Two New Graphs Kernels in Chemoinformatics

Abstract
Chemoinformatics is a well established research field concerned with the discovery of molecule’s properties through informational techniques. Computer science’s research fields mainly concerned by chemoinformatics are machine learning and graph theory. From this point of view, graph kernels provide a nice framework combining machine learning graph theory techniques. Such kernels prove their efficiency on several chemoinformatics problems. This paper presents two new graph kernels applied to regression and classification problems within the chemoinformatics field. The first kernel is based on the notion of edit distance while the second is based on sub trees enumeration. The design of this last kernel is based on a variable selection step in order to obtain kernels defined on parsimonious sets of treelets. Performances of both kernels are investigated through experiments.

Keywords:

1. Introduction
- Graph Kernels for chemoinformatics [1, 2, 3, 4].
- Edit Distance Kernels
- Graph Laplacian Kernel

2. Kernel from Edit Distance

2.1. Edit Distance
An edit path between two graphs $G$ and $G'$ is defined as a sequence of operations transforming $G$ into $G'$. Such a sequence may include vertex or edge addition, removal and relabeling. Given a cost function $c(.)$, associated to each operation, the cost of an edit path is defined as the sum of its elementary operation’s costs. The minimal cost among all edit paths transforming
$G$ into $G'$ is defined as the edit distance between both graphs. A high edit distance indicates a low similarity between two graphs while a small one indicates a strong similarity.

According to Bunke and Neuhaus[5], the computational cost of the exact edit distance grows exponentially with the size of graphs. Such a property limits the computation of exact edit distance to small graphs. To overcome this problem, Bunke and Riesen[6] defined a method to compute an approximate edit distance in $O(nd^2)$ where $n$ and $d$ are respectively equal to the number of nodes and to the maximal degree of both graph.

2.2. Graph Laplacian Kernel

Unfortunately, edit distance doesn’t define a metric and trivial kernels based on edit distance are not definite positive. Neuhaus and Bunke [5] proposed several method to overcome this important drawback, however the proposed kernels are not explicitly based on the minimization problem addressed by kernel methods. Such a minimization problem may be stated as follows: Given a kernel $k$ and a dataset of graphs $D = \{G_1, \ldots, G_n\}$, the Gram matrix $K$ associated to $D$ is an $n \times n$ matrix defined by $K_{ij} = k(G_i, G_j)$. Within the kernel framework, a classification or regression problem based on $K$ may be stated as the minimization of the following formula on the set of real vectors of dimension $n$:

$$f^* = \arg \min_{f \in \mathbb{R}^n} C\text{Loss}(f, y, K) + f^t K^{-1} f$$

(1)

where $C\text{Loss}(.,.,.)$ denotes a given loss function encoding the distance between vector $f$ and the vector of known values $y$.

Each coordinate $f_i$ of such a vector $f$ corresponds to a value attached to graph $G_i$. A vector $f$ may thus be considered as a function mapping each graph of the database $\{G_1, \ldots, G_n\}$ to a real value. As denoted by Steinke [7], the term $f^t K^{-1} f$ in equation 1 may be considered as a regularization term which counter balance the fit to data term encoded by function $C\text{Loss}(.,.,.)$. Therefore, the inverse of $K$ (or its pseudo inverse if $K$ is not invertible) may be considered as a regularization operator on the set of vectors of dimension $n$ and hence on the set of real functions defined on $\{G_1, \ldots, G_n\}$. Conversely, the inverse (or pseudo inverse) of any semi definite positive regularization operator may be considered as a kernel. We thus follow a kernel construction scheme recently introduced [8] which first builds a semi definite positive regularization operator on the set of functions mapping each graph
\{G_1, \ldots, G_n\} to a real value. The inverse, or pseudo inverse of this operator defines a kernel on the set \(\{G_1, \ldots, G_n\}\).

In order to construct this regularization operator, let us define a \(n \times n\) adjacency matrix \(W\) defined by

\[ W_{ij} = e^{-\frac{d(G_i, G_j)}{\sigma}} \] (2)

where \(d(., .)\) denotes the edit distance and \(\sigma\) is a tuning variable. The Laplacian of \(W\) is defined as \(l = \Delta - W\) where \(\Delta\) is a diagonal matrix defined by:

\[ \Delta_{i,i} = \sum_{j=1}^{n} W_{i,j} \] (3)

Classical results from spectral graph theory \([9]\) establish that \(l\) is a symmetric semi definite positive matrix whose minimal eigenvalue is equal to 0. Such a matrix is thus not invertible. To overcome this problem, Smola \([10]\) defines the regularized Laplacian \(\tilde{l} = I + \lambda l\) of \(W\) where \(\lambda\) is a regularization coefficient. The minimal eigen value of \(\tilde{l}\) is equal to 1 and this matrix is thus definite positive. Moreover, given any vector \(f\), we have:

\[ f^t \tilde{l} f = \|f\|^2 + \lambda \sum_{i,j=1}^{n} W_{ij} (f_i - f_j)^2 \] (4)

Intuitively, minimising equation 4, leads to build a vector \(f\) with a small norm which maps graphs with a small edit distance (and thus a strong weight) to close values. Such a constraint corresponds to the regularization term required by equation 1 in order to smoothly interpolate the test values \(y\) over the set of graphs \(\{G_1, \ldots, G_n\}\). Our un normalized kernel, is thus defined as: \(K_{un} = \tilde{l}^{-1}\).

Note that a regularized normalized Laplacian kernel may alternatively be considered by introducing the matrix

\[ \tilde{L} = \Delta^{-\frac{1}{2}} \tilde{l} \Delta^{-\frac{1}{2}} \] (5)

We have in this case, for any vector \(f\):

\[ f^t \tilde{L} f = \sum_{i=1}^{n} \frac{f_i^2}{\Delta_i} + \lambda \sum_{i,j=1}^{n} \frac{W_{ij}}{\sqrt{\Delta_i \Delta_j}} (f_i - f_j)^2 \]
The matrix \( \tilde{L} \) is definite positive and its associated kernel is defined as 
\[ K_{\text{norm}} = \tilde{L}^{-1}. \] 
Note that, our regularized normalized Laplacian kernel is not defined as the inverse of the regularized normalized Laplacian:
\[ I + \lambda \Delta^{-\frac{1}{2}} \Delta^{-\frac{1}{2}} \] (6)
This new formulation is consistent with the regularization constraint which should be added to equation 1 and provides significant advantages in the context of incoming data (Section 2.3).

Alternative regularization schemes [10, 11] may be applied to the Laplacian matrix. One of this regularization scheme of particular interest is the Laplacian exponential diffusion matrix \( \tilde{d} = \exp(\lambda l) \) associated to the kernel \( K = \exp(-\lambda l) \). This last kernel may be considered as a generalisation of the regularized Laplacian with further constraints on the derivatives of the function \( f \). However, this kernel does not allows to update efficiently kernel values in the context of incoming data (Section 2.3).

2.3. Incoming Data

Let us first consider a kernel defined from the un normalized Laplacian. Given our learning set \( D = \{G_1, \ldots, G_n\} \), the test of a new graph \( G \) within a regression or classification scheme requires to update the un normalized Laplacian \( l \) with this new graph and to compute the updated kernel defined as the inverse of the regularized and un normalized Laplacian \( K = (I + \lambda l)^{-1} \).

This direct method has a complexity equal to \( O((n + 1)^3) \), where \( n \) is the size of our data set. Such a method is thus computationally costly, especially for large datasets. In this section, we propose a method to reduce the complexity of this operation.

Given the regularized and un normalized Laplacian \( \tilde{l}_n = (I_n + \lambda(\Delta_n - W_n)) \) defined on the dataset \( D \), its updated version \( \tilde{l}_{n+1} \) defined on \( D \cup \{G\} \) may be expressed as follows:
\[ \tilde{l}_{n+1} = \left( \begin{array}{ccc} \tilde{l}_n - \delta_n & B \\ B^t & 1 - \sum_i B_i \end{array} \right) \]
where \( B = (-\lambda \exp(-\frac{d(G,G_i)}{\sigma}))_{i=1,\ldots,n} \) is deduced from the weights between the new input graph \( G \) and each graph \( (G_i)_{i=1,\ldots,n} \) of our dataset and \( \delta_n \) is a diagonal matrix with \( (\delta_n)_{i,i} = B_i \).

The minimal eigen value of \( \tilde{l}_{n+1} \) is equal to 1 (Section ??). This matrix is thus invertible, and its inverse may be expressed using a block inversion scheme:
\[
K_{\text{un}} = (\tilde{l}_{n+1})^{-1} = \begin{pmatrix}
\Gamma & \Theta \\
\Lambda & \Phi 
\end{pmatrix}
\]

\[
\begin{align*}
\Gamma &= E^{-1} + \Phi E^{-1} B B^t E^{-1} \\
\Theta &= -E^{-1} B \Phi \\
\Lambda &= -\Phi B^t E^{-1} \\
\Phi &= (1 - \sum_i B_i - B^t E^{-1} B)^{-1}
\end{align*}
\]

where \( E = \tilde{l}_n - \delta_n \). Note that \( \Phi \) corresponds to a scalar.

The computation of our new kernel, using equation 7, relies on the computation of the inverse of the matrix \( E = \tilde{l}_n + \delta_n \) which may be efficiently approximated using a development to the order \( K \) of \((I - \tilde{l}_n^{-1} \delta_n)^{-1}\):

\[
(\tilde{l}_n - \delta_n)^{-1} = \tilde{l}_n^{-1} (I - \tilde{l}_n^{-1} \delta_n)^{-1} \approx \sum_{k=0}^{K} \tilde{l}_n^{-k-1} \delta_n^k
\]

Such a sum converges since \( \|\tilde{l}_n^{-1} \delta_n\|_2 < 1 \), for \( \lambda < 1 \). Indeed:

\[
\|\tilde{l}_n^{-1} \delta_n\|_2 \leq \|\tilde{l}_n^{-1}\|_2 \|\delta_n\|_2 \leq \|\delta_n\|_2 \leq \lambda \max_{i=1,n} \exp\left(\frac{-d(G, G_i)}{\sigma}\right)
\]

The last term of this equation is strictly lower than one for any \( \lambda \) lower than one. Moreover, basic matrix calculus show that the approximation error is lower than \( \epsilon \) for any \( K \) greater than:

\[
\log(2\epsilon) \over \log(\max_{i=1,n} \exp\left(\frac{-d(G, G_i)}{\sigma}\right)).
\]

Equation 8 allows to approximate the inverse of \((\tilde{l}_n - \delta_n)\) by a sum of pre computed matrices \( l_n^{-k-1} \) multiplied by diagonal matrices. Using such pre calculus, the inverse of \((\tilde{l}_n - \delta_n)\) and hence the computation of our new kernel may be achieved in \( KN^2 \).

If we now consider the regularized normalized Laplacian (Section ??) \( \tilde{L} = \Delta^{-\frac{1}{2}} \tilde{l} \Delta^{-\frac{1}{2}} \), its inverse is defined as: \( \tilde{L}^{-1} = \Delta^{\frac{1}{2}} \tilde{l}^{-1} \Delta^{\frac{1}{2}} \) and we have:

\[
K_{\text{norm}} = \Delta^{\frac{1}{2}} K_{\text{un}} \Delta^{\frac{1}{2}}
\]

The update of the regularized and normalized Laplacian kernel may thus be deduced from the one of the regularized un normalized Laplacian kernel.
3. Treelet Kernel

Kernels based on edit distance rely on a direct comparison of each pair of graph. An alternative strategy consists to represent each graph by a bag of patterns and to deduce the similarity between two graphs from the similarity of their bags. This strategy may provide semi definite kernels hereby avoiding the necessity to regularize the whole gram matrix for each incoming data (Section 2.3). As mentioned in Section 1, most of kernels of this family are based on linear patterns (bags of paths, trails or walks). Shervashidze et al. [2] describe a method to enumerate for any input unlabelled graph, all its connected subgraphs composed of up to 5 nodes. This efficient method provides up to 2048 patterns composed of connected subgraphs (called graphlets) of size less than or equal to 5. We propose here to adapt this method to the enumeration of sub-trees of labelled and unlabelled acyclic graphs up to size 6. The resulting patterns are called treelets (Fig. 1).

3.1. Computing Embedded Distribution

3.1.1. Structural Analysis

Each treelet corresponds to a sub tree of some graph of our database. Such a treelet may be denoted as \( t = (V_t, E_t, \mu_t, \nu_t) \) where \( \mu_t \) and \( \nu_t \) denote vertex and edge labeling functions. The first step toward the enumeration of treelets consists to classify them according to their graph structures \( (V_t, E_t) \).
The cardinal of \( V_t \) being bounded, only a finite set of graph structures called tree patterns may correspond to a treelet.

Following [2], the identification of tree patterns is initiated by an enumeration of all paths whose length is lower than or equal to 6. A recursive depth first search method with a max depth equals to 6 is thus performed for each node of the graph. Note that, using such an enumeration, each path is retrieved from its two extremities and is thus counted twice. In order to prevent this problem, each path composed of at least two nodes is counted \( \frac{1}{2} \) times. This first step provides the distribution of patterns \( G_0, G_1, G_2, G_3, G_4 \) and \( G_5 \) (Fig. 1).

Our method to compute the distribution of remaining patterns, is based on the detection of nodes of degree 3 and 4. These nodes respectively denoted \( R_{3\text{-star}} \) and \( R_{4\text{-star}} \) are the center of 3-star and 4-star patterns. Note that a 4-star pattern \( (G_8) \) contains four 3-star pattern (Fig. 2). This first enumeration of nodes of degrees 3 and 4 provides the distribution of patterns \( G_6 \) and \( G_8 \). Patterns \( G_7, G_9, G_{10} \) and \( G_{12} \) are enumerated from the neighbourhood of 3-star patterns. For example, the tree pattern \( G_7 \) matches the neighborhood of a 3-star node if at least one of the neighbour of this 3-star has a degree greater or equal to 2. Pattern \( G_{11} \) is the only sub tree derived from a 4-star. Properties characterizing patterns with a 3 or 4 stars are summarized in Table 1. Note that pattern \( G_{12} \) is symmetric since it contains two centers of 3-star. Such a treelet will thus be counted twice (once from each of its 3-star) and must be counted for \( \frac{1}{2} \).

Conditions summarized in Table 1 define necessary conditions for the existence of a given pattern centered around a 3 or 4 star. However, such conditions does not guarantee the uniqueness of such a pattern. Fig. 3 shows such an example: the rightest node of \( G_9 \) has a degree equals to 4 within the input graph whereas a degree greater or equal to 2 is required to define

\[
(a) \ G_7 \hspace{2cm} (b) \text{Four decompositions of } G_7 \text{ into } G_4
\]

Figure 2: Tree pattern \( G_7 \) contains 4 \( G_4 \).
Table 1: Conditions characterizing patterns derived from 3-star and 4-star. \(N(v)\) and \(d(v)\) denote respectively the set of neighbours and the degree of vertex \(v\).

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Source pattern</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>(G_6)</td>
<td>3-star</td>
<td>({v; v \in N(R_3\text{-star}); d(v) \geq 2}) (\geq 1)</td>
</tr>
<tr>
<td>(G_9)</td>
<td>3-star</td>
<td>({v; v \in N(R_3\text{-star}); d(v) \geq 2}) (\geq 2)</td>
</tr>
<tr>
<td>(G_{10})</td>
<td>3-star</td>
<td>(\exists v_0 \in N(R_3\text{-star}); d(v_0) \geq 2) and ({v; v \in N(v_0) - {R_3\text{-star}}; d(v) \geq 2}) (\geq 1)</td>
</tr>
<tr>
<td>(G_{11})</td>
<td>4-star</td>
<td>({v; v \in N(R_4\text{-star}); d(v) \geq 2}) (\geq 1)</td>
</tr>
<tr>
<td>(G_{12})</td>
<td>3-star</td>
<td>({v; v \in N(R_3\text{-star}); d(v) \geq 3}) (\geq 1)</td>
</tr>
</tbody>
</table>

![Figure 3: Three permutations of \(G_9\) sharing the same core.](image)

treelet \(G_9\). Three different treelet \(G_9\) may thus be built from the same five nodes. This configuration thus induces to count \(G_9\) three times from the graph represented in Fig. 3(a).

One may easily check that no isomorphism exists between patterns depicted in Fig. 1. Moreover, has shown by Cayley [?], the number of different acyclic and unlabeled graphs with less than 6 vertices and a maximum degree of 4 is equal to 13. Therefore, tree patterns \(G_0\) to \(G_{12}\) (Fig. 1) represent, up to isomorphisms, all unlabeled acyclic graphs whose size is lower than 6 and whose vertex degree is bounded by 4. Adding \(G_{13}\) to this set provides all unlabeled acyclic graphs with a size lower than 6.

### 3.1.2. Labeling Information

Method described in Section 3.1.1 allows us to associate each treelet to a tree pattern. Using databases of unlabeled graphs this method can be used to enumerate the number of treelets corresponding to each tree pattern. However, using databases of labeled graphs, many treelets with different vertices and edge labels may be associated to a same tree pattern. We propose in this section, to identify each treelet by a code composed of two parts: A
strutural part corresponding to the index of its tree pattern (Section 3.1.1) and a canonical code defined as a sequence of vertices and edge labels. Such a sequence is specific to each tree pattern and designed so as two treelets with a same code must be isomorphic.

Such a code is trivial for linear patterns, i.e. paths. Each path may indeed be associated to two sequences composed of alternated vertices and edge’s labels encoding the two possible traversal of this path. By convention, the canonical code of such linear tree patterns is defined as the sequence with the lowest lexicographic order.

Let us consider a non linear treelet \( G = (V, E, \mu, \nu) \) where, \( \mu \) and \( \nu \) denote respectively vertices and edge labeling functions. Our canonical code for non linear patterns is based on the extended connectivity concept introduced by Morgan [12]. This concept is based on an additional vertex labeling function \( \lambda \) from \( V \) to \( \mathbb{N} \) called extended labeling. This function is defined by an iterative process which initialises each extended label \( \lambda(v) \) to the degree of \( v \). This initial labeling is then extended by assigning to each vertex the sum of extended labels of its neighbors. This summation process is iterated while it increases the number of different labels. Note that the resulting set of extended labels will be same for two isomorphic graphs and is unique for each tree pattern. Fig. 4 show the set of extended labels defined on treelet \( G_7 \). In this case since the number of extended labels does not increase after the first iteration, labels remain equal to the degree of vertices.

Since two adjacent vertices \( v \) and \( v' \) of a treelet may be compared according to \( \lambda(v) \) and \( \lambda(v') \), our extended label defines a partial order relationship between adjacent vertices of a treelet. Following Morgan [12], we encode this partial order relationships by a rooted tree. This tree is built as follows: we first connect each vertex to its neighbor with a maximal extended label. Note that, using our set of treelets (Fig. ??) a vertex is either a local maxima or adjacent to a single maxima. Such a process define a unique rooted tree on treelets \( G_6 \) to \( G_{11} \). Treelet \( G_{12} \) contains two local maxima connected by one edge. We define for this treelet two possible rooted trees rooted on each of these local maxima.

Our construction scheme of the canonical code of a treelet is based on a traversal of its rooted tree. The construction of our code requires to sort the children of each internal vertex of the tree in order to define a unique traversal and hence a unique code. This sorting step is achieved through the following recursive construction of our canonical code: The code of each leaf \( v \), \( \text{code}(v) \) is defined as the label \( \mu(v) \) of the associated vertex. For each
internal node $v$ of the tree, let us consider its set of children $\{v_1, \ldots, v_n\}$. This set is first sorted according to $\lambda(v_i)$ and then according to the chain defined as the concatenation of $\nu(v, v_i)$ and $\text{code}(v_i)$. Assuming such an order on $\{v_1, \ldots, v_n\}$, the code of vertex $v$ is defined as:

$$\text{code}(v) = \mu(v) \bigcirc_{i=1}^{n} \nu(v, v_i) \cdot \text{code}(v_i)$$  \hspace{1cm} (11)$$

where $\bigcirc$ denotes the concatenation operator.

The canonical code of treelets $G_6$ to $G_{12}$ is defined as the index of their tree pattern concatenated with the code of the root node of their associated rooted tree. Treelet $G_{12}$ is associated to two trees rooted on its two local maxima. We define the canonical code of this treelet as the index of its tree pattern concatenated with the lowest code of its two associated rooted trees.

Our canonical code is based on extended labels deduced from the structure of the treelet and from vertex and edge labeling functions. Hence, two isomorphic treelets are associated to a same canonical code. Conversely, since one may easily build a linear pattern from its canonical code, two linear patterns with a same canonical code must be isomorphic. Treelets corresponding to tree patterns $G_6$ to $G_5$ are thus uniquely determined by their canonical code.

Within our code construction scheme, the label of a vertex with an unique extended label is located at a fixed position within the canonical code of its treelet. The label of such a vertex may thus be retrieved without any ambiguity from a canonical code. However, a set of children $\{v_1, \ldots, v_n\}$ of a same parent node $v$ with a same extended label will be sorted according to the sequence of edge and vertex labels $\nu(v, v_i) \cdot \text{code}(v_i)$. This sorting step allows us to have an unique label for two isomorphic treelets but do not allows us to distinguish between any permutations of vertices $\{v_1, \ldots, v_n\}$. We have thus to check, for each treelet that all permutations of vertices allowed by our code correspond to isomorphic treelets.

Vertices with a same extended label within tree patterns $G_6, G_7, G_8, G_{10}$ and $G_{11}$ are represented by black squares (■) in Fig. 6. Using our code construction scheme, for each tree pattern these vertices of degree one are the child of the unique vertex they are connected to. Therefore, our code does not distinguish any permutation among these vertices. However, since these vertices have a degree one and are connected to a same vertex, any permutation exchanging two of these vertices leads to an isomorphic treelet.
Figure 4: Partial ordering of nodes on tree pattern $G_7$.

Figure 5: Rooted trees associated to treelets based on the extended labeling function $\lambda$. Values of $\lambda$ is indicated next to each vertex in (a) and (c).

The rooted tree associated to treelet $G_9$ does not allow to distinguish between branches $v_2v_3$ and $v_4v_5$ (Fig. 5(b)). However, the simultaneous permutation of $v_2$ with $v_4$ and $v_3$ with $v_5$ provides to an isomorphic graph. In the same way, treelet $G_{12}$ (Fig. 5(c)) does not allow us to distinguish permutations between branches rooted on $v_0$ and $v_3$ and between permutations between $v_1$, $v_2$ on one hand and $v_4$, $v_5$ on the other hand. All these permutations provide isomorphic treelets.

When all treelets from a graph $G$ have been enumerated, a vector representing the treelet distribution is computed. Each component of this vector,

Figure 6: Patterns with partial sort defined using Morgan partial ordering and FROM ATTACHMENT List. Possible permutations are represented by square nodes.
denoted the spectrum of $G$, is equal to the frequency of a given treelet in $G$:

$$f_i(G) = |(G_i \subset G)|$$

(12)

### 3.2. Treelet Kernel Definition

A first idea to define a kernel from treelets consists to perform the inner product of vectors encoding the spectrum of graphs. Unfortunately, the inner product doesn’t highlight spectrum similarities. For example, two graphs with nearly equal spectrum but with a low number of occurrences for each treelet may be considered as less similar than two graphs having a same high number of treelet $G_0$ (i.e., same size) but a distribution of others treelets highly dissimilar. We thus use RBF kernels in order to better highlight differences between two spectra:

$$k_{Treelet}(G, G') = \sum_{k=0}^{N} e^{-\frac{(f_k(G) - f_k(G'))^2}{\sigma}}$$

(13)

where $\sigma$ is a tuning variable used to weight the differences between treelet distribution and $N$ is the number of enumerated treelets. Our kernel may thus be considered as a basic RBF kernel between two vectors and is hence definite positive.

### 3.3. Treelet Weighting

Among the set of treelets found within a dataset, some of them may be irrelevant to explain a given property. It is thus interesting to reduce the set of treelets and keep the relevant ones according to a property to predict. A first trivial approach is to consider all possible permutations over the set of treelets. This approach implies to test $2^p$ different sets of treelets, where $p$ is the number of different treelets of the dataset. Such a brute force approach is untractable even for small values of $p$.

The Forward Selection for regression is an iterative method which starts with an empty set and add one treelet at each step. The chosen treelet is the one which produces the best regression result (Alg. 1). To evaluate the quality of the result, we can use the Residual Sum of Squares (RSS) defined as the quadratic sum of the prediction error made for each molecule. Another quite similar approach is to start from the whole set of treelets and to remove one treelet at each iteration. This second approach is called Backward Elimination (Alg. 2). Both methods imply to test $\frac{p(p+1)}{2}$ set of treelets.
The main difference between these two approaches is that the backward approach removes one treelet knowing the information brought by all others while the forward approach adds one treelet only knowing the contribution of current treelet. This characteristic may be interesting if a set of treelets is relevant when they’re considered together whereas they irrelevant alone. Considering this supposition, the backward approach will not remove one of these treelets since the forward approach will not add one of them since an isolated treelet doesn’t improve the RSS. So, the backward approach is able to keep set of relevant treelets which may increase the accuracy of our model.

**Algorithm 1** Forward Selection.

\[
P = \text{Treelets} \hspace{1cm} S = \emptyset \hspace{1cm} \text{nb}_{\text{treelets}} = |P|
\]

\[
\text{for } i = 0 \rightarrow \text{nb}_{\text{treelets}} \text{ do} \\
\hspace{1cm} t = \arg \min_t \text{RSS}(S \cup t), t \in P \\
\hspace{1cm} P = P \cap t \\
\hspace{1cm} S_{i+1} = S_i \cup t \\
\text{end for} \\
\text{return } \arg \min_{S_i} \text{RSS}(S_i), i \in [0, \text{nb}_{\text{treelets}}]
\]

**Algorithm 2** Backward Elimination.

\[
S = \text{Treelets} \hspace{1cm} \text{nb}_{\text{treelets}} = |S|
\]

\[
\text{for } i = 0 \rightarrow \text{nb}_{\text{treelets}} \text{ do} \\
\hspace{1cm} t = \arg \min_t \text{RSS}(S \cap t), t \in S \\
\hspace{1cm} S_{i+1} = S_i \cap t \\
\text{end for} \\
\text{return } \arg \min_{S_i} \text{RSS}(S_i), i \in [0, \text{nb}_{\text{treelets}}]
\]

Forward Selection and Backward Selection weight each treelet with a binary weight, ie. a treelet is included or not in the set of features. A more accurate solution to treelet weighting is to compute a weight for each treelet. In order to define these weights, let us consider the following objective function minimized by Kernel Ridge Regression method \[\]
\[
C(\alpha) = \lambda \alpha^t K \alpha + \|y - K \alpha\|^2
\] (14)
with \( \alpha \) is a vector \( \in \mathbb{R}^N \) equals to \((\lambda I + K)^{-1}y\) and \( \lambda \) is the regression coefficient. Our treelet kernel can be defined as a sum of \( T \) sub kernels where each sub kernel only takes into account one treelet among the \( T \) treelets found. Then, \( K \) is defined as follows:

\[
K = \sum_{k=1}^{T} K_k, \quad \text{with } K_{k,i} = e^{-\frac{(f_k(G_i) - f_k(G_j))^2}{\sigma^2}}
\]

and in a weighted way:

\[
K(w) = \sum_{k=1}^{T} w_k K_k
\]

where \( w \in \mathbb{R}_+^T \) is a vector encoding the positive weights associated to each treelet. So, considering Eq. 16, the Kernel Ridge Regression relies on minimizing the objective function which now depends on two parameters:

\[
C(\alpha, w) = \lambda \alpha^t K(w) \alpha + \|y - K(w)\alpha\|^2
\]

Then we insert a regularisation term which aims to sparsify \( w \) in order to decrease the number of treelets having a weight different from 0. So, we weight our objective function by adding the Manhattan norm of \( w \):

\[
C(\alpha, w) = \lambda \alpha^t K(w) \alpha + \|y - K(w)\alpha\|^2 + \|w\|_1
\]

The previous equation can be wrote as a linear tranformation of \( \alpha \) and \( w \):

\[
C(\alpha, w) = \lambda < A^t \alpha, w > + \|y - A w\|^2 + \|w\|_1
\]

The two vectors which give a local minima of the previous equation can be approximated using a double steepest gradient descent. This method consists to minimize the objective function according to each term alternatively, see Alg. 3.

First, for any \( w \), computing \( \alpha \) relies on the solution of classic Kernel Ridge Regression []:

\[
\alpha = (\lambda I + K(w))^{-1}y
\]

Second, the gradient according to \( w \) is equals to:

\[
\nabla_w C = A^t(2y - 2Aw - \lambda \alpha)
\]
\textbf{Algorithm 3} Double Steepest Gradient

\begin{itemize}
    \item $W_0 \in \mathbb{R}^T$
    \item $\alpha_0 \in \mathbb{R}^N$
    \begin{repeat}
        \item $\alpha^{t+1} = \arg\min_{\alpha} C(\alpha^t, w^t)$
        \item $w^{t+1} = \arg\min_w C(\alpha^t, w^t)$
    \end{repeat}
    \until convergence
\end{itemize}

So, to minimize Eq. 17 according to $w$, we use an iterative steepest descent until convergence with:

$$w^{t+1} = w^t + \tau^t A^t (2y - 2Aw - \lambda \alpha)$$ \hfill (22)

Where $\tau^t \in [0, \frac{1}{\|A^tA\|}]$ to ensure convergence \[.\] To take into account the regularization term $\|w\|_1$ of Eq.18, we include a soft thresholding defined as follows:

$$\text{ST}_\mu(x_i) = \left((1 - \frac{\mu}{|x_i|})_{+} x_i\right)_i$$ \hfill (23)

where $(.)_+$ is the proximal operator and $x$ a vector. Finally, our gradient descent relies to iterate using this equation:

$$w^{t+1} = \text{ST}_\mu \left(w^t + \tau^t A^t (2y - 2Aw - \lambda \alpha)\right)$$ \hfill (24)

until convergence.

4. Experiments

4.1. Classification Problem

Our first experiment evaluates our two graphs kernels on a classification problem. This problem is defined on the monoamine oxidase dataset(MAO)\(^1\) which is composed of 68 molecules divided into two classes: 38 molecules inhibits the monoamine oxidase (antidepressant drugs) and 30 does not. These molecules are composed of different types of atoms with simple bonds and are thus encoded as labeled graphs. Classification accuracy is measured for

\(^1\)All databases in this section are available on the TC15 Web page: http://www.greyc.ensicaen.fr/iapr-tc15/links.html#chemistry
Table 2: Results on MAO Dataset.

<table>
<thead>
<tr>
<th>Method</th>
<th>Classification Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>KMean [13]</td>
<td>80% (55/68)</td>
</tr>
<tr>
<td>KWMean [14]</td>
<td>88% (60/68)</td>
</tr>
<tr>
<td>Trivial Similarity Kernel from Edit Distance [5]</td>
<td>90% (61/68)</td>
</tr>
<tr>
<td>Normalized Standard Graph Laplacian Kernel (Eq. 10)</td>
<td>90% (61/68)</td>
</tr>
<tr>
<td>Normalized Fast Graph Laplacian Kernel (Eq. 10)</td>
<td>90% (61/68)</td>
</tr>
<tr>
<td>Random Walk Kernel [4]</td>
<td>82% (56/68)</td>
</tr>
<tr>
<td>Treelet Kernel</td>
<td>91% (62/68)</td>
</tr>
</tbody>
</table>

each method using a leave one out procedure with a two-class SVM. This classification scheme is made for each of the 68 molecules of the dataset.

Table 2 shows results obtained by graph Laplacian kernel using approximate graph edit distance [6] with node substitution and edge deletion costs set to 1 and edge substitution cost set to the sum of incident node substitution costs. Graph Laplacian kernel methods obtain a classification accuracy of 90% and we can note that the other method obtaining 90% of classification accuracy is also based on the edit distance. This last kernel may however be non definite positive.

We may additionally notice that the use of our fast inversion method (Section 2.3) does not modify graph Laplacian kernel’s classification accuracy (Table 2, lines 4 and 5). The number of iterations required by this fast inversion method is determined by equation 9. Our experiments performed on the MAO database show that a value of $\epsilon$ equal to $10^{-4}$ induces a maximum of 9 iterations hence allowing to update the gram matrix in $O(9N^2)$ instead of $O(N^3)$ using a standard matrix inversion method. The low value of $N$ on this dataset ($N = 68$) does not induce an important gain on execution time since the average time to update a Gram matrix using method described in Section 2.3 is 0.273$ms$ on the MAO database while this time is equal to 0.498$ms$ using a direct inverse matrix computation. The ratio between both execution times is nevertheless about 1.8 hence showing a significant gain. Our treelet kernel is not tested against this database since this kernel is devoted to unlabeled graphs.

PTC ou HIV ou classif Acyclique
Table 3: Boiling point prediction on alkane dataset.

<table>
<thead>
<tr>
<th>Method</th>
<th>Average error (°C)</th>
<th>Standard deviation (°C)</th>
<th>Correlation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Neural Network [15]</td>
<td>3.11453</td>
<td>3.69993</td>
<td>0.9827</td>
</tr>
<tr>
<td>KMean [13]</td>
<td>4.65536</td>
<td>6.20788</td>
<td>0.9918</td>
</tr>
<tr>
<td>Graph Laplacian Kernel</td>
<td>10.7948</td>
<td>16.4484</td>
<td>0.9412</td>
</tr>
<tr>
<td>Treelet Kernel</td>
<td>1.40663</td>
<td>1.91695</td>
<td>0.9992</td>
</tr>
</tbody>
</table>

4.2. Regression Problem

Our second experiment is based on a database of alkanes [15]. An alkane is an acyclic molecule solely composed of carbons and hydrogens. A common encoding consists to implicitly encode hydrogen atoms using the valency of carbon atoms. Such an encoding scheme allows to represent alkanes as acyclic unlabeled graphs. The alkane dataset described in [15] is composed of 150 molecules, associated to their respective boiling points. Using the same protocol than [15], we evaluate the boiling point of each alkane using several test sets composed of 10% of the database, the remaining 90% being used as training set.

Table 3 shows results obtained by different methods. Poor results obtained by graph Laplacian kernel can be explained by the lack of information when dealing with unlabeled graphs. Indeed, using such graphs, the heuristic used to approximate graph edit distance [6] maps the set of vertices of both graphs using uniquely the degree of vertices. Such a method thus consider several mappings as equivalent if several vertices with a same degree exist in both graphs. In this case, the sub optimal graph edit distance induces a poor graph discrimination. This lack of local information within unlabeled graphs also explains the poor results obtained by Kmean and random walk kernels. Indeed, these kernels are based on linear structures which are only discriminated by their lengths within unlabeled graphs. On the other hand, treelet kernel (with $\sigma = 0.25$) outperforms previous results of [15] based on neural networks combined with chemical descriptors. Note that we didn’t try to select relevant patterns since our set of treelet is composed of only 13 treelets. Indeed, no labelisation discrimination is performed since the dataset is composed of alkane molecules.

The second regression problem is based on a dataset which contains acyclic molecules []. In contrast to the previous one, the molecules of this dataset
Table 4: Boiling point prediction on acyclic molecule dataset.

<table>
<thead>
<tr>
<th>Method</th>
<th>Average error (C)</th>
<th>Standard deviation (C)</th>
<th>Correlation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Neural Network [16]</td>
<td>5.102</td>
<td></td>
<td>0.989</td>
</tr>
<tr>
<td>KMean [13]</td>
<td>7.27537</td>
<td>12.3692</td>
<td>0.96627</td>
</tr>
<tr>
<td>Graph Laplacian Kernel</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Gaussian Kernel</td>
<td>6.83768</td>
<td>9.04099</td>
<td>0.982124</td>
</tr>
<tr>
<td>Treelet Kernel</td>
<td>4.89039</td>
<td>7.80264</td>
<td>0.986716</td>
</tr>
<tr>
<td>Treelet Kernel Forward Selection</td>
<td>3.23422</td>
<td>4.36002</td>
<td>0.995871</td>
</tr>
<tr>
<td>Treelet Kernel Backward Elimination</td>
<td>2.62891</td>
<td>3.69775</td>
<td>0.997032</td>
</tr>
</tbody>
</table>

contains several hetero atoms and are thus represented as labeled graphs. This dataset Acyclic dataset
The selection method based on Backward Selection performs the best since it takes into account the information brought by combinations of treelets since the forward selection adds the treelets one by one and thus can’t take advantage of useful combinations of features.

5. Conclusion

We have proposed two new graphs kernel using different approaches. The first one uses the notion of edit distance between graphs in order to define a positive semi definite kernel. The second approach is based on the decomposition of the graph into a set of distinct substructures, called treelets. This second approach is different from the one presented by Mah and al. [17]. The first difference is the notion of tree walks vs subtrees. Our method computes the distribution of subtrees since the Mah’s method consider tree walks, ie trees which can contain a same node more than one time. In addition, we compute explicitly the distribution of each treelet which allow us to weight a given treelet since in tree patterns methods only the number of common tree patterns is computed.

Outlooks:

- Cycles
- Kernel Combinaison for classification problems.
References


[6] K. Riesen, H. Bunke, Approximate graph edit distance computation by


[16] D. Cherqaoui, D. Villemi, A. Mesbah, J.-m. Cense, V. Kvasnicka, Use of a Neural Network to determine the Normal Boiling Points of Acetals