

# Graph Kernels and Support Vector Machines for Pattern Recognition

**Léo Andéol\***

Master DAC, Sorbonne Université  
Paris, France

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.andéol@gmail.com

# Contents

<b>1</b>	<b>Introduction</b>	<b>4</b>
<b>2</b>	<b>Methodology</b>	<b>5</b>
2.1	Background . . . . .	5
2.1.1	Graphs . . . . .	5
2.1.2	Support Vector Machines . . . . .	6
2.1.3	Kernels . . . . .	7
2.2	State of the Art . . . . .	7
2.2.1	Introduction . . . . .	7
2.2.2	Graphlets . . . . .	8
2.2.3	Random walks . . . . .	8
2.2.4	Kernel definition . . . . .	9
2.2.5	Sylvester Equation . . . . .	9
2.2.6	Conjugate Gradient Method . . . . .	10
2.2.7	Fixed Point Iterations . . . . .	10
2.2.8	Spectral Decomposition . . . . .	11
2.2.9	Nearest Kronecker Product Approximation . . . . .	11
2.3	Contributions . . . . .	11
<b>3</b>	<b>Experiments</b>	<b>11</b>
3.1	Implementation . . . . .	11
3.1.1	Graphs . . . . .	11
3.1.2	Raw Kernel . . . . .	13
3.1.3	Inverse Kernel . . . . .	13
3.1.4	Sylvester Equation . . . . .	13
3.1.5	Conjugate Gradient Method . . . . .	13
3.1.6	Fixed Point Iterations . . . . .	13
3.1.7	Spectral Decomposition . . . . .	13
3.1.8	Nearest Kronecker Product . . . . .	14
3.2	Test of the random walk kernel . . . . .	14
3.2.1	Accuracy comparison between labelled and unlabelled graphs . . . . .	14
3.2.2	Efficiency of alternate methods . . . . .	14
3.2.3	Experiments on a biology dataset . . . . .	14
3.3	Comparison of kernels . . . . .	14
3.3.1	Label use . . . . .	14
3.3.2	Comparison of their gram matrices . . . . .	14
3.3.3	Complexity and Accuracy . . . . .	15
3.3.4	title . . . . .	15
3.4	Individual kernel analysis ? . . . . .	15
3.5	Nearest Kronecker product ? . . . . .	16
3.6	On molecules . . . . .	16
3.7	Improvements . . . . .	16
<b>4</b>	<b>Conclusion and Future Work</b>	<b>16</b>

<b>A</b>	<b>Appendix</b>	<b>16</b>
<b>B</b>	<b>Annex 1</b>	<b>16</b>
<b>C</b>	<b>Acknowledgements</b>	<b>16</b>

# 1 Introduction

Nowadays the world is turning to a new era where data is the most valuable resource, and we are working on algorithms to exploit those the best way possible. 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, which is the one we will be studying. Graphs can be very useful in various fields, but is partly known for its use in biology, as it 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 start of the millennium and significant progress was made : several methods were discovered and gave good results but met a big issue, the complexity of those methods. 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 that is graphs as already mentioned, but also Support Vector Machines (SVMs) which is the classification algorithm we will be using 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 objective 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 on to verify the claims made in the main publication studied, in different cases such as labeled and unlabeled graphs, or varying the size of either the database or the graph studied. 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.

introduction  
générale plus  
étendue : 1  
et 1/2 page  
: donner un  
aperçu/résumé  
du rapport  
: un para-  
graphe par  
élément -  
Ecrire à  
la fin

voir si j'en  
fait plus

## 2 Methodology

### 2.1 Background

#### 2.1.1 Graphs

Revoir  
toutes les  
refs

**Definition** A graph(cite) 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.

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

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

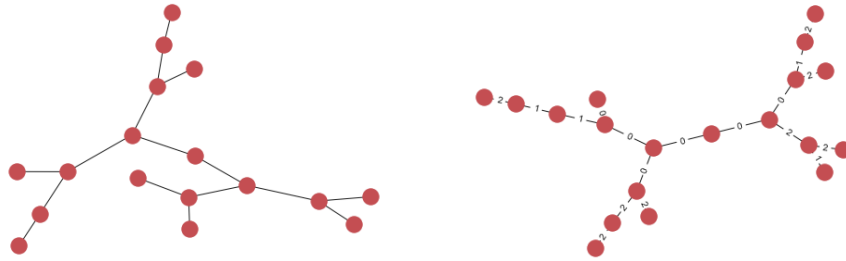


Figure 1: Two "tree" graphs, resp. unlabeled and labeled

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

- A graph is undirected if and only if :  $\forall (u, v) \in E \implies (v, u) \in E$
- 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 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  $(u, v) \in E'$  if and only if there was a vertex connecting the two former edges  $u$  and  $v$  together.

We will be focusing on undirected unlabeled graphs, and undirected graphs labeled by finite sets. However, we will only study edge-labeled graphs thus if a graph is node-labeled, simply taking its dual graph will do.

**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.

In the next part we will be introducing Support Vector Machines which is 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 (SVM) are a type of machine learning algorithms discovered in the early 90s (cite). It was originally a classification algorithm however it has been expanded since to regression and clustering too. It is especially powerful and widely used for two main reasons :

**Margin and support vectors** The Support Vector Machines were based on the model of the Perceptron (cite), 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 these issues, the SVM offered three fixes.

First, they added a margin in the loss 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.

$$L(y, x, w) = \max(0, -y * x \cdot w) \implies L(y, x, w) = \max(0, 1 - y * x \cdot w) \quad (2)$$

Then, seeing this solution wasn't enough, as there was still an infinity of possible solutions, it was proven that the margin between points and the decision boundary was inversely proportional to the norm of the weights.

$$\gamma = \frac{y_i(x_i \cdot w + w_0)}{\|w\|} \quad (3)$$

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

$$\min \frac{1}{2} \|w\|^2 \quad \text{s.t.} \quad y_i(x_i \cdot w + w_0) \geq 1 \quad \forall i \in \{1..n\} \quad (4)$$

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  $\xi$  was introduced as an error tolerance, as well as a factor  $C$  that would determine the balance between error tolerance and weights minimization.

$$\begin{aligned} \min \quad & \frac{1}{2} \|w\|^2 + C \sum_{i=1}^n \xi_i \\ \text{s.t.} \quad & \forall i \in \{1..n\} \quad y_i(x_i \cdot w + w_0) \geq 1 - \xi_i \end{aligned} \quad (5)$$

### 2.1.3 Kernels

In its dual form, the SVM problem only requires a dot product between the sample vectors. Then, we can use a nonlinear transformation  $\phi(x)$  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 dimensions 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  $\phi$  isn't defined. But it was found(cite) that the standard dot product can be replaced by another function as long as it is a bilinear positive semi definite form.

**Definition 1.** A kernel  $K$  is positive semi definite if and only if :

$$\sum_{i=1}^n \sum_{j=1}^n K(\mathbf{x}_i, \mathbf{x}_j) c_i c_j \geq 0 \quad \forall i \in \{1..n\} \quad c_i \in \mathbb{R} \quad (6)$$

Thus, the dot product can be replaced by a function  $K(x_1, x_2)$  which will indirectly (?) transform the data in larger dimensions, even infinite with the Gaussian kernel(cite) 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.

## 2.2 State of the Art

### 2.2.1 Introduction

In this chapter we will discuss the advance of research on graph kernels. Graph Kernels are a type of R-convolution kernels[2] applied to graphs, which are

Mettre explication comme "ici nous allons parler de ça ça " ou pas

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[3]. Since then, a lot of progress has been made, and several types of kernels have been discovered, such as random walk[3] and graphlet[4] kernels, the two we will be studying, but not the only ones. There are also graph kernels that use spanning trees, and several others that are unfortunately usually not positive semi-definite[5].

découverts  
ou inventés?

### 2.2.2 Graphlets

A graphlet[4] is a subgraph of 3,4 or 5 nodes taken from a much larger graph. The idea of a graphlet kernel is to count all types of graphlets that appear in a graph, and use that count to compare to another graph.  
insert list of graphlets

A faire plus  
tard

Citer deux  
fois un seul  
article ok ou  
pas?

### 2.2.3 Random walks

Graph kernels based on Random Walks have been studied since very early[3], several of them were created and were then united[6]. 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 and using it as a Markov chain. Moreover, it has also been shown[7] 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 2.** 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 [7, 8]:

$$W_{\times} = A \otimes A_2 \quad (7)$$

We will also need the *vec* operator that is very linked to the tensor product :

**Definition 3.** Let  $A$  be a matrix  $n \times n$ ,  $vec(A)$  is the vector obtained by stacking all columns of  $A$ .



### 2.2.4 Kernel definition

finir

The kernel is defined as follows

**Definition 4.** Let  $p_{\times}$  and  $q_{\times}$  be respectfully 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  $k$ .

$$k(G, G') = \sum_{k=0}^{\infty} \mu(k) q_{\times}^{\top} W_{\times}^k p_{\times} \quad (8)$$

### 2.2.5 Sylvester Equation

A Sylvester equation, also sometimes called Lyapunov equation (we will stick to Sylvester here) is a matrix equation of the following shape :

**Definition 5.** Let  $A$ ,  $B$ , and  $C$  be matrices of compatible shapes, then the Sylvester equation is

$$AX + XB = C \quad (9)$$

And the discrete-time Sylvester Equation is

$$AXB + C = X \quad (10)$$

Which can be generalized as

$$\sum_{i=0}^d A_i X B_i = X \quad (11)$$

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[6].

We can use this equation to solve our kernel faster by replacing the  $A$  and  $B$  matrices by our adjacency matrices  $A_1$  and  $A_2$ , adding  $\lambda$ , and  $C$  by our start probabilities  $p_{\times}$  while vectorizing the whole equation thus obtaining :

$$vec(M) = vec(\lambda A_1 M A_2) + p_{\times} \quad (12)$$

colon or not?

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

$$q_{\times}^{\top} vec(M) = q_{\times}^{\top} (I - \lambda W_{\times})^{-1} p_{\times} \quad (13)$$

partie speciale pour notations?

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.

donner des exemples = comme quoi

sous section 2 : state of the art : graph kernels et acceleration des random walk, donner les complexité, donner des exemples, discuter avantages et inconvenients chacun

## 2.2.6 Conjugate Gradient Method

The Conjugate Gradient Method 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 (usually) iterative algorithm. The Main idea is assumes as geometric  $\mu(k)$  function

## 2.2.7 Fixed Point Iterations

A fixed point is a value for which a function, returns that same value :

Faire usage de fct paragraph ??

**Definition 6.** Let  $S$  a set and  $f$  a map  $f : S \Rightarrow S$ ,  $x \in S$  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  $\epsilon$  is the acceptable level of error.

$$x_{n+1} = f(x_n) \quad (14)$$

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  $\xi$  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

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

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  $\lambda W_\times$

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

□

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:

$$k = O\left(\frac{\ln \epsilon}{\ln \lambda + \ln |\xi|}\right) \quad (15)$$

Finally, the fixed point iterations is more demanding than the others since it has a condition on the value of  $\lambda$  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.

verifier sur graphes

### 2.2.8 Spectral Decomposition

### 2.2.9 Nearest Kronecker Product Approximation

<https://mathematica.stackexchange.com/questions/91651/nearest-kronecker-product>  
<http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.42.1924&rep=rep1&type=pdf>[9]

## 2.3 Contributions

en anglais ça donne ?

## 3 Experiments

A big part of this project consisted in implementing the different algorithms from the main paper that I had studied [6], 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. 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

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

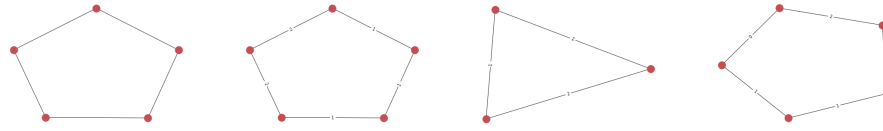


Figure 2: Ring graphs, resp. unlabelled, labelled, altered structured, altered labels

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.

The database generator works as follows: for each type of graph, it will create

Est-ce que je devrais dire ça ?

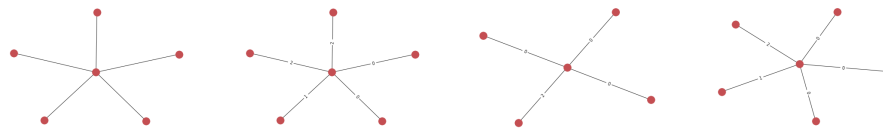


Figure 3: Star graphs, resp. unlabelled, labelled, altered structured, altered labels

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

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

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 ?

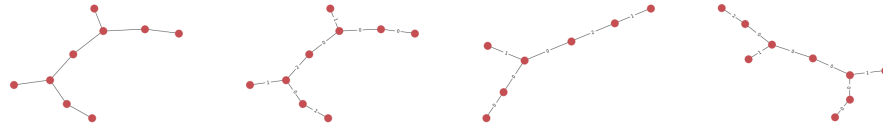


Figure 4: Tree graphs, resp. unlabelled, labelled, altered structured, altered labels

### 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 respectfully 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[6]. The same source also mentions a way to solve the generalized version[10], 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[11] 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 être très 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 Decomposition

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?](https://www.sciencedirect.com/science/article/pii/S0377042700003939?via%3Dihub)  
[via%3Dihub https://www.imageclef.org/system/files/CLEF2016\\_Kronecker\\_Decomposition.pdf](https://www.imageclef.org/system/files/CLEF2016_Kronecker_Decomposition.pdf) [http://dx.doi.org/10.1007/978-94-015-8196-7\\_17](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

### 3.2.2 Efficiency of alternate methods

	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

### 3.2.3 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.

$$M = (M - \min(M)) / \max(M) \quad (16)$$

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

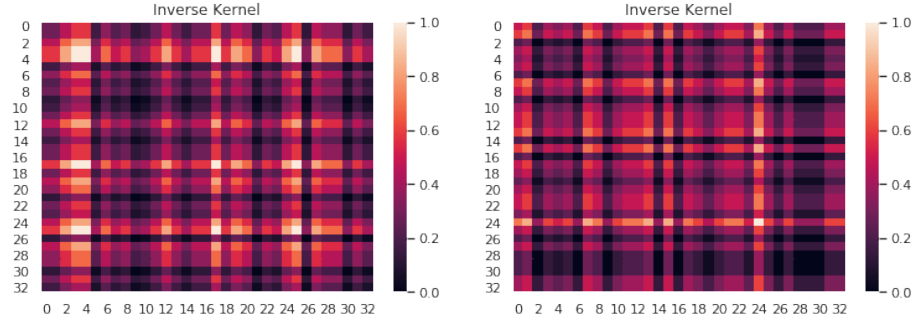


Figure 5: Two gram matrices computed using the Inverse Kernel on different datasets

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 (?)

### 3.3.3 Complexity and Accuracy

### 3.3.4 title

## 3.4 Individual kernel analysis ?

correct nom ?

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

1	Two "tree" graphs, resp. unlabeled and labeled . . . . .	5
2	Ring graphs, resp. unlabelled, labelled, altered structured, altered labels . . . . .	12
3	Star graphs, resp. unlabelled, labelled, altered structured, altered labels . . . . .	12
4	Tree graphs, resp. unlabelled, labelled, altered structured, altered labels . . . . .	13
5	Two gram matrices computed using the Inverse Kernel on different datasets . . . . .	15

## List of Tables

1	Time and Accuracy of learning resp. for unlabelled and labelled graphs . . . . .	14
2	Time and Accuracy of learning for the raw kernel and other methods	14
3	Approximate standard deviation of matrix entries (?) . . . . .	15

bibliographie et index

## References

- [1] Wikipedia, "Graph theory - Wikipedia," 2019.



- [2] D. Haussler, “Convolution kernels on discrete structures,” Technical Report UCS-CRL-99-10, University of California at Santa Cruz, Santa Cruz, CA, USA, 1999.
- [3] 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.
- [4] 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.
- [5] N. Shervashidze, *Scalable graph kernels*. Dissertation, Universität Tübingen, 2012.
- [6] 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.
- [7] W. Imrich and S. Klavžar, *Product graphs*. Wiley-Interscience Series in Discrete Mathematics and Optimization, Wiley-Interscience, New York, 2000. Structure and recognition, With a foreword by Peter Winkler.
- [8] Wikipedia, “Kronecker product,” Mar. 2019. Page Version ID: 887195556.
- [9] C. F. Van Loan and N. Pitsianis, *Approximation with Kronecker Products*, pp. 293–314. Dordrecht: Springer Netherlands, 1993.
- [10] 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.
- [11] Y. Nesterov, *Lectures on Convex Optimization*. Springer Optimization and Its Applications, Springer International Publishing, 2 ed., 2018.