \wedge v_{\alpha_1} = v\Big\}\Big|.$$

With this new definition at hand we can define a general graph kernel based on the detection of common tree-patterns in the graphs.

Definition 5 (Tree-pattern graph kernel) *The tree-pattern graph kernel K is given for the* graphs G_1 and G_2 by

$$K(G_1, G_2) = \sum_{t\in\mathcal{T}} w(t)\psi_t(G_1)\psi_t(G_2), \tag{1}$$

where $\mathcal{T}$ is a set of trees, $w : \mathcal{T} \to \mathbb{R}_+$ is a non-negative tree weighting functional and ψ_t is the tree-pattern counting function of Definition 4.

The kernel of Definition 5 is obviously positive definite since it can be written as a standard dot-product $K(G_1, G_2) = \langle\phi(G_1), \phi(G_2)\rangle$, where $\phi(G)$ is the mapping that maps any graph G to the feature space indexed by the trees of the set $\mathcal{T}$ as $\phi(G) = (\sqrt{w(t)}\psi_t(G))_{t\in\mathcal{T}}$. It is only defined for graphs in which the possibly infinite sum in (1) converges. Figure 4 illustrates this mapping.

4 Examples of tree-pattern graph kernels

In a recent work, Ramon and Gärtner (2003) proposed a particular tree-pattern graph kernel fitting the general Definition 5. In this section, we propose two different kernels with explicit feature spaces and inner products, discuss their practical computation, and highlight their differences with the kernel of Ramon and Gärtner (2003).

4.1 Kernel definition

According to Definition 5, two key elements enter in the definition of a tree-pattern graph kernel. Firstly, the set of trees $\mathcal{T}$ indexing the feature space where the graphs are mapped must be chosen. The kernels we consider in this section are based on the same feature space: the space indexed by the set of balanced trees of order h introduced in Definition 1, labeled according to the graphs labeling alphabet $\mathcal{A}$. We will refer to this set as $\mathcal{B}_h$ in the following. While the choice of balanced trees only may appear restrictive, it is convenient to derive efficient algorithm and is generalized to larger sets of trees in Sect. 5.1. Second, the tree weighting function w must be defined. A natural way to define such a functional is to take into account the structure of the trees, and accordingly, we propose to relate the weight of a tree to its size or its branching cardinality. In particular we propose to consider the following kernels:

Definition 6 (Size-based balanced tree-pattern kernel) For the pair of graphs G_1 and G_2, the *size-based balanced tree-pattern kernel of order h* is defined as

$$K^h_{\text{Size}}(G_1, G_2) = \sum_{t \in \mathcal{B}_h} \lambda^{|t|-h} \psi_t(G_1)\psi_t(G_2). \tag{2}$$

Definition 7 (Branching-based balanced tree-pattern kernel) For the pair of graphs G_1 and G_2, the *branching-based balanced tree-pattern kernel of order h* is defined as

$$K^h_{\text{Branch}}(G_1, G_2) = \sum_{t \in \mathcal{B}_h} \lambda^{\text{branch}(t)} \psi_t(G_1)\psi_t(G_2). \tag{3}$$

Note that the depth of a tree is a lower bound on its size, attained for a tree consisting of a linear chain of vertices. For such a tree, at depth h, we have $|t| - h = \text{branch}(t) = 0$, and we see that the corresponding tree-patterns are given a unit weight in the kernels of Definitions 6 and 7. The complexity of a tree naturally increases with its size and branching cardinality, and the λ parameter entering the kernel Definitions 6 and 7 has the effect of favoring tree-patterns depending on their degree of complexity. A value of λ greater than one favors the influence of tree-patterns of increasing complexity over the trivial linear tree-patterns, while they are penalized by a value of λ smaller than one. We can note, however, that while the size of a tree increases with its branching cardinality, the converse is not true. For any tree t of depth h, we therefore always have $|t| - h \geq \text{branch}(t)$, and the tree weighting is more pronounced in the size-based than in the branching-based kernel. In the case of balanced trees, this difference is particularly marked when the nodes with large out-degree are close to the root node. This is due to the fact that every leaf must be at depth h, and while the size of the tree necessarily increases by at least $h - 1$ along each path starting from the root, the branching cardinality does not.[3] The main difference in the feature space representations of the graphs is therefore induced by this particular type of tree-patterns, that can be interpreted as collections of regular subtree patterns merged in the root node. This suggests for instance that, for $\lambda < 1$, the branching-based formulation of the kernel may to some extent tolerate large, yet regular patterns, that would be strongly penalized in the size-based formulation. Figure 5 illustrates these tree weightings based on the size and branching cardinality.

[3]At the extreme, we have $|t| = 1 + (h-1) \times d^+(r(t))$ vs. $\text{branch}(t) = d^+(r(t)) - 1$.

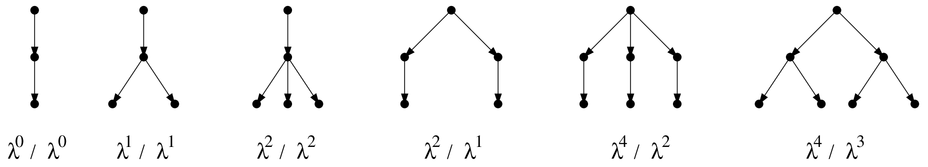


Fig. 5 A set of balanced trees of order 3, together with their size-based (*left*) and branching-based (*right*) λ weighting

When λ tends to zero, the complexity of the patterns is so penalized that only tree-patterns consisting of linear chains of graph vertices have non-vanishing weights, and the kernels of Definitions 6 and 7 boil down to a kernel based on the detection of common walks (Gärtner et al. 2003). More formally, if we define the set of walks of length n of the graph G as

$$\mathcal{W}_n(G) = \{(v_0, \dots, v_n) \in \mathcal{V}_G^{n+1} : (v_i, v_{i+1}) \in \mathcal{E}_G, 0 \le i \le n-1\},$$

and define for the graphs G_1 and G_2 the following walk-count kernel:

$$K^n_{\text{Walk}}(G_1, G_2) = \sum_{\substack{w_1 \in \\ \mathcal{W}_n(G_1)}} \sum_{\substack{w_2 \in \\ \mathcal{W}_n(G_2)}} \mathbf{1}(l(w_1) = l(w_2)), \tag{4}$$

where $\mathbf{1}(l(w_1) = l(w_2))$ is one if all pairs of corresponding edges and vertices are identically labeled in the walks w_1 and w_2, and zero otherwise, one easily gets that:

$$\lim_{\lambda \to 0} K^h_{\text{Size}}(G_1, G_2) = \lim_{\lambda \to 0} K^h_{\text{Branch}}(G_1, G_2) = K^{h-1}_{\text{Walk}}(G_1, G_2).$$

Increasing the value of λ relaxes the penalization on complex subtree features, and can therefore be interpreted as introducing tree-patterns of increasing complexity in the walk-based kernel of (4).

It should be noted finally that the parameters h and λ are directly related to the nature of the features representing the graphs and to their relative importance. Optimal values of the parameters are therefore likely to be dependent on the problem and data considered, and can hardly be chosen a priori. As an example, because of the variety of chemical compounds, the graphs considered in a chemical application can have a great structural diversity. This suggests that these parameters should be estimated from the data using, for example, cross-validation techniques.

4.2 Kernel computation

We now propose two factorization schemes to compute the kernels of Definitions 6 and 7. These factorizations are inspired by the dynamic programming (DP) algorithm proposed by Ramon and Gärtner (2003) to compute a slightly different graph kernel, discussed in the next subsection. The factorization relies on the following definition:

Definition 8 (Neighborhood matching set) The *neighborhood matching set* $\mathcal{M}(u, v)$ of two graph vertices u and v is defined as

$$\begin{aligned}\mathcal{M}(u, v) = \big\{R \subseteq \delta^+(u) \times \delta^+(v) \mid R \neq \emptyset \\ \wedge \big(\forall (a, b), (c, d) \in R : a = c \Leftrightarrow b = d\big) \\ \wedge \big(\forall (a, b) \in R : l(a) = l(b) \wedge l((u, a)) = l((v, b))\big)\big\}.\end{aligned}$$

Each $R \in \mathcal{M}(u,v)$ consists of one or several pair(s) of neighbors of u and v that are identically labeled and connected to u and v by edges of the same label. It follows from Definition 1 that such an element R corresponds to a pair of balanced tree-patterns of order 2 rooted in u and v, found in the graph(s) u and v belong to. Moreover, provided u and v have the same label, these patterns correspond to the same balanced tree. We can state the following propositions, whose proofs are postponed in Appendix A:

Proposition 1 (Size-based kernel computation) *The order h size-based tree-pattern kernel K^h_{Size} of Definition 6 between two graphs G_1 and G_2 can be computed as*:

$$K^h_{\text{Size}}(G_1, G_2) = \frac{1}{\lambda^h} \sum_{u \in \mathcal{V}_{G_1}} \sum_{v \in \mathcal{V}_{G_2}} k_h(u,v), \tag{5}$$

where $k_n, n = 1, \ldots, h$ is defined recursively by

$$\begin{cases} k_1(u,v) = \lambda \mathbf{1}(l(u) = l(v)), \\ k_n(u,v) = \lambda \mathbf{1}(l(u) = l(v)) \displaystyle\sum_{R \in \mathcal{M}(u,v)} \prod_{(u',v') \in R} k_{n-1}(u',v'), \quad n = 2, \ldots, h. \end{cases}$$

Proposition 2 (Branching-based kernel computation) *The order h branching-based tree-pattern kernel K^h_{Branch} of Definition 7 between two graphs G_1 and G_2 can be computed as*:

$$K^h_{\text{Branch}}(G_1, G_2) = \sum_{u \in \mathcal{V}_{G_1}} \sum_{v \in \mathcal{V}_{G_2}} k_h(u,v), \tag{6}$$

where $k_n, n = 1, \ldots, h$ is defined recursively by

$$\begin{cases} k_1(u,v) = \mathbf{1}(l(u) = l(v)), \\ k_n(u,v) = \mathbf{1}(l(u) = l(v)) \displaystyle\sum_{R \in \mathcal{M}(u,v)} \frac{1}{\lambda} \prod_{(u',v') \in R} \lambda k_{n-1}(u',v'), \quad n = 2, \ldots, h. \end{cases}$$

Not surprisingly, Propositions 1 and 2 show that the kernels K^h_{Size} and K^h_{Branch} of Definitions 6 and 7 have the same complexity. More precisely, for the pair of graphs G_1 and G_2, it follows from (5) and (6) that in both cases we need to evaluate $k_n(u,v)$ for all $u \in G_1$, $v \in G_2$ and $n = 1, \ldots, h$. The computation of a single $k_n(u,v)$ involves a sum over subsets R of matching pairs. The number of sets R of r matching pairs is upper bounded by $P_r^{d_+(u)} P_r^{d_+(v)}$, where $P_r^n = n!/(n-r)!$ counts the number of permutations of r elements of a sequence of n elements. Moreover each such subset requires a multiplication of r terms and an addition. If we denote by d an upper bound of the outer degrees of the vertices, the number of basic operations to compute $k_n(u,v)$ is therefore upper bounded by:

$$\sum_{r=1}^{d} (r+1)(P_r^d)^2 = \mathcal{O}(d^{2d}).$$

For two graphs G_1 and G_2, the computational complexity of the kernels is therefore upper bounded by

$$\mathcal{O}(|\mathcal{V}_{G_1}| \times |\mathcal{V}_{G_2}| \times h \times d^{2d}), \tag{7}$$

which shows a linear dependency with respect to the order h of the kernel. However, the super-exponential dependency on d suggests that these kernels are only useful in practice for graphs with very small connectivity. For example, in the case of chemical compounds, we have $d = 4$. The factor d^{2d} equals 65,536, and the complexity looks prohibitive. However this is only a worst-case complexity which is strongly reduced in practice because (i) the out-degree of the vertices is often smaller than four,[4] and (ii) the size of $\mathcal{M}(u, v)$ is reduced by the fact that vertices and edges can have distinct labels.

4.3 Relation to previous work

At this point, it is worth reminding the kernel formulation introduced by Ramon and Gärtner (2003) in order to highlight the differences with the kernels proposed in Definitions 6 and 7. In the context of graphs with labeled vertices and edges,[5] at order h, the kernel introduced in Ramon and Gärtner (2003), that we denote by K^h_{Ramon}, is formulated as follows:

$$K^h_{\text{Ramon}}(G_1, G_2) = \sum_{u\in\mathcal{V}_{G_1}} \sum_{v\in\mathcal{V}_{G_2}} k_h(u, v),$$

where k_n is defined by

$$\begin{cases} k_1(u, v) = \mathbf{1}(l(u) = l(v)) \\ k_n(u, v) = \mathbf{1}(l(u) = l(v))\lambda_u\lambda_v \displaystyle\sum_{R\in\mathcal{M}(u,v)} \prod_{(u',v')\in R} k_{n-1}(u', v'), \quad n = 2, \dots, h. \end{cases}$$

The first important difference lies in the fact that in their original definition, the neighborhood matching set $\mathcal{M}(u, v)$ includes the empty set as a special case. While this suggests that unbalanced trees are as well taken into account in their kernel, it is not clear how the product $\prod_{(u',v')\in R} k_n(u', v')$ is defined when $R = \emptyset$. Under the convention that this product is 0 if $R = \emptyset$, it turns out that only balanced-trees are taken into account in their kernel, as it is the case in our formulation. On the other hand, adopting the convention that this product is 1 if $R = \emptyset$, their formulation indeed makes it possible to take into account general trees. In Sect. 5.1, we propose an extension to our formulation that enables to consider general trees as well.

If we adopt our definition of the neighborhood matching set, given in Definition 8, it is clear that K^h_{Ramon} and the kernels of Definitions 6 and 7 have the same feature space. The second main difference with our formulations lies in the fact that here, a parameter λ_v is introduced for each vertex v of each graph. It can be checked that under this parametrization, each tree-pattern is weighted by the product of the parameters λ_v associated to its internal nodes. In the special case where these parameters are taken equal to a single parameter λ, each pattern is therefore weighted by λ raised to the power of its number of internal nodes. While this bears some similarity with the size-based weighting proposed in the kernel of Definition 6, we note for instance that the three leftmost trees of Fig. 5 are identically weighted, namely by a factor λ^2. As a result, the convergence to the walk-based kernel of (4) observed when λ tends to zero for the kernels of Definitions 6 and 7 does not hold with this formulation.

[4]For example, in the first dataset considered in our experiments in Sect. 6, the average out-degree of the vertices is 2.14.

[5]The original formulation considered graphs with labeled vertices only, and the definition of the neighborhood matching set is refined in this paper in order to handle labeled edges.

5 Extensions

The kernels introduced in the previous section arise directly from the adaptation of the algorithm proposed in Ramon and Gärtner (2003). In this section we introduce two extensions to this initial formulation. First, we extend the branching-based kernel of Definition 7 to a feature space indexed by a larger, and more general, set of trees. Second, we propose to eliminate a set of noisy tree-patterns from the feature space.

5.1 Considering all trees

The DP algorithms of Sect. 4.2 recursively extend the tree-patterns under construction until they reach a specified depth. Because they are based on the notion of neighborhood matching sets introduced in Definition 8, these algorithms add at least one child to every leaf node of the patterns under extension at each step of the recursive process. When they reach the specified depth, the patterns are therefore balanced, and the choice of the feature space associated to the kernels of Definitions 6 and 7 was actually dictated by their computation.

Rather than focusing on features of a particular size, standard representations of molecules involve structural features of different sizes. A prominent example is that of molecular fingerprints (Ralaivola et al. 2005) that typically represent a molecule by its exhaustive list of fragments of length up to 8, where a fragment is defined as a linear succession of connected atoms (see Fig. 4). In this section, we note that a slight modification of the DP algorithm of Proposition 2 generalizes the kernel of Definition 7 to a feature space indexed by the set of general trees up to a given depth, instead of the set of balanced-trees of the corresponding order. More precisely, if we let $\mathcal{T}_h$ be the set of trees of depth up to h, and if we define the *until-N extension* of the branching-based kernel of Definition 7 as

$$K_{\text{Branch}}^{\text{until-}h}(G_1, G_2) = \sum_{t \in \mathcal{T}_h} \lambda^{\text{branch}(t)} \psi_t(G_1)\psi_t(G_2), \tag{8}$$

we can state the following proposition, whose proof is postponed to Appendix B.

Proposition 3 (Until-N kernel computation) *The* until-N extension $K_{\text{Branch}}^{\text{until-}h}$ *of the branching-based kernel of order h of Definition 7 is given for the graphs G_1 and G_2 by*

$$K_{\text{Branch}}^{\text{until-}h}(G_1, G_2) = \sum_{u \in \mathcal{V}_{G_1}} \sum_{v \in \mathcal{V}_{G_2}} k_h(u, v),$$

where $k_n, n = 1, \ldots, h$ is defined recursively by

$$\begin{cases} k_1(u, v) = \mathbf{1}(l(u) = l(v)), \\ k_n(u, v) = \mathbf{1}(l(u) = l(v))\left(1 + \displaystyle\sum_{R \in \mathcal{M}(u,v)} \frac{1}{\lambda} \prod_{(u',v') \in R} \lambda k_{n-1}(u', v')\right), \\ \quad n = 2, \ldots, h. \end{cases}$$

The computation given in Proposition 3 follows that of Proposition 2, and this until-N extension comes at no extra cost. The feature space corresponding to this extended kernel has nevertheless a much larger dimensionality than that of the original branching-based kernel. Actually, because the set of trees $\mathcal{T}_h$ includes the set of balanced trees $\mathcal{B}_h$ as a special case, the feature space associated to the branching-based kernel is a sub-space of the feature

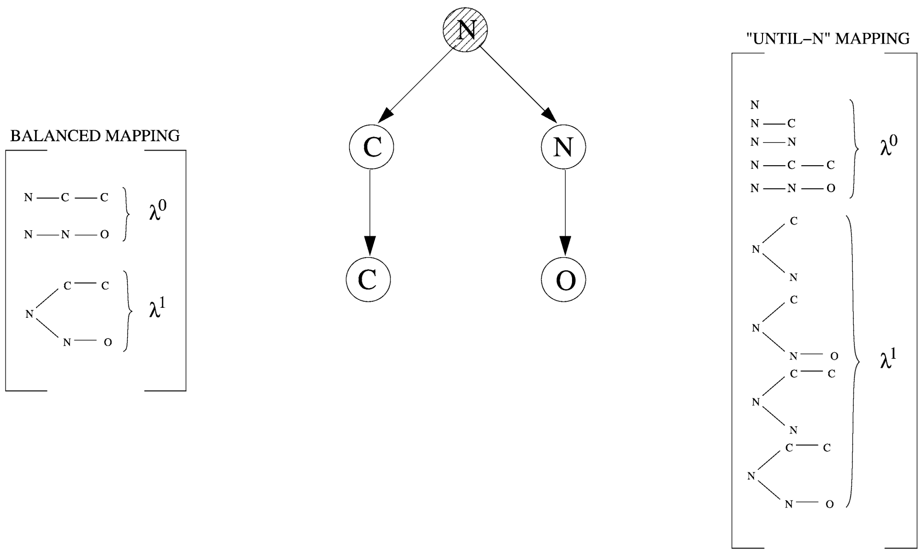


Fig. 6 A graph G (*middle*), the set of balanced trees of order 3 (*left*) and general trees of depth up to 3 (*right*) for which a tree-pattern rooted in the dashed vertex is found in G, together with their kernel weighting $\lambda^{\text{branch}(t)}$

space associated to its until-N extension. Figure 6 illustrates the different mappings. The behavior of this kernel with respect to λ follows that of the original branching-based kernel. In particular, when λ tends to zero, the set of tree-patterns with non-vanishing weights reduces to linear chain of vertices and the kernel boils down to a kernel based on the detection of common walks of length up to $h-1$. More formally, one can easily check that, in this case:

$$\lim_{\lambda\to 0} K_{\text{Branch}}^{\text{until-}h}(G_1, G_2) = \sum_{n=0}^{h-1} K_{\text{Walk}}^{n}(G_1, G_2),$$

where K_{Walk}^{n} is the kernel based on the detection of common walks of length n, defined in Sect. 4.1 (4).

Finally, we note that this extension is not directly applicable to the size-based kernel of Definition 6 because of a slight difference in the computations of Propositions 1 and 2. Indeed, note from Proposition 1 that in order to get the $\lambda^{|t|-h}$ weighting of the tree t proposed in Definition 6, the size-based kernel is initially computed from patterns weighted by their sizes, and is subsequently normalized by a factor λ^{-h}. As a result, while the above extension would still have the effect of extending the feature space to the space indexed by trees of $\mathcal{T}_h$, this λ^{-h} normalization would affect every tree-pattern regardless of their size, and the pattern weighting proposed in Definition 6 would be lost.

5.2 Removing tottering tree-patterns

The DP algorithms of Sects. 4.2 and 5.1 enumerate balanced tree-patterns of order h through the recursive extension of balanced tree-patterns of order 2 defined by neighborhood matching sets of pairs of vertices. According to Definition 8, the whole sets of neighbors of a pair of vertices enter in the definition of their neighborhood matching sets. As a result, it can be the case in a tree-pattern that a vertex appears simultaneously as the parent and a child of

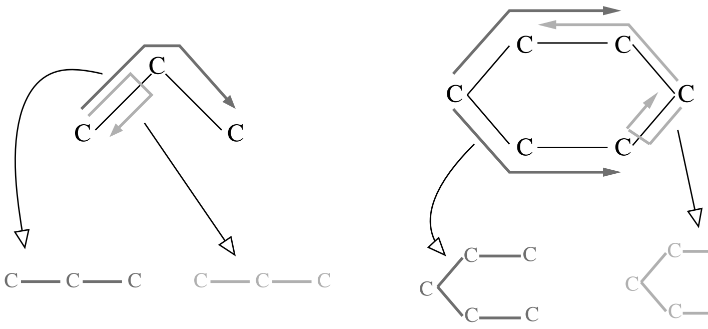


Fig. 7 *Left*: tottering (*light*) and no-tottering (*dark*) walks. *Right*: tottering (*light*) and no-tottering (*dark*) tree-patterns

a second vertex. This phenomenon is the tree counterpart of a phenomenon observed in the context of walk-based graph kernels, where a random walk under extension could return to a visited vertex just after leaving it. This behavior was called *tottering* in Mahé et al. (2005), and following this terminology, we refer to a tree-pattern in which a vertex appears simultaneously as the parent and a child of a second vertex as a *tottering tree-pattern*. Figure 7 illustrates the tottering phenomenon.

In many cases these tree-patterns are likely to be uninformative features. In particular they are not proper subgraphs of the initial graphs. Even worse, 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, suggesting that informative patterns corresponding to deep trees might be hidden by the profusion of tottering tree-patterns. In order to tackle this issue we now adapt an idea of Mahé et al. (2005) to filter out these spurious tottering tree-patterns in the kernels presented in Sects. 3 and 4. Tottering can be prevented by adding constraints in the tree-pattern counting function, according to the following definition.

Definition 9 (No-tottering tree-pattern counting function) From the tree-pattern counting function of Definition 4, a *no-tottering tree-pattern counting function* can be defined for the tree $t = (\mathcal{V}_t, \mathcal{E}_t)$, with $\mathcal{V}_t = (n_1, \ldots, n_{|t|})$, and the graph $G = (\mathcal{V}_G, \mathcal{E}_G)$, with $\mathcal{V}_G = (v_1, \ldots, v_{|\mathcal{V}_G|})$, as

$$\psi_t^{\mathrm{NT}}(G) = \Big|\Big\{(\alpha_1, \ldots, \alpha_{|t|}) \in [1, |\mathcal{V}_G|]^{|t|} : (v_{\alpha_1}, \ldots, v_{\alpha_{|t|}}) = \mathrm{pattern}(t) \\ \wedge (n_i, n_j), (n_j, n_k) \in \mathcal{E}_t \Rightarrow \alpha_i \neq \alpha_k\Big\}\Big|.$$

Following Definition 5, a graph kernel based on no-tottering tree-patterns can be defined from this no-tottering tree-pattern counting function.

Definition 10 (No-tottering tree-pattern kernel) A graph kernel K^{NT} based on no-tottering tree-patterns is given for the graphs G_1 and G_2 by

$$K^{\mathrm{NT}}(G_1, G_2) = \sum_{t \in \mathcal{T}} w(t) \psi_t^{\mathrm{NT}}(G_1) \psi_t^{\mathrm{NT}}(G_2), \tag{9}$$

where $\mathcal{T}$ is a set of trees, $w : \mathcal{T} \to \mathbb{R}$ is a tree weighting functional and ψ_t^{NT} is the no-tottering tree-pattern counting function of Definition 9.

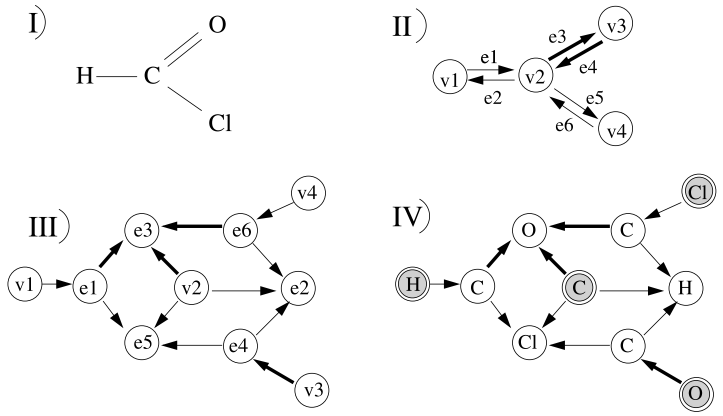


Fig. 8 The graph transformation. (**I**) The original molecule. (**II**) The corresponding graph $G = (\mathcal{V}_G, \mathcal{E}_G)$. (**III**) The transformed graph. (**IV**) The labels on the transformed graph. Note that different widths stand for different edges labels, and *gray* nodes are the nodes belonging to $\mathcal{V}_G$

This latter definition therefore extends the tree-pattern kernel of Definition 5 to the no-tottering case. However, due to the additional constraints on the set of acceptable patterns, the DP framework based on neighborhood matching set described in Sects. 4.2 and 5.1 does not hold any longer. In Mahé et al. (2005), the following graph transformation was introduced in order to filter tottering walks.

Definition 11 (Graph transformation) For a graph $G = (\mathcal{V}_G, \mathcal{E}_G)$, we let its *transformed graph* $G' = (\mathcal{V}_{G'}, \mathcal{E}_{G'})$ be defined by:

- $\mathcal{V}_{G'} = \mathcal{V}_G \cup \mathcal{E}_G$,
- $\mathcal{E}_{G'} = \mathcal{E}^1 \cup \mathcal{E}^2$ with:

$$\begin{cases} \mathcal{E}^1 = \{(v, (v,t)) \,|\, v \in \mathcal{V}_G, (v,t) \in \mathcal{E}_G\}, \\ \mathcal{E}^2 = \{((u,v), (v,t)) \,|\, (u,v), (v,t) \in \mathcal{E}_G, u \neq t\}, \end{cases}$$

and labeled as follows:

- for a node $v' \in \mathcal{V}_{G'}$ the label is either $l(v') = l(v')$ if $v' \in \mathcal{V}_G$, or $l(v') = l(v)$ if $v' = (u,v) \in \mathcal{E}_G$,
- for an edge $e' = (v'_1, v'_2)$ between two vertices $v'_1 \in \mathcal{V}_G \cup \mathcal{E}_G$ and $v'_2 \in \mathcal{E}_G$, the label is simply given by $l(e') = l(v'_2)$.

This graph transformation is illustrated in Fig. 8 for the graph corresponding to the chemical compound of Fig. 2. Based on this graph transformation, Mahé et al. (2005) proved that there is a bijection between the set of no-tottering walks of a graph and the set of walks of its transformed graph that start on a vertex corresponding to a vertex of the original graph. In a similar way, we show below that there is a bijection between the set of no-tottering tree-patterns found in a graph and the set of tree-patterns found in its transformed graph rooted in a vertex corresponding to a vertex of the original graph. This is summarized in the following proposition, whose proof is postponed to Appendix C.

Proposition 4 *If we let G'_1 (resp. G'_2) be the transformed graph of G_1 (resp. G_2), the no-tottering tree-pattern kernel of Definition 10 is given by*

$$\begin{aligned} K^{\mathrm{NT}}(G_1, G_2) &= \sum_{t\in\mathcal{T}} w(t)\psi_t^{\mathrm{NT}}(G_1)\psi_t^{\mathrm{NT}}(G_2) \\ &= \sum_{t\in\mathcal{T}} w(t)\psi_t^{\{V_{G_1}\}}(G'_1)\psi_t^{\{V_{G_2}\}}(G'_2), \end{aligned}$$

where, if G' is the transformed graph of G given by Definition 11, $V_G \subset \mathcal{V}_{G'}$ is the set of vertices of G' corresponding to the vertices of G, and $\psi_t^{\{v_1,\ldots,v_n\}}(G) = \sum_{i=1}^n \psi_t^{(v_i)}(G)$.

This proposition shows that we can compute no-tottering extensions of the kernels of Definitions 6 and 7, and of the until-N kernel extension of (8), using the graph transformation of Definition 11 and the original DP algorithms of Sects. 4.2 and 5.1. However, this operation comes at the expense of an increase in the cost of computing the kernel. More precisely, by definition of the graph transformation, we have $|\mathcal{V}_{G'}| = |\mathcal{V}_G| + |\mathcal{E}_G|$. Moreover, as noticed by Mahé et al. (2005), the maximum out-degree of the vertices of the transformed graph is equal to that of the original graph. As a result, the worst case complexity of evaluating the functional $k_h(u, v)$ of Propositions 1, 2 and 3 is the same if u and v belong to $\mathcal{V}_{G'_1}$ and $\mathcal{V}_{G'_2}$, or $\mathcal{V}_{G_1}$ and $\mathcal{V}_{G_2}$. It follows that for two graphs G_1 and G_2, the complexity of computing the non-tottering extension of any of the kernels given in (2), (3) and (8) is upper bounded by the complexity of computing the original kernel multiplied by the factor:

$$\frac{(|\mathcal{V}_{G_1}| + |\mathcal{E}_{G_1}|)(|\mathcal{V}_{G_2}| + |\mathcal{E}_{G_2}|)}{|\mathcal{V}_{G_1}||\mathcal{V}_{G_2}|}. \tag{10}$$

6 Experiments

We performed two series of experiments to validate the kernels proposed in this paper and test the effect of the extensions proposed. The first series is a small-scale experiment on two benchmark problems of toxicity prediction. We performed extensive experiments in order to assess the effects of the different parameters and extensions on this series. The second series of experiments is meant to assess the performance of the proposed kernels on a large-scale problem, that of predicting the anti-cancer effects of molecules on 60 cancer cell lines.

All classification experiments were carried out with a support vector machine as a learning algorithm, using the LibSVM implementation (Chang and Lin 2001) and the PyML python machine learning framework.[6] The kernels were pre-computed with the ChemCPP software,[7] a free and publicly available C++ toolbox for chemoinformatics where we implemented all kernels used in this study. We report results in terms of area under the ROC curve (AUC) for all datasets, as well as accuracy for the large-scale experiments. The performance is assessed in a cross-validation setting, using ten folds for the first series of experiments, and, for computational reasons, five folds for the second series of experiments. Within each cross-validation fold, the regularization parameter C of the SVM is chosen over the grid $\{10^{-3}, 10^{-2}, 10^{-1}, 10^0, 10^1, 10^2, 10^3\}$ as the value maximizing the mean AUC obtained by cross-validation over the training set of the fold.

[6]Available at http://pyml.sourceforge.net.

[7]Available at http://chemcpp.sourceforge.net.

6.1 Toxicity prediction

For our first series of experiments, we consider two small public datasets of chemical compounds together with informations about the toxicity of the molecules. Both datasets gather results of mutagenicity assays, and while the first one (King et al. 1996) is a standard benchmark for evaluating chemical compounds classification, the second one (Helma et al. 2004) was introduced more recently. The first dataset contains 188 chemical compounds tested for mutagenicity on *Salmonella typhimurium*. The molecules of this dataset belong to the family of aromatic and hetero-aromatic nitro compounds. They are split into two classes: 125 positive examples with high mutagenic activity (positive levels of log mutagenicity), and 63 negative examples with no or low mutagenic activity, and they are made of 26 atoms and 27.9 covalent bonds in average. The second database considered consists of 684 compounds classified as mutagens or non-mutagens according to a test known as the *Salmonella*/microsome assay. This dataset is well balanced with 341 mutagens compounds for 343 non-mutagens ones, made of 14.1 atoms and 14.6 covalent bonds in average. Although the biological property to be predicted is the same, the two datasets are fundamentally different. While King et al. (1996) focused on a particular family of molecules, the second dataset involves a set of very diverse chemical compounds, qualified as *noncongeneric* in the original paper. To predict mutagenicity, the model therefore needs to solve different tasks: in the first case it has to detect subtle differences between homogeneous structures, while in the second case it must seek regular patterns within a set of structurally different molecules. Our objective with this first series of experiments is to study in detail the introduction of subtree patterns in walk-based kernels. Walk-based kernels have notably been studied extensively in Swamidass et al. (2005) and Mahé et al. (2005). We refer the interested reader to these references for a detailed comparison between walk-based kernels and alternative approaches.

We first tested the classification performance reached by the size-based (Definition 6) and branching-based (Definition 7) formulations of the kernels. AUC values obtained for different parameters $0 \leq \lambda \leq 1$ and order h taken between 2 and 9 are displayed in Figs. 9 and 10 for both datasets, respectively. In both formulations, the parameter λ controls the relative weights given to subtree patterns in the kernel. In particular, for $\lambda = 0$ only linear subtrees are considered, and we recover the classical walk-based kernels. The fact that most performance curves, for different orders h, start by increasing when λ increases from 0 suggests that the introduction of subtree patterns is often beneficial to walk kernels. More precisely, an improvement can be observed for all orders h in the case of the size-based kernel (except for $h = 2$ in the first dataset), and for orders h between 2 and 7 for the branching-based kernel. For a given order h between 2 and 7, the optimal values obtained with the size- and branching-based kernels are similar, although the optimal λ values are systematically smaller for the branching-based formulation. This is due to the fact that, as noted in Sect. 4.1, the size-based penalization is stronger than the branching-based penalization. As a result, optimal λ values observed using the size-based kernel are shifted towards zero using the branching-based kernel. We can also note that optimal values of λ tend to decrease for increasing values of h. This is probably due to the fact that the number of tree-patterns increases exponentially with h, and that the kernels therefore need to limit their individual influence. We observe in fact that higher order patterns, with $h > 7$, can only be considered for sufficiently small values of λ. For example, the size-based kernel computation did not converge for patterns of order 9 and λ greater than 0.15. In the case of branching-based kernel, due to the weaker pattern penalization, this phenomenon is even emphasized, and in that case, 10^{-4} was the largest value acceptable for λ. Additionally, we note that because the size- and branching-based penalization of balanced trees of order 2 is the same, the

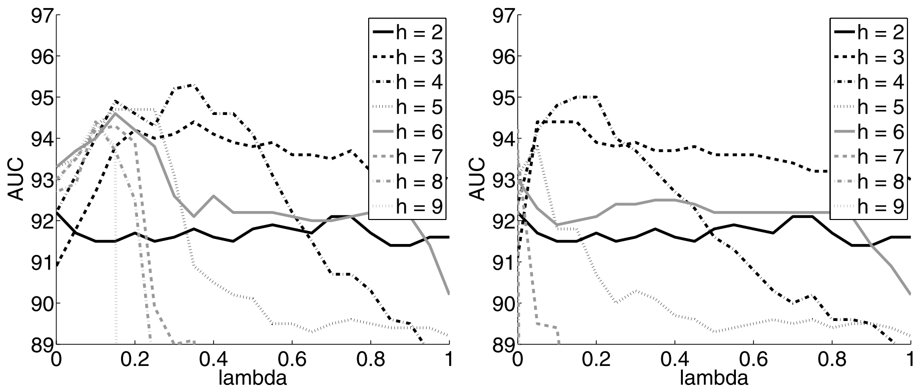


Fig. 9 First toxicity dataset. Evolution of the AUC with respect to λ at different orders h. *Left*: size-based kernel (2); *Right*: branching-based kernel (3)

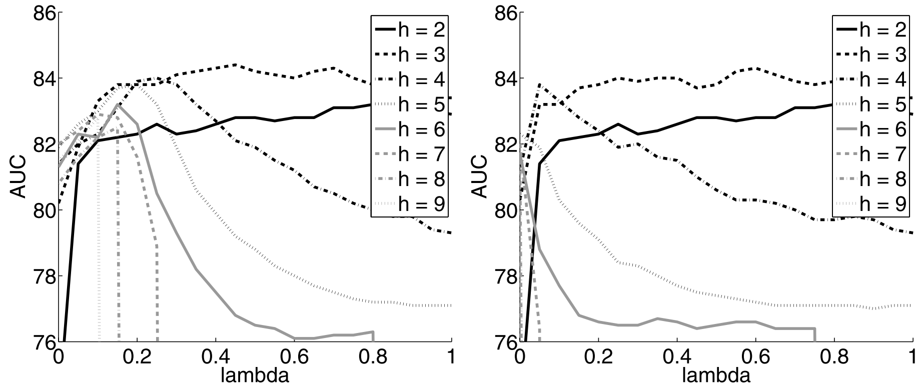


Fig. 10 Second toxicity dataset. Evolution of the AUC with respect to λ at different orders h. *Left*: size-based kernel (2); *Right*: branching-based kernel (3)

results obtained for $h = 2$ are identical with the two kernels. Interestingly, both datasets behave differently in that case: while the introduction of tree patterns brings no improvement over the walk-based kernel for the first dataset, an important relative improvement of 12% is observed for the second dataset. This suggests that different molecular substructures are relevant within each dataset.

Figure 11 presents the results of the until-N extension (8) of the branching-based kernel (3) for both datasets. No clear difference can be detected between these curves and the curves corresponding to the kernels without the until-N extensions (right-hand side plots in Figs. 9 and 10). The fact that the differences between the two kernel formulations are barely noticeable is surprising since their associated feature spaces are intuitively quite different. In Sect. 5.1, we mentioned that the feature space associated to the branching-based kernel is a subspace of the feature space associated to its until-N extension. Figure 11 therefore suggests that the extra features related to the until-N extension do not bear additional information into the kernel. This hypothesis seems to be confirmed by the fact that the differences between corresponding walk-based kernels, observed for $\lambda = 0$, are not significant neither. This might be explained by the fact that the dimensions of the corresponding feature space

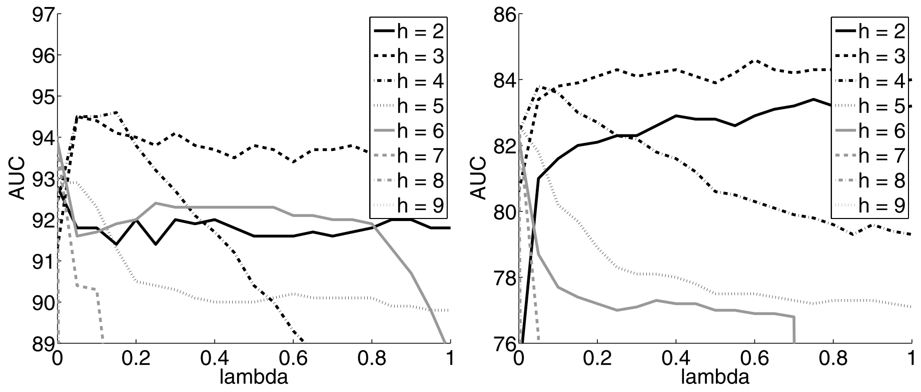


Fig. 11 Evolution of the AUC with respect to λ at different orders h, for the until-N extension (8) of the branching-based kernel (3) for the first toxicity dataset (*left*) and the second one (*right*)

are probably strongly correlated due to the relation of inclusion existing between trees and walks patterns of orders n, and those of order $n+1$. Another possible explanation for the lack of improvement of the until-N extension lies of course in the difficulty of learning in high dimension, suggesting that discriminating patterns of a given order are lost within the flood of patterns of greater orders taken into account by this until-N extension.

We finally tested the effect of filtering the tottering subtrees in the different kernel formulations. Figures 12, 13 and 14 show the results of the no-tottering extension (9) of the size-based (2), branching-based (3), and until-N branching-based kernels (8), respectively. Each figure shows the plots corresponding to both datasets. If we compare the results of the no-tottering extensions of the size-based and branching-based kernels (Figs. 12 and 13), we can first note that the introduction of tree-patterns is now systematically beneficial for $h > 2$ in both formulations, and for both datasets. Moreover, we note that the kernel computations remain feasible even for $h = 9$ and $\lambda = 1$, which means that the no-tottering extension limits the combinatorial explosion observed in the original formulation, and therefore allows the inclusion of larger subtree patterns in the feature space.

Both datasets clearly differ in terms of optimal order h. While optimal results were obtained for $h = 4$ using the original kernels for the first dataset, we observe that after the no-tottering extension the performance gradually increases from $h = 3$ to an optimum value obtained for $h = 8$. In the case of the second dataset, however, the optimal value $h = 3$ remains the same after the no-tottering extension, confirming that the most discriminative features are likely to be different between the two datasets. This is likely due to the fact that the compounds are structurally similar in the first dataset, and different (or *noncongeneric*) in the second dataset: while the kernel needs to detect subtle structural differences in the first case, it must identify more regular patterns in the second one. This observation supports the intuition that the choice of the kernel order h is problem-dependent and should be related to or learned from each particular dataset. In terms of absolute performance, optimal AUC values are close to 96.5% for the first dataset and improve over the values around 95% observed with the initial formulation. Importantly, we note that these optimal values are obtained using parametrizations of the kernels that lead to a combinatorial explosion in their initial formulation. For the second dataset, however, the best AUC are near 84%, bringing no improvement over the initial formulation.

Figure 14 shows similar performance curves for the no-tottering extension (9) of the until-N branching-based kernel (8). Analyzing the results for the first dataset (left-hand

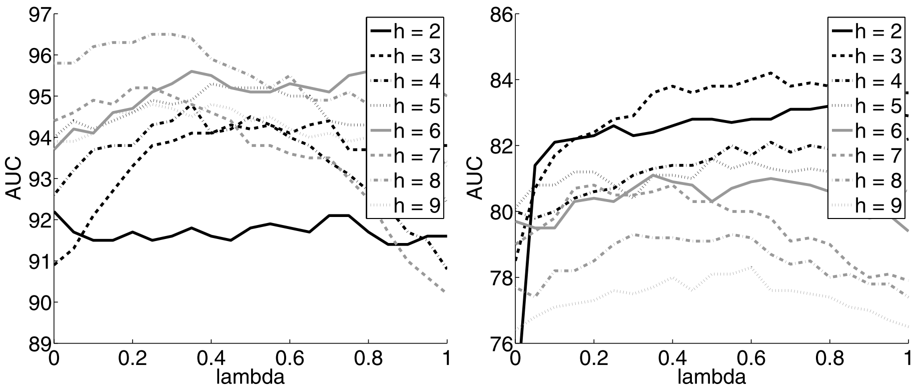


Fig. 12 AUC as a function of λ for different orders h for the no-tottering extension (9) of the size-based kernel (2) (first dataset on the *left*, second dataset on the *right*)

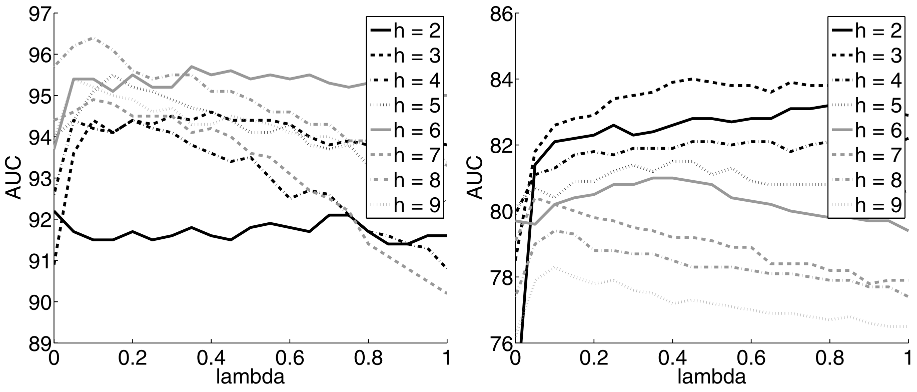


Fig. 13 AUC as a function of λ for different orders h for the no-tottering extension (9) of the branching-based kernel (2) (first dataset on the *left*, second dataset on the *right*)

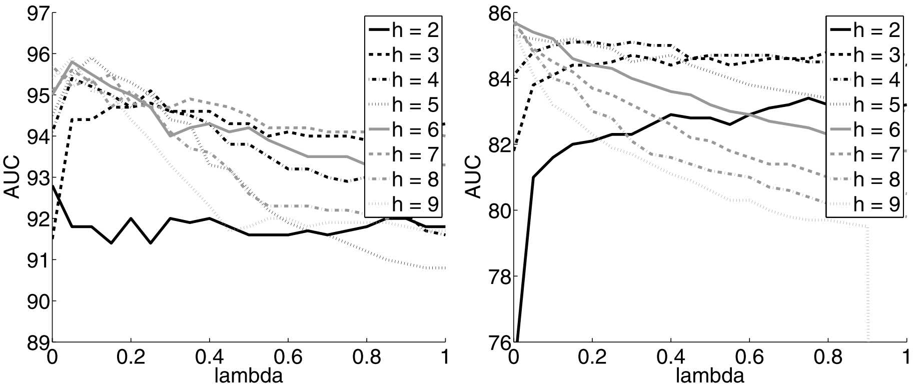


Fig. 14 AUC as a function of λ for different orders h for the no-tottering extension (9) of the until-N branching-based kernel (2) (first dataset on the *left*, second dataset on the *right*)

side), we can first notice that conclusions similar to those related to the no-tottering extension of the branching-based kernel can be drawn: an improvement over the corresponding walk-based kernel is systematically observed for tree-patterns of order greater than 2, the kernel behaves more nicely (no combinatorial explosion), and the no-tottering extension consistently improves over the initial until-N branching-based kernel. Interestingly however, we note that optimal results obtained for $4 \leq h \leq 9$ tend to converge to an optimal value around 95.5% (between 95.3 and 95.9%) for a λ value around 0.05. While this global optimum is not as good as the overall optimal result obtained with the no-tottering branch-based kernel (Fig. 13), it still remains competitive (95.5% vs 96.5%). This observation contrasts with the results obtained with the until-N extension in the tottering case, where patterns of a given order seemed to be lost in the amount of patterns of greater orders taken into account by the kernel. This is due to the fact the no-tottering extension limits the number of patterns to be detected, and suggests that patterns of different orders can now be considered simultaneously in the kernel. This fact therefore suggests that in the no-tottering case, the until-N extension can help solving the problem of pattern order selection by taking a maximal pattern order large enough (here, $h > 4$). In the case of the second dataset (right-hand side of Fig. 14), results are less clear. In that case, the introduction of the tree-patterns only improves the results for patterns of limited order, and for patterns of order greater than 4, results systematically decrease. We can however note the interesting point that optimal results obtained for patterns of order 5 to 9 converge to a global optimal value between 85 and 86%. This therefore tends to confirm that in the no-tottering case, the until-N extension can help solving the problem of pattern order selection by considering a maximal pattern order large enough (here, $h > 4$). Nevertheless, the striking difference with the results obtained with the first dataset is that in this case, when $h > 4$, the introduction of tree-patterns could not further improve the results obtained by the until-N walk-based kernel, that constitute the overall best performance we could observe for this dataset.

6.2 Anti-cancer activity prediction

The second series of experiments is meant to evaluate the performance of the subtree kernel on the NCI anticancer activity dataset. This dataset, made available by the Developmental Therapeutics Program (DTP) of the National Cancer Institute (NCI), provides screening results for the ability of about 70,000 compounds to suppress or inhibit the growth of a panel of 60 tumor cell lines, collectively known as the NCI-60 cell lines. We used the dataset corresponding to the concentration parameter GI50, essentially the concentration that causes 50% growth inhibition. We retrieved this dataset from the ChemDB database (Chen et al. 2005). For each of the 60 cell lines, an average of about 3,500 molecules are available and classified as inhibitory or not, depending on the GI50 value. The 60 datasets are well balanced between positive and negative examples, and therefore provide an interesting benchmark to assess the performance of classification algorithms in a realistic situation. Across the different cell lines, the molecules are made of 23.5 atoms and 25.4 covalent bonds on average.

For the sake of computational burden we limited our investigations to a comparison between a baseline walk-based kernel with a subtree-based kernel. Preliminary experiments with the walk-based kernel suggested that an order $h = 6$ gave good results.[8] Following our analysis on the small toxicity datasets, we therefore compared (i) the walk-based kernel,

[8]A walk-based kernel of order h is based on walks involving h vertices and $h - 1$ edges.

with $h = 6$ and the non-tottering extension, to (ii) the subtree-based kernel, with $h = 6$, the non-tottering extension, $\lambda = 0.4$ and the size-based formulation. In other words we tested whether, for a given order $h = 6$ observed to be good for the walk-based kernels, the introduction of subtrees (with a parameter $\lambda = 0.4$) would increase performance. Of course better results than the ones reported below may be obtained by a better tuning of the subtree kernel parameters.

For each of the 60 cell lines we evaluated the AUC and accuracy of the classification with both kernels. Results are shown in Fig. 15. They reveal the consistent improvement brought by the introduction of subtree patterns: on each of the 60 experiments, the subtree formulation outperforms the walk formulation, both in terms of AUC and accuracy. The average accuracy over the 60 lines increases from 68.9% to 70.0% between the two kernels (this improvement is significant with a P-value $< 2 \times 10^{-6}$ using a Wilcoxon paired one-sided test), while the average AUC increases from 74.3% to 76.1% (improvement significant with P-value $< 4 \times 10^{-11}$). This demonstrates the relevance of subtree patterns over linear patterns.

On the same dataset, Swamidass et al. (2005) recently reported state-of-the-art results using a voted perceptron coupled with various kernels based on linear fragments extracted from the 2D structure of the molecules. These kernels are variants of the non-tottering walk kernels that we investigated, with different ways to compute the kernels from the vectors of walk features. With an average accuracy (resp. AUC) over the 60 lines of 72.3% (resp. 78.7%), the best results reported in that study were obtained using the so-called "Min-Max" kernel, a variant of the Tanimoto coefficient that is widely used in the chemoinformatics community, computed from fragments involving a maximum of 10 covalent bonds. The Min-Max kernel is based on similar features as the walk kernel, namely the counts of occurrences of all linear fragments in a molecule. The only conceptual difference with the walk kernel resides in the way the kernel is computed from the feature vector representation: in the walk kernel a dot product between feature vectors is used, while in the Min-Max formulation, the kernel between two graphs is a sum over features of the ratio between the maximum and the minimum values of the features between the two graphs.

While the results of Swamidass et al. (2005) are better than those we obtained with the subtree pattern kernel, we wanted to investigate whether the difference was mainly due to the kernel itself, or to other features such as the algorithm used, the way parameters are selected, or the experimental protocol. Therefore we re-implemented the Min-Max kernel of Swamidass et al. (2005), using the exact same linear features as those we used in our walk kernel, and simply replacing the dot product computation by a sum of min/max ratios. We made this implementation publicly available in the ChemCpp toolbox.[9] We then ran classification experiments on the 60 cell lines for this Min-Max kernel with the same fragment length ($h = 6$) and experimental protocol as those used for the walk-based and subtree pattern kernels. Over the 60 experiments the Min-Max kernel reached an average accuracy of 70.9% and an average AUC of 77.5%, outperforming the subtree pattern kernel (respectively 70.0% and 76.1%). Although these results are slightly below those reported by Swamidass et al. (2005), they show that the Min-Max formulation of the linear fragment kernel outperforms the subtree pattern kernel both in accuracy (P-value $< 2 \times 10^{-5}$) and AUC (P-value $< 2 \times 10^{-7}$).

The overall conclusions of this large-scale study are that, in a controlled experiment: (i) the introduction of subtree patterns significantly improves the performance of the kernel

[9] http://chemcpp.sourceforge.net

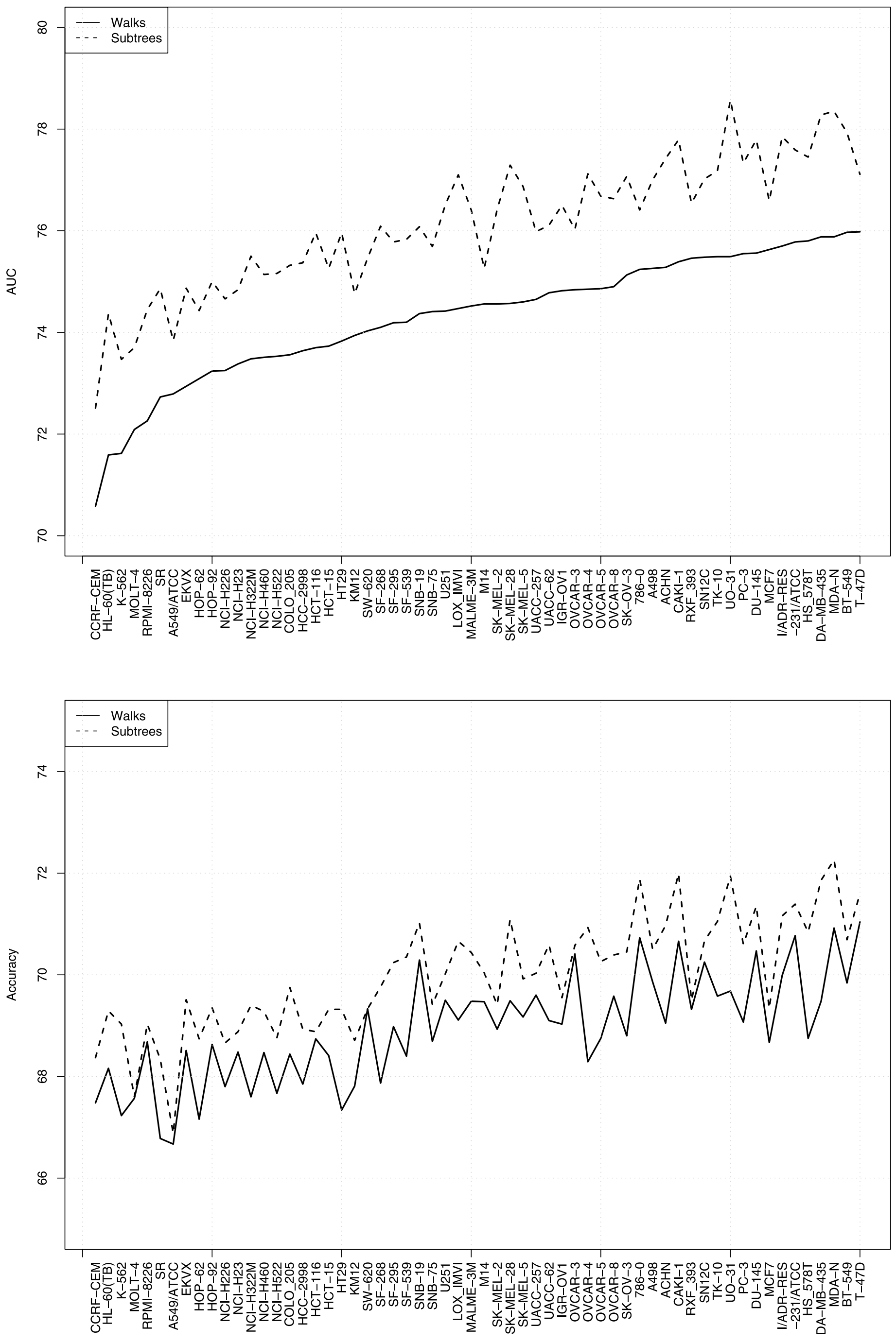


Fig. 15 AUC and accuracy for the 60 NCI cancer cell lines benchmark. Cell lines are sorted by increasing AUC of the walk kernel

based on the count of common linear walks when a dot product between feature vectors is used in both cases; (ii) the Min-Max kernel formulation outperforms the inner product formulation when linear pattern are considered, as already pointed out by Swamidass et al. (2005); (iii) the Min-Max kernel formulation with linear fragments also outperforms the inner product formulation for subtree patterns. These conclusions suggest that both the kernel formulation (inner product or Min-Max) and the feature used (linear or subtree patterns) have an influence on the final performance. An interesting direction for future research would be to combine both improvements, i.e., to propose a Min-Max formulation based on subtree patterns.

6.3 Computation times

We conclude this section discussing computation times. Our goal is to compare the time required, on the one hand, to compute size-based (2) and branching-based (3) tree-pattern kernels in their original and no-tottering formulations,[10] and, on the other hand, to compute such tree-pattern kernels and their walk-based counterparts (4). To do so we base our empirical analysis on the time needed to compute Gram matrices associated to the second toxicity dataset,[11] using a computer equipped with an AMD-64 bi-opteron 2.2 GHz processor and 4 GB RAM.

The left part of Fig. 16 shows the evolution of the computation time of the size- and branching-based tree-pattern kernels, in their original formulation, with respect to the order of the kernel. This curve shows a linear dependency between the computation time and the order of the kernel, which is consistent with the theoretical complexity derived in (7). The right part of Fig. 16 shows the ratio between the times needed to compute the no-tottering extension (9) and the original formulation of the size- and branching-based kernels, as a function of their order. This curve shows that, in all cases, this ratio is between 6 and 7, in accordance with the complexity analysis of Sect. 5.2, stating that the kernel complexities are proportional (10). More precisely, for a pair of graphs $G_1 = (\mathcal{V}_{G_1}, \mathcal{E}_{G_1})$ and $G_2 = (\mathcal{V}_{G_2}, \mathcal{E}_{G_2})$, the theoretical proportionality factor equals $\frac{(|\mathcal{V}_{G_1}|+|\mathcal{E}_{G_1}|)(|\mathcal{V}_{G_2}|+|\mathcal{E}_{G_2}|)}{|\mathcal{V}_{G_1}||\mathcal{V}_{G_2}|}$. Taking $\mathcal{V}_{G_1}$ and $\mathcal{V}_{G_2}$ (resp. $\mathcal{E}_{G_1}$ and $\mathcal{E}_{G_2}$) to be the average number of vertices (resp. edges) of the graphs of the dataset, this factor is approximately 9, which is consistent with the values 6 to 7 that are empirically observed.

Figure 17 is equivalent to Fig. 16 for the kernel based on the count of common walks (4), which, as explained in Sect. 4.1, corresponds to the size-based (2) and branching-based (3) kernels when the parameter λ entering their definition tends to zero.[12] For moderate walk lengths, this type of kernel can be computed efficiently using an algorithm derived from that used in the implementation of spectrum string kernels, based on trie-tree structures (Leslie et al. 2002; Shawe-Taylor and Cristianini 2004). Each internal node of a trie-tree has its children indexed by a given alphabet. Taking the alphabet to be the set of vertex labels, it is easy to see that if we consider such a trie-tree of depth $n + 1$, there is a direct correspondence between its set of leaves and the set of walk labels of length n that can possibly be

[10]Recall from Sects. 4.2 and 5.1 that the size- and branching-based kernels have the same complexity, and that the "until-N" extension comes at no extra cost.

[11]This dataset is made of 684 molecules, and the times reported correspond to the evaluation of $\frac{684\times685}{2} = 234,270$ kernel values.

[12]Recall from Sect. 4.1 that, when $\lambda = 0$, the size-based and branching-based kernel of order n correspond to the kernel based on the count of common walks of length $n - 1$.

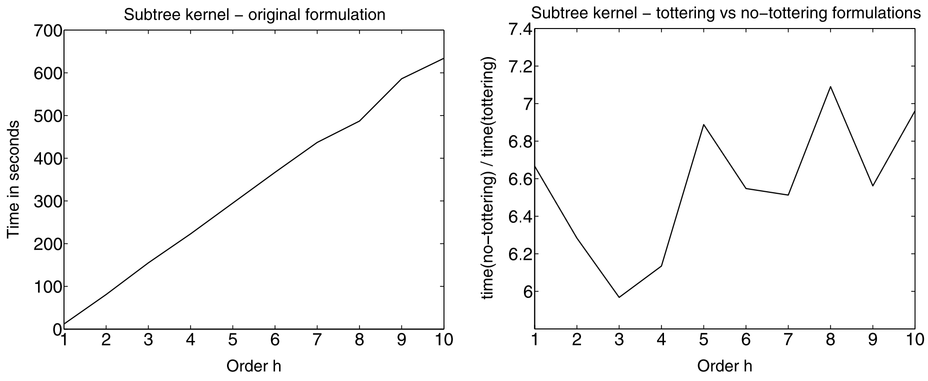


Fig. 16 Computation times of the size-based (2) and branching-based (3) tree-pattern kernels, versus their order. *Left*: computation time in their original formulation. *Right*: computation time with the no-tottering extension (9), in comparison with their original formulation

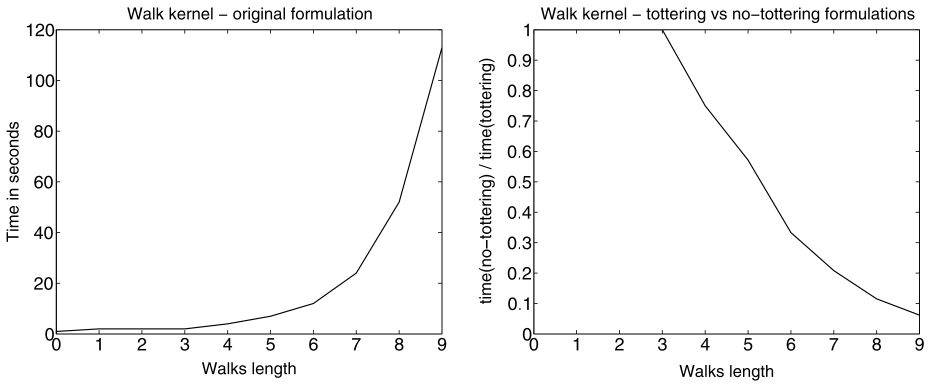


Fig. 17 Computation times of walk-based kernels, versus the length of the walks considered. *Left*: computation time in the original formulation of the kernel (4). *Right*: computation time when tottering walks are filtered, in comparison with the original formulation

found in the graphs.[13] Computing the kernel amounts to recursively traversing the trie-tree while maintaining, within each graph, a set of pointers referencing the occurrences of the corresponding labeled walks. When reaching a leaf node, these pointers reference the occurrences of walks to be taken into account in the kernel, that is, walks of length n having a particular label, and the entries of the kernel matrix can be updated. In practice, the number of distinct walk labels that can be found among the graphs of a dataset is often smaller than the number of possible walk labels. As a result, the whole trie-tree needs not be traversed, and the complexity of this implementation is expressed as the number of leaves of the trie-tree that need to be visited in order to compute the kernel. Making the assumption that the graphs are fully connected, and letting $\alpha(G)$ (resp. $\beta(G)$) represent the number of distinct vertex (resp. edge) labels of the graph G, it follows that the complexity of computing the

[13]For simplicity we forget about the edge labels. In reality, the children of a node are indexed by a pair vertex label/edge label.

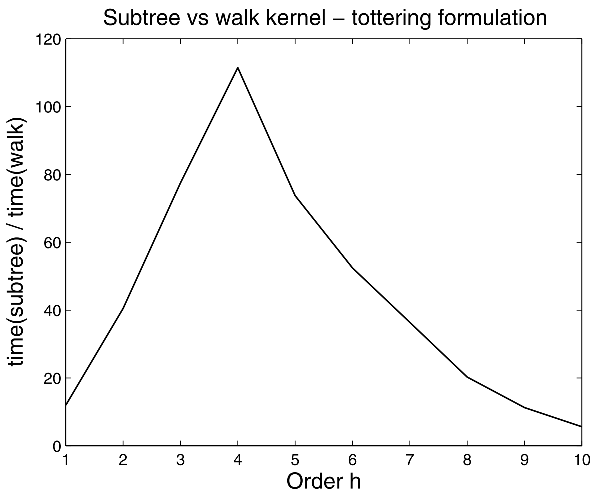


Fig. 18 Compared evolution of the computation times of the subtree and walk-based kernels with respect to the order of the kernel

kernel matrix for a set of graphs $\{G_i\}_{1\le i\le l}$ is upper bounded by

$$\max_{1\le i\le l} \alpha(G_i)^{n+1}\beta(G_i)^n,$$

where n is the length of the walks considered. In the case of graphs representing chemical compounds, the assumption of fully connectivity is obviously false, and the true complexity is much smaller. Nevertheless, this analysis is consistent with the left hand side of Fig. 17, that shows an exponential dependency between the computation time of the walk-based kernel (4) and the length of the walks considered, in opposition to the linear dependency observed in Fig. 16. Extending the above algorithm to filter tottering walks does not necessarily involve the graph transformation introduced in Sect. 5.2. Indeed, this can be done at almost no extra cost with the introduction of an additional condition to be verified when updating the set of pointers along the trie-tree traversal process. Interestingly, this extension has the effect of reducing the number of leaves of the trie-tree that need to be visited and, as a result, the right hand side of Fig. 17 shows a drastic decrease of the computation time in the no-tottering formulation of the kernel, which contrasts with what was observed in Fig. 16.

Finally, Fig. 18 show the ratio between the time needed to compute tree-pattern and walk-based kernel in their original formulations (that is, without no-tottering extensions), with respect to the order of the kernels. This curve shows that for small orders, tree-patterns kernels are much more costly to compute than their walk-based counterparts. However, because the complexity of the tree-pattern kernel increases linearly with the order, while it increases exponentially for the walk-based kernel, the ratio becomes smaller at high orders. At order 10 however, it is still nearly 6 times more costly to compute tree-patterns rather than walk-based graph kernels in their original formulations, and this factor becomes 630 in their no-tottering formulation.

7 Discussion

This paper introduces a family of graph kernels based on the detection of common tree patterns in the graphs. In a first step, we revisited an initial formulation presented in Ramon and

Gärtner (2003), from which we derived two kernels with explicit feature spaces and inner products. A parameter λ enters their definition and makes it possible to control the complexity of the features characterizing the graphs. At the extreme, admissible tree-patterns consist of linear chains of graph vertices, and the kernels resume to a classical graph kernel based on the detection of common walks (Gärtner et al. 2003). Walk-based graph kernels are therefore generalized to a wider class of kernels defined by features of increasing levels of complexity. In a second step we introduced two modular extensions to this initial formulation. On the one hand, the set of trees initially indexing the feature space is enriched by the set of their subtrees with an *until-N* extension, leading to a wider and more general feature space. On the other hand, a *no-tottering* extension prevents spurious tree-patterns to be detected, based on the notion of "tottering" initially introduced in the context of walk-based graph kernels (Mahé et al. 2005).

With respect to chemical applications, experiments on two toxicity datasets and one large anti-tumor activity dataset demonstrate that the tree-pattern graph kernels under their initial formulation improve over their walk-based counterpart. The experiments on the two toxicity datasets allowed us to investigate in detail the effects of the various parameters and extensions. In particular, while no significant difference was detected between the different weighting schemes of the subtrees (size- or branching-based), the no-tottering extension allowed the computation of the kernels for more parameters and improved the results in the first toxicity dataset, where discriminative subtrees of order 8 seem to exist.

A problem often encountered with structure kernels is the problem of *diagonal dominance*, i.e., the fact that the elements on the diagonal of the kernel Gram matrix are much larger than other elements. We were not confronted with this issue in our experiments, in spite of the large dimension of the feature spaces used, for several reasons. First, the graphs used to represent molecules have usually a very small degree in average, and therefore a limited number of walk or tree patterns. Second, the no-tottering extension we proposed further reduces the number of patterns present. On other applications where graphs with larger degrees may be analyzed, such as image or multimedia documents, diagonal dominance may become a problem and require specific algorithmic solutions.

Among the possible extensions to our work, we note that it might be relevant in the context of chemical applications to incorporate chemical knowledge in the graph representation of the molecules. For instance, it is well known that physico-chemical properties of atoms are related to their position in the molecule, and as a first step in this direction, an enrichment of atom labels by their Morgan indices led to promising results in the context of walk-based kernels (Mahé et al. 2005). However, this particular approach is likely to have a lesser impact in this context, because the information encoded by the Morgan indices is at some extend already incorporated in the tree-patterns. Alternatively, we note that the kernel implementation could easily be extended in order to introduce a flexible matching between tree-patterns based on measures of similarity between pairs of vertices and edges, following for instance the construction of the marginalized kernel between labeled graphs (Kashima et al. 2004). Such an extension would induce an increase in the cost of computing the kernel, but is likely to make sense for chemical applications, where atoms of different types can exhibit similar properties.

Last but not least, we note that on the NCI dataset, subtree kernels were outperformed by the state-of-the-art walk-based kernels of Swamidass et al. (2005). This suggests, as a promising direction for future work, to introduce tree patterns in the family of kernels presented in Swamidass et al. (2005), in particular in the so-called Min-Max formulation.

Open Access This article is distributed under the terms of the Creative Commons Attribution Noncommercial License which permits any noncommercial use, distribution, and reproduction in any medium, provided the original author(s) and source are credited.

Appendix A: Proof of Propositions 1 and 2

In Propositions 1 and 2, we want to prove that for the graphs G_1 and G_2

$$\sum_{t\in\mathcal{B}_h} w(t)\psi_t(G_1)\psi_t(G_2) = \alpha(h)\sum_{u\in\mathcal{V}_{G_1}}\sum_{v\in\mathcal{V}_{G_2}} k_h(u,v), \tag{11}$$

where in Proposition 1, $\alpha(h)=\lambda^{-h}$ and $w(t)=\lambda^{|t|-h}$, while in Proposition 2, $\alpha(h)=1$ and $w(t)=\lambda^{\text{branch}(t)}$.

From Definition 4 we have $\psi_t(G)=\sum_{u\in\mathcal{V}_G}\psi_t^{(u)}(G)$. As a result,

$$\sum_{t\in\mathcal{B}_h} w(t)\psi_t(G_1)\psi_t(G_2) = \sum_{u\in\mathcal{V}_{G_1}}\sum_{v\in\mathcal{V}_{G_2}}\left(\sum_{t\in\mathcal{B}_h} w(t)\psi_t^{(u)}(G_1)\psi_t^{(v)}(G_2)\right),$$

and in order to prove (11) we just need to prove

$$\sum_{t\in\mathcal{B}_h} w(t)\psi_t^{(u)}(G_1)\psi_t^{(v)}(G_2) = \alpha(h)k_h(u,v). \tag{12}$$

A.1 Proof of Proposition 1

In order to prove Proposition 1, it follows from (12) that we just need to prove that

$$\frac{1}{\lambda^h}k_h(u,v) = \sum_{t\in\mathcal{B}_h}\lambda^{|t|-h}\psi_t^{(u)}(G_1)\psi_t^{(v)}(G_2),$$

or equivalently:

$$k_h(u,v) = \sum_{t\in\mathcal{B}_h}\lambda^{|t|}\psi_t^{(u)}(G_1)\psi_t^{(v)}(G_2), \tag{13}$$

where k_h is defined recursively by $k_1(u,v)=\lambda\mathbf{1}(l(u)=l(v))$ and for $h>1$:

$$k_h(u,v) = \lambda\mathbf{1}(l(u)=l(v))\sum_{R\in\mathcal{M}(u,v)}\prod_{(u',v')\in R} k_{h-1}(u',v'). \tag{14}$$

We prove (13) by induction on h. The case $h=1$ is rather trivial. Indeed, a tree of depth one is just a single node, and $\psi_t^{(u)}(G_1)$ is therefore equal to 1 if $l(u)=l(r(t))$, 0 otherwise. It follows that

$$\begin{aligned}\sum_{t\in\mathcal{B}_1}\lambda^{|t|}\psi_t^{(u)}(G_1)\psi_t^{(v)}(G_2) &= \sum_{t\in\mathcal{B}_1}\lambda\mathbf{1}(l(r(t))=l(u))\mathbf{1}(l(r(t))=l(v))\\ &= \lambda\mathbf{1}(l(u)=l(v)),\end{aligned}$$

which corresponds to $k_1(u,v)$.

Let us now assume that (13) is true at order $h-1$, and let us prove that it is then also true at order $h>1$. Combining the recursive definition of k_h (14) with the induction hypothesis (13) at level $h-1$ we first obtain:

$$k_h(u,v) = \lambda\mathbf{1}(l(u)=l(v))\sum_{R\in\mathcal{M}(u,v)}\prod_{(u',v')\in R}\sum_{t'\in\mathcal{B}_{h-1}}\lambda^{|t'|}\psi_{t'}^{(u')}(G_1)\psi_{t'}^{(v')}(G_2). \tag{15}$$

Second, for any graph G, let us denote by $\mathcal{P}_n^{(u)}(G)$ the set of balanced tree-patterns of order n rooted in $u \in \mathcal{V}_G$, and for any tree-pattern $p \in \mathcal{P}_n^{(u)}(G)$ let $t(p) \in \mathcal{B}_n$ denote the corresponding tree. With these notations we can rewrite, for any $n \geq 1$ and $(u, v) \in G_1 \times G_2$:

$$\sum_{t \in \mathcal{B}_n} \lambda^{|t|} \psi_t^{(u)}(G_1) \psi_t^{(v)}(G_2) = \sum_{p_1 \in \mathcal{P}_n^{(u)}(G_1)} \sum_{p_2 \in \mathcal{P}_n^{(v)}(G_2)} \lambda^{|t(p_1)|} \mathbf{1}(t(p_1) = t(p_2)). \tag{16}$$

Indeed both sides of this equation count the number of pairs of similar tree-patterns rooted in u and v. Plugging (16) into (15) we get:

$$\begin{aligned} k_h(u, v) &= \lambda \mathbf{1}(l(u) \\ &= l(v)) \sum_{R \in \mathcal{M}(u,v)} \prod_{(u',v') \in R} \sum_{p_1 \in \mathcal{P}_{h-1}^{(u')}(G_1)} \sum_{p_2 \in \mathcal{P}_{h-1}^{(v')}(G_2)} \lambda^{|t(p_1)|} \mathbf{1}(t(p_1) = t(p_2)). \end{aligned} \tag{17}$$

Now we use the fact that any tree-pattern p of order h can be uniquely decomposed into a tree-pattern p' of order 2 and a set of tree-patterns of order $h - 1$ rooted at the leaves of p'. We note that matching two tree-patterns is equivalent to matching the tree-patterns in their decomposition, and that the sets of leaves of tree-patterns of order 2 rooted respectively in u and v matching each other are exactly given by $\mathcal{M}(u, v)$. In other words, (17) performs a summation over pairs of matching tree-patterns of depth h, rooted respectively in u and v: the corresponding pairs of patterns of order 2 are implicitly matched by the summation over $\mathcal{M}(u, v)$ and the condition $\mathbf{1}(l(u) = l(v))$, and the subsequent pairs of patterns (p_1, p_2) of order $h - 1$ are matched by the product of conditions $\mathbf{1}(t(p_1) = t(p_2))$.

The tree-pattern p_1 in G_1 of such a matching pair of tree-patterns of order h rooted in (u, v) decomposes as a pattern of depth 2 rooted in u with leaves in some $R \in \mathcal{M}(u, v)$, and a set of patterns $p_1(u')$ of depth $h - 1$ rooted in the leaves $u' \in R$. By (17), to each such matching pair is associated the weight $\lambda \times \prod_{(u',v') \in R} \lambda^{|t(p1(u'))|}$, which is exactly equal to $\lambda^{|t(p_1)|}$ since we obviously have $|t(p_1)| = 1 + \sum_{(u',v') \in R} |t(p_1(u'))|$. As a result, (17) can be rewritten as:

$$k_h(u, v) = \sum_{p_1 \in \mathcal{P}_h^{(u)}(G_1)} \sum_{p_2 \in \mathcal{P}_h^{(v)}(G_2)} \lambda^{|t(p_1)|} \mathbf{1}(t(p_1) = t(p_2)),$$

which combined with (16) proves (13).

A.2 Proof of Proposition 2

The proof of Proposition 2 is a straightforward variant of the proof of Proposition 1. By (12) we need to show that

$$k_h(u, v) = \sum_{t \in \mathcal{B}_h} \lambda^{\mathrm{branch}(t)} \psi_t^{(u)}(G_1) \psi_t^{(v)}(G_2), \tag{18}$$

where k_h is defined recursively by $k_1(u, v) = \mathbf{1}(l(u) = l(v))$ and for $h > 1$:

$$k_h(u, v) = \frac{\mathbf{1}(l(u) = l(v))}{\lambda} \sum_{R \in \mathcal{M}(u,v)} \prod_{(u',v') \in R} \lambda k_{h-1}(u', v'). \tag{19}$$

We proceed again by induction over h to prove (18). The case $h = 1$ is easily done by checking, using an argument similar to that of the previous proof, that (18) is one if $l(u)$

and $l(v)$ are identical, zero otherwise, which corresponds to the definition of $k_1(u,v)$. If we assume that (18) is true at the level $h-1$, we can plug it in (19) to obtain:

$$\begin{aligned} k_h(u,v) = {} & \frac{\mathbf{1}(l(u)=l(v))}{\lambda} \\ & \times \sum_{R\in\mathcal{M}(u,v)} \prod_{(u',v')\in R} \sum_{t'\in\mathcal{B}_{h-1}} \lambda^{1+\text{branch}(t')} \psi_{t'}^{(u')}(G_1)\psi_{t'}^{(v')}(G_2). \end{aligned} \tag{20}$$

We can then follow exactly the same line of proof as in the previous section and obtain the following equations

$$\begin{aligned} & \sum_{t\in\mathcal{B}_n} \lambda^{\text{branch}(t)} \psi_t^{(u)}(G_1)\psi_t^{(v)}(G_2) \\ & \quad = \sum_{p_1\in\mathcal{P}_n^{(u)}(G_1)} \sum_{p_2\in\mathcal{P}_n^{(v)}(G_2)} \lambda^{\text{branch}(t(p_1))} \mathbf{1}(t(p_1)=t(p_2)), \end{aligned} \tag{21}$$

and

$$\begin{aligned} k_h(u,v) = {} & \frac{\mathbf{1}(l(u)=l(v))}{\lambda} \\ & \times \sum_{R\in\mathcal{M}(u,v)} \prod_{(u',v')\in R} \sum_{p_1\in\mathcal{P}_{h-1}^{(u')}(G_1)} \sum_{p_2\in\mathcal{P}_{h-1}^{(v')}(G_2)} \lambda^{1+\text{branch}(t(p_1))} \mathbf{1}(t(p_1)=t(p_2)), \end{aligned} \tag{22}$$

that correspond respectively to (16) and (17). The only difference with the previous proof is in the exponent of λ to form the weight of a matching pair of tree-patterns. By analogy with the previous proof, we consider the tree-pattern p_1 in G_1 of a pair of matching tree-patterns of depth h rooted in (u,v), that decomposes as a pattern of depth 2 rooted in u with leaves in some $R\in\mathcal{M}(u,v)$, and a set of patterns $p_1(u')$ of depth $h-1$ rooted in the leaves $u'\in R$. By (22), to each such matching pair is associated the weight $\frac{1}{\lambda}\prod_{(u',v')\in R}\lambda^{1+\text{branch}(t(p_1(u')))} = \lambda^{-1+\sum_{(u',v')\in R}1+\text{branch}(t(p_1(u')))}$. We observe that the number of leaves of a tree t, that we note leaves(t), is equal to $1+\text{branch}(t)$. The weight associated to the above pair of matching tree-patterns can therefore be written as:

$$\lambda^{-1+\sum_{(u',v')\in R}\text{leaves}(t(p_1(u')))}.$$

Finally, because the number of leaves of the tree-pattern p_1 is equal to the sum of the leaves of the patterns $p_1(u')$, it follows that this expression is equal to $\lambda^{-1+\text{leaves}(t(p_1))} = \lambda^{\text{branch}(t(p_1))}$. As a result, we can write (22) as

$$k_h(u,v) = \sum_{p_1\in\mathcal{P}_h^{(u)}(G_1)} \sum_{p_2\in\mathcal{P}_h^{(v)}(G_2)} \lambda^{\text{branch}(t(p_1))}\mathbf{1}(t(p_1)=t(p_2)),$$

which, combined with (21), concludes the proof.

Appendix B: Proof of Proposition 3

The proof presented in this section is very similar to the proofs of Propositions 1 and 2. Based on the observations made in the beginning of Appendix A, it follows from (12) that in order to prove Proposition 3, we just need to prove that

$$k_h(u,v) = \sum_{t \in \mathcal{T}_h} \lambda^{\mathrm{branch}(t)} \psi_t^{(u)}(G_1) \psi_t^{(v)}(G_2), \tag{23}$$

where k_h is defined recursively by $k_1(u,v) = \mathbf{1}(l(u) = l(v))$ and for $h > 1$

$$k_h(u,v) = \mathbf{1}(l(u) = l(v)) \left(1 + \sum_{R \in \mathcal{M}(u,v)} \frac{1}{\lambda} \prod_{(u',v') \in R} \lambda k_{h-1}(u',v') \right). \tag{24}$$

We proceed again by induction over h to prove (23). The case $h = 1$ directly follows from the proof of Proposition 2. If we assume that (23) is true at the level $h-1$, we can plug it in (24) to obtain:

$$\begin{aligned} k_h(u,v) &= \mathbf{1}(l(u) = l(v)) \\ &\quad \times \left(1 + \sum_{R \in \mathcal{M}(u,v)} \frac{1}{\lambda} \prod_{(u',v') \in R} \sum_{t' \in \mathcal{T}_{h-1}} \lambda^{1+\mathrm{branch}(t')} \psi_{t'}^{(u')}(G_1) \psi_{t'}^{(v')}(G_2) \right). \end{aligned} \tag{25}$$

By analogy with the construction of the previous proof, for any graph G, let us denote by $\mathcal{P}_n^{(u)}(G)$ the set of tree-patterns of depth 1 to n rooted in $u \in \mathcal{V}_G$, and for any tree-pattern $p \in \mathcal{P}_n^{(u)}(G)$ let $t(p) \in \mathcal{T}_n$ denote the corresponding tree. Note that $\mathcal{P}_n^{(u)}(G)$ corresponds here to general tree-patterns of depth 1 to n, in opposition to the balanced-tree patterns of order n involved in the previous proofs. With these notations we obtain similarly, for any $n \geq 1$ and $(u,v) \in G_1 \times G_2$:

$$\begin{aligned} &\sum_{t \in \mathcal{T}_n} \lambda^{\mathrm{branch}(t)} \psi_t^{(u)}(G_1) \psi_t^{(v)}(G_2) \\ &\quad = \sum_{p_1 \in \mathcal{P}_n^{(u)}(G_1)} \sum_{p_2 \in \mathcal{P}_n^{(v)}(G_2)} \lambda^{\mathrm{branch}(t(p_1))} \mathbf{1}(t(p_1) = t(p_2)), \end{aligned} \tag{26}$$

and, plugging (26) into (25), we get:

$$\begin{aligned} k_h(u,v) &= \mathbf{1}(l(u) = l(v)) \\ &\quad \times \left(1 + \sum_{R \in \mathcal{M}(u,v)} \frac{1}{\lambda} \prod_{(u',v') \in R} \sum_{p_1 \in \mathcal{P}_{h-1}^{(u')}(G_1)} \sum_{p_2 \in \mathcal{P}_{h-1}^{(v')}(G_2)} \lambda^{1+\mathrm{branch}(t(p_1))} \right. \\ &\quad \left. \times \mathbf{1}(t(p_1) = t(p_2)) \right), \end{aligned} \tag{27}$$

which can be further decomposed into:

$$\begin{aligned} k_h(u,v) &= \mathbf{1}(l(u) = l(v)) + \frac{\mathbf{1}(l(u) = l(v))}{\lambda} \\ &\quad \times \sum_{R \in \mathcal{M}(u,v)} \prod_{(u',v') \in R} \sum_{p_1 \in \mathcal{P}_{h-1}^{(u')}(G_1)} \sum_{p_2 \in \mathcal{P}_{h-1}^{(v')}(G_2)} \lambda^{1+\mathrm{branch}(t(p_1))} \mathbf{1}(t(p_1) = t(p_2)). \end{aligned} \tag{28}$$

The second term in the sum of the right-hand side of (28) matches pairs of tree-patterns of depth 2 to n rooted in (u,v). It follows directly from the proof of Proposition 2 that such a pair (p_1, p_2) of matching tree-patterns is weighted by $\lambda^{\text{branch}(t(p_1))}$. The first part of the right member of (28) matches the trivial pair of tree-patterns of depth 1 rooted in (u, v) consisting of the single nodes (u, v). The corresponding tree has a zero branching cardinality, and we can therefore write

$$\mathbf{1}(l(u) = l(v)) = \sum_{t \in \mathcal{T}_1} \lambda^{\text{branch}(t)} \psi_t^{(u)}(G_1) \psi_t^{(v)}(G_2).$$

Taken together, these two arguments show that (28) can be written as

$$k_h(u, v) = \sum_{t \in \mathcal{T}_h} \lambda^{\text{branch}(t)} \psi_t^{(u)}(G_1) \psi_t^{(v)}(G_2),$$

which concludes the proof.

Appendix C: Proof of Proposition 4

The proof is derived from results presented in Mahé et al. (2005). The sets of walks and no-tottering walks of the graph $G = (\mathcal{V}_G, \mathcal{E}_G)$ are respectively defined by $\mathcal{W}(G) = \bigcup_{n=0}^{\infty} \mathcal{W}_n(G)$ and $\mathcal{W}^{\text{NT}}(G) = \bigcup_{n=0}^{\infty} \mathcal{W}_n^{\text{NT}}(G)$, where

$$\mathcal{W}_n(G) = \{(v_0, \ldots, v_n) \in \mathcal{V}_G^{n+1} : (v_i, v_{i+1}) \in \mathcal{E}_G, 0 \le i \le n-1\}$$

is the set of walks of length n defined is Sect. 4.1, and

$$\mathcal{W}_n^{\text{NT}}(G) = \{(v_0, \ldots, v_n) \in \mathcal{W}_n(G) : v_i \neq v_{i+2}, 0 \le i \le n-2\}$$

is the set of no-tottering walks of length n defined in Mahé et al. (2005). We start by stating the following lemma.

Lemma 1 *A tree-pattern p of the graph G associated to the tree t is no tottering if, and only if, any walk of G defined as a succession of vertices of p corresponding to nodes of t forming a path from its root to one of its leaves is no-tottering.*

Proof of Lemma 1 According to Definition 9, let $(v_1, \ldots, v_{|t|}) \in \mathcal{V}_G^{|t|}$ be a no-tottering tree pattern of the graph $G = (\mathcal{V}_G, \mathcal{E}_G)$ corresponding to the tree $t = (\mathcal{V}_t, \mathcal{E}_t)$, where $\mathcal{V}_t = (n_1, \ldots, n_{|t|})$. Let $(n_{i_0}, \ldots, n_{i_k}) \in \mathcal{V}_t^{k+1}$ be a path from the root of t to one of its leaves. By Definition 3, it is clear that $(v_{i_0}, \ldots, v_{i_k}) \in \mathcal{W}(G)$. Moreover, by the definition of paths we have $(n_{i_m}, n_{i_{m+1}}), (n_{i_{m+1}}, n_{i_{m+2}}) \in \mathcal{E}_t$ for $0 \le m \le k-2$. By Definition 9, this implies that $v_{i_m} \neq v_{i_{m+2}}$ for $0 \le m \le k-2$, meaning that $(v_{i_0}, \ldots, v_{i_k}) \in \mathcal{W}^{\text{NT}}(G)$. Conversely, let $p \in \mathcal{V}_G^{|t|}$ be a tree-pattern of the graph $G = (\mathcal{V}_G, \mathcal{E}_G)$ corresponding to the tree $t = (\mathcal{V}_t, \mathcal{E}_t)$. Consider the set of walks of G defined as successions of vertices of p associated to nodes of t forming paths from its root to its leaves. If these walks are not tottering, it is clear from Definition 9 that the tree-pattern itself is not tottering. □

We can now state the proof of Proposition 4.

Proof of Proposition 4 If, according to Definition 11, we let G' be the transformed graph of G, Mahé et al. (2005) showed that there is a bijection between $\mathcal{W}^{\text{NT}}(G)$ and the set of walks of G' starting in a vertex corresponding to a vertex of G, which can be formally defined as

$$\mathcal{W}^{\{V_G\}}(G') = \{(v_0, \ldots, v_n) \in \mathcal{W}(G') : v_0 \in \{V_G\}, n \in \mathbb{N}\},$$

if we let $V_G \subset \mathcal{V}_{G'}$ be the subset of $\mathcal{V}_{G'}$ that corresponds to $\mathcal{V}_G$. It follows from Lemma 1 that there is a bijection between the set of no-tottering tree-patterns of G and the set of tree-patterns of G' rooted in a vertex of V_G. Finally, Mahé et al. (2005) showed that a walk in $\mathcal{W}^{\mathrm{NT}}(G)$ and its image in $\mathcal{W}^{\{V_G\}}(G')$ are identically labeled, which enables to count no-tottering labeled walks in G, by counting identically labeled walks in G' starting in a vertex of V_G. It follows that counting no-tottering tree-patterns in G is equivalent to counting tree-patterns in G' rooted in a vertex of V_G. As a result, we have $\psi_t^{\mathrm{NT}}(G) = \psi_t^{\{V_G\}}(G')$, which concludes the proof. $\square$

References

Borgwardt, K. M., Ong, C. S., Schönauer, S., Vishwanathan, S. V. N., Smola, A. J., & Kriegel, H.-P. (2005). Protein function prediction via graph kernels. *Bioinformatics*, *21*(Suppl. 1), i47–i56.

Chang, C.-C., & Lin, C.-J. (2001). *LIBSVM: a library for support vector machines*.

Chen, J., Swamidass, S. J., Dou, Y., Bruand, J., & Baldi, P. (2005). ChemDB: a public database of small molecules and related chemoinformatics resources. *Bioinformatics*, *21*(22), 4133–4139.

Deshpande, M., Kuramochi, M., Wale, N., & Karypis, G. (2005). Frequent substructure-based approaches for classifying chemical compounds. *IEEE Transactions on Knowledge and Data Engineering*, *17*(8), 1036–1050.

Gärtner, T., Flach, P., & Wrobel, S. (2003). On graph kernels: hardness results and efficient alternatives. In B. Schölkopf & M. Warmuth (Eds.), *Lecture notes in computer science: Vol. 2777. Proceedings of the sixteenth annual conference on computational learning theory and the seventh annual workshop on kernel machines* (pp. 129–143). Heidelberg: Springer.

Helma, C., Cramer, T., Kramer, S., & De Raedt, L. (2004). Data mining and machine learning techniques for the identification of mutagenicity inducing substructures and structure activity relationships of noncongeneric compounds. *Journal of Chemical Information and Computer Sciences*, *44*(4), 1402–1411.

Horváth, T., Gärtner, T., & Wrobel, S. (2004). Cyclic pattern kernels for predictive graph mining. In *Proceedings of the tenth ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 158–167). New York: ACM Press.

Inokuchi, A., Washio, T., & Motoda, H. (2003). Complete mining of frequent patterns from graphs: mining graph data. *Machine Learning*, *50*(3), 321–354.

Karklin, Y., Meraz, R. F., & Holbrook, S. R. (2005). Classification of non-coding RNA using graph representations of secondary structure. In *Pacific symposium on biocomputing* (pp. 4–15).

Kashima, H., Tsuda, K., & Inokuchi, A. (2004). Kernels for graphs. In B. Schölkopf, K. Tsuda, & J. P. Vert (Eds.), *Kernel methods in computational biology* (pp. 155–170). Cambridge: MIT Press.

King, R. D., Muggleton, S. H., Srinivasan, A., & Sternberg, M. J. (1996). Structure-activity relationships derived by machine learning: the use of atoms and their bond connectivities to predict mutagenicity by inductive logic programming. *Proceedings of the National Academy of Sciences of the United States of America*, *93*(1), 438–442.

Leach, A. R., & Gillet, V. J. (2003). *An introduction to chemoinformatics*. Dordrecht: Kluwer Academic.

Leslie, C., Eskin, E., & Noble, W. S. (2002). The spectrum kernel: a string kernel for SVM protein classification. In R. B. Altman, A. K. Dunker, L. Hunter, K. Lauerdale, & T. E. Klein (Eds.), *Proceedings of the Pacific symposium on biocomputing 2002* (pp. 564–575). Singapore: World Scientific.

Mahé, P., Ueda, N., Akutsu, T., Perret, J.-L., & Vert, J.-P. (2005). Graph kernels for molecular structure-activity relationship analysis with support vector machines. *Journal of Chemical Information and Modeling*, *45*(4), 939–951.

Menchetti, S., Costa, F., & Frasconi, P. (2005). Weighted decomposition kernels. In L. De Raedt & S. Wrobel (Eds.), *Proceedings of the twenty-second international conference on machine learning* (ICML 2005) (pp. 585–592). New York: Assoc. Comput. Mach.

Ralaivola, L., Swamidass, S. J., Saigo, H., & Baldi, P. (2005). Graph kernels for chemical informatics. *Neural Networks*, *18*(8), 1093–1110.

Ramon, J., & Gärtner, T. (2003). Expressivity versus efficiency of graph kernels. In T. Washio & L. De Raedt (Eds.), *Proceedings of the first international workshop on mining graphs, trees and sequences* (pp. 65–74).

Schölkopf, B., & Smola, A. J. (2002). *Learning with kernels: support vector machines, regularization, optimization, and beyond*. Cambridge: MIT Press.

Shawe-Taylor, J., & Cristianini, N. (2004). *Kernel methods for pattern analysis*. Cambridge: Cambridge University Press.

Swamidass, S. J., Chen, J., Bruand, J., Phung, P., Ralaivola, L., & Baldi, P. (2005). Kernels for small molecules and the prediction of mutagenicity, toxicity and anti-cancer activity. *Bioinformatics*, *21*(Suppl. 1), i359–i368.