Graph Kernels for Molecular Structure−Activity Relationship Analysis with Support Vector Machines

Pierre Mahé,*,† Nobuhisa Ueda,‡ Tatsuya Akutsu,‡ Jean-Luc Perret,‡ and Jean-Philippe Vert†

Ecole des Mines de Paris, 35 rue Saint Honoré, 77305 Fontainebleau, France, and Bioinformatics Center, Institute for Chemical Research, Kyoto University, Uji, Kyoto 611-0011, Japan

Received February 2, 2005

The support vector machine algorithm together with graph kernel functions has recently been introduced to model structure−activity relationships (SAR) of molecules from their 2D structure, without the need for explicit molecular descriptor computation. We propose two extensions to this approach with the double goal to reduce the computational burden associated with the model and to enhance its predictive accuracy: description of the molecules by a Morgan index process and definition of a second-order Markov model for random walks on 2D structures. Experiments on two mutagenicity data sets validate the proposed extensions, making this approach a possible complementary alternative to other modeling strategies.

1. INTRODUCTION

Accurate predictive models applied early during the drug design process can lead to substantial savings in terms of time and costs for the development of new drugs. While processing of chemical data involves many different tasks, including for instance clustering, regression, classification, or ranking, most of them are related to *Structure−Activity Relationship (SAR) analysis*, that is, finding a relationship between the *structures* of molecules and their *activity*. We employ the term activity here in a broad sense to refer to a particular biological property the molecules exhibit, such as their ability to bind to a particular biological target, their toxicity properties, or their ADME (Absorption, Distribution, Metabolism, Excretion) properties.

To build a model relating structure to activity, machine learning methods require a set of molecules with known activity, usually called the *training set*. Once adjusted on the training set, the model can then be used to predict the activity of new molecules. Decades of research in machine learning and statistics provide many different methods to fit a model. Each of them has its own specificities, and the choice of a particular model is usually related to the final objectives of the analysis (e.g., efficiency of the prediction versus interpretability of the model). Nevertheless, an important topic common to all models concerns the way chemical compounds are represented.

While many descriptors related to global physicochemical properties or to the 3D structures of molecules are often used, we focus in this paper on the simpler 2D representation of molecules, which we see as a labeled graph with atoms as vertices and covalent bonds as edges. While this representation might appear too restrictive for some applications, it appears sufficient to build state-of-the-art predictors for properties such as mutagenicity.[1]

On the methodological side, working directly with this representation requires the analysis, comparison, and classification of labeled graphs. Among the many different ways to tackle this problem, two mainstream research directions have emerged during the past decades. One direction involves the use of graph algorithms to compare or classify graphs, for instance by finding maximal or frequent common subgraphs. Such approaches usually suffer from their computational complexity (NP-hardness of subgraph isomorphism, exponential number of potential subgraphs) and are usually based on heuristics or restricted to small-size graphs and data banks. The second mainstream direction, particularly in chemoinformatics, consists of transforming the graphs into vectors using molecular descriptors, before applying the whole panoply of statistical or machine learning tools to the vector representations. This usually requires the selection of a small number of features of interest, which is known to be a difficult task, especially for noncongeneric data sets.[1]

An alternative direction has been explored recently in the pioneering papers refs 2 and 3, using the theory of positive definite kernels and support vector machines (SVM).[4−6] SVM, an increasingly popular classification method in machine learning and chemoinformatics[7,8] for its good performance on many real-world problems, possesses two important properties. First, it theoretically allows learning in very high—potentially infinite—dimensions thanks to a heavy use of regularization;[5] second, instead of an explicit representation of data as vectors, it only requires inner products between such representations, through what is usually referred to as a positive definite kernel function. Ushering in the avenue opened by SVM to implicitly map molecules to infinite dimensional spaces, refs 2 and 3 introduce positive definite kernels between labeled graphs, based on the detection of common fragments between different graphs. These kernels correspond to a dot product between the graphs mapped to an infinite-dimensional feature space indexed by all possible finite-length fragments but can be computed in polynomial time with respect to the graph sizes. Encouraging experimental results suggest that this

* Corresponding author phone: (+33) 1 64 69 49 94; e-mail: pierre.mahe@ensmp.fr.
† Ecole des Mines de Paris.
‡ Kyoto University.

10.1021/ci050039t CCC: $30.25
Published on Web 00/00/0000 PAGE EST: 12.6

approach might be a valid alternative to the two mainstream directions.

These graph kernels, however, are subject to several limitations. First, the graph kernel has a computational complexity roughly proportional to the product of the sizes of the two graphs to be compared, which results in slow implementation for real-world problems. Second, it might not be optimal to use all fragments to characterize each graph for at least two reasons: on one hand, some fragments may contain relatively little information (e.g. a sequence $C - C - C$), while on the other hand, many fragments as defined in refs 2 and 3 are irrelevant because they represent "tottering paths" on the graph, that is, paths which return to a visited vertex immediately after leaving it.

The purpose of this paper is to propose two extensions of the original graph kernel, which try to address these issues. The first extension is to relabel each vertex automatically in order to insert information about the environment of each vertex in its label. This has both an effect in terms of feature relevance, because fragments of such labels contain information about the environment of each atom, and computation time, because the number of identical fragments between two molecules significantly decreases. Second, we show how to modify the random walk model proposed in ref 2 in order to remove totters, without increasing the complexity of the implementation. Each method is validated on two benchmark experiments of mutagenicity prediction, showing the potentiality of this approach by improving the original graph kernel both in terms of speed and accuracy and reaching state-of-the-art prediction performance.

The paper is organized as follows. After a brief review of support vector machines and kernels in section 2, section 3 presents the graph kernels introduced in refs 2 and 3. The following two sections introduce two extensions to these graph kernels, with the double goal to reduce their computational complexity and to increase their relevance as a similarity measure. Finally, we conclude with experimental results in section 6. The current work is an expanded version of a conference proceeding paper.[9] We have included new experimental results and a more thorough discussion.

2. SUPPORT VECTOR MACHINE

SVM[4−6] is a machine learning framework for supervised binary classification originally developed in the 1990s by V. Vapnik and co-workers, although extensions to multiclass classification, regression, and density estimation also exist. We focus here on support vector machines for binary classification, the task considered in our experiments. Interested readers can find a more thorough presentation of this algorithm and many related ones in ref 6.

Formally, given a set of n objects (e.g., molecules) $x_1, ..., x_n \in \mathcal{X}$, and associated binary labels (e.g., active/nonactive, toxic/nontoxic) denoted $y_1, ..., y_n \in \{-1, 1\}$, SVM produces a classifier $f: \mathcal{X} \rightarrow \{-1, 1\}$ that can be used to predict the class of any new data $x \in \mathcal{X}$. As mentioned in the Introduction, the common approach in chemoinformatics consists of representing a molecule by a set of descriptors, and the molecular space $\mathcal{X}$ usually corresponds to the Euclidian space R^d. In such vector spaces, the classifier output by SVM is based on the sign of a linear function: $f(x) = \mathrm{sign}(\langle w, x\rangle + b)$, for some $(w, b) \in \mathcal{X} \times \mathrm{R}$ defined below. Geometrically, a hyperplane $\langle w, x\rangle + b = 0$ separates the input space $\mathcal{X}$ into two half-spaces, and the prediction of the class of a new point depends on the position of the point on the one or on the other side of the hyperplane. When the data set is *separable*, i.e., when a hyperplane exists such that the positive and negative examples lie on distinct sides of the hyperplane, SVM chooses among the infinity of separating hyperplanes the one with the largest distance to the closest data point. This distance is known as the *margin* of the hyperplane, and this particular hyperplane defines the *maximum margin* classifier, as illustrated in Figure 1.

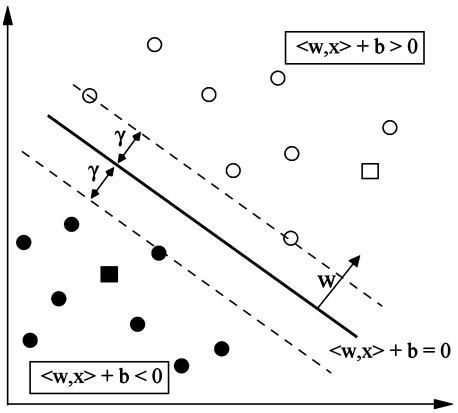

Figure 1. SVM finds the hyperplane $\langle w, x\rangle + b = 0$ that separates positive (white circles) and negative (black circles) examples with a maximum margin γ. Here, a new example represented by the white (respectively black) square is predicted as positive (respectively negative).

More generally, in particular for nonseparable data sets, the SVM algorithm finds the separating hyperplane by solving the following optimization problem

$$\min_{w,b} \frac{1}{2}||w||^2 + C\sum_{i=1}^{n}(y_i(\langle w, x_i\rangle + b) - 1)_+ \qquad (1)$$

where C is a parameter and $(u)_+ = \max(u, 0)$. The rational behind this optimization problem is to find a linear classifier that reaches a tradeoff between the amount of errors on the training set (as quantified by the second term of this sum) and the smoothness of the classifier (as quantified by the first term). In the extreme case, when $C = +\infty$ and the data set is separable, then no error is allowed on the training set, and the classifier with largest margin is found. Classical Lagrangian optimization theory shows that this problem is equivalent to the following dual problem:

$$\max_{\alpha} \sum_{i=1}^{n}\alpha_i - \frac{1}{2}\sum_{i,j=1}^{n}\alpha_i\alpha_j y_i y_j \langle x_i, x_j\rangle$$

$$\text{subject to : } \sum_{i=1}^{n}\alpha_i y_i = 0 \text{ and } 0 \le \alpha_i \le C,\ i \in [1{:}n] \qquad (2)$$

When the optimum α^* is met, w writes as $w = \sum_{i=1}^{n}\alpha_i^* y_i x_i$, and the decision function f becomes $f(x) = \mathrm{sign}(\sum_{i=1}^{n}\alpha_i^*\langle x, x_i\rangle + b^*)$, the value b^* being computed from the α_i^* and the x_i.

A striking point of SVM lies in the fact that they can be generalized to nonvectorial spaces $\mathcal{X}$, and in particular to the space of molecular compounds represented by their 2D

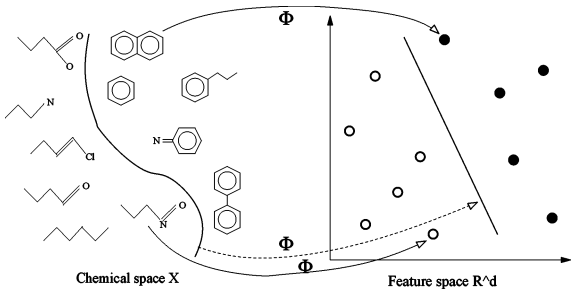

Figure 2. A feature map Φ from the chemical space $\mathcal{X}$ to the vector space R^d.

or 3D structures, by embedding them to a vector space $\mathcal{H}$ (often called the feature space) through a mapping $\Phi:\mathcal{X}\rightarrow\mathcal{H}$ and applying the linear SVM in $\mathcal{H}$ to the training points $\Phi(x_i)$, $i = 1, ..., n$. Figure 2 illustrates this mapping process. [Note that the step of feature extraction usually required by machine learning algorithms can be seen as a particular mapping $\phi:\mathcal{X}\rightarrow\mathcal{H} = R^d$.] Indeed, an important remark is that in the dual formulation (2), the data are only present through dot-products: pairwise dot-products between the training points during the learning phase, and dot-products between a new data and the training points during the test phase. This means that instead of explicitly knowing $\Phi(x)$ for any $x \in \mathcal{X}$, it suffices to be able to compute inner products of the form $k(x, x') = \langle\Phi(x),\Phi(x')\rangle$ for any $x, x' \in \mathcal{H}$, in which case the optimization algorithm rewrites

$$\max_{\alpha} \sum_{i=1}^{n}\alpha_i - \frac{1}{2}\sum_{i,j=1}^{n}\alpha_i\alpha_j y_i y_j k(x_i, x_j)$$

$$\text{subject to : } \sum_{i=1}^{n}\alpha_i y_i = 0 \text{ and } 0 \le \alpha_i \le C, i \in [1 : n] \quad (3)$$

and the classification function becomes $f(x) = \text{sign}\,(\sum_{i=1}^{n}\alpha_i^* k(x, x_i) + b^*)$. The function k is called a kernel. A classical result states that any function $k: \mathcal{X}\times\mathcal{X}\rightarrow R$ can be plugged in the SVM optimization problem as long as it is symmetric and positive definite, the key point being that it is sometimes easier to compute directly the kernel between two points than computing their explicit representations as vectors in $\mathcal{H}$.

This property offers at least two major advantages. First, it enables the straightforward extension of the linear SVM to model nonlinear decision functions by using a nonlinear kernel, while keeping its nice properties intact (e.g. unicity of the solution, robustness to overfitting, etc.). Second, it offers the possibility to directly apply SVM to nonvectorial data, provided a kernel function exists to compare them. For these reasons, an important research topic has emerged in the past few years, focusing on the design of kernel functions dealing with structured data[10] such as strings, trees, or graphs. Clearly choosing a kernel amounts to choosing an implicit representation of the objects, and prior knowledge might serve as a guide to design a suitable representation for a given problem. A second and often contradictory constraint to take into account is the necessity to have fast kernel computations, as a naive SVM implementation might require the computation of $n(n + 1)/2$ kernel values.

In particular, different families of kernels for graphs have recently been proposed, that naturally allow the molecules

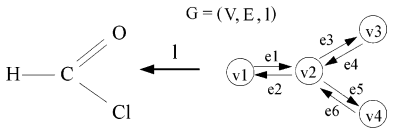

Figure 3. A chemical compound seen as a labeled graph.

to be represented by their 2D structures. Similarity is usually assessed by walks occurring concurrently in the graphs, since subgraphs characterization suffers from computational complexity. The first approach is to simply count the number of common walks. Gärtner introduced in ref 11 several kernels that count the number of walks with identical starting and ending nodes, and in ref 3, he defines with co-workers another family of kernels counting globally identical walks. A somehow similar but more flexible approach is proposed in ref 2 where the graph similarity is defined as the sum of pairwise similarities of walks found in the two graphs, weighted by a coefficient standing for the probability that this particular pair of walks occur simultaneously in the two graphs. This formulation, which forms the starting point of our work, is presented in more details in the next section.

3. MARGINALIZED GRAPH KERNELS

In this section we define the basic notations and briefly review the graph kernel introduced in refs 2 and 3, upon which are based the extensions that will be presented in sections 4 and 5.

3.1. Labeled Graphs. A *labeled graph* $G = (V, E)$ is defined by a finite set of *vertices* V, a set of *edges* $E \subset V \times V$, and a labeling function l: $V \cup E \rightarrow A$ which assigns a *label* $l(x)$ taken from an alphabet A to any vertex or edge x. We let $|V|$ be the number of vertices of G, and $|E|$ its number of edges. We assume below that a set of labels A has been fixed and consider different labeled graphs (each labeled graph corresponding to a particular molecular compound). More precisely, if we associate a labeled graph to a chemical compound, the set of vertices V corresponds to the set of atoms of the molecule, the set of edges E to its covalent bonds, and these graph elements are labeled according to an alphabet A consisting of the different types of atoms and bonds. Note that we consider directed graphs here, so that a pair of edges of opposite direction is introduced in the graph for each covalent bond of the molecule. Figure 3 shows a chemical compound seen as a labeled graph.

For a given graph $G = (V, E)$, we denote by $d(v)$ the number of edges emanating from the vertex v (i.e., the number of edges of the form (v, u)), and by $V^* = \cup_{n=1}^{\infty} V^n$ the set of finite-length sequences of vertices. A *path* $h \in V^*$ is a finite-length sequence of vertices $h = v_1 ... v_n$ with the property that $(v_i, v_{i+1}) \in E$ for $i = 1, ..., n - 1$. The length of a path is equal to the number of edges it is made of, and we note $|h|$ is the length of the path h. We define $H(G) \subset V^*$ to be the set of all paths of a graph G. The labeling function l: $V \cup E \rightarrow A$ can be extended as a function l: $H(G) \rightarrow A^*$ where the label $l(h)$ of a path $h = v_1 ... v_n \in H(G)$ is the succession of labels of the vertices and edges of the path: $l(h) = (l(v_1), l(v_1, v_2), l(v_2), ..., l(v_{n-1}, v_n), l(v_n)) \in A^{2n-1}$.

3.2. Marginalized Graph Kernels. As briefly mentioned in the previous section, the kernel introduced in ref 2 is derived from the general *marginalized kernels* formulation.[12]

Marginalized kernels define a global similarity measure by means of a simpler one expressed on auxiliary variables introduced in the problem. In our case, these latent variables consist of substructures of the graphs, and more precisely they are paths, which are easier to handle than subgraphs. In labeled graphs, label sequences are associated with the paths of the graph, and their similarity is assessed by a string kernel. Moreover, paths are here considered as random walks, so that probability distributions are associated with the set of paths of the graphs. The kernel between two graphs is then defined as the expectation of the pairwise paths similarity, according to their probability distributions.

The kernel introduced in ref 2 boils down to the following formula where p_{G1} and p_{G2} are probability distributions on

$$K(G_1, G_2) = \sum_{(h_1,h_2)\in V_1^* \times V_2^*} p_{G1}(h_1)p_{G2}(h_2)K_L(l(h_1),l(h_2)) \quad (4)$$

the set of paths V_1^* and V_2^*, and the function K_L:$A^* \times A^* \rightarrow$ R is a (string) kernel between label sequences.

Reference 2 focuses on the particular case where the kernel K_L in (4) is the Dirac function δ

$$K_L(l_1, l_2) = \delta(l_1, l_2) = \begin{cases} 1 & \text{if } l_1 = l_2, \\ 0 & \text{otherwise} \end{cases} \quad (5)$$

thus accounting for a perfect similarity between paths if they share the same label and null otherwise; and where, for a graph $G = (V, E)$, the probability p_G on V^* factorizes as a first-order random walk model:

$$p_G(v_1 \ldots v_n) = p_s(v_1)\prod_{i=2}^{n} p_t(v_i|v_{i-1}) \quad (6)$$

To ensure that (6) defines a probability distribution on V^* (i.e., $\sum_{v\in V^*} p_G(v) = 1$), we must impose constraints on the emission and transition probabilities p_s and p_t.

This can be done, for example, by choosing parameters $0 < p_q(v) < 1$ for $v \in V$, an initial probability distribution p_0 on V ($\sum_{v\in V} p_0(v) = 1$), a transition matrix p_a on $V \times V$ ($\sum_{u\in V} p_a(u|v) = 1$ for $v \in V$) positive only along edges ($p_a(v|u) > 0 \Rightarrow (u, v) \in E$), and by setting, for any $u, v \in V^2$

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

Under these conditions it can easily be checked that (6) is a probability distribution on V^* corresponding to a random walk on the graph with initial distribution p_0, transition probability p_a, and stopping probability p_q at each step. In particular, this implies that only paths have positive probabilities under p: $p_G(h) > 0 \Rightarrow h \in H(G)$. Figure 4 shows an example of this particular probabilistic model.

Following the definition introduced in ref 1, we let $\mathcal{S}(A)$ be the set of *linear molecular fragments* taken from A, i.e., the set of sequences of bonds-connected atoms based on the labels of A. For a given fragment $s \in \mathcal{S}(A)$ we introduce a mapping ϕ_s:$\mathcal{X} \rightarrow$ R defined for a given graph G as $\phi_s(G) = \sum_{h\in H(G)} p_G(h)\delta(l(h),s)$. If we let K_L be the Dirac kernel, eq 4 can be written as a standard dot-product

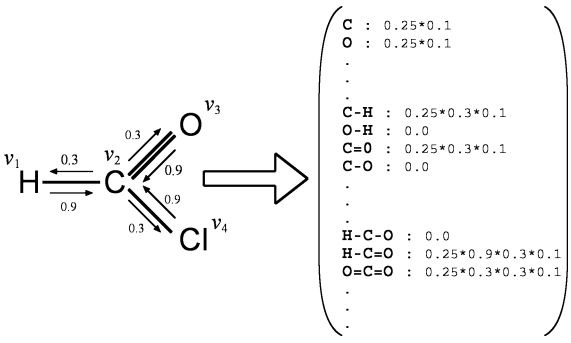

Figure 4. A molecular graph G (left) and its feature-space representation $\phi(G)$ (right). Here, $\forall i$, $p_q(v_i) = 0.1$, $p_a(v_j|v_i) = 1/d$ (v_i) iff (v_i, v_j) $\in E$, and p_0 can be chosen to be the unifom distribution, i.e., $p_0(v_i) = 1/|V| = 0.25$. The values $(1 - p_q(v_i))p_a(v_j|v_i)$ are shown along each edge (v_i, v_j) of the graph. (Note that $\sum_j p_a(v_j|v_i) = 1\ \forall i$). For the path $h = (v_1, v_2, v_3)$, we therefore have $l(h) = (H, -, C, =, O)$ and $p(h) = 0.25 * 0.9 * 0.3 * 0.1$. The right-hand side of the picture shows such examples of paths possibly occurring in the graph, together with their associated probabilities.

based on the molecular fragments of A:

$$K(G_1, G_2) = \langle \Phi(G_1), \Phi(G_2) \rangle = \sum_{s\in \mathcal{S}(A)} \phi_s(G_1)\phi_s(G_2)$$

The kernel (4) therefore maps the graphs into an infinite-dimensional space where each dimension corresponds to a particular substructure. This shows an analogy with the fingerprints characterization of molecules widely used in chemoinformatics.[13] Molecular fingerprints encode a molecule as a (finite length) vector where each dimension corresponds to a particular molecular substructure. Each substructure is represented by a bit or an integer, that either indicates the presence of the substructure in the molecule or counts the number of times it appears.

There are however several important differences with the approach illustrated above. First, while marginalized kernels take into account every single molecular fragment to compare the molecules, which is equivalent to dealing with an infinite-dimensional feature vector, fingerprints only involve a smaller number of features. [Usually between 150 and 2500, but hashed-fingerprints provide a way to consider a much larger number of features.] These substructures are carefully chosen (based on prior chemical knowledge), and by focusing on a restricted set of substructures, fingerprint description of molecules may be more efficient if these features are indeed relevant according to the learning task to perform. On the other hand, because they consider a larger set of substructures, marginalized kernels can detect features not taken into account by standard fingerprints that may be useful to account for molecular similarity. Another important difference lies in the way of dealing with the substructures. Instead of just checking the presence of molecular fragments, marginalized kernels can quantify their occurrence in the graph according to probability distributions. This approach offers a more flexible way to evaluate the influence of the substructures in the graph similarity. Note that the method presented in ref 3 is equivalent to defining an infinite-dimensional fingerprint counting the frequency of appearance of the molecular fragments in the graphs.

3.3. Graph Kernel Computation. While the kernel definition (4) involves a summation over an infinite number of paths, it can be computed efficiently using product graphs and matrix inversions introduced in ref 3 and briefly recalled below.

Given two labeled graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, their *product graph* is defined as the labeled graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ whose vertices $\mathcal{V} \subset V_1 \times V_2$ are pairs of vertices with identical labels ($(v_1, v_2) \in \mathcal{V}$ iff $l(v_1) = l(v_2)$), and an edge connects the vertices (u_1, u_2) and (v_1, v_2) iff $(u_i, v_i) \in E_i$, for $i = 1, 2$, and $l(u_1, v_1) = l(u_2, v_2)$.

Let us now define a functional π on the set of paths $H(\mathcal{G})$ by

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

with

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

where $p_s^{(1)}$ and $p_t^{(1)}$ (respectively $p_s^{(2)}$ and $p_t^{(2)}$) are the functions used to define the probabilities of random paths in (6) on the graph G_1 (respectively G_2).

If the label kernel K_L is chosen to be the Dirac kernel (5), then the kernel (4) only involves paths that can be found concurrently in the two graphs. By construction of the product graph, there is a bijection between this set of common paths and the set of paths $H(\mathcal{G})$ of the product graph. Using the definition of the functional π, it can then be shown that

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

Define now the $|\mathcal{V}| \times |\mathcal{V}|$ transition matrix $\Pi_t = (\pi_t(v|u))_{(u,v)\in\mathcal{V}^2}$. Paths in the product graph can be generated by raising this matrix to a particular power. If one now defines the $|\mathcal{V}|$-dimensional vector $\pi_s = (\pi_s(v))_{v\in\mathcal{V}}$, it can be checked that

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

where $\mathbf{1}$ is the $|\mathcal{V}|$-dimensional vector with all entries equal to 1 and therefore

$$K(G_1, G_2) = \sum_{n=1}^{\infty}\Big(\sum_{h \in H(\mathcal{G}), |h| = n} \pi(h)\Big)$$

$$= \pi_s{}^T (I - \Pi_t)^{-1}\mathbf{1}$$

The direct computation of a matrix inversion has a complexity cubic in the size of the matrix. In the case of a product graph, the size of the matrix Π_t is at worst $|V_1| \times |V_2|$, and this approach can be time-consuming. However, this matrix is typically sparse, and savings can be achieved using an approximation of the matrix inverse based on the first terms of its power series expansion: $(I - \Pi_t)^{-1} \approx \sum_{i=0}^{N}\Pi_t{}^i$. Generally speaking, if we note $|M|$ the number of nonzero elements of a matrix M, and $d(M)$ its maximum number of nonzero elements per line, computing the product of two $(n \times n)$ sparse matrices A and B has a complexity of $O(|A|d(B))$. Moreover, if we note $d = d(A)$, we have $d(A^k) \leq \min(d^k, n)$. From these two observations, it follows that computing the sum $\sum_{i=0}^{N}A^i$ has a complexity of $O(|A|\sum_{i=1}^{N-1}\min(d^i, n))$.[14] Note that if no hypothesis is made about the value of d, this complexity reduces to $O(|A|nN)$.

By construction of the product graph $G = G_1 \times G_2$, we have $|\mathcal{V}| \leq |V_1| \times |V_2|$ and $|\mathcal{E}| \leq |E_1| \times |E_2|$. Moreover, if d_1 and d_2 are the maximum degrees of the nodes of G_1 and G_2, it follows that the maximum degree of the nodes of the graph $\mathcal{G}$ is less or equal than d_1d_2. This means that the size of the matrix Π_t is bounded by $|V_1| \times |V_2|$, its maximum number of nonzero elements by $|E_1| \times |E_2|$, and its maximum nonzero elements per line by d_1d_2. It therefore follows that the approximation of the matrix $(I - \Pi_t)^{-1}$ by the first N terms of its power series expansion has a complexity of $O(|E_1||E_2|\sum_{i=1}^{N-1}\min((d_1d_2)^i, |V_1||V_2|))$.

In the case where many vertices have identical labels, the product graph used to compute the graph kernel has many vertices too, since the number of vertices in the product graph corresponds to the number of pairs of vertices with identical labels. As a result, the computation of the graph kernel can be time-consuming, and this method may be difficult to use on large chemical data banks involving several hundred thousand molecules. As an example, the computation can take several hundred milliseconds on a recent desktop computer to compute the kernel between two chemical compounds with a moderate number of atoms (typically between 10 and 50). Moreover, one might expect the search of common path labels to be too naive to detect interesting patterns between chemical compounds.

These two points constitute important issues to tackle in order to use this type of graph kernels in real-world applications. We now present two modifications of the original kernel with the goals to increase its relevance as a similarity measure between molecular compounds, usually denoted as its expressive power, and to reduce its computational complexity.

4. LABEL ENRICHMENT WITH THE MORGAN INDEX

One possibility to address both issues simultaneously is to increase the specificity of labels, for example by including contextual information about the vertices in their labels. This has two important consequences. First, as the label specificity increases, the number of common label paths between graphs automatically decreases, which shortens the computation time. Second, this is likely to increase the relevance of the features used to compare graphs, as paths are replaced by paths labeled with their environment.

For the kind of applications we focus on in this paper—classification of chemical compounds—it seems natural to consider the chemical environment of atoms. For instance it makes sense to distinguish between atoms with similar labels but that belong to different functional groups. As a first attempt to define such a local environment, we propose to introduce information related to the topological environment of the vertices in the labeling function of the graphs.

To do so, we compute for each vertex of the graph an index called the *Morgan index*,[15] that is defined by a simple

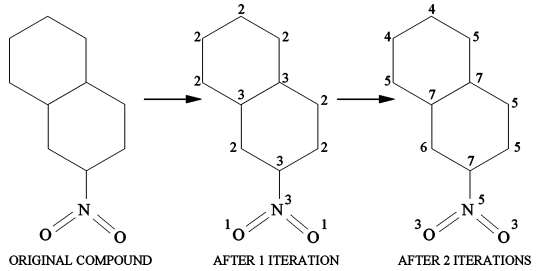

Figure 5. Morgan index process.

iterative procedure. Initially, the Morgan indices are equal to 1 for every vertex. Then, at each iteration, the Morgan index of each vertex is defined as the sum of the Morgan indices of its adjacent vertices. Mathematically, if we let M_i be the vector of the Morgan indices computed at the ith iteration, this reads $M_0 = \mathbf{1}$ and $M_{n+1} = A_{dj}M_n$, where A_{dj} is the graph adjacency matrix and $\mathbf{1}$ the unity vector. This process is illustrated in Figure 5. The Morgan index was initially developed to determine canonical representations of molecules and is considered a good and fast solution to detect graph isomorphism.

Note that the Morgan index associated with a particular vertex after n iterations actually counts the number of paths of length n that start in that vertex and end somewhere in the graph. This vertex descriptor has already been studied in chemical graph theory and is known as the *atomic length-n walk-count* descriptor in the literature.[16]

Finally, given the Morgan indices after n iterations, we propose to augment the label of a vertex by its value, before computing the marginalized graph kernel. This results in a family of kernels $(K_n)_{n\geq 0}$, indexed by the number of iterations for the Morgan index computation.

When the number of iterations increases, the topological information vehiculated by the Morgan index becomes more and more specific to the graphs. Pairs of vertices having at the same time identical atom type and topological properties are therefore less and less likely to occur. This results in a systematic decrease of the computation time, because the number of nodes of the product graph automatically decreases, but, on the other hand, the similarity between molecules may be difficult to assess if their description becomes too specific. This suggests that the step of the Morgan process that performs the optimum trade off between the uniform and the molecular-specific descriptions of vertices needs to be found.

5. PREVENTING TOTTERS

A second avenue to modify the original graph kernel is to modify the probability (6). This probability is the distribution of a first-order Markov random walk along the edges of the graph, killed with some probability after each step. We propose to modify the random walk model to prevent "totters", that is, to avoid any path of the form $h = v_1, ..., v_n$ with $v_i = v_{i+2}$ for some i. The motivation here is that such excursions are likely to add noise to the representation of the graph. For example, the existence of a path with labels C−C−C might either indicate the presence of a succession of 3 C-labeled vertices in the graph or just a succession of 2 C-labeled vertices visited by a tottering random walk. By preventing totters, the second possibility disappears. Figure 6 illustrates this idea.

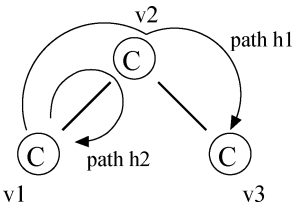

Figure 6. Illustration of the process of prevention of the tottering paths on a toy example. Paths h_1 and h_2 are both labeled as C−C−C, but path h_2 corresponds to a tottering path.

5.1. Modification of the Random Walk. A natural way to carry out this modification is to keep the general kernel definition (4) but modify the probability model (6) as follows

$$p(v_1 \dots v_n) = p_s(v_1)p_t(v_2|v_1)\prod_{i=3}^{n} p_t(v_i|v_{i-2}, v_{i-1}) \quad (7)$$

where $p_i(.)$, $p_t(.|.)$, and $p_t(.|.,.)$ satisfy for any $(u, v) \in V^2$:

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

Here we assume that $0 < p_q(v), p_q^{(0)}(v) \leq 1$ for each vertex v, $p_a(\cdot|v)$ is a probability on V that is only positive on the neighbors of v, and $p_a(\cdot|w, v)$ is a probability on V that is only positive on the neighbors of v different from w. This model is simply the distribution of a second-order Markov random walk, killed at each step with some probability $p_q(v)$ (or $p_q^{(0)}(v)$ after the first vertex, see section 6), which cannot follow excursions of the form $u \rightarrow v \rightarrow u$. In other words, only paths belonging to

$$H_0(G) = \{h = v_1 \dots v_n :\ v_i \neq v_{i+2}, i = 1, \dots, n-2\} \quad (8)$$

can have a positive probability under this model. Given this new random walk model, the function (4) is still a valid kernel, but the implementation described in section 3.3 cannot be used directly anymore.

5.2. Computation of the New Kernel. While paths have been previously defined as the succession of vertices they are made of, one can see a path as a starting vertex followed by a succession of connected edges. In such a definition, a pair of connected edges provides information about a triplet of vertices of the path: the starting vertex of the first edge, the vertex that connects them, and the ending vertex of the second edge. A second-order information about the succession of vertices therefore resumes to a first-order one based on the succession of edges. This suggests it should be possible to deal with second-order "vertex-based" random walks models by means of a first-order ones involving edges of the graphs.

Based on this consideration, we now derive an explicit way to perform the computation of the kernel (4) under the model (7). To do so, we introduce a graph transformation such that the second-order random walk (7) in the original graphs factorizes as a first-order Markov process (6) in the transformed ones.

More precisely, for a graph $G = (V, E)$, let the transformed graph $G' = (V', E')$ be defined by

$$V' = V \cup E$$

and

$$E' = \{(v,(v, t))|v \in V,(v, t) \in E\} \cup \{((u, v),(v, t))|(u, v),(v, t) \in E, u \neq t\} \quad (9)$$

The vertices of the transformed graph G' can therefore correspond either to edges or vertices of the original graph G. Among all paths $H(G')$ on G', let us consider the subset of paths that start on an arbitrary vertex in V, that is the set

$$H_1(G') = \{h' = v'_1 \dots v'_n \in H(G') : v'_1 \in V\} \quad (10)$$

Note that from the definition of the transformed graph edges, it is easy to check that any path $h' = v'_1 \dots v'_n \in H(G')$ starting with a vertex $v'_1 \in V$ must be made of edges: $v'_i \in E$, $i = 2, \dots, n$. This construction is illustrated in Figure 7.

We define the labeling function l' of the transformed graph G' as follows:

− for a node $v' \in V'$ the label is either $l'(v') = l(v')$ if $v' \in V$ or $l'(v') = l(v)$ if $v' = (u, v) \in E$.

− for an edge $e' = (v'_1, v'_2)$ between two vertices $v'_1 \in V \cup E$ and $v'_2 \in E$, the label is simply given by $l'(e') = l(v'_2)$.

This labeling is also illustrated in Figure 7.

Let us consider the map f: $H_0(G) \rightarrow (V')^*$ defined by

$$f(v_1 \dots v_n) = v'_1 \dots v'_n$$

with

$$\begin{cases} v'_1 = v_1 \in V, \\ v'_i = (v_{i-1}, v_i) \in E, \text{ for } i = 2, \dots, n \end{cases} \quad (11)$$

This definition gives rise to the following proposition, whose proof can be found in the Appendix.

Proposition 1. *f is a bijection between $H_0(G)$ and $H_1(G')$ and for any path $h \in H_0(G)$ we have $l(h) = l'(f(h))$.*

Finally, let the functional p': $(V')^* \rightarrow$ R be derived from (7) by

$$p'(v'_1 \dots v'_n) = p'_s(v'_1)\prod_{i=2}^{n} p'_t(v'_i|v'_{i-1}) \quad (12)$$

with

$$p'_s(v') = \begin{cases} p_s(v') & \text{if } v' \in V, \\ 0 & \text{if } v' \in E \end{cases}$$

and

$$p'_t(v'|u') = \begin{cases} p_t(v|\, u') & \text{if } u' \in V \\ & \text{and } v' = (u', v) \in E, \\ p_t(v|\, t, u) & \text{if } u' = (t, u) \in E \\ & \text{and } v' = (u, v) \in E \end{cases}$$

Note that only paths belonging to $H_1(G')$ have a positive value under p'.

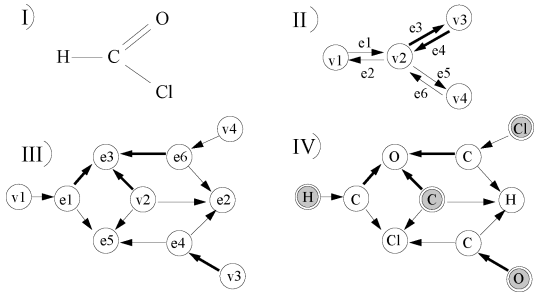

Figure 7. The graph transformation. (I) The original molecule. (II) The corresponding graph $G = (V, E)$. (III) The transformed graph. (IV) The labels on the transformed graph. Note that different widths stand for different edges labels, and gray nodes are the nodes belonging to V.

Based on the definitions of f and p', we can state the following result, whose proof is postponed in the Appendix.

Theorem 1. *Under the bijection f: $H_0(G) \rightarrow H_1(G')$ defined in (11), for any path $h \in H_0(G)$ we have $p(h) = p'(f(h))$.*

We have defined a graph transformation showing a one to one correspondence between a particular subset of the paths of the transformed graph (the set $H_1(G')$) and the set of nontottering paths of the original graph (the set $H_0(G)$). Moreover we introduced a first-order Markov functional (12) on the transformed graph, positive only on this particular subset of paths $H_1(G')$, that corresponds to the second-order probability distribution (7) that was previously defined on the original graph to prevent totters. We can therefore immediately deduce the following.

Corollary 1. *For any two graphs G_1 and G_2, the kernel (4) can be expressed in terms of the transformed graphs G_1' and G_2' by*

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

This shows that computing the $K(G_1, G_2)$ under the second-order Markov model (7) for the random walk is equivalent to computing a kernel between the transformed graphs G'_1 and G'_2 under a first-order Markov random walk (12). This can therefore be carried out using the computation scheme described in section 3.3, at the expense of an increased complexity.

More precisely, consider a graph $G = (V, E)$, whose maximum node degree is d, and the graph $G' = (V', E')$ resulting from its transformation. By definition of V', $|V'| = |V| + |E|$. Moreover, from the two steps appearing in the definition of E', we also have $|E'| \leq |E| + (d - 1)|E| = d|E|$. Finally, it is easy to check that the node of maximum degree in the transformed graph is precisely the node of maximum degree in the original graph. This is due to the fact that the nodes of G' corresponding to nodes of G have the same degree that their homologues and that the nodes in G' corresponding to edges of G have a degree equal to those of the nodes being reached by the edges in G minus one (to prevent tottering). From section 3.3, the complexity of the kernel between two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ writes as $O(|E_1||E_2|\sum_{i=1}^{N-1}\min((d_1d_2)^i,|V_1||V_2|))$. As a result, if we now consider the graphs $G'_1 = (V'_1, E'_1)$ and $G'_2 = (V'_2, E'_2)$ obtained by transforming G_1 and G_2, this complexity is of order $O(d_1d_2|E_1||E_2|\sum_{i=1}^{N-1}\min((d_1d_2)^i,(|V_1| + |E_1|)(|V_2| + |E_2|)))$.

6. EXPERIMENTS

In our experiments, we adopted the parametrization proposed in ref 2. A single parameter p_q is used to define the first-order random walk model as follows, for any $u, v \in V$:

$$p_0(v) = 1/|V|$$

$$p_q(v) = p_q < 1$$

$$p_a(v|u) = \begin{cases} 1/d(u) \text{ if } (u, v) \in E \\ 0 \text{ otherwise} \end{cases}$$

This way, the emission probability distribution is chosen to be uniform over the set of edges, a constant ending probability is introduced for every node of the graph, and transition probabilities are made uniform over the set of neighbors of the nodes. When we do not have prior knowledge about the data, this seems to be a natural way to parametrize the model.

To filter tottering paths, we adapt this model to define in a similar way the second-order random walk model (7) introduced in section 5.1. The main differences between the two models concern the functional $p_q(v)$, $p_q^{(0)}(v)$, and $p_a(u|w, v)$. Indeed, in the first step of the random walk process, the walk is not subject to tottering, and we can consider the same first-order transition functional $p_a(u|v)$ and ending probabilities $p_q^{(0)}(v)$. In the following steps however, we may have to set the ending probability $p_q(v)$ to one, to explicitly kill random walks when reaching a node with only one neighbor, because in this case, the only possibility to continue the walk is to "totter" to the previous node. The definition of $p_a(u|w, v)$ also reflects the modification required to prevent totters: the number of possible edges to follow from a node v is only $d(v) - 1$, because one edge has already been used to reach v.

This leads to the following second-order Markov model, for any $u, v, w \in V$:

$$p_0(v) = 1/|V|$$

$$p_a(u|v) = \begin{cases} 1/d(v) \text{ if } (v, u) \in E \\ 0 \text{ otherwise} \end{cases}$$

$$p_a(u|\, w, v) = \begin{cases} 1/(d(u) - 1) \text{ if } (v, u) \in E \text{ and } u \neq w \\ 0 \text{ otherwise} \end{cases}$$

$$p_q^{(0)}(v) = p_q$$

$$p_q(v) = \begin{cases} 1 \text{ if } d(v) = 1 \\ p_q \text{ otherwise} \end{cases}$$

The classification experiments described below were carried out with a support vector machine based on the different kernel tested. Each kernel was implemented in C++, and we used the free and publicly available GIST (http://microarray.cpmc.columbia.edu/gist) software to perform SVM classification. No optimization of the parameters required by GIST was carried out. The only option specified was the *-radial* option, which converts the kernel into a radial basis function, a standard way to normalize the data. Two data sets of chemical compounds were used. Both gather results of mutagenicity assays, and while the first one[17] is a standard benchmark for evaluating chemical compounds classification, the second one[1] was introduced more recently.

Generally speaking, focusing only on the global accuracy is hazardous to analyze classification results and may lead to wrong conclusions. This is particularly true when the data set is *unbalanced*, which means that one of the classes is overrepresented compared to the other. A safer approach is to describe the classifier using *ROC* analysis[18] and consider the *sensitivity/specificity* rates. Sensitivity is defined as the ratio between the correctly classified positive data (the *true positives*) and the total number of positive data (the sum of *true positive* and *false negative* data). It therefore accounts for the proportion of positive data that will be retrieved by the algorithm. Similarly, specificity accounts for the proportion of negative data that the algorithm will correctly find (the ratio between *true negative* data and the whole negative data, i.e., the *true negative* plus the *false positive*). Clearly, a good classifier will show a high sensitivity together with a good specificity.

Moreover, the SVM algorithm actually computes a score to make the predictions. If this score is positive, the prediction is $+1$, otherwise it is -1. This fact makes it possible to draw the evolution of the true positive rate versus the false positive rate in a curve denoted as the *ROC curve*. A good indicator can be derived from this curve: the *AUC*, Area Under the (ROC) Curve. The AUC of an ideal classifier would be 1 (the positive data would be the first to be recognized as positive according to their scores), while for a random classifier it would be 0.5.

6.1. First Data Set. This data set contains 230 chemical compounds (aromatic and hetero-aromatic nitro compounds) tested for mutagenicity on *Salmonella typhimurium*. A SAR analysis on this data set was first conducted by ref 17, which identified two subsets of the data: 188 compounds considered to be amenable to regression and 42 compounds that could not easily be fitted by regression. In this study we mainly focus on the first set of 188 compounds. These compounds can be 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. Each chemical compound is represented as a graph with atoms as vertices and covalent bonds as edges. This subset of 188 compounds was already used in the original paper ref 2, and in a similar way, kernels are evaluated here by their leave-one-out error. AUC will be our quality criterion.

Table 1 shows the results we can get with the kernel as it is formulated in ref 2. They will be our reference results to evaluate the impact of the proposed extensions. This table shows a consistent increase in the AUC when the parameter p_q decreases, that is to say when the kernel favors long paths.

Figure 8 shows the effect of removing the tottering paths with the original kernel formulation for distinct values of p_q. The curve reveals that the relationship between small p_q and high AUC observed with the original formulation of the kernel does not rigorously hold any longer when tottering paths are filtered. Indeed, we can find in this case an optimum value of p_q around 0.1. Above this value, we can notice on one hand a small but consistent increase on classification when tottering paths are removed, and on the other hand that the effect of this extension becomes smaller when p_q

Table 1. Classification of the First Data Set, with the Original Formulation of the Kernel Function for Different Values of the Parameter p_q

p_q	accuracy	sensitivity	specificity	AUC
0.01	89.4	88.8	90.4	94.4
0.05	89.4	88.8	90.4	94.2
0.1	89.9	89.6	90.4	94.0
0.2	90.4	90.4	90.4	93.8
0.3	90.4	90.4	90.4	93.6
0.4	88.8	88.0	90.4	93.4
0.5	88.8	88.0	90.4	93.0
0.6	88.3	87.2	90.4	92.7
0.7	87.2	85.6	90.4	92.2
0.8	86.7	84.8	90.4	91.2
0.9	83.5	82.4	85.7	89.2

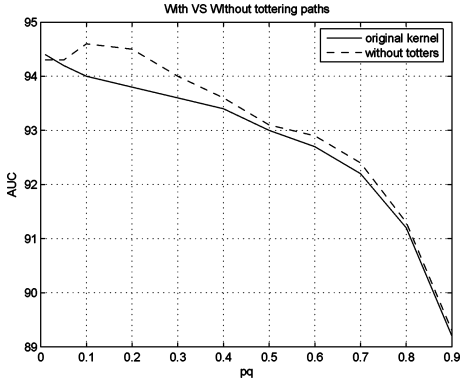

Figure 8. AUC for distinct values of p_q, with and without filtering the tottering paths, first data set.

Table 2. AUC for the 10 First Morgan Indices and Different Ending Probabilities, First Data Set

p_q	0.01	0.05	0.1	0.3	0.5	0.7	0.9
MI = 0	94.4	94.2	94.0	93.6	93.0	92.2	89.2
MI = 1	94.4	94.2	93.8	93.2	92.7	92.2	92.0
MI = 2	**96.1**	**96.0**	**95.9**	**95.2**	94.3	93.6	93.1
MI = 3	94.6	94.7	94.7	94.9	**94.8**	**94.8**	**94.6**
MI = 4	93.3	93.3	93.2	93.3	93.1	93.0	92.6
MI = 5	92.3	92.4	92.5	92.8	93.2	93.4	93.5
MI = 6	91.6	91.8	92.0	92.6	92.8	92.9	92.8
MI = 7	90.2	90.1	90.1	90.1	90.1	90.1	90.2
MI = 8	86.9	87.1	87.3	87.7	88.1	88.3	88.4
MI = 9	80.5	80.8	81.5	81.6	81.7	81.9	81.7
MI = 10	72.8	72.8	73.7	76.2	77.1	77.6	77.9

increases. This is probably due to the fact that when p_q increases, the paths taken into account by the kernel tend to become shorter and are therefore less likely to totter. When $p_q \leq 0.1$, the AUC decreases and becomes smaller than the one obtained with the original formulation for $p_q = 0.001$.

Table 2 shows the AUC results for distinct values of p_q combined with the introduction of Morgan indices. It reveals that the introduction of Morgan indices can always increase the classification results, and interestingly, the optimal index to be used depends on the value of p_q: it is generally smaller for little values of p_q. This reflects the fact that we have to add more specificity in the atoms labels for large p_q, since only paths involving a few atoms will be taken into account. However, no prior rule can define a precise relation between the Morgan index and the parameter p_q.

Table 3 shows the AUC results when tottering paths have been filtered. The classification results show the same behavior of the kernel with respect to p_q and the Morgan indices when the tottering paths have been filtered. The

Table 3. AUC for the 10 First Morgan Indices and Different Ending Probabilities, When Tottering Paths Have Been Filtered, First Data Set

p_q	0.01	0.05	0.1	0.3	0.5	0.7	0.9
MI = 0	94.3	94.3	94.6	94.0	93.1	92.4	89.3
MI = 1	94.0	95.1	94.2	94.0	93.2	92.3	92.0
MI = 2	**94.9**	**94.9**	**95.2**	**95.4**	94.5	93.6	93.1
MI = 3	93.3	93.4	93.6	94.7	**94.8**	**94.8**	**94.6**
MI = 4	92.5	92.4	92.7	93.3	93.2	92.9	92.6
MI = 5	90.5	90.7	91.0	92.6	93.1	93.4	93.4
MI = 6	88.2	88.5	89.9	92.1	92.8	92.9	92.8
MI = 7	84.7	86.4	88.5	90.2	90.1	90.2	90.2
MI = 8	69.2	73.9	81.4	87.4	88.0	88.2	88.3
MI = 9	57.1	60.6	70.7	81.3	81.8	81.7	81.7
MI = 10	49.2	48.8	52.7	73.8	76.8	78.2	78.3

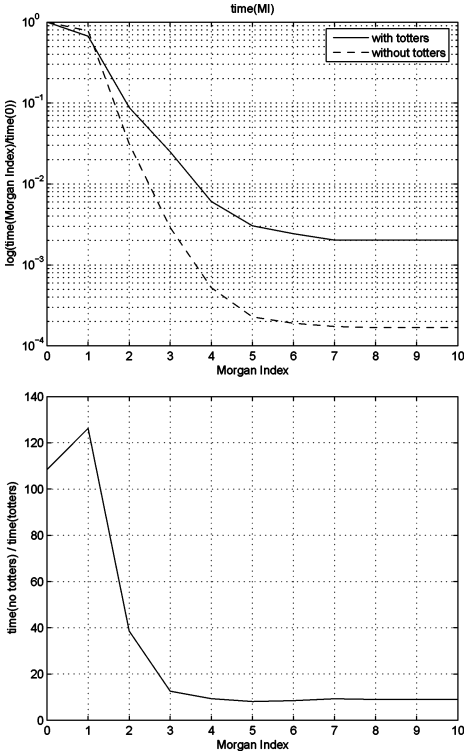

Figure 9. Top: Time needed to compute the kernel for the 10 first iterations of the Morgan process. Bottom: Ratio between computation times with or without filtering the totters for the 10 first iterations of the Morgan Process, first data set.

values obtained are actually sensibly equal, which suggests that filtering the tottering paths provides little additional information. We can however notice that performances are globally reduced when p_q becomes smaller. Tottering between atoms made specific to the molecule therefore accounts for graphs similarity when long paths are taken into account.

Finally, results about computation times are presented in Figure 9. The top curve plots the evolution of the time needed to compute the kernels when different Morgan indices were introduced. More precisely it plots the ratio between the time needed for a given iteration of the Morgan process and the time initially required. Note that the *y*-axis is in log-scale, so that we can notice a drastic decrease in the computational cost. For example, at the third iteration of the process, computation time is reduced by a factor around 40 when tottering paths have not been filtered. Remember that when Morgan indices are introduced, atoms are made more specific to the molecule they belong, and as a consequence fewer atoms are apariated in the product graph, which makes the

Table 4: Accuracy Results Obtained for the 10-Fold Cross-Validation of the Mutag Data Set[a]

Lin.Reg	*DT*	*NN*	*Progol1*	*Progol2*	*Sebag*	*Kramer*	graph kernels
89.3%	88.3%	89.4%	81.4%	87.8%	93.3%	95.7%	91.2%

[a] *Lin.Reg* (linear regression), *DT* (decision tree), *NN* (neural network), and *Progol1/2* (inductive logic programming): ref 19; *Sebag*: ref 21; *Kramer*: ref 20.

Table 5: Accuracy Results Obtained for the Leave-One-Out Classification of the "Unfriendly Part" of the Mutag Data Set[a]

Lin.Reg	*Lin.Reg+*	*DT*	*NN*	*Progol1*	*Progol2*	graph kernels
66.7%	71.8%	83.3%	69.0%	85.7%	83.3%	88.1%

[a] *Lin.Reg* (Linear Regression), *DT* (Decision Tree), *NN* (Neural Network), and *Progol1/2* (Inductive Logic Programming): ref 19.

matrix inversion cheaper. This effect is even stronger when filtering the totters. The bottom curve presents the impact of totter removal on the computation times. The curve shows the ratio between the computation times of the original formulation and the totters filtering. This ratio becomes smaller with the Morgan process, but while the computation without totters was initially more than a hundred time longer than with totters, it remains at least 10 times longer with high Morgan indices.

As a comparison, Table 4 gathers 10-fold cross-validated accuracy results already obtained for the classification of this set of 188 compounds. These methods can be split in three categories: those relying on global molecular properties (*Lin.Reg, NN, DT*) [For instance, the molecular hydrophobicity (logP), the energy of the lowest unoccupied molecular orbital (LUMO), and two additional binary descriptors coding for the presence of particular features in the molecule.], those considering the structure of the molecules as a set of atoms and connecting bonds (*Progol1*), and those involving the two representations (*Progol2, Sebag, Kramer*). The best 10-fold cross-validated accuracy corresponding to our previous experiments is 91.2%. As we can notice from Table 4, this result is better than those based on one of the two molecular representations, but it is below those obtained by methods that combine both representations. This table reveals that there is a significant gap between the *Progol1* and *Progol2* results, which are obtained using the same algorithm when the global descriptors are considered or not as an additional source of information. This suggests that the information contained in the two descriptions may be complementary. Moreover, the best result reported (ref 20) deals with the structure of the molecule via a fragment-based characterization, which, as we already mentioned, shows some similarities with the graph kernel approach. It seems therefore reasonable to draw the hypothesis that the results of the graph kernel approach may be improved if such a combination of information about the molecules is used.

Little work has been carried out on the 42 compounds that constitute the "nonregression friendly part" of the data set. To our knowledge, the only results were published in ref 19 and are summarized in Table 5. The fundamental difference with the "friendly part" of the data set lies in the fact that here the best result was obtained using only the 2D structure of the compounds (with the *Progol1* method). Using our graph kernels, we can reach 88.1% of the correct classification using a similar leave-one-out procedure. Outperforming all the results from Table 5, this result shows that the graph kernel approach is indeed efficient when the relevant information is to be sought in the structure of the molecules.

Independently of our work, related graph kernels for chemoinformatics applications were recently introduced.[22] Their formulation is driven by the usual molecular fingerprinting process, and several kernel functions are proposed based on variations of the Tanimoto coefficient. Different ways of fingerprinting the molecules are considered, and, in particular, some experiments compare standard hashed fingerprints (such as *Daylight* fingerprints) with exhaustive fingerprints, for paths up to a given length (which was set to 10). In exhaustive fingerprints, a dimension is introduced for every possible path, which is closely linked to the description of molecules related to the graph kernels introduced here. Although the performances of these different configurations are similar, this study tends to reveal that the hashing process leads to a decrease in the classification accuracy. More precisely, the best result for exhaustive fingerprints reaches 87.8% of correct leave-one-out classification, while it is 87.2% when the fingerprints are hashed. Using our graph kernels, we can reach a leave-one-out accuracy of 91%, which indicates that the marginalized graph kernels approach may compare favorably to classical hashed-fingerprints. Note however that results in ref 22 were not obtained using SVM but using the Voted Perceptron algorithm, an algorithm known to provide comparable results, and that further refinements of their kernels lead to an optimal accuracy of 91.5%.

6.2. Second Data Set. The second database considered was recently introduced in ref 1. It consists of 684 compounds classified as mutagens or nonmutagens according to a test known as the *Salmonella*/microsome assay. The classes are well balanced with 341 mutagens compounds for 343 nonmutagens ones.

Although the biological property to be predicted is the same as the one of the previous section, the two data sets are fundamentally different. While ref 17 focused on a particular family of molecules (aromatic and heteroaromatic nitro compounds), this data set 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 for regular patterns within a set of structurally different molecules. As stated in ref 1, toxicity is a very complex and multifactor mechanism, so that diverse data sets need to be considered in order to be able to predict mutagenicity in real-world applications. Finally, note that this data set is public, and a further description can be found in ref 1.

We applied different graph kernels to this data set in order to compare our approach to the results presented in ref 1. Several machine learning algorithms have been used in that paper (namely SVM, decision trees, and rule learning), based on a molecular-fragment characterization of molecules. In their method, a set of substructures occurring frequently in mutagenic compounds but seldomly in nonmutagens ones is defined, and molecules are represented by bit-strings

Table 6. Classification Results for the Second Mutagenicity Data Set, with the Tottering Paths

MI	p_q	training accuracy	test accuracy	test sensitivity	test specificity
0	0.1	94.47	76.02	73.34	78.67
	0.2	93.85	75.14	72.65	77.77
	0.3	92.65	74.20	71.14	77.26
	0.4	92.06	73.36	70.76	75.90
	0.5	92.77	73.32	71.61	75.03
1	0.1	96.03	79.01	77.06	80.97
	0.2	96.00	79.32	77.55	81.08
	0.3	96.08	78.70	77.65	79.73
	0.4	96.11	78.40	76.45	80.30
	0.5	96.01	77.17	76.00	78.20
2	0.1	97.20	78.09	76.95	79.25
	0.2	97.13	77.89	77.33	78.47
	0.3	97.02	78.70	78.43	78.93
	0.4	96.91	78.34	78.11	78.56
	0.5	96.82	78.20	77.71	78.70
3	0.1	97.42	75.32	73.62	76.95
	0.2	97.14	75.00	73.31	76.67
	0.3	96.92	75.88	74.55	77.21
	0.4	96.71	75.00	74.04	76.12
	0.5	96.57	75.27	73.81	76.69
4	0.1	98.03	71.45	68.53	74.41
	0.2	97.55	71.80	69.47	74.20
	0.3	97.29	72.10	69.77	74.50
	0.4	97.10	71.52	69.79	73.21
	0.5	96.87	72.18	69.86	74.53
5	0.1	97.65	66.76	64.32	69.30
	0.2	97.24	67.73	65.56	69.80
	0.3	96.94	66.99	64.65	69.27
	0.4	96.58	66.85	65.11	68.72
	0.5	96.38	66.73	64.62	68.86

Table 7. Classification Results for the Second Mutagenicity Data Set, When Tottering Paths Were Removed

MI	p_q	training accuracy	test accuracy	test sensitivity	test specificity
0	0.1	95.60	77.37	76.59	78.15
	0.2	95.11	75.46	72.38	78.63
	0.3	94.48	74.93	71.95	77.82
	0.4	93.65	75.13	71.81	78.44
	0.5	93.22	74.18	71.58	76.77
1	0.1	97.22	77.87	74.90	80.90
	0.2	96.43	78.89	76.65	81.16
	0.3	96.21	79.06	77.28	81.00
	0.4	96.20	78.54	77.25	79.84
	0.5	96.22	78.47	76.65	80.03
2	0.1	97.72	76.59	72.03	81.21
	0.2	97.52	77.95	75.35	80.47
	0.3	97.20	78.04	77.01	79.08
	0.4	97.04	78.28	77.77	78.80
	0.5	96.90	78.03	77.54	78.53
3	0.1	97.97	75.18	69.82	80.54
	0.2	97.79	75.87	72.99	78.69
	0.3	97.53	75.48	74.36	76.63
	0.4	97.06	75.54	73.98	77.10
	0.5	96.78	75.62	75.22	76.00

indicating the presence or absence of these substructures. Tables 6 and 7 gather results on this data set using the original and totters-filtering versions of the kernel, for several values of p_q and different iterations of the Morgan process. Following ref 1, we performed classifications by a 10-fold cross-validation procedure, and performances are evaluated according to the *accuracy*, *sensitivity*, and *specificity* values of the models.

A quick inspection of these two tables reveals that, similarly to the original paper, the test sensitivity and specificity rates are always similar. This means that the different models obtained can be used to predict either mutagenicity or nonmutagenicity, with a similar degree of confidence. From this consideration, we base our analysis of the results on the global test accuracy of the models.

Table 8. Diversity Values for Different Subsets of the Data, Computed from the Kernels Obtained Using the Tottering Paths, the First Iteration of the Morgan Process, and Two Different Values of p_q[a]

	All	*Pos.*	*Neg.*	*Correct*	*Correct & Pos.*	*Correct & Neg.*
$p_q = 0.1$	0.862	0.849	0.864	0.854	0.826	0.860
$p_q = 0.2$	0.868	0.856	0.870	0.860	0.834	0.864

[a] *All*: whole data set; *Pos.*: positive compounds; *Neg.*: negative compounds; *Correct*: correctly classified compounds; *Correct & Pos.*: correctly classifed positive compounds; *Correct & Neg.*: correctly classified negative compounds.

Several conclusions can be drawn from these tables. First, when no Morgan indices are introduced (i.e. $MI = 0$), we can note from both tables that test accuracy systematically increases when the parameter p_q decreases. This is consistent with the experiments carried out with the previous data set and suggests that it is worth considering long paths. Moreover, when we compare the two tables, we note that filtering the totters systematically enhances the classification, which comforts the intuition that this kind of paths adds noise to the description of the molecules.

Concerning the introduction of the Morgan indices, we can note from the two tables that, for any value of p_q considered, classification is improved for the first iteration of the process, after what it systematically decreases. This means that although the first step of the Morgan process could improve the expressive power of the kernel, the information introduced into the description of the molecule becomes too specific from the second iteration. Interestingly, we can also notice that for a given index of the Morgan process, the optimal value of p_q is not the smallest one any longer.

Filtering the totters after the introduction of the Morgan indices have a somehow ambiguous effect. It does not show a consistent trend with respect to the parameter p_q. However, the two optimal results show a 79.32% and 79.06% test accuracy, so that results are globally similar.

Finally, note that the computation times needed to compute the different kernels follow the same behavior as the results presented in the previous subsection.

Reference 1 pointed out the need to consider structurally diverse data sets such as this one in order to be able to model multifactor mechanisms such as toxicity. Although the classification accuracy provides a general measure of the effectiveness of the algorithm, it is of limited help to quantify its ability to handle the diversity of the data set. For instance, a situation where the subset of correctly classified data shows a smaller diversity compared to the global data set actually makes sense. This situation means that the algorithm is only efficient in a particular subspace of the chemical space defined by the whole data set, which is actually likely to occur, and reveals that the method fails to handle the diversity of the data set. Analyzing the diversity of the classification results is therefore useful to give fair conclusions about the method. Table 8 shows the values of a diversity criterion

measured on the whole data set and on several subsets: the subsets of positive compounds, negative compounds, correctly classified compounds, positive compounds that were correctly classified, and negative compounds that were correctly classified. These values were computed for two kernels that correspond to optimal results in the previous tables: those obtained using the tottering paths, the first iteration of the Morgan process, and values of p_q of 0.1 and 0.2. [The kernels used actually correspond to the kernels used for the classification, i.e., the kernels obtained after applying the *-radial* option of the *GIST* software to the original kernels.]

To evaluate the diversity of a subset, we use the average distance of the points of this subset to their center of mass, in the feature-space associated with the kernel. Recall from section 2 that a kernel function corresponds to a dot-product between the data mapped to a vector space $\mathcal{F}$(the so-called *feature-space*): $k(x_1, x_2) = \langle\phi(x_1),\phi(x_2)\rangle$. A distance in the feature-space is therefore implicitly defined by the kernel function: $d_{\mathcal{F}}(x_1, x_2) = ||\phi(x_1) - \phi(x_2)||^2 = k(x_1, x_1) + k(x_2, x_2) - 2k(x_1, x_2)$. Our diversity criterion, for the subset S (of cardinality n_S), therefore writes as $D(S) = 1/n_S\sum_{i\in S}d_{\mathcal{F}}(x_i, M)$, where M is the center of mass of S, i.e., $M = 1/n_S\sum_{i\in S}\phi(x_i)$, which leads to $D(S) = 1/n_S\sum_{i\in S}k(x_i, x_i) - 1/n_S^2\sum_{i,j\in S}k(x_i, x_j)$.

Table 8 reveals that the diversities of these different subsets are very similar. Two main conclusions can be drawn from this observation. First, the fact that the diversity of the whole data set equals that of the positive and negative subsets reveals that the supports of these subsets largely overlap and indicates that the classification problem is not trivial. Indeed, in a particularly simple configuration the diversities of the positive and negative subsets would be significantly smaller than that of the data set as a whole: the two classes of compounds would be clearly separated in the chemical space. Second, the fact that all these values are similar shows that this algorithm is able to correctly classify data regardless of the class they belong to nor their location in the chemical space. This accounts for the fact that the method is indeed able to handle noncongeneric data sets.

Finally, we can note that Tables 6 and 7 compare quite favorably to the results presented in ref 1. Many configurations have been tested in ref 1, and the best model reported has an accuracy of 78.5%. With an optimal accuracy of 76%, the original graph kernel with SVM shows a slightly smaller performance, but the extensions introduced here could raise this figure up to more than 79%. Based on our pair of extensions, we have therefore been able to propose models with state-of-the-art performance and even models performing slightly better. We can however note a slight difference between the two family of models: models from ref 1 tend to have a higher sensitivity, while ours show a better specificity. This means that the models from ref 1 will correctly classify a larger fraction of the positive data, but this comes at the expense of a larger false positive rate, whereas our models may miss sensibly more true positive data, but the confidence in positive predictions is higher.

7. CONCLUSION

Based on a recently introduced family of graph kernels, we validated in this paper the graph kernel approach for SAR analysis. Experiments revealed in a consistent way that optimal results are obtained when long paths are considered, and this insight is worth solving the problem of model parametrization.

We introduced two extensions to the general formulation and showed they can actually improve the SAR models in terms of accuracy of the predictions and/or computation times. These two extensions are formulated as preprocessing steps of the algorithm and are therefore completely modular. Moreover, they are based on general graph considerations, and we believe they can be useful in other problems.

The fundamental difference between this approach and other SAR algorithms lies in the fact that the step of feature selection inherent to all other methods is avoided here. In this sense, these kernels provide a kind of universal way to compare molecules. Together with the panel of kernel methods algorithms, this family of graph kernels could be used straightaway to solve different SAR problems, such as clustering or regression tasks for instance, which otherwise typically involve multiple feature selection tasks.

On top of that, since it deals with every molecular fragment of the molecules, this model can benefit from structural patterns responsible for activity that have not been discovered yet and are therefore not included in the set of traditional descriptors. This property however comes at the expense of the interpretability of the model, which has a great interest in medicinal chemistry. Indeed, an interpretable model can give clues to explain the causes of activity or nonactivity and therefore provide chemists with worthy feedback to carry out molecular optimization in a rational way. The next challenge to these graph kernels for chemoinformatics is to be able to extract information from the infinite dimensional feature-space associated with the kernels and to formalize it in terms of chemical knowledge.

ACKNOWLEDGMENT

This work was supported by a French-Japanese *SAKURA* grant. Jean-Luc Perret is partly supported by a grant from the *Swiss National Science Foundation*.

8. APPENDIX

1. Proof of Proposition 1. For any path $h = v_1 \dots v_n \in H_0(G)$, let $f(h) = v'_1 \dots v'_n$ defined by (11). By definition (9), $(v'_1, v'_2) = (v_1,(v_1, v_2)) \in E'$, and $(v'_i, v'_{i+1}) = ((v_{i-1}, v_i),(v_i, v_{i+1})) \in E'$ because $v_{i+1} \neq v_{i-1}$ for $i > 1$. Hence $f(h)$ is a path in G'. Moreover $v'_1 \in V$ and $v'_i \in E$ by (11), hence $f(h) \in H_1(G')$ by (10).

Conversely, for any $h' = v'_1 \dots v'_n \in H_1(G')$, we have $v'_1 = v_1 \in V$ and by easy induction using the definition of edges (9), $v'_i = (v_{i-1}, v_i) \in E$ with $v_{i-1} \neq v_{i+1}$. Hence $h' = f(h)$ with $h = v_1 \dots v_n \in H_0(G)$, therefore f is surjective. By definition of f (11), it is also clear that $f(h) = f(h') \Rightarrow h = h'$. f is therefore a bijection from $H_0(G)$ onto $H_1(G')$.

Moreover, by definition of the labeling l' on G', we obtain for any $h = v_1, \dots, v_n \in H_0(G)$:

$$l'(f(h)) = l'(v_1,(v_1, v_2), \dots ,(v_{n-1}, v_n))$$

$$= l(v_1)l(v_2) \dots l(v_n)$$

$$= l(h)$$

2. Proof of Theorem 1. From the definition of p', for any $h = v_1, ..., v_n \in H_0(G)$, we obtain:

$$p'(f(h)) = p'(v_1,(v_1, v_2), ..., (v_{n-1}, v_n))$$

$$= p'_s(v_1)p'_t((v_1, v_2)|v_1)$$

$$\prod_{i=3}^{n} p'_t((v_{i-1}, v_i)|(v_{i-2}, v_{i-1}))$$

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

$$= p(h)$$

Note Added after ASAP Publication. This paper was released ASAP on May 27, 2005. $v'_1 \in E$ should be $v'_i \in E$ in the sentence following equation 10. The correct version was posted on June 6, 2005.

REFERENCES AND NOTES

(1) Helma, C.; Kramer T.; Kramer S.; DeRaedt L. Data Mining and Machine Learning Techniques for the Identification of Mutagenicity Inducing Substructures and Structure−Activity Relationships of Noncongeneric Compounds. *J. Chem. Inf. Comput. Sci.* **2004**, *44*, 1402−1411.

(2) Kashima, H.; Tsuda, K.; Inokuchi, A. Marginalized Kernels between Labeled Graphs. In *Proceedings of the Twentieth International Conference on Machine Learning*; Fawcett T., Mishra, N., Eds.; AAAI Press: 2003; pp 321−328.

(3) Gärtner, T.; Flach, P.; Wrobel, S. On Graph Kernels: Hardness Results and Efficient Alternatives. In *Computational Learning Theory and Kernel Machines*; Schölkopf, B., Warmuth, M. K., Eds.; Springer-Verlag: Heildeberg, 2003; pp 129−143.

(4) Boser, B. E.; Guyon, I. M.; Vapnik, V. N. A Training Algorithm for Optimal Margin Classifiers. In *Proceedings of the 5th Annual ACM Workshop on Computational Learning Theory*; ACM Press: 1992; pp 144−152.

(5) Vapnik, V. N. *Statistical Learning Theory*; Wiley: New York, 1998.

(6) Schölkopf, B.; Smola, A. J. *Learning with Kernels*; MIT Press: Cambridge, MA, 2002.

(7) Warmuth, K. W.; Liao, J.; Rätsch, G.; Mathieson, M.; Putta, S.; Lemmen, C. Active Learning with Support Vector Machines in the Drug Discovery Process. *J. Chem. Inf. Comput. Sci.* **2003**, *43*(2), 667−673.

(8) Burbidge, R.; Trotter, M.; Buxton, B.; Holden, S. Drug Design by Machine Learning: Support Vector Machines for Pharmaceutical Data Analysis. *Comput. Chem.* **2001**, *26*(1), 5−14.

(9) Mahé, P.; Ueda, N.; Akutsu, T.; Perret J. L.; Vert J. P. Extensions of Marginalized Graph Kernels. In *Proceedings of the Twenty-First International Conference on Machine Learning*; Greiner, R., Schuurmans, D., Eds.; ACM Press: 2004; pp 552−559.

(10) Gärtner, T. A Survey of Kernels for Structured Data. In *ACM SIGKDD Explor. Newsl.* **2003**, *5*(1), 49−58.

(11) Gärtner, T. Exponential and Geometric Kernels for Graphs. In *NIPS Workshop on Unreal Data: Principles of Modeling Nonvectorial Data*; 2002.

(12) Tsuda, K.; Kin, T.; Asai, K. Marginalized Kernels for Biological Sequences. *Bioinformatics* **2002**, *18* (suppl. 1), 268−275.

(13) Gasteiger, J.; Engel, T. *Chemoinformatics: a Textbook*; Wiley-VCH: 2003.

(14) Smola, A.; Kondor, R. Kernels and Regularization on Graphs. In *Computational Learning Theory and Kernel Machines*; Schölkopf, B., Warmuth, M. K., Eds.; Springer-Verlag: Heidelberg, 2003; pp 144−158.

(15) Morgan, H. L. The Generation of Unique Machine Description for Chemical Structures − A Technique Developed at Chemical Abstracts Service. *J. Chem. Doc.* **1965**, *5*, 107−113.

(16) Rücker, G.; Rücker, C. Counts of All Walks as Atomic and Molecular Descriptors. *J. Chem. Inf. Comput. Sci.* **1993**, *33*, 683−695.

(17) Debnath, A. K.; Lopez de Compadre, R. L.; Debnath, G.; Schusterman, A. J.; Hansch, C. Structure−Activity Relationship of Mutagenic Aromatic and Heteroaromatic Nitro Compounds. Correlation with Molecular Orbital Energies and Hydrophobicity. *J. Med. Chem.* **1991**, *34*, 786−797.

(18) Gribskov, M.; Robinson, N. L. Use of Receiver Operating Characteristic (ROC) Analysis to Evaluate Sequence Matching. *Comput. Chem.* **1996**, *20*(1), 25−33.

(19) King, R. D.; Muggleton, S. H.; Srinivasan, A.; Sternberg, M. J. E. Structure−Activity Relationship Derived by Machine Learning: The Use of Atoms and Bonds Connectivities to Predict Mutagenicity by Inductive Logic Programming. *Proc. Natl. Acad. Sci. U.S.A.* **1996**, *93*, 438−442.

(20) Kramer, S.; De Raedt, L. Feature Construction with Version Spaces for Biochemical Applications. In *Proceedings of the Eighteenth International Conference on Machine Learning*; Brodley, C. E., Pohoreckyj Danyluk, A., Eds.; Morgan Kaufmann, 2001; pp 258−265.

(21) Sebag, M.; Rouveirol, C. Tractable 1nduction and Classification in First-Order Logic via Stochastic Matching. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence*; Morgan Kaufmann: 1997; pp 888−893.

(22) Ralaivola, L.; Swamidass, S.; Saigo, H.; Baldi, P. Graph Kernels for Chemical Informatics. *Neural Networks*, 2005, to appear.

CI050039T