Graph Kernels and Support Vector Machines for Pattern Recognition

Léo Andéol*
Master DAC, Sorbonne Université
Paris, France

Supervised by: Prof. Hichem Sahbi

May 2019

Abstract

The problem of having a framework to classify graphs is becoming increasingly important in the era of data. The issue has been tackled since the beginning of the millennium and there have been significant progress. This report introduces all necessary knowledge to understand the topic and then reviews different graph kernels while focusing on a random walk kernel and its optimization. Some experiments are then conducted to verify the information given by the state of the art, and some attempts are made to accelerate the different methods to compute the kernel.

*leo.andeol@gmail.com

Contents

1 Introduction

introduction générale plus étendue : 1 et 1/2 page :donner un aperçu/résumé du rapport : un paragraphe par élément - ECRIRE A LA FIN

Nowadays the world is shifting towards a new era where data is the most valuable resource, and researchers are working on algorithms to exploit it the best possible way. Most research is currently focused on the so-called "big data", huge quantities of data which are very sparse and of poor quality. However, there also exist structured databases of good quality, in the form of graphs, this very form being the one studied throughout this project. Graphs can be very useful in various fields, but are partly known for their use in biology, as they can be used to represent proteins or other types of molecules.

Classification of such graphs is an important problem of pattern recognition which affects a lot of sectors such as medicine, biology, and more. It was studied since the beginning of the millennium and significant progress has been made : several methods were discovered and gave good results but met a big issue, the complexity of these methods. Indeed, they do not scale well to large graphs as their complexity tends to be several times polynomial. Since then, the main objective of research on this topic has been to either find more computationally efficient algorithms, or to find methods to accelerate the ones already in use.

The first part of this report will introduce all the necessary background knowledge which are graphs as already mentioned, but also Support Vector Machines (SVMs) which is a powerful classification algorithm, and kernels which are transformations of data usually used to increase the accuracy of SVMs. However, graphs being non-vector data, kernels have also the purpose of becoming a metric of comparison between two instances of graphs. Thus, the main principles of graph kernels will be introduced as well as their definitions. Afterwards, the different methods used to improve the computation complexity of the kernel will be introduced together with their own complexities, advantages and disadvantages.

The second part of this report will be focused on the technical side and experiments conducted during the project. A synthetic graph database made of toy data was required to conduct simple and quick experiments. Indeed, it was made in order to verify the claims made in the main publication studied thanks to easy control on matters such as labeled and unlabeled graphs, and variations of the size of either the database or the graphs inside of it. Different challenges and problems met during the implementation will be explained, as well as their solution, if one has been found. Then, the main subject of this part will be discussed : the accuracy and computation time of different methods, estimation of their complexities and different attempts to approximate those methods. Finally, an experiment on a real database of proteins is conducted, and the results analyzed.

voir si j'en fait plus

2 Methodology

2.1 Background

2.1.1 Graphs

Revoir toutes les refs

Definition A graph[1] is a type of mathematical structure that represents connections between objects. It is more precisely an ordered pair $G = (V, E)$ of two sets: vertices V (or nodes) and edges E that connect two vertices together.

$$E \subseteq \{(u, v) : (u, v) \in V^2\} \tag{1}$$

Vertices represent objects and are usually depicted as circles or spheres whereas edges link pairs or vertices.

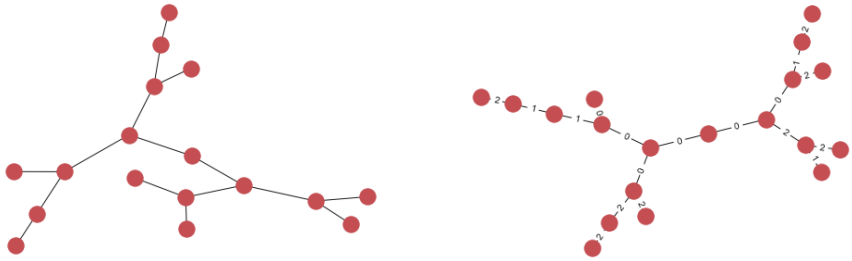

Figure 1: Two "tree" graphs, resp. unlabeled and labeled

Properties Graphs have a lot of properties that will be used in the rest of this project and will be introduced here

- A graph if undirected if and only if : $\forall (u, v) \in E \implies (v, u) \in E$
- $|V|$ is the number of vertices and $|E|$ the number of edges
- The degree of a vertex $d(v)$ is the number of vertices it is connected to, or the number of distinct edges connected to it.
- A path is a sequence of connected edges linking distinct vertices.
- A cycle is a path where the first and last vertex is the same.
- A graph can have labels (sometimes called colors) either on its vertices or edges (or both). they can take various forms such as being integers, more generally elements of a finite or infinite set and even continuous (such as $\in \mathbb{R}$).

- A graph is said to be connected if all vertices are connected by a path with any other vertex.
- A graph is said to be a tree if it is connected and doesn't have any cycle.
- A dual graph $G' = (V', E')$ of a graph $G = (V, E)$ is composed of a set of vertices V' of a new vertex for each unique $e \in E$ and a set of edges E' where there is an edge $(u, v) \in V'^2$if and only if there was a vertex connecting the two former edges u and v together.
- A subgraph $G' = (V', E')$ of a graph $G = (V, E)$ is that graph restricted to a subset of nodes $V' \in V$ while keeping all the edges with those vertices $E' = \{(u, v) : (u, v) \in E \wedge (u, v) \in V'^2\} \subseteq E$

Our methods are focused on undirected unlabeled graphs, and undirected graphs labeled by finite sets. Moreover, only edge-labeled graphs will be studied thus if a graph if node-labeled, simply taking its dual graph will do.

Adjacency Matrix The adjacency matrix of an undirected graph represents the presence or absence of an edge between two specific vertices. It is defined as the matrix A of dimension $|V| \times |V|$ where entries are given by

$$A_{i,j} = \begin{cases} 1 & \text{if } i \neq j \text{ and } (i, j) \in E \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

History Graphs were first used in their modern form to represent the problem of Seven Bridges of Königsberg (cite), and have been ever since used to represent maps, and thus path-finding algorithms were developed. They have also been used to represent flow problems, scheduling problems, networking routing and many others(cite). Graphs can also be used, with labels, to represent different types of molecules and interactions between them, or more simply to represent molecules by considering atoms as vertices and bonds as edges.
The next part will introduce Support Vector Machines, the algorithm used in this project to classify new unknown graphs into known categories. It has been shown(cite) that it can be used effectively on proteins and enzymes.

2.1.2 Support Vector Machines

Support Vector Machines (SVMs)[2] are a type of machine learning algorithms discovered in the early 90s[3]. It was originally a classification algorithm however it has been expanded since to regression and clustering too.

Classification Classification is one of the two problems of supervised learning that aims to automatically recognize and classify observations $\mathbf{x}$, such as recognizing handwritten digits. It can be split into two tasks where the first task is to learn from training data how to decide in which class (or group) to classify an example from its attributes and the second is to predict the class

of new examples. All of that, while minimizing errors. This problem can be formalized as

Parler de la theorie de l'apprentissage ? risque structurel etc ? et dire quelle loss on utilise ou pas ?

Definition 1. Classification is the problem of finding the best function $f : \mathbb{R}^d \longrightarrow \{0..k\}$ among a set of functions F while minimizing a risk function R

$$f\star = \text{argmin}_f R(f) \tag{3}$$

where k is the number of classes of the problem and d the number of features of data

SVMs are especially powerful at this task and widely used for the two following reasons.

Margin and support vectors The Support Vector Machines are based on the model of the Perceptron[4], another classification algorithm that tried to find an hyperplane that discriminates the two sets.The main issue with the Perceptron is that it had no formal guarantee to find an optimal hyperplane, and more often than not would not find it. Moreover, if the data weren't linearly separable, the algorithm would not converge.
In order to tackle theses issues, the SVM offered three fixes. First, a margin was added in the loss L function in order to not only classify well the data samples, but also with the maximum certainty, i.e. as far from the decision boundary as possible.

$$L(y_i, \mathbf{x}_i, w) = \max(0, -y_i \mathbf{x}_i \cdot w + w_0) \implies L(y_i, \mathbf{x}_i, w) = \max(0, 1 - y_i \mathbf{x}_i \cdot \mathbf{w} + w_0) \tag{4}$$

Where x is the features vector of an instance, y its class and $\mathbf{w}$ and w_0 respectively the weight vector and bias calculated by the algorithm. Then, seeing this solution wasn't enough, as there was still an infinity of possible solutions, it was proven that the margin γ between points and the decision boundary was inversely proportional to the norm of the weights.

$$\gamma_i = \frac{y_i(\mathbf{x}_i \cdot \mathbf{w} + w_0)}{\|\mathbf{w}\|} \tag{5}$$

These two fixes gave us the first version the SVM, the hard-margin SVM

$$\min \frac{1}{2}\|\mathbf{w}\|^2 \quad \text{s.t.} \quad y_i(\mathbf{x}_i \cdot \mathbf{w} + w_0) \geq 1 \quad \forall i \in \{1..n\} \tag{6}$$

However, if the data weren't linearly separable the algorithm wouldn't be optimizable since the quadratic programming problem requires all points to be correctly classified. Then, a new term ξ was introduced as an error tolerance, as well as a factor C that would determine the balance between error tolerance and weights minimization.

$$\begin{aligned} \min \quad & \tfrac{1}{2}\|\mathbf{w}\|^2 + C \sum_{i=1}^{n} \xi_i \\ \text{s.t.} \quad & \forall i \in \{1..n\} \qquad y_i(\mathbf{x}_i \cdot \mathbf{w} + w_0) \geq 1 - \xi_i \end{aligned} \tag{7}$$

Then, the next section will introduce the second advantage of SVMs.

2.1.3 Kernels

In its dual form, the SVM problem only requires a dot product between the sample vectors.

$$\begin{aligned} \max \quad & \sum_{i=1}^{n} \alpha_i - \tfrac{1}{2} \sum_{j=1}^{n} \sum_{i=1}^{n} \alpha_i \alpha_j y_i y_j \mathbf{x_i}^\top \mathbf{x_j} \\ \text{s.t.} \quad \forall i \quad & 0 \leq \alpha_i \leq C \\ & \sum_{i=1}^{n} \alpha_i y_i = 0 \end{aligned} \tag{8}$$

Where n is the number of instances. Then, a nonlinear transformation $\phi(x)$ can be used to replace the dot-product $\mathbf{x}_i \cdot \mathbf{x}_j$ by $\phi(\mathbf{x_i}) \cdot \phi(\mathbf{x}_j)$, effectively augmenting the size of our vectors and thus the expressiveness of our model by enhancing the separability of the data. However, for complex transformations, such as in infinite dimensions, the function ϕ isn't defined. But it was found that the standard dot product can be replaced by another function as long as it is a bilinear positive semi definite form.

Definition 2. A kernel K is positive semi definite if and only if

$$\sum_{i=1}^{n} \sum_{i=1}^{n} \kappa(\mathbf{x_i}, \mathbf{x_j}) c_i c_j \geq 0 \qquad \forall i \in \{1..n\} \quad c_i \in \mathbb{R} \tag{9}$$

Thus, the dot product can be replaced by a function $K(x_1, x_2)$ which will indirectly map the data to higher dimensions, even infinite with the RBF kernel and return the dot product of those transformations, while remaining computable in all cases. This replacement is usually called the Kernel Trick. Ever since the foundation of SVMs, the kernel trick became a big focus of the machine learning community as it naturally fits in the algorithm and allows supervised learning on very complex data, and enjoying greater accuracy than most algorithms.
The fact that the dot product can be replaced by a function without specifying the map ϕ is the reason why the Kernel Trick became a big focus of research. In particular, it will be possible to make a kernel for non vector data such as graphs, as shown in the next section.

Est-ce qu'il faut rentrer dans le détail ? rajouter quoi?

2.2 State of the Art

This section will review the state of the art kernels for graphs. Particular attention will be given to the random walk kernel.

2.2.1 Graph Kernels

Graph Kernels are a type of R-convolution kernels[5] applied to graphs, which are kernels that are based on decompositions of complex objects and comparisons of those decompositions.
Graph Kernels have been studied since Support Vector Machines started getting popular[6]. Since then, a lot of progress has been made, and several types of

kernels have been discovered, such as random walk and graphlet[7] kernels, the two main families of kernels (and the ones studied in this section), but not the only ones. There are also graph kernels that use subtrees[8], shortest paths[9] and several others that are unfortunately usually either too complex to compute or not positive semi-definite[10]. However it has been shown[11] that in some cases non positive semi definite kernels can still be used efficiently either by using them as a dissimilarity measure. They can also be artificially made positive semi definite.

developper plus les autres graphes?

2.2.2 Graphlets

A graphlet is a non-empty graph of $k = 3$, 4 or 5 nodes. The idea of the graphlet kernel κ is to generate a set S all possibles graphlets of a certain size k and calculate for each of them the frequency of occurrence in a certain graph $G = (V, E)$ to build a vector $\mathbf{f}_G$ of size $|S|$. The kernel is then defined as follows

Definition 3. Let G and G_2 be two graphs, $\mathbf{f}_G$ and $\mathbf{f}_{G_2}$ the frequency vectors of respectively G and G_2, then the kernel κ is defined as

$$\kappa(G, G_2) = \mathbf{f}_G^\top \mathbf{f}_{G_2} \tag{10}$$

However the complexity of this calculating a vector $\mathbf{f}_G$ is $O(n^k)$ where $n = |V|$ and is thus computationally extremely expensive. In order to address this issue, a sampling method was introduced. Indeed, if enough samples are drawn, the Law of Large numbers indicates that the sampled distribution will converge to the actual (theoretical) distribution. However a bound has been found on the complexity of such sampling[12] allowing better precision.

2.2.3 Random walks

Graph kernels based on Random Walks have been studied since very early[6], several of them were created and were then united[13]. Random walks are a type of rather intuitive algorithms : the idea is to randomly walk through a graph and then compare it to random walks in another graph. Actually, all the random walks are computed at the same time by making the adjacency matrix stochastic by dividing each column of the matrix by its sum (normalizing it) and using it as a Markov chain.
Moreover, it has also been shown[14] that performing a walk on two separate graphs at the same time is the same as performing a walk on the product graph. The product graph is in simpler terms the Cartesian product of the two graphs, taking all possible combinations of nodes and linking those nodes when there is an edge between two specific nodes in both the original graphs.

Definition 4. Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be two graphs, the product graph $G_\times = (V_\times, E_\times)$ is then defined as

- $V_\times = \{(v, w) : v \in V_1, w \in V_2\}$

- $E_\times = \{((v_1, w_1), (v_2, w_2)) : v_1, v_2 \in V_1^2, w_1, w_2 \in V_2^2\}$

However, technically, the adjacency matrix $W_\times$ of a such graph is the result of the Kronecker (tensor) product of the two adjacency matrices A and A_2 of the two graphs [15]

$$W_\times = A \otimes A_2 \tag{11}$$

In case the graphs are labeled, then the adjacency matrix $W_\times$ is the sum of the Kronecker products of adjacency matrices restricted to each label l

$$W_\times = \sum_{l=1}^{d} A_1^{(l)} \otimes A_2^{(l)} \tag{12}$$

Where d is the size of the label set and $A_x^{(l)}$ the adjacency matrix of G_x restricted to the label li which is defined as follows

$$A^{(l)} = \begin{cases} 1 & \text{if } i \neq j \text{ and } (i,j) \in E \text{ and } (i,j) \text{ is labeled } l \\ 0 & \text{otherwise} \end{cases} \tag{13}$$

The *vec* operator will also be used as it is very linked to the tensor product. It is the vector obtained by stacking all columns of A.

Kernel definition The idea of the kernel is to perform random walks simultaneously on two graphs in order to compare the walks they have in common. As shown above, that can be achieve by simply performing a random walk on the product graph. By making the adjacency matrix stochastic, and computing the k-th power of the matrix $W_\times$, the probabilities obtained are the ones of walks in common (starting and ending at the two same nodes) between the two graphs of length k. By taking into account the start $p_\times$ and end $q_\times$ probabilities, the sum of paths of all possible length can be calculated. Moreover using a power series $\mu(k)$ helps de-emphasizing longer walks and makes the sum converge.

Definition 5. Let $p_\times$ and $q_\times$ be respectively the start and end probability of each node, and let $W_\times$ be the adjacency matrix of the product graph of G and G', and finally $\mu(k)$ be a convergent function of k.

$$\kappa(G, G') = \sum_{k=0}^{\infty} \mu(k) q_\times^\top W_\times^k p_\times \tag{14}$$

This kernel can be computed in $O(kn^6)$ where k is the number of iterations

Inverse Kernel There is a specific case of the kernel following provided a power series is used. Indeed, by replacing the power series as follows $\mu(k) = \lambda^k$ the following kernel is obtained

$$\kappa(G_1, G_2) = q_\times^\top (I - \lambda W_\times)^{-1} p_\times \tag{15}$$

It can also be computed in $O(n^6)$ however in practice the hidden factor is much lower which makes it a better option for very small graphs and databases and the accuracy much better since there isn't any tradeoff on the number of iterations. Moreover, it will inspire some of the following methods.

Sylvester Equation A Sylvester equation, also sometimes called Lyapunov equation is a matrix equation of the following shape

Definition 6. Let A, B, and C be matrices of compatible shapes, then the Sylvester equation is

$$AX + XB = C \tag{16}$$

And the discrete-time Sylvester Equation is

$$AXB + C = X \tag{17}$$

Which can be generalized as

$$\sum_{i=0}^{d} A_i X B_i = X \tag{18}$$

We will be exclusively using the discrete-time Sylvester Equation and its generalization and will be referring to it as Sylvester equation. These equations are solved usually using Schur decompositions in $O(n^3)$ for the basic Sylvester equation and in unknown time for the generalized version[13].
We can use this equation to solve our kernel faster if we assume a geometric function $\mu(k) = \lambda^k$ with λ an hyperparameter. It can be achieved by replacing the A and B matrices by our adjacency matrices A_1 and A_2, adding λ, and C by our start probabilities $p_\times$ while vectorizing the whole equation thus obtaining

$$vec(M) = vec(\lambda A_1 M A_2) + p_\times \tag{19}$$

From which we can obtain (where I is the identity matrix)

$$q_\times^\top vec(M) = q_\times^\top (I - \lambda W_\times)^{-1} p_\times \tag{20}$$

Thus, having calculated the inverse kernel by an alternative method. The advantage of this method is that it is very fast, but unfortunately limited to unlabeled graphs, unless there is an implementation for the generalized Sylvester equation. Moreover, compared to the others, this option is very simple and does not require specific parameters, but like many others, suffers when it encounters singular matrices.

Conjugate Gradient Method The Conjugate Gradient Method[16] is an optimization algorithm used to find approximate solutions of linear systems that have a positive semi definite matrix. As its name suggests, it is a gradient based iterative algorithm. The Main idea is that normal gradient descent only takes into account the gradient at the current step of the algorithm, and depending on the step size may cancel progress made in the previous step. The Conjugate Gradient method was introduced as a fix for that issue. The idea of the algorithm is to keep the former gradients in order improve the convergence speed by making the new gradient orthogonal to all the former one through the Gram-Schmidt process, at each step. Thus, unless there are approximation

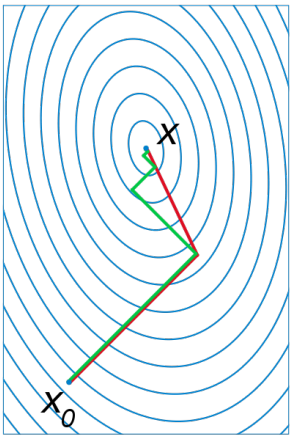

Figure 2: Conjugate gradient (red) compared to Gradient Descent (green)

errors, the algorithm guarantees converge in n steps, being the number of dimensions of the problem, here the size of our graph.
However, this algorithm has limitations, it can only be used to solve linear systems of equations as $Ax = b$ under the constraint that A is symmetric positive definite. In our case, this algorithm can be used to solve the following problem

$$(I - \lambda W_\times x = p_\times \tag{21}$$

And the kernel value can be obtained by computing $q_\times^\top x$. The complexity of that method is $O(rn^3)$ for unlabeled graphs and $O(rdn^3)$ for labeled graphs where r is the effective rank of the matrix $W_\times$. It is important to note that this kernel also assumes a geometric $\mu(k)$ function.
This method is thus more slow than the Sylvester Equation, however it is easily applied to labeled graphs, which is its main advantage.

Fixed Point Iterations A fixed point is a value for which a function, returns that same value

Definition 7. Let S a set and f a map $f : S \implies S$,
$x \in S$ is a fixed point of f if and only if $x = f(x)$

Fixed point iterations is a method of computing a fixed point of a function by applying repeatedly the following equation until $\|x_{n+1} - x_n\| < \epsilon$ where ϵ is the acceptable level of error.

$$x_{n+1} = f(x_n) \tag{22}$$

There is also a guarantee of convergence

Theorem 1. *In order for the fixed-point iterations to converge, it is necessary that $\lambda < |\xi|^{-1}$ where ξ is the largest eigenvalue of $W_\times$*

le mettre en appendix?

Proof. Let $x_0 = p_\times$ and $t >> 0$

$$\begin{aligned}
x_{t+1} - x_t &= p_\times + \lambda W_\times x_t - x_t \\
&= p_\times + (\lambda W_\times - 1)(p_\times + \lambda W_\times x_{t-1}) \\
&= p_\times - p_\times + \lambda W_\times p_\times + (\lambda W_\times)^2 x_{t-1} - \lambda W_\times x_{t-1} \\
&= \lambda W_\times (p_\times - x_{t-1} + \lambda W_\times x_{t-1}) \\
&= \lambda W_\times (p_\times - \lambda W_\times x_{t-2} - p_\times + \lambda W_\times (\lambda W_\times x_{t-2} + p_\times) \\
&= \lambda W_\times (\lambda W_\times (\lambda W_\times x_{t-2} + p_\times - x_{t-2})) \\
&= (\lambda W_\times)^2 (\lambda W_\times x_{t-2} + p_\times - x_{t-2})
\end{aligned}$$

We find a pattern

$$\implies x_{t+1} - x_t = (\lambda W_\times)^t (\lambda W_\times x_0 + p_\times - x_0)$$

And because $x_0 = p_\times$

$$= (\lambda W_\times)^{t+1} p_\times$$

Which decreases only when

$$\xi_1 < 1$$

The largest magnitude eigenvalue of$\lambda W_\times$

$$\implies \lambda < |\xi_1|^{-1}$$

□

Since this method requires the computation of $W_\times$, the kernel value can be computed for any type of labeling. For unlabeled graphs, the complexity is $O(kn^3)$ and $O(kdn^3)$ for labeled graphs, where d is the number of labels and k the number of iterations which can be estimated by

$$k = O\left(\frac{\ln \epsilon}{\ln \lambda + \ln |\xi|}\right) \tag{23}$$

Finally, the fixed point iterations is more demanding than the others since it has a condition on the value of λ and thus assumes as geometric $\mu(k)$ function, and the number of iterations can easily vary a lot. The only advantage of this method is that it is very simple while still being better than the raw kernel.

Spectral Decomposition The Spectral Decomposition, usually called the eigendecomposition of a matrix M is a factorization in the form VDV^{-1} where D is the diagonal matrix of eigenvalues and V the matrix of eigenvectors. This decomposition can reduce the computation time of the kernel by making most of the matrix operations trivial

$$\kappa(G, G_2) = \sum_{k=0}^{\infty} \mu(k) q_\times^\top (V_\times D_\times V_\times^{-1})^k p_\times = q_\times^\top V_\times \left(\sum_{k=0}^{\infty} \mu(k) D_\times^k\right) V_\times^{-1} p_\times \tag{24}$$

Indeed, the power now only applies to a diagonal matrix which can be computed much faster. Moreover, this method unlike the others preserves the prior $\mu(k)$

without any conditions. If $\mu(k) = \lambda^k$ is chosen then the kernel becomes

$$\kappa(G, G_2) = q_\times^\top V_\times (I - \lambda D_\times)^{-1} V_\times^{-1} p_\times \tag{25}$$

That kernel is even more trivial to compute as the inverse of a diagonal matrix is simply the inverse of each of its entries and is thus computed in linear time. Finally, by using $\mu(k) = \frac{\lambda^k}{k!}$ as prior, the following new kernel is obtained

$$\kappa(G, G_2) = q_\times^\top V_\times e^{\lambda D_\times} V_\times^{-1} p_\times \tag{26}$$

However, the complexity of the computation of the eigendecomposition is as large as the raw kernel $O(n^6)$, but, for unlabeled graphs, a big improvement is possible by simply calculating the eigendecompositions of the original adjacency matrices A_1 and A_2 thus cutting the time to $O(n^3)$, thanks to a property of the Kronecker product

$$A_1 \otimes A_2 = (V_1 D_1 V_1^{-1}) \otimes (V_2 D_2 V_2^{-1}) = (V_1 \otimes V_2)(D_1 \otimes D_2)(V_1 \otimes V_2)^{-1} \tag{27}$$

By rewriting the spectral decomposition kernel taking advantage of this property, the following new kernel is obtained

$$\kappa(G_1, G_2) = (q_1^\top V_1 \otimes q_2^\top V_2)(\sum_{k=0}^{\infty} \mu(k)(D_1 \otimes D_2)^k)(V_1^{-1} p_1^\top \otimes V_2^{-1} p_2^\top) \tag{28}$$

Which can also be altered by choosing specific $\mu(k)$ as described above. This method ends up being the most efficient of all for unlabeled graphs as it is computed in $O(pn^3)$ where p is the computation time of the power series μ. Moreover this complexity can further improve the computation time of the gram matrix because the eigendecomposition of an adjacency matrix only needs to be computed once. However, this method cannot be expanded to labeled graphs as it encounters difficulties applying the property mentioned above.

Nearest Kronecker Product Approximation This method is more particular as it is not a new way to compute the kernel, but rather a way to generalize all the other methods. Indeed the Nearest Kronecker Product Approximation[17] is used to approximate two matrices A and B as if their Kronecker product was equal to the Kronecker product of two labeled graphs' adjacency matrices $W_\times \approx A \otimes B$. This methods allows using any of the methods mentioned above since it returns only two matrices and thus brings back the problem to unlabeled graphs. This method tries to minimize the Frobenius norm of the difference between the two $\|W_\times - A \otimes B\|_F$. It can be computed in $O(dn^2)$ since $W_\times$ is a sum of Kronecker products.
However this method also brings its own set of drawbacks. Since it required the Kronecker product of each couple of adjacency matrices to be computed, the previous advantage of the Spectral Decomposition (only computing once the eigendecomposition for each adjacency matrix) is lost as the obtained matrices are unique for each $W_\times$.

2.2.4 Digest

remplir

3 Experiments

experiences : détailler db, tests, méthodes, les parametres, construction label, donner tableaux, resultats, teps de calculs, précision, discuter tout cela

A big part of this project consisted in implementing the different algorithms from the main paper that I had studied [13], verifying the results obtained in that article, and making new attempts to accelerate those algorithms while keeping the best accuracy possible.

3.1 Implementation

Implementing code from the main paper was a challenge in itself. Several problems aren't really documented and finding functions to solve them, or even explanations on how to solve them is sometimes difficult.

3.1.1 Graphs

The first thing that had to be done was to create a graph database generator, of very simple and standard graphs that had expectable behaviors and could thus be used for tests. Those graphs would also be altered using simple transformations, in order to create different graphs belonging to the same class.
For simplicity, it was decided to only use 3 classes to stay as simple as possible.

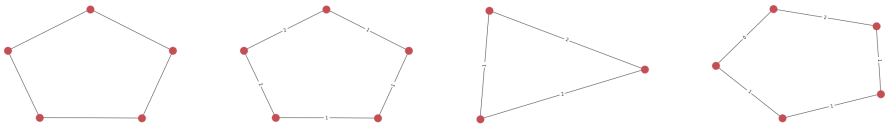

Figure 3: Ring graphs, resp. unlabelled, labelled, altered structured, altered labels

We studied 3 standard types of graphs : the ring, the star and the tree. A ring graph is a connected graph where each vertex is connected to exactly two vertices, it thus forms one big cycle. A star graph is composed of a central node, and of several other nodes that are all and only connected to the central node, it should look like a star, or perhaps a flower.. Finally, a tree is a famous type of graph : it is a connected graph without any cycles, but here we will be studying a special type of tree, since each vertex is at most of degree 3.

Est-ce que je devrais dire ça ?

The database generator works as follows: for each type of graph, it will create a graph of predetermined size with randomly generated edge labels in a certain given interval. Then, for each graph, it will alter it a predefined number of times, and each time will create a new graph that has been both altered on structure and on labels. Alteration on structure involves removing or adding a

verifier si anglais, et c'est ok mettre deux points ?

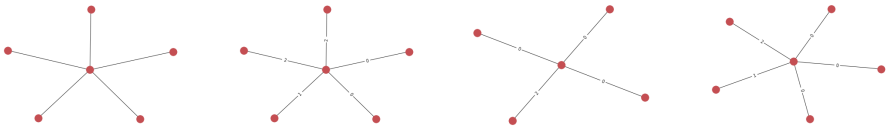

Figure 4: Star graphs, resp. unlabelled, labelled, altered structured, altered labels

predefined number of nodes in respect of the type of the graph thus staying in the same class as the source.

1. Ring graphs are simply regenerated slightly longer or shorter randomly.
2. Star graphs are also regenerated either with more or less nodes, randomly.
3. Tree graphs are either expanded by adding randomly leaves anywhere in the graph, or reduced by randomly removing leaves.

Alteration on labels will randomly switch a predefined number of edge labels. Once this is done, the generator will create two databases out of the set of previously generated graphs, one will be made of the adjacency matrix of each graph without looking at labels, and the second one will be an array of adjacency matrices taken from graphs induced by selecting only edges with a specific label, and that for all labels in the label set.

predefined beaucoup répété, c'est un probleme ?

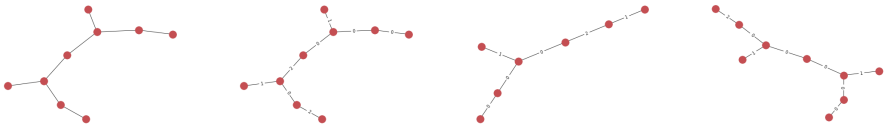

Figure 5: Tree graphs, resp. unlabelled, labelled, altered structured, altered labels

3.1.2 Raw Kernel

3.1.3 Inverse Kernel

3.1.4 Sylvester Equation

The Sylvester Equation is a good example of poorly documented problems (compared to the others). There are very few resources on the subject, and even less implementation of solvers. We could theoretically use the Sylvester Equation on both labeled and discrete-labeled graphs which are respectively represented as Sylvester and Generalized Sylvester Equations (involving a sum of matrices), however, the only library available solves the first one, but not the second

one. There seems to be confusing terminology since there is a solver for the Generalized Sylvester Equation, but a different one than the one described in our source[13]. The same source also mentions a way to solve the generalized version[18], however, this option wasn't explored.

correct ?

changé ça ?

Rajouter papier à citer

3.1.5 Conjugate Gradient Method

Note : Preconditioner random vector : $x \cdot x^T$ fonctionne bien
The Conjugate Gradient Method is however very well documented[16] and there are several libraries implementing this algorithm (we will be using scipy).

a faire

3.1.6 Fixed Point Iterations

Rapidité de la convergence : géométrique avec un ratio $\lambda 1/\lambda 2$??? si lambda trop proche de l'inverse de la valeur propre peut etre tres lent à converger quand la matrice d'adjacence est très dense
The Fixed Point Iterations method was very easy to implement and gave rapidly good results, however there is a constraint on the lambda(the geometric factor) used in the kernel as we have proved. Experiments showed that a lambda very close to the constraint increases significantly the time of computation to convergence.

image

3.1.7 Spectral Decompostion

can't inverse Left eigenvectors : `https://arxiv.org/pdf/1708.00064.pdf`

3.1.8 Nearest Kronecker Product

`https://www.sciencedirect.com/science/article/pii/S0377042700003939?via%3Dihub` `https://www.imageclef.org/system/files/CLEF2016_Kronecker_Decomposition.pdf` `http://dx.doi.org/10.1007/978-94-015-8196-7_17` `https://mathematica.stackexchange.com/questions/91651/nearest-kronecker-product`

3.2 Test of the random walk kernel

3.2.1 Accuracy comparison between labelled and unlabelled graphs

Même score, car donne même matrice ???

	Unlabelled	Labelled
Accuracy	?	?
Time	?	?

Table 1: Time and Accuracy of learning resp. for unlabelled and labelled graphs

	Raw kernel	Inverse Kernel	Sylvester Equation	Conjugate Gradients	Fixed points	Spectral Decomp.	Nearest Kronecker Product
Accuracy	67%	67%	?	67%	67%	?	?
Time	18.40s	5.72	?	7.62s	5.94s	?	?

Table 2: Time and Accuracy of learning for the raw kernel and other methods

3.2.2 Efficiency of alternate methods

3.2.3 Experiments on a biology dataset

`https://ls11-www.cs.tu-dortmund.de/staff/morris/graphkerneldatasets`
`http://members.cbio.mines-paristech.fr/~nshervashidze/code/` enzymes

3.3 Comparison of kernels

3.3.1 Label use

Comparison using labels and no labels

3.3.2 Comparison of their gram matrices

The algorithms approximate the kernel using very different techniques, thus the gram matrices obtained from the kernels were similar in appearance, but had significant differences of scale. It was then decided to normalize very simply the gram matrices so they could be more easily compared.

$$M = (M - min(M))/max(M) \tag{29}$$

Afterwards, in order to verify all the algorithms gave similar results, it was

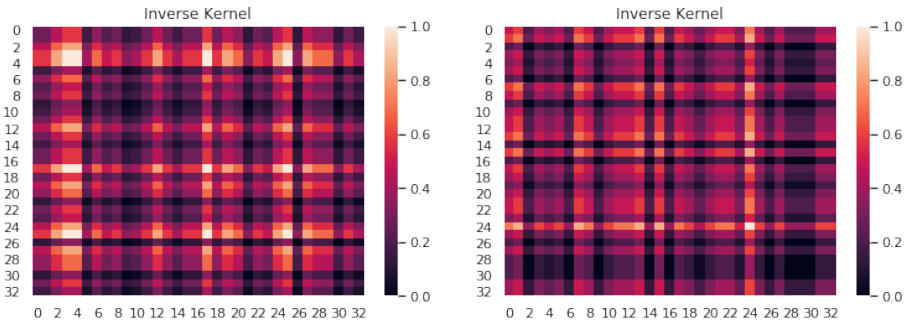

Figure 6: Two gram matrices computed using the Inverse Kernel on different datasets

decided to compute the Frobenius norm of the difference of two gram matrices (divided by the size of the matrix, thus giving the mean standard deviation). The following results were obtained and were satisfying. Indeed, the

	Raw kernel	Inverse Kernel	Sylvester Equation	Conjugate Gradients	Fixed points	Spectral Decomp.
Raw.	0	1.1e-4	9.8e-5	8.9e-5	1.0e-4	1.0e-04
Inv.	-	0	2.1e-5	7.9e-5	4.0e-6	6.8e-6
Syl.	-	-	0	8.0e-5	1.7e-5	1.4e-5
Con.	-	-	-	0	7.9e-5	7.9e-5
Fix.	-	-	-	-	0	2.8e-6
Spe.	-	-	-	-	-	0

Table 3: Approximate standard deviation of matrix entries (?)

correct nom ?

3.3.3 Complexity and Accuracy

3.3.4 title

3.4 Individual kernel analysis ?

mettre ici les images variations degré d'approximation

3.5 Nearest Kronecker product ?

3.6 On molecules

3.7 Improvements

4 Conclusion and Future Work

experiences : détailler db, tests, méthodes, les parametres, construction label, donner tableaux, resultats, teps de calculs, précision, discuter tout cela section 4 : 1 page ou page et demi : conclusion et discussion

A Appendix

B Annex 1

C Acknowledgements

This work was done during the first year of my master.

List of Figures

List of Tables

References

[1] J. A. Bondy, U. S. R. Murty, *et al.*, *Graph theory with applications*, vol. 290. Citeseer, 1976.

[2] C. J. C. Burges, "A Tutorial on Support Vector Machines for Pattern Recognition," *Data Min. Knowl. Discov.*, vol. 2, pp. 121–167, June 1998.

[3] C. Cortes and V. Vapnik, "Support-vector networks," *Machine Learning*, vol. 20, pp. 273–297, Sept. 1995.

[4] Y. Freund and R. E. Schapire, "Large margin classification using the perceptron algorithm," *Machine learning*, vol. 37, no. 3, pp. 277–296, 1999.

[5] D. Haussler, "Convolution kernels on discrete structures," Technical Report UCS-CRL-99-10, University of California at Santa Cruz, Santa Cruz, CA, USA, 1999.

[6] H. Kashima, K. Tsuda, and A. Inokuchi, "Marginalized kernels between labeled graphs," in *Proceedings of the Twentieth International Conference on International Conference on Machine Learning*, ICML'03, pp. 321–328, AAAI Press, 2003.

[7] N. Shervashidze, S. V. N. Vishwanathan, T. Petri, K. Mehlhorn, and K. Borgwardt, "Efficient graphlet kernels for large graph comparison," in *Artificial Intelligence and Statistics*, pp. 488–495, Apr. 2009.

[8] J. Ramon and T. Gärtner, "Expressivity versus efficiency of graph kernels," Citeseer.

[9] K. M. Borgwardt and H.-P. Kriegel, "Shortest-path kernels on graphs," in *Fifth IEEE international conference on data mining (ICDM'05)*, pp. 8–pp, IEEE, 2005.

[10] N. Shervashidze, *Scalable graph kernels*. PhD thesis, Universität Tübingen, 2012.

[11] B. Schlkopf and A. J. Smola, *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. Cambridge, Mass: The MIT Press, 1st edition ed., Dec. 2001.

[12] T. Weissman, E. Ordentlich, G. Seroussi, S. Verdu, and M. J. Weinberger, "Inequalities for the l1 deviation of the empirical distribution," *Hewlett-Packard Labs, Tech. Rep*, 2003.

[13] S. V. N. Vishwanathan, N. N. Schraudolph, R. Kondor, and K. M. Borgwardt, "Graph Kernels," *Journal of Machine Learning Research*, vol. 11, no. Apr, pp. 1201–1242, 2010.

[14] W. Imrich and S. Klavzar, *Product graphs: structure and recognition*. Wiley, 2000.

[15] P. M. Weichsel, "The kronecker product of graphs," *Proceedings of the American mathematical society*, vol. 13, no. 1, pp. 47–52, 1962.

[16] Y. Nesterov, *Lectures on Convex Optimization*. Springer Optimization and Its Applications, Springer International Publishing, 2 ed., 2018.

[17] C. F. Van Loan and N. Pitsianis, "Approximation with kronecker products," in *Linear algebra for large scale and real-time applications*, pp. 293–314, Springer, 1993.

[18] L. De Lathauwer, B. De Moor, and J. Vandewalle, "Computation of the canonical decomposition by means of a simultaneous generalized schur decomposition," *SIAM Journal on Matrix Analysis and Applications*, vol. 26, no. 2, pp. 295–327, 2004.