Measures of similarity between graphs

Mathieu Senelle,
Marco Saerens,
François Fouss,
Louvain School of Management
Measures of similarity between graphs

Mathieu Senelle, Louvain School of Management
Marco Saerens, Louvain School of Management
François Fouss, Louvain School of Management

Summary

This work introduces new ways to compare graphs, and more specifically focuses on finding and analyzing new similarity measures between graphs, which are efficient in a computational point of view (how the time to solve the problem evolves with respect to the size of the underlying graph), but also lead to accurate results, compared to state-of-the-art methods. This work also shows that the suggested measures can be applied successfully to supervised classification of graphs.

Keywords: Graph similarity, Graph kernels, Classification.

JEL Classification: C19, C65.

Corresponding author:
Mathieu Senelle
Center for Excellence PRISME
Louvain School of Management / Campus Mons
Université catholique de Louvain
Chaussée de Binche, 151
B-7000 MONS, BELGIUM
Email: m.senelle@uclouvain-mons.be
Abstract. This work introduces new ways to compare graphs, and more specifically focuses on finding and analyzing new similarity measures between graphs, which are efficient in a computational point of view (how the time to solve the problem evolves with respect to the size of the underlying graph), but also lead to accurate results, compared to state-of-the-art methods. This work also shows that the suggested measures can be applied successfully to supervised classification of graphs.

Keywords: Graph similarity, Graph kernels, Classification.

1 Introduction

1.1 Overview of the research topic

Bioinformatics, chemoinformatics, drug discovery are only few examples of domains where the need for algorithms to analyze and classify data is constantly increasing. Indeed, various applications, like predicting the properties of molecules, such as toxicity or anti-cancer activity, are using databases with millions of molecules to handle. An efficient way to model these molecules is to use structured objects like graphs, because in those cases, we need to take into account not only their sub-elements (atoms, which are represented as nodes), but also the structure between them (bonds, which are represented as edges). We can see an example of molecule modelled by a graph on the figure 1.

In this context, two problems can be studied: firstly, we can try to find how similar two nodes in a graph are, and secondly, how similar two graphs, or a set of graphs, are. In our work, we will study different ways to answer the second question. We will focus on finding and analyzing new similarity measures...
between graphs, which are efficient in a computational point of view (how the
time to solve the problem evolves with respect to the size of the underlying
graph), but also lead to accurate results, compared to state-of-the-art methods.
This work will also show that the suggested measures can be applied successfully
to supervised classification, which will let us assess the quality of those measures.

Some notions necessary to the understanding of this work will be defined
hereunder.

1.2 Theoretical Introduction

Graph Theory As already mentioned above, graphs can represent many real
life objects, like molecules, social networks, telecommunication networks,... But
what do these graphs really are ? In fact, they can be defined as mathematical
objects containing nodes (or vertices) and edges. On the figure 2, nodes are
modelled by circles and edges by links going from one node to another. There
are several kind of graphs : labelled or not, directed or not. Labelled graphs bear
information (labels) on their nodes, and/or on their edges. Directed graphs have
edges that are directed, i.e. they have a direction assigned to them. The figure
1 is a simple example of labelled undirected graph : the labels on the nodes
are the type of atom in the molecule (Hydrogen, Carbon,...), and the labels on
the edges are the type of bound existing between these nodes (here simple and
double bond).

![Fig. 2: From left to right : undirected, directed, and labelled directed graphs.](image)

Graph Comparison The methods used to compare graphs can be grouped in
several categories. Three of them are reviewed in this section : feature based,
frequent substructure based, and kernel based approaches.

Feature based methods extract several indicators from a graph, and use them
to construct a vector which describes that graph. Those indicators can be com-
puted from the structure of the graph (like number of nodes, number of edges,...) or
can be obtained from the application studied, for instance biological activity
of molecules. This kind of graph comparison is widely used in bio or chemoin-
formatics, but suffers from several drawbacks : the most discriminant set of
Descriptors often vary from one application to another, and their computational complexity can be quite high (polynomial or even exponential).

Frequent subgraphs comparison consists in retrieving the most frequent subgraphs appearing in a graph, and comparing that collection of subgraphs to the ones from all the other graphs in the dataset. Some methods, like the Gspan software [6], have been developed for this purpose, using for instance branch-and-bound search strategies, a well-known technique in graph theory. But again, the main drawback of these methods is that they are plagued by computational slowness, as it can scale exponentially with the size of graphs in the worst case.

Kernel methods are one of the latest - and most promising - techniques used in data mining and machine learning to quantify similarities between structured objects. While most of data analysis methods define a vector-based representation of the objects studied, which can be tedious to construct, kernel methods do not represent data individually, but through a set of pairwise comparison. More formally, suppose that we possess a dataset $X$ where each object $x_i$ is, for instance, a molecule. In classical methods, a representation $\phi(x_i)$ should be found for each object, leading to the set of representations $\phi(X)$. For a molecule, it could be the types of atoms present in that compound and thus $\phi(X) = (\text{COO}, \text{HCl}, ...)$.

As already said, kernel methods are based on a drastically different idea, i.e. the whole dataset is simply represented as a kernel matrix $K$, with $K_{i,j} = k(x_i, x_j)$ $k$ being a similarity measure between $i$ and $j$ objects (see next section). This kind of representation has several advantages. Firstly, no matter the nature of the objects studied, the dataset will always be described as a real-valued matrix. As a consequence, an algorithm designed to handle such matrices will be able to process molecules as well as pictures. Secondly, it is often easier to find a comparison "score" between two complex objects than an exact representation of the latter. And thirdly, the size of the kernel matrix is only a function of the number of objects, and does not grow with their complexity.

**Classification and Support Vector Machines** Support Vector Machines (SVMs) belong to the family of kernel methods, and are used in this work to perform supervised classification of molecules. Classification is a data mining technique used to predict which class an object belongs to, and the term supervised stands for data where each instance consists of a pair composed of an input value and a desired output value. SVMs are often employed in binary classification problems: given a set of training objects $x_i \in \mathcal{X} = \mathbb{R}^d$, $d \in \mathbb{N}$ associated with class labels $y_i \in \mathcal{Y} = \{-1, 1\}$ the goal is to build a classifier function $f: \mathcal{X} \rightarrow \mathcal{Y}$ in order to predict the label of any unknown object presented to that classifier.

**Linearly Separable Problems**

When data can be linearly separated into two classes, classification can be done by introducing a hyperplane (or a plane, if the data lie in three dimensions) between the classes $y = 1$ and $y = -1$ (see Figure 3). The class corresponding to an object $x_i$ is determined by its location with respect to the hyperplane. If
a dataset is linearly separable, an infinity of hyperplanes correctly classify the
data, and are described by the equation $\langle w, x \rangle + b = 0$, where $w$ is a vector
perpendicular to the hyperplane (and thus is an indicator of the direction of the
plane), and $b$ is a constant $\in \mathbb{R}$. These hyperplanes correspond to the following
decision function:

$$f(x) = \text{sgn}(\langle w, x \rangle + b),$$

where $f(x)$ correspond to the class predicted for an object $x$, i.e. +1 or -1. But
among all these hyperplanes, how to choose the “best” one? One way to do that
is to define a margin, which is the minimum distance between the hyperplane
and the nearest points from both classes, and to select the hyperplane which
maximizes that margin. It will be the optimal hyperplane, as it minimizes the
risk of misclassification of data.

![Fig. 3: An example of linearly separable problem, with a hyperplane separating two
classes of objects (triangles and circles), and the corresponding margin (dashed lines)](image)

This optimization problem can be formulated as follow:

$$\begin{align*}
\text{minimize} & \quad \frac{1}{2} \|w\|^2 \\
\text{subject to} & \quad y_i(\langle w, x_i \rangle + b) \geq 1 \text{ for all } i \in \{1, 2, \ldots, m\}.
\end{align*}$$

The minimization of $w$ correspond to the objective of maximizing the margin.
Indeed, it can be shown that the latter is proportional to $\frac{1}{\|w\|}$. The constraints
$y_i(\langle w, x_i \rangle + b) \geq 1$ illustrates the fact that any object must lie outside the margin.
Using the Lagrangian, we can solve the corresponding dual problem:

\[
\maximize_{\alpha} \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j y_i y_j \langle x_i, x_j \rangle
\]  

subject to \( \sum_{i=1}^{m} \alpha_i y_i = 0 \) and \( \alpha_i \geq 0 \) for all \( i \in \{1, 2, \ldots, m\} \).  

and we also obtain:

\[
w = \sum_{i=1}^{m} \alpha_i y_i x_i.
\]  

One interesting fact about these equations is that the dual problem only takes into account the dot product of the initial data set, it does not need the data in itself. This kind of Support Vector Machines is called hard margin SVM, because by construction, it does not tolerate an object to be misclassified.

**Non-Linearly Separable Problems**

Unfortunately, the majority of the datasets are hardly linearly separable, and it is often impossible to find a hyperplane that correctly separate the objects in two classes. In order to overcome this problem, soft margin SVMs were developed. They allow some data to be misclassified, in order to improve the generalization performance of the SVM (the performance over previously unseen data)(see Figure 4). In 1995, Cortes and Vapnik [2] proposed a technique which tolerates misclassification, called C-SVM. It aims at minimizing the number of errors by introducing new slack variables, \( \xi_i \), and a penalty term that penalizes high slack variables, \( C \). The formulation of the primal problem now becomes:

\[
\minimize_{w, b, \xi} \frac{1}{2} \|w\|^2 + C \sum_{i=1}^{m} \xi_i
\]  

subject to \( y_i (\langle w, x_i \rangle + b) \geq 1 - \xi_i \) for all \( i \in \{1, 2, \ldots, m\} \).  

\( C \) is a constant that controls the tradeoff between classification errors and the width of the margin. The tuning of this parameter’s value is a problem, as it has no intuitive meaning. In order to find its optimal value, it is often necessary to perform an exhaustive search in the parameter space, using a cross validation for instance (see below).

But even with the soft margin technique, it is still not possible to solve every classification problem. For instance, if one class of the data lies in a circle, and the other outside, it is not possible to find a hyperplane that classifies correctly the objects. One idea is to map the objects in another space, called the feature space, which is often higher-dimensional, with the help of a non-linear mapping \( \phi \). The objective is to find a feature space in which the data are linearly separable. We saw previously that the dual formulation of the optimization problem only depends on the dot product of the data \( \langle x_i, x_j \rangle \). If those data are mapped to the feature space, this product becomes \( \langle \phi(x_i), \phi(x_j) \rangle \).
If a function $k$, called kernel function, is defined by

$$k(x_i, x_j) = \langle \phi(x_i), \phi(x_j) \rangle,$$

the decision function becomes

$$f(x) = \text{sgn} \left( \sum_{i=1}^{m} y_i \alpha_i \langle \phi(x_i), \phi(x_j) \rangle + b \right) = \text{sgn} \left( \sum_{i=1}^{m} y_i \alpha_i k(x_i, x_j) + b \right).$$

The optimization problem is now

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

subject to $\alpha_i \geq 0$ for all $i = 1, \ldots, m$ and $\sum_{i=1}^{m} \alpha_i y_i = 0.$

We can see in this classification method that the objects to classify are mapped to a higher-dimensional feature space, using a non-linear map $\phi$. But it is not necessary to know explicitly $\phi$, because only the kernel function is needed. This is called the Kernel Trick. The kernel function has to meet certain conditions to represent a dot product in a higher-dimensional space. It must be symmetric:

$$k(x_i, x_j) = k(x_j, x_i)$$

and positive semidefinite

$$\sum_{i=1}^{m} \sum_{j=1}^{m} c_i c_j k(x_i, x_j) \geq 0.$$
In fact, when using a SVM, the key task is to choose the kernel adapted to the situation. Some of them are widespread in the machine learning community:

- the polynomial kernel: \( k = (\langle x_i, x_j \rangle + c)^d \). If \( c = 0 \) and \( d = 1 \), this kernel is known as the linear kernel.
- the Gaussian kernel: \( k = \exp \left( -\frac{\|x_i - x_j\|^2}{2\sigma^2} \right) \)

In section 2, some well-known kernels specifically designed to compare graphs are exposed, and new ones are introduced in section 3.

1.3 Applications

The discovery of new drugs is a long and expensive process. Indeed, it takes 10 to 15 years to develop a drug and the average cost associated to it is about 800 million euro [3]. Pharmaceutical companies therefore try to diminish those costs, and the use of machine learning and data mining tools become prominent to achieve that goal. In this section, we will detail the traditional way used to discover new drugs, and how computational techniques can improve this process.

The primary goal of new drug research is to identify the origin of the apparition of a disease, and this factor is usually an active protein (called the target). We thus try to find a compound which will interact with that protein, and will inhibit its activity. Of course, that compound must not only be active towards the protein, but also be harmless for patients who will ingest the corresponding drug. We will now develop each step of the process leading to the discovery of a new drug. The first step of target identification involves deep knowledge in biology and chemistry. We will not go further in this subject, as it is not impacted...
with bioinformatics techniques used in our research, but, as an example, the work realized on the Human Genome Project, which aimed at decrypting the complete DNA of the human genome, permitted the identification of many new targets [5].

The second step of the discovery process is to find a molecular compound which will bind to the target defined in the first step. But where do these compounds come from? In fact, during the eighties, a technique called combinatorial chemistry was set up. It uses a set of elementary chemical blocks, and combines them in every possible way. This operations allows scientists to synthesize huge sets of molecules. For example, if we consider 20 building blocks (for instance amino acids), and we would like to synthesize molecules constituted by 5 blocks, we can therefore create $20^5 = 3200000$ molecules! Thereby, we can then obtain sets of millions molecules, which will have to be screened to determine the ones that presents the best characteristics (i.e. interaction with the target). The screening step consists in sending the molecule studied on the target, see if it binds to it, and to quantify the strength of the binding. We can imagine the time it would take to screen millions of molecules by hand. Fortunately, a technique allowing to screen molecules in a very fast way exists: the high throughput screening (HTS), consisting in a parallelization of the screening. As an order of magnitude, the latest version of HTS is able to treat 100 000 molecules a day. The third step of the discovery process is to verify that candidate molecules, which are an output of the HTS step, possess the adequate characteristics to be used as a treatment to human patient. Indeed, not only these molecules must interact with the target, but they must also have low toxicity, good pharmacokinetics properties (ability to enter the bloodstream, to be eliminated from the body, . . .).

Let us now describe how computational biology can intervene and improve the drug discovery process described above. First of all, the sets of molecular compounds created by combinatorial chemistry are not much diversified. Indeed, a lot of molecules present almost the same structure, and it is known that a similar structure is correlated to a similar function. So if the screening is performed on all molecules, a lot of redundancy will be found. A way to avoid this is to group the similar molecules together, by creating clusters. This implies to possess a similarity measure between the compounds, which is exactly the goal of our work. We will represent molecules as graphs, and check their similarity with a measure we will set up. As the database is constituted with the graphs representing the molecules, we will then have clusters that regroup similar molecules. The screening step will therefore be simplified, as we will only have to check a representative molecule of a cluster, and if this molecule does not bind well to the target, we will drop that cluster and check another. If the interaction between the molecule and the target is strong enough, then the other molecules belonging to the cluster will also be screened, in order to determine the best candidate for a drug. This process is known as Library Design. Another way to improve the drug discovery cycle using computational tools is the following: when the structure (the graph representation) of a reference molecule, which is
known to bind strongly to the target, is accessible, we can also perform what is called a Virtual Screening. The principle is quite simple: we compare molecules present in the database to the one of reference. Again, as similar structure gives similar effects, we will first virtually (i.e., on a computer) select the molecules the most similar to the reference, and then perform the physical screening only on that restrained pool of molecule. This is an enormous time saving, compared to the case where all the molecules have to be physically screened.

We can see that our work fits well in a research area which is very interesting for pharmaceutical companies. Indeed, by performing the operations above, the huge sets of molecules created by combinatorial chemistry are restrained to a set of good candidates, and the physical screening is only required on this smaller set of compounds. Our main contribution in the context of drug discovery is to improve existing similarity measures between graphs (representing molecules) and to create new ones. Ideally, a measure of similarity should meet two conditions: firstly it should be computationally efficient, because it is important when handling large molecular databases, and secondly the measure of similarity should separate molecules considering the activity at interest. Finding such similarity measures is one of the current challenges in chemoinformatics and bioinformatics, and any improvement in this domain could lead to enormous savings for pharmaceutical companies.

2 State-of-the-art

This section introduces some classical graph kernels: the marginalized graph kernel, which was conceived by Kashima et al. [4], and is one the most widely used kernel in the graph community, the shortest path kernel [1] and the Weisfeiler-Lehmann kernel [7]. These three methods serve either as inspiration for the new kernels that are described in the next section, or as baselines for the experiments.

2.1 Marginalized Kernel

The main idea behind this kernel is to perform random walks on labelled graphs (where both nodes and edges are labelled). While a random walker goes through a graph, he produces sequences of labels as he traverses nodes and edges, like the one hereunder:

\[(A, b, C, a, D, e, C)\]

\[(A, C, D)\] being node labels, and \((a, b, e)\) being edge labels. This algorithm extracts all the label sequences from a graph, which can be numerous, especially when the size of the graph increases. The length of these walks can also be infinite, if there are loops in the graph. We can already foresee a major drawback of this kernel, namely the scalability. All those sequences from a graph \(G_1\) are then compared to the ones of a graph \(G_2\). More precisely, the node label \(n_{1,G_1}\) from \(G_1\) is compared to node label \(n_{1,G_2}\) of \(G_2\), then edge label \(e_{1,G_1}\) is compared to \(e_{1,G_2}\), etc. A Dirac kernel is used to evaluate the similarity of those elements, so
the result is one if the labels are the same (on nodes or edges), and zero if the labels are different:

\[ K = \delta(n_{1,G_1}, n_{1,G_2}) \]

Applied to two toy sequences \((A, b, A)\) and \((A, b, C)\), this would give:

\[ K = \delta(A, A) \cdot \delta(b, b) \cdot \delta(A, C) = 1 \cdot 1 \cdot 0 = 0. \]

The label sequences are also weighted by their probability of appearance. Indeed, if a random walk is performed on a graph, each node in that graph has a probability \(p_s\) that this walk starts at this particular node, a transition probability \(p_t\) of going from that node to a neighbour node, and a termination probability \(p_q\), which represents the probability that the walk ends at this node. If there is no prior information about these probabilities, \(p_s\) can be set to the uniform distribution, \(p_t\) is the inverse of the degree of the initial node (before transition), and \(p_q\) is a parameter that can be tuned, to favor short or long walks. The weighted similarities between two walks are then summed over all sequences of the two graphs, and this gives us a global similarity score:

\[
K(G, G') = \sum_{l=1}^{\infty} \sum_{w_{G_1}} \sum_{w_{G_2}} p_s(w^{G_1}_{i}) \prod_{i=1}^{l} p_t(w^{G_1}_{i+1|w^{G_1}_{i-1}}) p_q(w^{G_1}_{i}) \cdot p_s(w^{G_2}_{i}) \prod_{i=1}^{l} p_t(w^{G_2}_{i+1|w^{G_2}_{i-1}}) p_q(w^{G_2}_{i}) \\
\cdot K(n_{w^{G_1}_{1}}, n'_{w^{G_2}_{1}}) \prod_{k=2}^{l} K(e_{w^{G_1}_{k-1}, w^{G_1}_{k-1}}, e'_{w^{G_2}_{k-1}, w^{G_2}_{k-1}}) K(n_{w^{G_1}_{k}}, n'_{w^{G_2}_{k}}),
\]

where \(w^{G_1}\) is a walk in \(G_1\), \(w^{G_2}\) is a walk in \(G_2\), \(w_i\) is the \(i\)th vertex in the sequence, and \(n, e\) are the labels on the nodes and edges. The enumeration of all the label sequence is unfortunately very computationally expensive, and the pair comparison between all the sequences from two graphs makes the time to compute this kernel grows strongly. Indeed, the complexity of this kernel is \(O(n^6)\), where \(n\) is the number of nodes (if \(G_1\) and \(G_2\) have the same number of nodes). This makes this kernel not tractable with databases containing more than a few hundred molecules.

A modification of the marginalized kernel [9], brought the complexity down to \(O(n^3)\), thanks to the use of direct graph product, and the resolution of a linear equations system. Although more efficient, this version of the marginalized kernel is still limited to be applied to relatively small graphs.

2.2 Shortest Path Kernel

The first step of this kernel consists in transforming an original graph \(G\) into a “Shortest-Path (SP) graph” \(G^{SP}\). This new graph has the same set of vertices (or nodes) than the original one. The set of edges is redefined: there is an edge
between two vertices if there was a path connecting them in the original graph. The labels on the new edges are the shortest-path distances between the nodes in $G$. If the edges of the original graphs do not bear any label which correspond to a distance, the shortest path between two nodes can simply be considered as the minimum number of edges between these two nodes. The all-pairs shortest paths can be computed using a Floyd-Warshall algorithm, which has a complexity of $O(n^3)$.

Then each 1-walk (made of an edge, the two corresponding vertices, and their labels) of a graph $G_{SP}^1$ is compared to those of graph $G_{SP}^2$, using, for example, a Dirac kernel, as in the marginalized kernel. As the shortest path distances are real numbers, a gaussian (RBF) kernel can be used to compare them:

$$k = \exp \left( -\frac{\|d^G_1 - d^G_2\|^2}{2\sigma^2} \right),$$

where $d^G_1$ is a SP distance in graph $G_1$ and $d^G_2$ is a SP distance in graph $G_2$. The comparison between two walks, which provides a similarity measure, is the product between the node kernels and the edge kernel. It will give a zero result if one (or both) of the pairwise node labels are different, or if the distances are the same. The sum of the comparison on all the 1-walks is used as a similarity measure between these two graphs. The comparison of all the SP walks can be performed in $O(n^4)$, which is global complexity of the SP kernel. Indeed, after the transformation to shortest path graphs, the latter are complete. And if both graphs have $n$ nodes each, then the pairwise comparison of all the SP distances can only be done in $n^4$ operations.

### 2.3 Weisfeiler-Lehman Kernel

The Weisfeiler-Lehman kernel, developed in [7], is inspired by a test of isomorphism between two graphs, the aptly-named Weisfeiler-Lehman test. The key idea of this algorithm is to concatenate the label of a node with the ones of its neighbours, to obtain a label representing the local structure of the graph around that node. The first step of this test is to list all types of nodes present in the graph, and to associate them to an integer, via a hash table for instance. Each new label is then aggregated with its nearest neighbours, to form a new list of labels, representing the local neighbouring of the nodes. At each iteration, the size of the neighbourhood increases, taking into account wider and wider subgraph structures. When the respective lists of labels of the two graphs differ, then the test is able to say that the two graphs are not isomorphic. If the set of labels is still the same after $n$ iterations, the algorithm cannot tell if the graphs are isomorphic or not.

This isomorphism test was transformed to a graph kernel simply by counting the number of common transformed labels between two graphs. Each graph is represented as a vector, in which each element represent the number of occurrence
of a newly created label type. This vector is updated at each iteration of the algorithm, which goes a step forward to the next “ring” of neighbours. In the end, one obtains a matrix $M$ in which each row represent a graph, each column a type of “concatenated” label, and each element of $M$ is the number of times a label occurs in the graph. The global measure of similarity between two graphs can be obtained by using a linear kernel, which amounts to multiply the matrix $M$ with its transposed matrix.

This kernel’s runtime complexity is of $O(hm)$, where $h$ is the number of iterations, i.e. the size of the neighbourhood, and $m$ is the number of edges. It is one of the most accurate and fastest kernel known in the literature, and it is used for comparison throughout the different experiments we performed.

3 Experiments

This chapter describes the datasets used to assess the quality of the kernels developed, as well as the procedure applied for this evaluation.

3.1 Datasets

Four datasets are used during the experiments. They all contain molecular graphs stored as adjacency matrices, adjacency lists and node labels lists, and a binary vector encoding the activity of a molecule concerning a particular task. The first one is the MUTAG dataset. It contains 188 molecular graphs related to molecules that can present a mutagenic activity on bacterium *Salmonella Typhimurium*. The second one is the PTC MR dataset, which contains 344 molecules resulting from pharmaceutical experiments on anticancer activity. The original dataset included experiments on male and female rats and mice, but we only used the male rat (MR) part here. This dataset is known to be hard to classify. The third dataset is the NCI1 one and contains about four thousands graphs also representing compounds possessing an anticancer activity or not. And at last, the D&D (Dobson & Doig) dataset is composed of about one thousand protein structures. Here, the nodes of the molecular graphs represent amino acids. There is a link between two nodes if they are sufficiently close (less than 6 Ångstroms). We will try here to predict if a protein is an enzyme or not. Below is a recapitulative table including the most important informations about the datasets:

<table>
<thead>
<tr>
<th>Name</th>
<th>MUTAG</th>
<th>PTC</th>
<th>NCI1</th>
<th>D&amp;D</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of graphs</td>
<td>188</td>
<td>344</td>
<td>4110</td>
<td>1178</td>
</tr>
<tr>
<td>Max number of nodes</td>
<td>28</td>
<td>64</td>
<td>111</td>
<td>5748</td>
</tr>
<tr>
<td>Average number of nodes</td>
<td>18</td>
<td>14</td>
<td>30</td>
<td>284</td>
</tr>
</tbody>
</table>

Fig. 6: Characteristics of datasets
3.2 Experimental setup

We assessed the behaviour of our kernels on these four datasets with a binary classification task.

Each dataset includes a target vector indicating if a compound present a desired activity or not. We performed the classification using a support vector machine (SVM) from the Matlab version of LIBSVM. These results were compared with some classical graph kernels found in the literature, representing diverse families: the marginalized kernel (based on walks), the shortest-path kernel (based on paths), and the Weisfeiler-Lehman kernel (based on the identification of local structures present in the graph). Double cross-validation is used: the datasets are divided in ten folds, one remains for the purpose of test (called the test set) while the other nine folds, forming the training set, are used for training. The training set was again divided into ten folds, and cross-validation is performed in order to identify the combination of parameters that gives the best accuracy results. Those parameters, i.e. C from the SVM, \( \sigma \) from the RBF kernel, . . . , are then used on the external test set. This double cross-validation process is repeated ten times, in order to reduce the variability of the results, due to the small sizes of the datasets, in particular MUTAG and PTC.

4 New Kernels

This section is devoted to the presentation of some new kernels defined for comparison of graphs. It does not cover the comparison of nodes belonging to the same graph but the comparison of whole different graphs. In order to compare the performance of the various kernels, real life datasets, referenced by many authors in the literature (see section 3), are used.

4.1 Modified Weisfeiler-Lehmann Kernel

As a reminder, the original Weisfeiler-Lehman kernel computes the global similarity between two graphs simply by performing a dot product between the vectors representing the graphs (linear kernel). This is quite efficient, as all the vector products can be done in one pass, multiplying the global matrix with its transposed matrix. But one can wonder if the comparison between the label vectors can be improved. One idea to refine this process is to consider the vectors as frequency distribution of labels, per type of label. Each element of those vectors would represent the frequency of apparition of a label in a graph. It then becomes possible to compare these distribution wihth well-known tools, as the Kullback-Leibler divergence, or its symmetrized version, the Jensen-Shannon divergence:

\[
JS(p_1, p_2) = H\left(\frac{p_1 + p_2}{2}\right) - \frac{1}{2}\left(\frac{H(p_1)}{2} + \frac{H(p_2)}{2}\right),
\]

where \( p_1 \) and \( p_2 \) are two probability measures, and \( H \) is the Shannon entropy, defined by:
\[ H(X) = - \sum_{i=1}^{n} P(X = x_i) \log(P(X = x_i)) = - \sum_{i=1}^{n} p_i \log(p_i). \]

For a discrete random variable \( X \), comporting \( i \) symbols, \( p_i \) is the probability of appearance of object \( x_i \), here a particular label. We still have to transform that divergence into a similarity measure, or a kernel, as shown in [6]:

\[ k^{JS}(p_1, p_2) = \ln(2) - JS(p_1, p_2). \]

We modified the original WL kernel in that direction, computing the Jensen-Shannon divergence for all pairs of graphs, and converting the results into a kernel.

4.2 Randomized Shortest Path Kernel

The first idea that led to this kernel is similar to the one used in the shortest paths kernel [1]. In that paper, the initial graphs are transformed to shortest paths graphs, using the Floyd-Warshall algorithm, for instance. Our idea consists in the use of another distance than the SP one, by introducing some entropy into the computation of those distances. This idea was inspired by a paper of Yen et al. [11], in which a new dissimilarity between the nodes of a graph is introduced, generalizing the shortest path and the commute-time distances. This algorithm lets, for instance, a traveller explores the graph and discovers new “roads”, instead of going straight to another node with a shortest path distance. This concept is also helpful in domains where artificial intelligence is needed, as chess game. In order to avoid too straightforward strategies from the computer, one can introduce some entropy, or disorder, to create more unpredictable decisions.

Our aim is therefore to adapt the dissimilarity measure presented above (applied for ongoing pairs of nodes), to obtain a distance measure between pairs of graphs, by using the same methodology as Borgwardt. For each dataset, we therefore transform original graphs into “randomized shortest path” (RSP) ones, computing the all-pairs RSP distances on these graphs. The “amount” of entropy injected in the calculation is controlled by the parameter \( \theta \), according to the desired degree of randomness. When \( \theta = 0 \), the distances reduce to the commute-time distances, and when \( \theta \) is large, they tend to shortest-path distances. The resulting graphs are complete, since every node is connected to every other by an edge labelled with the RSP distance. Thus, a list of distances is obtained for each graph, which has to be compared in order to obtain the global similarity between two graphs. In [1], this is done by comparing “1-walks” (see section 2) between them.

Since quite computationally demanding, computing all the pairwise randomized shortest paths between two graphs has to be avoided. The distances are therefore discretized in order to produce frequency distributions of distances. At first, all the edge labels existing in a dataset, per type on nodes labels, are
collected. Then, based on the percentiles, the boundaries of each distance class are determined, and applied to each RSP graphs. Distributions of frequencies concerning the distances, one for each type of nodes label, are thus obtained. These distributions are weighted by the a-priori probabilities to find a type of node label in the whole dataset. Finally, the Jensen-Shannon kernel between two graphs is computed.

4.3 Graph Invariants Kernel

As real life graphs can count several thousands of nodes (and more), it is essential that a kernel present a good scalability. In this vein, the Graph Invariant (GI) Kernel assumes that a single, fast to compute, global characteristic of the graphs, could lead to a measure of similarity. The first characteristic to be checked is the Frobenius norm of the graphs’s adjacency matrices. A RBF kernel is used to compare those norms, and the classification is done via a SVM (LibSVM on Matlab). The results are surprisingly good on some datasets, since very close to those obtained by state-of-the-art methods. Moreover, the speed is clearly one of the strengths of this kernel, as it can be computed in linear time of the number of nodes. Note that this aspect could still be improved, since using a RBF kernel implies the comparison of all the pairs of feature vectors. Instead, using a linear kernel permits to compute all the similarities in one pass, using just a matrix product. Another advantage of the latter kernel is the absence of parameter to tune, unlike the RBF kernel, which depends on the $\sigma$ parameter. We then tried to obtain higher accuracy by adding other invariant characteristics of the graphs, as the number of nodes, edges, the density of the graphs, the degree distribution,... Only characteristics that could be computed in linear time were chosen and the final feature vector used in the experiments is composed of:

- the Frobenius norm of the graph’s Laplacian matrix:

$$\|L\|_F = \sqrt{\sum_{i=1}^{n} \sum_{j=1}^{n} |l_{i,j}|^2},$$

- the Manhattan norm:

$$\|L\|_1 = \sum_{i=1}^{n} \sum_{j=1}^{n} |l_{i,j}|,$$

- the number of nodes in the graph,
- the distribution of the degrees of the nodes,
- the distribution of the labels of the nodes,
- the density of the graph (number of edges of the graph divided by the number of edges of a complete graph with the same number of nodes),
- the minimum and maximum degree of the nodes in the graph.
As the optimal set of characteristics can be different from one dataset to another, we developped a simple feature selection algorithm. This algorithm computes the correlation, with a built-in Matlab tool, of a feature vector with the class label vector, for all the graph features. Then all the features that present a correlation coefficient that is above a certain threshold are selected to participate to the elaboration of the kernel matrix. The threshold is determined via cross-validation.

Moreover, as all the characteristics do not have the same order of magnitude, each of them is normalized separately, by removing the mean and dividing by the standard-deviation of each feature vector.

## 5 Results

The results hereunder are the accuracy (percentage of good classification), and total computation time, obtained when applying double cross validation, except for the * cases, where a double CV was not performed, because of the slowness of the kernel.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>MUTAG</th>
<th>PTC</th>
<th>NCI1</th>
<th>D&amp;D</th>
</tr>
</thead>
<tbody>
<tr>
<td>WL Mod Kernel</td>
<td>88.19</td>
<td>59.36</td>
<td>86.03</td>
<td>80.31</td>
</tr>
<tr>
<td>RSP Kernel</td>
<td>87.09</td>
<td>56.23</td>
<td>63.78*</td>
<td>77.75*</td>
</tr>
<tr>
<td>GI Kernel</td>
<td>89.69</td>
<td>60.72</td>
<td>68.43</td>
<td>79.10</td>
</tr>
<tr>
<td>WL Kernel</td>
<td>88.20</td>
<td>64.18</td>
<td>86.04</td>
<td>79.78</td>
</tr>
<tr>
<td>Marginalized Kernel</td>
<td>87.84</td>
<td>60.14</td>
<td>/</td>
<td>/</td>
</tr>
</tbody>
</table>

**Fig. 7: Accuracy in percents**

<table>
<thead>
<tr>
<th>Dataset</th>
<th>MUTAG</th>
<th>PTC</th>
<th>NCI1</th>
<th>D&amp;D</th>
</tr>
</thead>
<tbody>
<tr>
<td>WL Mod Kernel</td>
<td>24s</td>
<td>64s</td>
<td>2h04min</td>
<td>25min</td>
</tr>
<tr>
<td>RSP Kernel</td>
<td>10s</td>
<td>21s</td>
<td>1h16min</td>
<td>17h06min</td>
</tr>
<tr>
<td>GI Kernel</td>
<td>0.1s</td>
<td>0.2s</td>
<td>6s</td>
<td>9s</td>
</tr>
<tr>
<td>WL Kernel</td>
<td>6s</td>
<td>10s</td>
<td>7min</td>
<td>11min</td>
</tr>
<tr>
<td>Marginalized Kernel</td>
<td>50s</td>
<td>5min56s</td>
<td>&gt; 2days</td>
<td>&gt; 2days</td>
</tr>
</tbody>
</table>

**Fig. 8: Computation time**

Figure 7 shows that the results obtained by the modified WL kernel (WL Mod Kernel) are not significantly better than the original WL kernel, and the RSP kernel displays the worse results. The GI kernel provides the best results.
on the MUTAG dataset, is competitive on the D&D dataset but behaves not so well on the two last datasets.

Concerning the computation time (Figure 8), GI kernel is way fastest than the other kernels, even compared to the Weisfeiler-Lehman, which is one of the fastest graph kernel in the literature. This is a consequence of the simplicity of the features used in our kernel. WL Mod kernel is slower compared to the original WL one, because it does not use a linear kernel that computes the kernel matrix in one pass. Instead, it compares graphs pairwise, and is slower as the number of graphs increases. RSP kernel, at last, is plagued by the computation of the RSP distances, which increases as the size of the graph grows.

It should be noted that, while we were developing the GI kernel, we saw that Li et al. [5] had published a paper using almost the same idea, except that they use more complex features extracted from the graphs. Their results are thus better (accuracy), but the computational time is also slower than our GI Kernel.
References

[10] X. Yan and J. Han. gSpan: Graph-Based Substructure Pattern Mining. International Conference in Data Mining (ICDM 2002).