TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter One

Context and related works

In this chapter, we first introduce graphs and how they arise from real world networks. Then we present the graph classification problem, along with applications in which it arises. Finally, we proceed to detailing state-of-the-art methods and their limitations, before stating our contribution.

1.1 Graph structures and its presence in real world

The need for graphs and their analysis can be traced back to 1679, when G.W. Leibniz wrote to C. Huygens about the limitations of traditional analysis methods of geometric figures and said that "we need yet another kind of analysis, geometric or linear, which deals directly with position, as algebra deals with magnitude", (**Graph_application**). This lead to graphs, mathematical objects that provide a pictorial and efficient form to represent data with many inter-connections.

Formally, a graph of size v is a pair $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \{u_1, ..., u_v\}$ is a set of v graph nodes (or vertices), and $\mathcal{E} \in \mathcal{V} \times \mathcal{V}$ is the set of edges between these nodes, i.e. $(u_i, u_j) \in \mathcal{E}$ means that the graph has an edge between node u_i and node u_j.

Graph structures are used to model a set of objects and their interactions/relations. While the nature of these objects and their interactions vary with the application, the underlying

Network	Nodes	Node features	Edges	Edge features
Transportation System	cities	registered cars	Routes	Length, cost
Banking Network	Account holders	account status	Transactions	Transaction value
Social Network	users	name, country	Interactions	type (like, comment)

Table 1.1 Some real world graphs

modeling paradigm is the same for all applications: objects are represented by nodes, and a relation between two objects is represented by an edge between the corresponding two nodes. **For instance, in a social network like Facebook, nodes are .. and edges are... In a biological network such as the brain, nodes are brain regions and edges are.. In a transportation network such as the subway, nodes are .. and edges are.. (see Table.1.1 for a list of different examples).

These graphs, if not too large, can be visually represented in order to provide an intuitive understanding of the existing interactions. Such an illustration is in Fig. 1.1 in the application of chemical reactions.

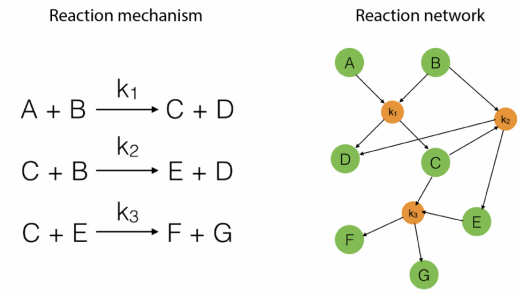

Figure 1.1 Graph structures in representing chemical reactions mechanisms

1.2 Graph classification problem

Graph classification can be understood in several ways. Here, we place ourselves in the context of *supervised learning*, where we suppose we have access to a set of pre-labeled graphs

$(\mathcal{X} = \{\mathcal{G}_1, \ldots, \mathcal{G}_n\}, \mathcal{Y} = \{y_1, \ldots, y_n\})$, where each graph $\mathcal{G}_i$ is *a priori* known to belong to the class with label y_i. Stated simply, the graph classification problem we are interested in in this work may be stated as: given this prior information, design a classification algorithm that, given in input any graph (importantly, any graph belonging or not to $\mathcal{X}$), outputs the label of the class to which it belongs.

More formally, consider the set $\mathcal{D}$ of all graphs $\mathcal{G}$ that can occur in some real-world application, a fixed set of classes $\beta = \{\beta_1, \ldots, \beta_l\}$ of finite size l, and a mapping function $f : \mathcal{D} \mapsto \beta$ which maps each graph $\mathcal{G}$ in $\mathcal{D}$ to the class $\beta_{\mathcal{G}}$ it belongs to. Graph classification is the problem of estimating the mapping function f in the case where it is only known on a subset $\mathcal{X} \subset \mathcal{D}$. Formally, we have a dataset $(\mathcal{X} = \{\mathcal{G}_1, \ldots, \mathcal{G}_n\}, \mathcal{Y} = \{y_1, \ldots, y_n\})$ of size n such that $\mathcal{X} \in \mathcal{D}^n$ and $\mathcal{Y} \in \beta^n$, where for each graph $\mathcal{G}_i \in \mathcal{X}$ we have that $y_i = f(\mathcal{G}_i)$ is the class of $\mathcal{G}_i$. The classification task is to have a **predictive model** which can predict well, based on some-predefined metric, the class for any new graph $\mathcal{G}$ in $\mathcal{D}$. This prediction functionality of the model is gained using the dataset $(\mathcal{X}, \mathcal{Y})$ to optimize the parameters of the model that is believed to govern the behavior of the mapping function f on $\mathcal{D}$. This optimization completed in this paradigm is called the learning algorithm.

Note that graph classification as considered here, has nothing to do with the more common problem of *node classification* in a graph, in which there exists only one graph and the goal is to separate the node set in a partition of communities. In our work, graphs are classified, not nodes. This being said, the extra information that the nodes and/or edges may have in some applications (gender, age for instance for nodes of a social network; maximum bandwidth, number of channels for instance for edges of a communication network; etc.) could in principle be used along with the graph structure to classify different graphs into different classes. However, as the existence of such extra-information is very application-dependent, we prefer to focus here on the case where nodes and edges do not carry such information: the only information one has access to for classification is the graph structure.

This problem has been addressed in many different fields of research, such as:

- ***Marketing analytics:*** advertisers and marketers are interested in detecting the influential people communities in Social Networks in the sense that addressing their products' advertisements to such groups would be a more rewarding investment. This can be approached with graph classification applied on these networks (**marketing_analytics**).

- ***Banking security:*** graph classification is used to catch unusual patterns of fraudulent transactions (**banking_security**).

- ***Biology and genomics:*** graphs are based on proteins such that nodes correspond to amino acids which compound the protein and a pair of amino acids are linked by an edge if they are less than 6 Angstroms apart. The task is to detect whether a protein is an enzyme or not (**protein_application**), to mention a few.

1.3 State-of-the-art methods for graph classification

We present here existing algorithms for the graph classification problem and discuss their limitations. In general, these algorithms can be classified in four main categories: set based, frequent sub-graph based, kernel based, and graph neural networks based algorithms.

Set based algorithms. This type of algorithms is only applicable to cases where nodes/edges are supplied with features or attributes, as they completely disregard the graph's structure. Based on the provided feature vectors, a distance function of interest between the graphs is computed. The drawback of this method is that it does not take the structure (topology) of the graph itself into consideration. For example, if we just compare how much the edges' features of one graph are similar to the edges' features of another, we can have two graphs with the same set of edge features, which will lead to maximum similarity, even though their graph structures can be arbitrarily different. On the other hand, a strength of these algorithms is their low computations cost that is usually linear or quadratic in the number

of nodes and edges (**graphlet_kernel**).

Frequent sub-graph based algorithms. These algorithms contain two steps. First, the graph dataset $\mathcal{X}$ is analyzed to enumerate the frequent sub-graphs occuring in the different graphs. Then, another analysis is done to choose the most discriminative sub-graphs out of the ones found during the first step. The disadvantage of using this method is the computational cost that grows exponentially with the graph size (**graphlet_kernel**).

Graph kernels based algorithms. It is a middle ground between both previous methodologies, where the graph structure is well considered, and in most cases, these algorithms are designed in a way that the computational time is a polynomial function of the graph size (**graphlet_kernel**). However, some effective and competitive kernels still require exponential time, and this is in short the problem we approach in this work using random features to approximate these kernels or to compete with them in notably lower computational time.

Graph neural networks (GNNs) based algorithms. GNNs compute a representation vector (embedding vector) for every node in a graph, where this vector is recursively computed by aggregating the representation vectors of neighboring nodes. The goal of this aggregation technique is that nodes that are neighbors (or close) to each other in the graph are more likely to have similar representations (with respect to some similarity function) and vice versa. On the graph level, a representation vector is computed by aggregating its nodes' representation vectors. This aggregated vector now representing the graph itself is used as a usual feature vector which can be fed to a typical deep neural network to learn the classification task. Traditional GNNs such as graph convolutional networks (GCNs) and GraphSAGE fail to provide high performance classifying graphs whose node/edges don't include any original feature vectors, and that even applies on graphs with simple topology (**GCN_powerful**). I could not re-write this last sentence: I don't understand it However,

another GNN structure was developed to overcome this weakness point complete failure is more than a "weakness", and it is referred to by Graph Isomorphism Network (GIN). Regarding the computational time, it is mainly a matter of the layers number in the network, since this parameter in reality represents how far from a node we want to go in order to compute its representation vector.

1.4 Our contribution

One of the methods of the kernel-based algorithms (the third out of the four categories listed), called graphlet kernel, has proven to be competitive for graph classification. Theoretically and empirically, it was shown that a desired performance or a required amount of information to be preserved from the original graph can be reached with sufficiently large k. **hold your horses! we need at least a few sentences actually *explaining* what is the graphlet kernel you are talking about. For instance, we have yet no clue what k is.** However, the computational cost becomes prohibitive as k (the graphlet size) and/or v (the size of the graph) become too large. Thus it cannot be applied on large-scale graph datasets.

The advent of Optical Processing Units (OPUs) opened a new horizon solving this problem, since it can apply enormous number of *Random Projections* in light speed. **hold your horses! The reader has no clue *why* making random projections in light speed is actually useful for your problem. You need at least one sentence explaining what you mean by random projections

In this work, we did the sufficient mathematical analysis to prove that OPUs' light-speed random feature projections compete the k-graphlet kernel with respect to *Maximum Mean Discrepancy (MMD)* Euclidean metric. Moreover, we empirically tested this hypothesis and made sure that the the theoretical MMD error is aligned with the empirical one with respect to the parameters introduced in the problem (sampling technique, number of sampled subgraphs, number of random features, etc). Instead of this last paragraph, I invite you to write

"Our contributions are the following:" followed by an "enumerate" environment to make a clear and thorough list of your achievements. After that enumeration, I invite you to write "On top of these contributions, we have also:" followed by an "enumerate" environment to make a list of things you did that we cannot call "contributions" yet as they either did not work or are work in progress.

Chapter Two

Background

In this chapter we present the necessary background that the reader should have in order to proceed through the next two chapters, which are the core of this work. After a brief overview of kernel methods in machine learning, we will present random features and graph kernels. In the next chapter, these different notions will be combined in the proposed algorithm.

2.1 Kernel methods and random features

We first start by an overview of kernel methods.

2.1.1 Kernel methods and kernel trick

Kernel methods is a family of classic algorithms in machine learning that learn models as a combination of similarity functions between data points (x, x'), defined by positive semi-definite (psd) *kernels* $\kappa(x, x')$. Denote by $\mathcal{D}$ the set of all possible data points. A symmetric function $\kappa : \mathcal{D} \times \mathcal{D} \mapsto \mathbb{R}$ is said to be a positive semi-definite kernel if:

$$\forall n \in \mathbb{N},\ \forall \alpha_1, \ldots, \alpha_n \in \mathbb{R},\ \forall x_1, \ldots, x_n \in \mathcal{D}, \quad \sum_{i,j}^{n} \alpha_i \alpha_j \kappa(x_i, x_j) \geq 0\,. \tag{2.1}$$

Let us now illustrate how kernels can be incorporated to learning models and how that is useful to learn more complex functions. To do that let us consider a classical supervised

learning setting for classification: take $\mathcal{D} = \mathbb{R}^d; d \in \mathbb{N}$, and let $\mathcal{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_n)$ be a set of n datapoints in $\mathbb{R}^d$, along with the vector of their associated known labels $\mathbf{y} = (y_1, \ldots, y_n)^T \in \mathbb{R}^n$ where, for each i, y_i is the label of the class to which $\mathbf{x}_i$ belongs. For simplicity, we set ourselves in the context of two classes, and we set $y_i = -1$ is $\mathbf{x}_i$ belongs to class 1, and $y_i = 1$ if $\mathbf{x}_i$ belongs to class 2.

The classification task is: given this *a priori* known labeled data $(\mathcal{X}, \mathbf{y})$, design a classifier that is able to classify any new data point in $\mathbb{R}^d$.

Many learning models designed to solve this problem, like Support Vector Machine (SVM) add ref and Perceptron binary classifier add ref, rely on the inner product as a measure of similarity between data points: during the training, they only use inner products $\mathbf{x}_i^T \mathbf{x}_j$, and then produce classifiers with the following form (**inner_product**):

$$\hat{y}(\mathbf{x}) = \text{sign}\left\{ \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i^T \mathbf{x} \right\} \text{ with } \alpha_i \in \mathbb{R} \tag{2.2}$$

where the values $\{\alpha_i\}_{i=1,\ldots,n}$ in Eq. 2.2 are optimized based on the dataset $(\mathcal{X}, \mathbf{y})$ by the learning algorithm. The intuition behind Eq. 2.2 is that the output class for every new data point $\mathbf{x}$ is expected to be the same class of nearby points in the input set $\mathcal{D}$. This is achieved by introducing the inner product $\mathbf{x}_i^T \mathbf{x}$ to control how much the class y_i contributes in the output $\hat{y}(\mathbf{x})$. The parameter α_i controls how strongly the data point x_i can affect other neighboring points. They mainly depend on how both classes are distributed in the input set $\mathcal{D}$, and how much the dataset $(\mathcal{X}, \mathbf{y})$ is noisy.

A *kernel method* consists in replacing every inner products $\mathbf{x}_i^T \mathbf{x}_j$ by a psd kernel $\kappa(\mathbf{x}_i, \mathbf{x}_j)$ during training, and similarly $\mathbf{x}_i^T \mathbf{x}$ by $\kappa(\mathbf{x}_i, \mathbf{x})$ during prediction. Let us now explain the intuition behind this, starting by rewriting Eq. 2.2 as

$$\hat{y}(\mathbf{x}) = \text{sign}\{\mathbf{x}^T (\mathbf{X}^T \; diag([\alpha]_{i=1}^n) \; \mathbf{y})\},$$

where $\mathbf{X} \in \mathbb{R}^{n \times d}$ is the matrix whose i-th row corresponds to data-point $\mathbf{x}_i$, and where $diag([\alpha]_{i=1}^n)$ is the diagonal matrix with values $[\alpha]_{i=1}^n$.

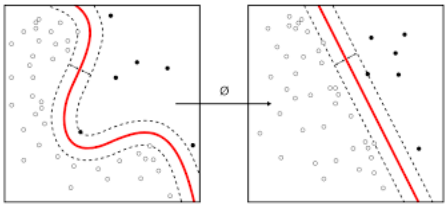

Figure 2.1 The left figure shows a case where the input data in their original space are not separable by a linear boundary. The right figure shows the same data transformed to a new space using a lifting function φ, and we can see that different classes are now separable using linear boundary. in fact, I would remove φ from this figure and only use this figure to show an example on the right of a linearly separable dataset; and an example on the left where it is not the case. Justifying the use of a mapping φ is the message of Fig. 2.2

To get the decision boundary of such classifiers (the boundary in $\mathbb{R}^d$ that separates $\mathbb{R}^d$ into a part associated to class 1 and another associated to class 2), one solves $\mathbf{x}^T\mathbf{q} = \mathbf{0}$, where $\mathbf{q} = (\mathbf{X}^T \; diag([\alpha]_{i=1}^n) \; \mathbf{Y}) \in \mathbb{R}^d$. It is the equation of a hyper-plane in the input space $\mathbb{R}^d$, also referred to as a *linear* decision boundary. The question here is: what if the two classes are not separable by a hyper-plane (as illustrated on the left of Fig. 2.1)?

One common solution to this problem is to map the data points from $\mathbb{R}^d$ to another space $\mathbb{R}^m$ through a proper mapping function φ such that the two classes become separable with a linear decision boundary in $\mathbb{R}^m$. Then we can apply the same learning models specified in Eq. 2.2 but on the transformed data. Let us consider for example the dataset shown in Fig. 2.2, we can use the following mapping function $\varphi : (x_1, x_2) \mapsto (\sqrt{2}x_1x_2, x_1^2, x_2^2)$ to move from $\mathbb{R}^2$ on the left, where data are not linearly separable, to $\mathbb{R}^3$ on the right, where they are.

Learning such a function φ is what is typically done by neural networks using complex optimization methods. Kernel methods are much simpler (and elegant) methods to perform this mapping. They are justified by the following key theorem.

Theorem 1 (Mercer theorem)**.** *To every positive semi-definite kernel* $\kappa : \mathbb{R}^d \times \mathbb{R}^d \mapsto \mathbb{R}$*, there exists a Hilbert space* $\mathbb{H}$ *and a feature map* $\phi : \mathbb{R}^d \mapsto \mathbb{H}$ *such that for all* $\mathbf{x}, \mathbf{x}' \in \mathbb{R}^d$ *one has::*

$$\kappa(\mathbf{x}, \mathbf{x}') =< \phi(\mathbf{x}), \phi(\mathbf{x}') >_{\mathbb{H}} \tag{2.3}$$

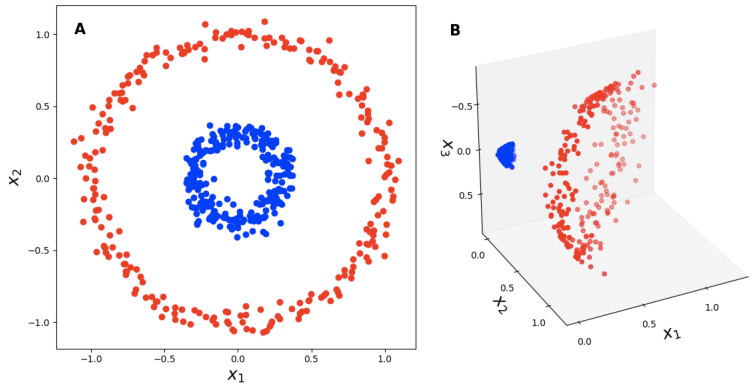

Figure 2.2 Using the mapping function $\varphi : (x_1, x_2) \mapsto (\sqrt{2}x_1x_2, x_1^2, x_2^2)$ to map the data on the left in $\mathbb{R}^2$ to $\mathbb{R}^3$ where they are linearly separable

where $< \phi(\mathbf{x}), \phi(\mathbf{x}') >_{\mathbb{H}}$ *is the inner product defined in* $\mathbb{H}$.

This theorem states that replacing the inner product $\mathbf{x}_i^T\mathbf{x}$ in Eq. 2.2 by a positive semi-definite kernel $\kappa(\mathbf{x}_i, \mathbf{x})$ is equivalent to implicitly map the data from the original input space $\mathcal{D}$ to another feature space $\mathbb{H}$ and then apply the classical inner product. Therefore, one *does not need to know explicitely the mapping* ϕ nor the new feature space $\mathbb{H}$, instead, it is sufficient to evaluate the kernel κ for pairs of data points in the original input space $\mathcal{D}$. This main feature of kernel methods is known as the *kernel trick*. It has two main advantages:

- Kernels allow us to transform data to a new Hilbert space of very high or even infinite dimensionality, which can make the learning model able to represent more complex functions.

- Kernels are often computationally cheaper, since they save the time required to compute the explicit co-ordinates of the data in the new feature space by directly calculating the inner product between the transformed data.

To better illustrate these benefits, we take the Gaussian kernel as an example, which is one of the most classical kernels in $\mathbb{R}^d$, defined as:

$$\kappa_G(\mathbf{x}, \mathbf{x}') = \exp^{-\frac{\|\mathbf{x}-\mathbf{x}'\|^2}{2\sigma^2}} \tag{2.4}$$

where $\sigma > 0$ is the bandwidth parameter of the kernel. The lifting function ϕ_G of this kernel is located in a Hilbert space of infinite dimension add ref, but the kernel can be easily evaluated for any pair $(\mathbf{x}, \mathbf{x}') \in \mathbb{R}^d \times \mathbb{R}^d$.

Despite their advantages, kernel methods still have some drawbacks, notably in the computation of the Gram matrix. The Gram matrix associated to a given kernel κ and a given set $\mathcal{X}$ of n datapoints in $\mathbb{R}^d$, is a matrix of size $n \times n$ whose $(i,j)_{th}$ entry equals the kernel evaluated between points $\mathbf{x}_i$ and $\mathbf{x}_j$: $\kappa(\mathbf{x}_i, \mathbf{x}_j)$. This Gram matrix is typically needed to learn the parameters $\{\alpha_i\}$ of the classifier. Computing this matrix takes however $\mathcal{O}(dn^2)$ operations and requires $\mathcal{O}(n^2)$ memory space. As n and/or d increase, these computation and memory costs may become prohibitive. ** this last paragraph needs some work still. The memory argument is not really an argument. In fact, the GRam matrix, whatever the dimension of the embedding, always needs $\mathcal{O}(n^2)$ to be stored. What RFF truly provide is a low rank approximation of the Gram matrix.

To overcome these difficulties, random feature projections is a technique developed to approximate kernels, often requiring less computational time and less memory storage.

2.1.2 Random features

Random features (RF) (**rahimi2008random**) is an approach developed to approximate kernel methods with reduced computational time. The idea is that, instead of considering the true lifting function ϕ in Eq. 2.3, we explicitly map the data points using an appropriate randomized feature map $\varphi : \mathcal{D} \to \mathbb{C}^m$, such that the kernel evaluated for two data points $\mathbf{x}, \mathbf{x}'$ is approximated by the inner product of their random features with high probability:

$$\kappa(\mathbf{x}, \mathbf{x}') =< \phi(\mathbf{x}), \phi(\mathbf{x}') >_{\mathbb{H}} \approx \varphi(\mathbf{x})^* \varphi(\mathbf{x}') \tag{2.5}$$

where * stands for the conjugate transpose. Considering this approximation, we can transform the input vectors of $\mathcal{X}$ with φ and then apply a linear learning method as in Eq. 2.2 to have a similar learning power as the original non-linear kernel machine, while often avoiding

the cost of explicitly constructing the Gram matrix. Note that with RF we do not use the kernel trick anymore, but construct an explicit (even if random) mapping φ to approximate the kernel κ.

Most RF constructions are known as Random *Fourier* Features (RFF), and are based on the following theorem.

Theorem 2 (Bochner's theorem)**.** *A continuous and shift-invariant kernel* $\kappa(\mathbf{x}, \mathbf{x}') = \kappa(\mathbf{x} - \mathbf{x}')$ *on* $\mathbb{R}^d$ *is positive definite if and only if* κ *is the Fourier transform of a non-negative measure.*

As a direct consequence, we can easily scale any shift-invariant kernel to obtain $\kappa(0) = \int p = 1$, so that its Fourier transform $p(\mathbf{w})$ is a correct probability distribution. We obtain that any shift-invariant psd kernel is of the form:

$$\kappa(\mathbf{x} - \mathbf{x}') = \int_{\mathbb{R}^d} p(\mathbf{w}) e^{j\mathbf{w}^T(\mathbf{x}-\mathbf{x}')} d\mathbf{w} = E_{\mathbf{w}\sim p}[\xi_{\mathbf{w}}(\mathbf{x})^* \xi_{\mathbf{w}}(\mathbf{x}')] \tag{2.6}$$

where $\xi_{\mathbf{w}}(\mathbf{x}) = e^{-j\mathbf{w}^T\mathbf{x}}$ and where $E_{\mathbf{w}\sim p}$ stands for the expectation over $\mathbf{w}$ drawn from the probability distribution $p(\mathbf{w})$. Note that, since κ is a real-valued function, from Eq. 2.6 one can also prove that:

$$\kappa(\mathbf{x} - \mathbf{x}') = \int_{\mathbb{R}^d} p(\mathbf{w}) cos(\mathbf{w}^T(\mathbf{x} - \mathbf{x}')) d\mathbf{w} = E_{\mathbf{w}\sim p}[\tilde{\xi}_{\mathbf{w}}(\mathbf{x})\tilde{\xi}_{\mathbf{w}}(\mathbf{x}')] \tag{2.7}$$

where $\tilde{\xi}_{\mathbf{w}}(\mathbf{x}) = \sqrt{2} cos(\mathbf{w}^T\mathbf{x} + b)$ such that $\mathbf{w}$ is drawn from p and b is drawn uniformly from $[0, 2\pi]$, such that we can use a real-valued mapping if desired.

As a result, for $\mathbf{w}$ a random variable drawn from $p(\mathbf{w})$, $\xi_{\mathbf{w}}(\mathbf{x})^* \xi_{\mathbf{w}}(\mathbf{x}')$ is an unbiased estimate of $\kappa(\mathbf{x}, \mathbf{x}')$. The RF methodology consists in averaging m instances of the estimator with different random frequencies $\mathbf{w}_j$ drawn identically and independently (iid) from p, that is, define

$$\varphi(\mathbf{x}) = \frac{1}{\sqrt{m}} (\xi_{\mathbf{w}_j}(\mathbf{x}))_{j=1}^m \in \mathbb{C}^m \tag{2.8}$$

such that $\varphi(\mathbf{x})^*\varphi(\mathbf{x}') = \frac{1}{m}\sum_{j=1}^{m} \xi_{\mathbf{w}_j}(\mathbf{x})^*\xi_{\mathbf{w}_j}(\mathbf{x}')$, which converges to $\kappa(\mathbf{x},\mathbf{x}')$ by the law of large numbers. Moreover, Hoeffding's inequality guarantees exponentially fast convergence in m between $\varphi(\mathbf{x})^*\varphi(\mathbf{x}')$ and the kernel's true value:

$$\forall \epsilon > 0 \qquad Pr(|\varphi(\mathbf{x})^*\varphi(\mathbf{x}') - \kappa(\mathbf{x},\mathbf{x}')| \geq \epsilon) \leq 2e^{\frac{-m\epsilon^2}{4}}, \tag{2.9}$$

that is, for any error $\epsilon > 0$, the probability that the estimation is off by more than ϵ is controlled by an exponentially decaying term.

One could use the union bound on all couples $(\mathbf{x},\mathbf{x}') \in \mathcal{X}^2$ to study the concentration of *all* entries of the Gram matrix, not only a given couple as in the previous Hoeffding inequality:

Theorem 3 (add ref)**.** *Let $\epsilon \in (0,1)$ and $\delta \in (0,1)$. Consider a dataset $\mathcal{X} = (\mathbf{x}_1,\ldots,\mathbf{x}_n)$ of n elements, and a psd shift-invariant kernel κ. The random embedding $\varphi(\mathbf{x}) \in \mathbb{C}^m$ defined in Eq.* (**??**) *enables a controlled approximation of* all *the elements of the Gram matrix with probability larger than $1-\delta$,* i.e.

$$Pr\left(\forall(\mathbf{x},\mathbf{x}') \in \mathcal{X}^2 \quad |\varphi(\mathbf{x})^*\varphi(\mathbf{x}') - \kappa(\mathbf{x},\mathbf{x}')| \leq \epsilon\right) \geq 1-\delta$$

provided that

$$m \geq \mathcal{O}\left(\frac{1}{\epsilon^2}\log\frac{n}{\delta}\right).$$

As an illustration, consider the Gaussian kernel in Eq. 2.4. This kernel is shift-invariant and known to be positive semi-definite. It is already correctly normalized since $\kappa(0) = 1$, and its Fourier transform is also a Gaussian but with inverted variance:

$$p(w) = FT\big(\kappa_G\big)(\mathbf{w}) = \left(\frac{\sigma^2}{2\pi}\right)^{\frac{d}{2}} e^{-\frac{\sigma^2\|\mathbf{w}\|^2}{2}} \tag{2.10}$$

Thus, in practice, in order to approximate the Gram matrix of κ_G on a dataset $\mathcal{X}$ of size n, one i/ draws m iid frequencies from this probability distribution, with m as in Theorem 3; ii/ uses these frequencies to associate to each element $\mathbf{x} \in \mathcal{X}$ its associated random feature

vector $\varphi(\mathbf{x}) \in \mathbb{C}^m$ as defined in Eq. (**??**) (or its real-valued equivalent in $\mathbb{R}^m$); iii/ uses $\varphi(\mathbf{x})^* \varphi(\mathbf{x}')$ as an approximation of $\kappa_G(\mathbf{x}, \mathbf{x}')$ where necessary in any kernel-based learning algorithms.

2.2 Graphlet kernel

Kernel methods are a flexible set of tools, since psd kernels can be defined on any set of objects rather than on vectors $\mathbf{x} \in \mathbb{R}^d, d \in \mathbb{N}$. Naturally, for machine learning tasks on graphs such as graph classification or regression, authors have developed kernels on graphs $\kappa(\mathcal{G}, \mathcal{G}')$ (**kriege_graph_kernels**). This section gives a brief overview of graph kernels, focusing on the so-called *graphlet kernel*, which will be our main inspiration for this work. Present the table of contents: section 221 will... In section 222 the.. will be detailed before we present, in section 223, ...

2.2.1 Notations of graphs and graphlets

Recall that a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is formed by a set of nodes and a set of edges connecting them. A graph $\mathcal{F} = (\mathcal{V}_\mathcal{F}, \mathcal{E}_\mathcal{F})$ is said to be a subgraph (also called *graphlet*) of $\mathcal{G}$, written $\mathcal{F} \sqsubseteq \mathcal{G}$, if and only if there exists an injective function $g : \mathcal{V}_\mathcal{F} \to \mathcal{V}$ such that $(u, u') \in \mathcal{E}_\mathcal{F} \Leftrightarrow (g(u), g(u')) \in \mathcal{E}$.

Any edge (u_i, u_i) is called a self loop. In a general graph two vertices u_i and u_j may be connected by more than one edge. A simple graph is a graph with no self loops or multiple edges. Here we always consider simple graphs. A (simple) graph can equivalently be represented by an adjacency matrix $\mathbf{A}$ of size $v \times v$. The $(i, j) - th$ entry of $\mathbf{A}$ is 1 if an edge (u_i, u_j) exists and zero otherwise.

Two graphs $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and $\mathcal{G}' = (\mathcal{V}', \mathcal{E}')$ are said to be *isomorphic*, written $\mathcal{G}' \cong \mathcal{G}$, if there exists a bijective function $g : \mathcal{V} \to \mathcal{V}'$ such that $(u_i, u_j) \in \mathcal{E}$ iff $(g(u_i), g(u_j)) \in \mathcal{E}'$. Deciding if two graphs are isomorphic is known to be a difficult problem: it is actually an

open question if this problem is solvable in polynomial time or is NP-complete add ref. It is equivalent to test if two adjacency matrices are a permutation of each other. This gives us a clue why the isomorphism test is expensive: in the worst case where the graphs to be compared don't have a specific structure which can used as *a prior*, the brute force method considers all the $v!$ permutation matrices. There are efficient, specific methods for small graphs (**graphlet_kernel**), but the general case is still open add ref. We denote by $C_k^{\cong}$ the computational cost of testing the isomorphism between two graphs of size k.

As we will see, for a given graph, the graphlet kernel is defined by counting small subgraphs of size k, also called graphlets. We here introduce some useful notations. Let us denote by $\mathfrak{H} = \{\mathcal{H}_1, ..., \mathcal{H}_{N_k}\}$ the set of all possible graphlets of size k. Depending on the context, there is two choices in defining this set. Either we count all possible adjacency matrices and treat isomorphic graphs as different graphs, in which case we have $N_k = 2^{k(k-1)/2}$ different graphlets. We refer to this set as the set of graphlets *with repetition*. Or we do not distinguish isomorphic graphs, in which case $N_k < 2^{k(k-1)/2}$ but it is still exponential in k. We call this set the set of graphlets *without repetition*. The classical graphlet kernel uses graphlets without repetition, which can require expensive isomorphism tests. We will see that some methods on graphlets *with* repetition also perform well in practice.

Say we sample a graphlet $\mathcal{F}$ of size k from a given graph $\mathcal{G}$. A possibly expensive operation is to find, in the set $\mathfrak{H}$ of all possible graphlets of size k, which one it matches. We define the matching function φ_k^{match} which, in the case of graphlets with repetition is defined as:

$$\varphi_k^{match}(\mathcal{F}) = \left[1_{(\mathcal{F}=\mathcal{H}_i)}\right]_{i=1}^{N_k} = \left[1_{(\mathbf{A}_\mathcal{F}=\mathbf{A}_{\mathcal{H}_i})}\right]_{i=1}^{N_k} \in \{0,1\}^{N_k}$$

where $1_{(\Omega)}$ is the indicator function: it equals 1 if the proposition Ω is true, and 0 otherwise. In words, $\varphi_k^{match}(\mathcal{F})$ is a Boolean vector of dimension N_k and has a 1 in the coordinate i if the adjacency matrices of both graphs $\mathcal{F}$ and $\mathcal{H}_i$ are equal, and 0 otherwise. Clearly the cost of each test is k^2, and the cost of applying φ_k^{match} to any graph $\mathcal{F}$ is[1] $\mathcal{O}(N_k k^2)$.

[1]This is in fact the cost of a naive implementation. First of all, as soon as a match is found, one does

In the case of graphlets *without repetition*, φ_k^{match} is defined as:

$$\varphi_k^{match}(\mathcal{F}) = \left[1_{(\mathcal{F} \cong \mathcal{H}_i)}\right]_{i=1}^{N_k} \in \{0,1\}^{N_k}$$

which means that φ_k^{match} puts a 1 in the coordinate i if $F \cong \mathcal{H}_i$, and 0 otherwise. The cost global cost of applying φ_k^{match} to any graph $\mathcal{F}$ is now $\mathcal{O}(N_k C_k^{\cong})$, with an isomorphic test cost $C_k^{\cong}$ exponential in k thus much larger than k^2, although for a N_k smaller than the case with repetition.

Let $\mathfrak{F}$ be any collection of size-k graphs. We define the function φ_k^{hist} which counts, for each graphlet $\mathcal{H}_i$ in $\mathfrak{H}$, how many matches it has in $\mathfrak{F}$. This definition exists for both versions of φ_k^{match}, and reads:

$$\varphi_k^{hist}(\mathfrak{F}) = \frac{1}{s} \sum_{\mathcal{F} \in \mathfrak{F}} \varphi_k^{match}(\mathcal{F}) \in \mathbb{R}^{N_k} \tag{2.11}$$

where the term $\frac{1}{s}$ is introduced for normalization purposes: in order for $\varphi_k^{hist}(\mathfrak{F})$ to sum to 1.

2.2.2 Convolutional graph kernels

I only partially corrected the notations in this section Recall that traditional kernel machines are applied to problems with vector-valued input data, where they compare different data points $\mathbf{x}, \mathbf{x}' \in \mathbb{R}^d$, often through their Euclidean distance. Based on that, these kernels cannot be used directly on (a vector representation of the) graphs: indeed, isomorphic graphs have different adjacency matrices representing the same structure. As a result it is necessary to measure distances between graphs in ways that are insensitive to isomorphism: ideally, if $\mathcal{G}_1 \cong \mathcal{G}_1'$ and $\mathcal{G}_2 \cong \mathcal{G}_2'$, then $\kappa(\mathcal{G}_1, \mathcal{G}_2)$ should be equal to, or at least very close to, $\kappa(\mathcal{G}_1', \mathcal{G}_2')$. One observes that the concept of isomorphism is critical in learning algorithms

not need to check for other possibilities and may stop the algorithm. Also, ordering all possible adjacency matrices according to a smart tree-search algorithm would certainly decrease this cost. We will not delve into this here.

on graphs, not only because there is no known polynomial-time algorithm for testing graph isomorphism (except for graphs with specific structures), but simply testing isomorphism is also too strict for learning in a similar way to learning with equality operator not able to re-write this last sentence: I do not understand it (**kriege_graph_kernels**).

Since it is simpler to define kernels on *small* graphs, most of the graph kernels in the literature belong to the family of *convolution kernels*: given two graphs, the trick is to divide each into smaller subgraphs and then to pairwise compute a kernel between the resulted subgraphs.

Definition 1 (Convolution Kernel)**.** *let $\mathcal{R} = \mathcal{R}_1 \times ... \times \mathcal{R}_d$ denote a space of components such that a composite object $X \in \mathcal{X}$ decomposes into elements of $\mathcal{R}$. Let $R : \mathcal{R} \to \mathcal{X}$ denote the mapping from components to objects, such that $R(x) = X$ iff the components $x \in \mathcal{R}$ make up the object $X \in \mathcal{X}$, and let $R^{-1}(X) = \{x \in \mathcal{R} : R(x) = X\}$. then, the R-convolution kernel is:*

$$K_{CV}(X, Y) = \sum_{x \in R^{-1}(X)} \sum_{y \in R^{-1}(Y)} \underbrace{\prod_{i=1}^{d} k_i(x_i, y_i)}_{k(x,y)} \tag{2.12}$$

with k_i is a kernel on $\mathcal{R}$ for $i \in \{1, ..., d\}$.

Applying this definition on graphs, $R^{-1}(\mathcal{G} = (\mathcal{V}, \mathcal{E}))$ includes all the components in graph $\mathcal{G}$ that we want to compare with the components $R^{-1}(\mathcal{G}' = (\mathcal{V}', \mathcal{E}'))$ in graph $\mathcal{G}'$. One example of these kernels is the node label kernel, where for two graphs $\mathcal{G}, \mathcal{G}'$, the mapping function R maps the features $x_u \in \mathcal{R}$ of each node $u \in \mathcal{V} \cup \mathcal{V}'$ to the graph that u is a member of. Another example, which will be our main source of inspiration and will be further described in the next section, is the k-graphlet kernel, where R here maps the subgraphs of size k to the graph in which they occur.

The advantage of using convolution kernel framework with graphs is that kernels are permutation invariant, non-sensitive to on the graphs level as long as they are permutation invariant on the components level. As a drawback, the sum in Eq. 2.12 iterates over every

possible pair of components. As a result, when the base kernel has high value between a component and itself while it is low between two different components, each graph becomes drastically similar to itself but distant from any other graph. Thus, a set of weights is usually added to counter-balance this problem.

2.2.3 Graphlet Kernel

As mentioned above, the graphlet kernel is a special instance of convolution kernel equivalently described as follows: one enumerates all the subgraphs of size k of each graph (where k is small), counts them to build a histogram of their frequencies of apparition, and takes the inner product between the two histograms to obtain the final kernel. In this context, the subgraphs are called "graphlets", as an analogy with classical wavelets, which are individual components of more traditional signals.

For a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, denote by $\mathfrak{F}_\mathcal{G} = \{\mathcal{F}_1, \mathcal{F}_2, \dots, \}$ the exhaustive collection of *all* size-k subgraphs existing in $\mathcal{G}$: $\mathfrak{F}_\mathcal{G}$ thus has $\binom{v}{k}$ elements. Let us define the k-spectrum of $\mathcal{G}$: the vector $\mathbf{f}_\mathcal{G}$ of size N_k obtained when applying the function φ_k^{hist} as defined in Eq. (2.11) to $\mathfrak{F}_\mathcal{G}$:

$$\mathbf{f}_\mathcal{G} = \varphi_k^{hist}(\mathfrak{F}_\mathcal{G}) \in \mathbb{R}^{N_k}$$

In words, the i-th of $\mathbf{f}_\mathcal{G}$ equals the frequency of appearance of graphlet $\mathcal{H}_i$ in $\mathcal{G}$. Classically, the graphlets are considered without repetition. We will sometimes refer to $\mathbf{f}_\mathcal{G}$ simply as the histogram of $\mathcal{G}$.

The graphlet kernel between two graphs is the inner product between their histograms:

Definition 2 (Graphlet Kernel)**.** *Given two graphs $\mathcal{G}$ and $\mathcal{G}'$ of size $\geq k$, the graphlet kernel $\mathcal{K}_\mathcal{G}$ is defined as (***graphlet_kernel***):*

$$\mathcal{K}_g(\mathcal{G}, \mathcal{G}') = f_\mathcal{G}^T f_{\mathcal{G}'}. \tag{2.13}$$

In this case the distance between graphs in the kernel space is just the Euclidean metric

between histograms $d_\kappa(\mathcal{G}, \mathcal{G}') = \|\mathbf{f}_\mathcal{G} - \mathbf{f}_{\mathcal{G}'}\|_2$. Also, from now on, unless otherwise specified, κ will refer to this graphlet kernel.

The computational cost is a major drawback of this kernel, as computing the k-spectrum vector $\mathbf{f}_\mathcal{G}$ of any graph $\mathcal{G}$ is very costly, even for moderate k: the sum of Eq. (2.11) is over $\mathfrak{F}_\mathcal{G}$, thus over $\binom{v}{k}$ elements; and since graphlets are taken without repetition isomorphism tests have to be performed. Thus, the total cost for computing $\mathbf{f}_\mathcal{G}$ can be written as:

$$C_{gk} = \mathcal{O}(\tbinom{v}{k} N_k C_k^{\cong}). \tag{2.14}$$

As a result, there is a trade-off between a more accurate representation of the graph (larger value of k) and the computational cost. However, one can take advantage of some techniques in order to handle this limitation. In the next section, we focus on random sampling.

2.2.4 Graph sampling to approximate the k-graphlet spectrum

The graphlet kernel can be interpreted as follows: if one draws a subgraph uniformly at random from $\mathcal{G}$, then one has a probability $(\mathbf{f}_\mathcal{G})_i$ of obtaining $\mathcal{H}_i$, *i.e.*:

$$\mathbf{f}_\mathcal{G} = \mathbb{E}_{F \sim \text{unif}}\ \varphi_k^{match}(F)$$

where $\mathbb{E}_{F\sim\text{unif}}$ stands for the expectation over the subgraphs F of size k drawn uniformly at random. Note that we use the notation F, instead of $\mathcal{F}$, when the subgraph it refers to is a random variable.

It is thus natural to approach $\mathbf{f}_\mathcal{G}$ with a sample average, built by i/ first uniformly sampling at random s subgraphs of size k from $\mathcal{G}$ to form the collection

$$\hat{\mathfrak{F}}_\mathcal{G}^{\text{u}} = \{F_1, ..., F_s\}$$

and ii/ then estimate the k-spectrum vector from these random samples:

$$\hat{\mathbf{f}}_\mathcal{G}^{\text{u}} = \varphi_k^{hist}(\hat{\mathfrak{F}}_\mathcal{G}^{\text{u}}) = \frac{1}{s} \sum_{F \in \hat{\mathfrak{F}}_\mathcal{G}^{\text{u}}} \varphi_k^{match}(F). \tag{2.15}$$

In words, the i-th entry of $\hat{\mathbf{f}}_{\mathcal{G}}^{\mathrm{u}}$ counts the number of times $\mathcal{H}_i$ appears among the samples. The law of large numbers states that $\hat{\mathbf{f}}_{\mathcal{G}}^{\mathrm{u}} \xrightarrow[s\to\infty]{} \mathbf{f}_{\mathcal{G}}$ with probability 1. The question here is how many samples we should consider in order to have a desired certainty in our estimation. In other words: how fast is the concentration of $\hat{\mathbf{f}}_{\mathcal{G}}^{\mathrm{u}}$ around $\mathbf{f}_{\mathcal{G}}$? The following theorem answers this question:

Theorem 4 ((**graphlet_kernel**)). *Let $\mathbf{f}_{\mathcal{G}}$ be the k-spectrum of a graph $\mathcal{G}$. Let $\hat{\mathfrak{F}}_{\mathcal{G}}^{\mathrm{u}} = \{F_1, ..., F_s\}$ be a collection of s iid random subgraphs of size k uniformly drawn from $\mathcal{G}$ and $\hat{\mathbf{f}}_G^{\mathrm{u}}$ the associated estimation of $\mathbf{f}_G$ as defined in Eq. (2.15). Then, for a given $\epsilon > 0$ and $\delta > 0$, one only needs:*

$$s_{min} = \left\lceil \frac{2(N_k \log(2) + \log(\frac{1}{\delta}))}{\epsilon^2} \right\rceil \tag{2.16}$$

samples to ensure that $Pr(\|\mathbf{f}_{\mathcal{G}} - \hat{\mathbf{f}}_G^{\mathrm{u}}\|_1 \geq \epsilon) \leq \delta$, where $\lceil . \rceil$ is the ceiling function.

add comment?

Other random sampling techniques. In fact, many other random sampling techniques can be used to sample subgraphs from all the possible ones (**graph_sampling**): uniform sampling is only one such possibility. A *sampling method* will generically be denoted by S_k. We will denote by $F \sim S_k(\mathcal{G})$ a random subgraph of size k of $\mathcal{G}$ sampled by S_k. Note that each sampling method defines a potentially *different histogram*[2] $\mathbf{f}_{\mathcal{G}} = \mathbb{E}_{F\sim S_k(\mathcal{G})}\varphi_k^{match}(F)$, that can in turn be estimated with a sample average as in the previous uniform case. These different histograms capture different kinds of information from the graph $\mathcal{G}$ and may all potentially be used to solve graph learning problems. We describe two examples that we will use in the experiments.

- **Uniform sampling** is the simplest sampling method. We select a subset of k nodes uniformly at random among the $\binom{v}{k}$ possible choices, and extract the subgraph induced

[2]We use generically the notation $\mathbf{f}_{\mathcal{G}}$ for histograms. The underlying sampling method should be clear from context.

by these nodes. As we have seen, this is the sampling method that converges to the classical k-spectrum (and thus the classical graphlet kernel) (2.13) as $s \to \infty$.

- **Random walk sampling**: here we sample the subgraph node by node, we first randomly choose a node u from $\mathcal{V}$ to be the starting node. Then, and until we collect k nodes, we choose the next node randomly from the current node's neighbors in $\mathcal{G}$, and with probability $p_{flyback}$ we go back to the starting node and repeat the same from there, or we stay at the recently chosen node.

The difference between the two methods is that unlike uniform sampling, random walk sampling generates connected subgraphs, *i.e.* there is a path of edges between any pair of the subgraph. This difference is important when we want to sample large graphs where the average degree is small, as uniform sampling in this case generates sparse subgraphs that don't have any edge or just few with high probability. Taking this into account, it is necessary to use random walk when possible since the frequencies of sparse graphlets is high in all the graphs and thus not discriminative to be used in a learning algorithm. *this last paragraph could be a bit clearer*

Impact of graphlet random sampling on the computational cost. The computational cost of approximating the k-graphlet spectrum of a given graph $\mathcal{G}$ via graph sampling includes the cost of sampling a single subgraph with the process S_k, denoted by C_S, iterating over all s sampled subgraphs $F_j \sqsubseteq \mathcal{G}$, then applying φ_k^{match} as before. So the computational cost of this method is:

$$C_{gk+gs} = \mathcal{O}\left(sC_S N_k C_k^{\cong}\right) \tag{2.17}$$

In the case of uniform sampling, C_S is negligible, s should be set proportional to N_k according to Theorem 4, such that the total cost of computing a reasonable estimation of $\mathbf{f}_{\mathcal{G}}$ is $\mathcal{O}\left(N_k^2 C_k^{\cong}\right)$, which is an improvement over the cost of the exact computation of Eq. (2.14).

Chapter Three

Fast graph kernel classifier based on optical random features

Graphlet kernel is a good method to solve graph classification problem but as we have seen in chapter 2, it suffers from a high computational cost. In this chapter, we take inspiration from graph sampling and averaging to propose a family of fast graph kernels that generalizes the graphlet kernel paradigm. We show how random features can be incorporated within the new framework to get a faster and competitive algorithm in graph classification. Finally, we describe how Optical Processing Units (OPUs) can be used to eliminate some significant computational cost altogether.

3.1 Proposed algorithm

We recall from chapter 2 that the computational cost of graphlet kernel is $C_{gk} = O(\binom{v}{k} N_k C_k)$. As an attempt to lower this cost, using graph sampling we can compute an empirical approximation of k-spectrum vector so the new that cost becomes $C_{gk+gs} = O(C_S s N_k C_k)$. We can see that $\binom{v}{k}$ is replaced with $C_S s$, and recall that the minimum number of samples $s \sim N_k$ required to ensure some certainty sharply increases as the graphlet size increase. It is clear then that the number of graph samples is not the only bottleneck here but also the cost to

compute φ_k^{match}, denoted by $C_{\varphi_k^{match}} = O(N_k C_k)$.

So we propose to replace φ_k^{match} with another user-defined function $\varphi : \mathcal{H} \mapsto \mathbb{R}^m$ and keep everything else as it is. We obtain a family of algorithms referred to as Graph Sampling and Averaging (GSA-φ), described in Alg. 1, whose main user-defined methods are the samping method S_k and the feature map φ.

Algorithm 1: Graph Sampling and Averaging (GSA-φ)

Input: 2-Classes labelled graph dataset $\mathcal{X} = (\mathcal{G}_i, y_i)_{i=1,\dots,n}$

Output: Trained model to classify graphs

Tools *Graph random sampler S_k, a function φ, linear classifier (ex. SVM)*

Hyperparameters *k:graphlet size, s:#graphlet samples per graph*

Algorithm

Random initialization of SVM weights

for $\mathcal{G}_i$ *in* $\mathcal{X}$ **do**

  $\varphi_i = 0$

  for $j = 1 : s$ **do**

    $F_{i,j} \leftarrow S_k(\mathcal{G}_i)$

    $\varphi_i \leftarrow \varphi_i + \frac{1}{s}\varphi(F_{i,j})$

$\mathcal{D}_\varphi \leftarrow (\varphi_i, Y_i)_{i=1,\dots,n}$

Train the linear classifier on the new vector-valued dataset $\mathcal{D}_\varphi$

For a set $\mathfrak{F} = \{F_1, \dots, F_s\}$ and a feature map φ, similarly to φ^{hist} we define $\varphi(\mathfrak{F}) = \frac{1}{s}\sum_i \varphi(F_i)$. The GSA-$\varphi$ algorithm computes, for each graph $\mathcal{G}$ in the dataset, an embedding $\varphi(\mathfrak{F}_\mathcal{G})$ where $\mathfrak{F}_\mathcal{G}$ is a set of s subgraphs drawn *iid* from $S_k(\mathcal{G})$, then trains a classifier on it.

We note that within the new paradigm, the defined φ does not necessarily respect the isomorphism between sampled subgraphs: if $\varphi(F) = \varphi(F')$ whenever $F \cong F'$, then we are in the framework of graphlet *without* repetition, otherwise we are in the other case. As we will see, choosing a randomized φ presents both theoretical and practical advantages, however it does not respect isomorphism. Nevertheless, it is possible to apply some preprocessing

function Q invariant by permutation before passing it to a randomized φ, and in this case isomorphism is respected without any condition on φ. An example of such function is $Q : \mathbb{R}^{k\times k} \mapsto \mathbb{R}^k, Q(F) = Sort(Eigenvalues(\mathbf{A}_F))$, that is, the sorted eigenvalues of the adjacency matrix.

NK: A word on the generic computational cost of the method ? $O(C_S s C_\varphi)$**, emphasizing that** C_φ **is now the main focus in the rest (intuitively, trade off between computational cost and discriminative power)**

3.2 Using random features framework in our algorithm

In the previous section, we introduced a generic family of algorithms, GSA-φ. Besides the sampling method, the performance and computational cost of the algorithm mainly depends on the choice of feature map φ: choosing $\varphi = \varphi_k^{match}$ produces the classical graphlet kernel but it is expensive to compute. In this section, we motivate the choice of φ as kernel random features and relate it to a new notion of metric between graphs, using the so-called *Maximum Mean Discrepancy* (MMD) (**gretton**), a kernel metric between probability distributions.

Let us recall a few notions about kernel RF (see Section **??**). Assume that we have a psd (often shift-invariant) kernel on $\mathbb{R}^{k\times k}$, that can be written in the form:

$$\mathcal{K}(F, F') = \mathbb{E}_{w\sim p(w)} \xi_w(F)^* \xi_w(F') \tag{3.1}$$

for some mapping $\xi_w : \mathbb{R}^{k\times k} \to \mathbb{C}$ parameterized by w distributed according to $p(w)$. Although this kernel technically takes (the adjacency matrix of) a graphlet as an input, we do not require it to be permutation-invariant, since it will be combined with the graph sampling process. For instance, we will see in experiments that even a simple Gaussian kernel on adjacency matrices performs well!

As we have seen, the RF method defines:

$$\varphi(F) = \frac{1}{\sqrt{m}} (\xi_{w_j}(F))_{j=1}^m \in \mathbb{C}^m, \quad m \in \mathbb{N} \tag{3.2}$$

and we can write:

$$\mathcal{K}(F, F') \approx \varphi(F)^* \varphi(F')$$

Now, how does the embedding computed by GSA-φ relates to the base kernel $\mathcal{K}$? To examine that, in the following we define another kernel, called the *mean kernel* $\mathcal{K}_{mk}$ (**gretton**), with its corresponding metric called the *Maximum Mean Discrepancy (MMD)*. Next, we show with the aid of concentration inequalities how using the random features map φ of $\mathcal{K}$ in our algorithm GSA-φ will lead to an approximation of $\mathcal{K}_{mk}$ concentrated around its true value with high probability.

The mean kernel methodology allows to *lift* a kernel from a domain $\mathcal{H}$ to a kernel on *probability distributions* on $\mathcal{H}$. Given a base kernel $\mathcal{K}$ on $\mathcal{H}$ and two probability distribution $\mathcal{P}, \mathcal{Q}$, it is defined as:

$$\mathcal{K}_{mk}(\mathcal{P}, \mathcal{Q}) = \mathbb{E}_{x \sim \mathcal{P}, y \sim \mathcal{Q}} \mathcal{K}(x, y) \tag{3.3}$$

In other words, the mean kernel is just the expectation of the base kernel with respect to each term. The metric associated to the mean kernel is usually referred to as the *Maximum Mean Discrepancy (MMD)* in the literature (**gretton**), and is defined as:

$$MMD(\mathcal{P}, \mathcal{Q}) = \sqrt{\mathcal{K}_{mk}(\mathcal{P}, \mathcal{P}) + \mathcal{K}_{mk}(\mathcal{Q}, \mathcal{Q}) - 2\mathcal{K}_{mk}(\mathcal{P}, \mathcal{Q})} \tag{3.4}$$

It should be noticed here that $\mathcal{K}_{mk}(\mathcal{P}, \mathcal{P}) = \mathbb{E}_{x \sim \mathcal{P}, x' \sim \mathcal{P}} \mathcal{K}(x, x') \neq \mathbb{E}_{x \sim \mathcal{P}} \mathcal{K}(x, x)$.

The link between the mean kernel and graph sampling can be easily constructed, since considering a random sampling method S_k, the pair $(S_k, \mathcal{G})$ introduces a probability distribution $f_{\mathcal{G}} = \mathbb{E}_{F \sim S_k(\mathcal{G})} \varphi_k^{match}(F)$ on the set of size-k graphlets $\mathcal{H}$. Thus, for two graphs $\mathcal{G}, \mathcal{G}'$, the mean kernel between these distributions, denoted by $\mathcal{K}_{mk}(\mathcal{G}, \mathcal{G}') = \mathcal{K}_{mk}(f_{\mathcal{G}}, f_{\mathcal{G}'})$ for simplicity, can be reformulated as:

$$\mathcal{K}_{mk}(\mathcal{G}, \mathcal{G}') = \mathbb{E}_{F \sim S_k(\mathcal{G}), F' \sim S_k(\mathcal{G}')} \mathcal{K}(F, F') \tag{3.5}$$

Remark that the mean kernel also reduces to:

$$\mathcal{K}_{mk}(\mathcal{G}, \mathcal{G}') = \sum_{i,j}^{N_k} (f_{\mathcal{G}})_i (f_{\mathcal{G}'})_j \mathcal{K}(\mathcal{H}_i, \mathcal{H}_j)$$

We denote by $MMD(\mathcal{G}, \mathcal{G}')$ the corresponding MMD, which is a new notion of distance between graphs that generalizes the graphlet kernel metric.

Let us now integrate random features with the mean kernel, assuming (3.1) and (3.2), and show how GSA-φ relates to the MMD we have just defined. We combine the decomposition of the base kernel $\mathcal{K}$ in Eq. (3.1) with Eq. (3.5) to get:

$$\mathcal{K}_{mk}(\mathcal{G}, \mathcal{G}') = \mathbb{E}_{F \sim S_k(\mathcal{G}), F' \sim S_k(\mathcal{G}')} \mathbb{E}_w \xi_w(F) \xi_w(F') \tag{3.6}$$

The corresponding MMD metric in this case is:

$$MMD(\mathcal{G}, \mathcal{G}')^2 = \mathbb{E}_\omega \Big(|\mathbb{E}_{S_k(\mathcal{G})} \xi_\omega(F) - \mathbb{E}_{S_k(\mathcal{G}')} \ \xi_\omega(F')|^2 \Big) \tag{3.7}$$

Until now, what we have in Eq. 3.6 is the true value of the mean kernel, where the expectations there implies that we should consider infinite number of both graph samples and random features. However, what we really want is to approximate this value using our algorithm GSA-φ, which includes using the finite-dimensional map φ and a finite number of *iid* samples drawn with S_k. First let's consider only a finite number s of samples: $\mathfrak{F}_\mathcal{G} = \{F_1, \ldots, F_s\}$ and $\mathfrak{F}_{\mathcal{G}'} = \{F'_1, \ldots, F'_s\}$. Then we have:

$$\mathcal{K}_{mk}(\mathcal{G}, \mathcal{G}') \approx \frac{1}{s^2} \sum_{i,j=1}^{s} \mathbb{E}_w \xi_w(F_i) \xi_w(F'_j) \tag{3.8}$$

$$MMD(\mathcal{G}, \mathcal{G}')^2 \approx \frac{1}{s^2} \mathbb{E}_\omega (|\xi_\omega(F) - \xi_\omega(F')|^2)$$

No we consider both a finite number s of (iid) samples, and a feature map φ with finite number m of random features defined as (3.2), so the formula of our final approximation is:

$$\mathcal{K}_{mk}(\mathcal{G}, \mathcal{G}') \approx \frac{1}{s^2} \sum_{i,j=1}^{s} \varphi(F_i)^* \varphi(F'_j) = \Big(\frac{1}{s} \sum_{i=1}^{s} \varphi(F_i) \Big)^* \ \Big(\frac{1}{s} \sum_{i=1}^{s} \varphi(F'_i) \Big) = \varphi(\mathfrak{F}_\mathcal{G})^* \varphi(\mathfrak{F}_{\mathcal{G}'}) \tag{3.9}$$

$$MMD(\mathcal{G}, \mathcal{G}')^2 \approx \|\varphi(\mathfrak{F}_\mathcal{G}) - \varphi(\mathfrak{F}_{\mathcal{G}'})\|_2^2$$

Indeed, we just proved that, when using random features, GSA-φ theoretically represents an unbiased estimation of the mean kernel $\mathcal{K}_{mk}$. The corresponding MMD is then the key

to the performance of the algorithm: if the graphs are well-separated by this metric, then a machine learning algorithm will be able to classify them.

Now, we have two issues to discuss: how is this estimation concentrates around its expected value $\mathcal{K}_{mk}$, and what is the computational cost of GSA-φ in this case. While the first question can be addressed for any kernel $\mathcal{K}$, the second one is really case-dependent as it is mainly related to the computational cost of computing the term $\varphi(F_i)$ in Eq. (3.9) which differs from one φ to another **NK: I would define the generic cost C_φ in the previous section and keep the discussion focused on concrete examples of RF here (Gaussian, gaussian with eigenvalues...)**. We start answering the first question.

Theorem 5. *Let $\mathcal{G}$ and $\mathcal{G}'$ be two graphs, $\{F_i\}_{i=1}^s$ (resp. $\{F'_i\}_{i=1}^s$) be iid size-k graphlet samples drawn from $S_k(\mathcal{G})$ (resp. $S_k(\mathcal{G}')$). Assume a kernel of the form* (3.1) *and a random feature map* (3.2)*. Assume that $|\xi_w(F)| \leq 1$ for any w, F.*

We have that, for all $\delta > 0$, with probability at least $1-\delta$:

$$\left| \|\varphi(\mathfrak{F}_{\mathcal{G}}) - \varphi(\mathfrak{F}_{\mathcal{G}'})\|^2 - MMD(\mathcal{G}, \mathcal{G}')^2 \right| \leq \frac{4\sqrt{\log(6/\delta)}}{\sqrt{m}} + \frac{8\left(1+\sqrt{2\log(3/\delta)}\right)}{\sqrt{s}}$$

this tells that the Euclidean distance between embedding computed by GSA-φ converges to the MMD in $O(1/\sqrt{m} + 1/\sqrt{s})$. The proof of this theorem is provided in section 3.4.

«««< HEAD To calculate **the computational cost of $GSA-\varphi$**, we considered three cases of φ in our experiments, hence $\mathcal{K}$, in both theoretical and practical sides: Gaussian random features applied on the adjacency matrix, Gaussian random features applied on the sorted Eigenvalues of the adjacency matrix , and finally the optical random features of OPUs. The case of OPU random features is presented in section 3.3.

Hashem: here am supposed after few words to reach the point: $C_{\varphi_{Gs}} = O(mk^2)$ for the adjacency matrix and $C_{\varphi_{Gs}} = O(mk^2 + C_{Eigen_extraction})$ for the adjacency matrix NK: you can use the theorem above to say that m should be of the order of s (since the error rate is the same in m and s), which is high. Hence the huge gain with the OPU. ======= To calculate **the computational cost of $GSA-\varphi$**,

we considered three cases of φ, hence $\mathcal{K}$, in both theoretical and practical sides: Gaussian random features applied on the adjacency matrix, Gaussian random features applied on the Eigenvalues of the adjacency matrix , and finally the optical random features of OPUs. The case of OPU random features is presented in section 3.3.

For a flattened version of an adjacency matrix $\mathbf{A}_{flat}$ of a subgraph F, we recall that the Gaussian kernel stated in Eq. 2.4 can be approximated by the inner product of the following random features map:

$$\varphi(\mathbf{A}_{flat}) = \frac{1}{\sqrt{m}}(\sqrt{2}cos(w_j^T \mathbf{A}_{flat} + b))_{j=1}^{m} \in \mathbb{C}^m$$

where $w_j \in \mathbb{R}^{k^2}$ drawn from the Gaussian distribution in Eq. 2.10. Hence, The computational cost needed in this case for each subgraph to compute its $\varphi(\mathbf{A}_{flat})$ is $O(mk^2)$. Which is the cost of matrix multiplication supposing that we have a fast method to evaluate the *cos* function. Finally, the processing cost per graph $\mathcal{G}$ is: $C_{Gs} = O(smk^2)$.

Gaussian random features applied on the Eigenvalues of the adjacency matrix is similar to the one applied on the adjacency matrix, the difference is that instead of passing the adjacency matrix as an input, we pass a vector of its Eigenvalue $\boldsymbol{\lambda} \in \mathbb{R}^k$. So what changes per subgraph process is the input dimension and the cost of Eigenvalues extraction, which is $O(k^3)$. In a similar way, we can write: $C_{Gs+Eig} = O(s(mk + k^3))$.

»»»> 212a4937ba12b07559998ab673b99f81e8f360f4

3.3 Fast $GSA - \varphi$ with optical random features

Optical processing units (OPUs) are physical units developed to apply optical random features projections in light-speed. In this section and before introducing $GSA - \varphi$ in its ultimate and fastest form $GSA - \varphi_{OPU}$, we explain what optical random features projection is. We start by presenting the mathematical model of these projections φ_{OPU}, next we explain how OPUs can perform such projections in light-speed from hardware point of view.

Finally, we merge OPUs' random features with our proposed algorithm and compare the corresponding computational cost with previously mentioned methods.

3.3.1 Optical random features model

Random projections is one of the important techniques in machine learning and in signal processing. However, traditional random projection methods need a large memory to store the corresponding random matrix $\mathbf{W}$ and a huge computational time to project the input data points $\mathbf{x}$, i.e. to compute $\mathbf{Wx}$. Optical processing units (OPU's) is the technology developed to solve the previous two drawbacks: where an OPU complete random projections at the speed of light without the need to store any random matrix. In general, Random projections are the result of two procedures: the first one is the linear-random projections and the second one is non-linear mapping. Mathematically speaking, OPU's perform the following operation (**saade_opu**):

$$\varphi_{OPU}(\mathbf{x}) = |\mathbf{Wx} + \mathbf{b}|^2;\ \mathbf{W} \in \mathbb{R}^{m\times d}, \mathbf{b} \in \mathbb{R}^m, \mathbf{x} \in \mathbb{R}^d \tag{3.10}$$

Where $\mathbf{b}$ is a bias vector, $\mathbf{x}$ is an input point, m is the number of random features, d is the input space dimension and the amplitude function $|.|$ is taken element wise in $\varphi_{OPU}(\mathbf{x})$. The matrix $\mathbf{W}$ is a random *iid* complex matrix with Gaussian real and imaginary parts.
In the limit where the number of random features $m \to \infty$, it can be proven by the concentration of the measure that the inner product between the projected data points ($\varphi_{OPU}(\mathbf{x} \in \mathbb{R}^m)$ in the new feature space tends to a kernel function that depends only on the input points in the original feature space ($\mathbf{x} \in \mathbb{R}^d$)

3.3.2 OPU structure and functionality

Eq. 3.10 still imply that an OPU need to store and multiply by the random projection matrix. But in an OPU, a heterogeneous material, as a paper or any white translucid material, is used to scatter incident light in a very complex way. The behavior of the scattering process

is considered random because of the extremely high complexity. One can argue that light scattering is a linear, deterministic, and reproducible phenomenon, but what can be said here is that the unpredictable behavior of the process makes it effectively a random process. That is why these materials are called opaque since all information embedded within the incident light is seemingly lost during the propagation through the material (**saade_opu**). An example used to demonstrate and justify the resulted randomness is a cube of edge length $100\mu m$, such cube can include $\approx 10^7$ paint nanoparticles, all the positions and shape of these particles must be known in order to predict its effect on light. Propagation through such a layer can be seen as a random walk because of frequent scattering with the nanoparticles, where light explores the whole volume and undergoes tens of thousands of such scattering steps before coming out from the other side in a few picoseconds.

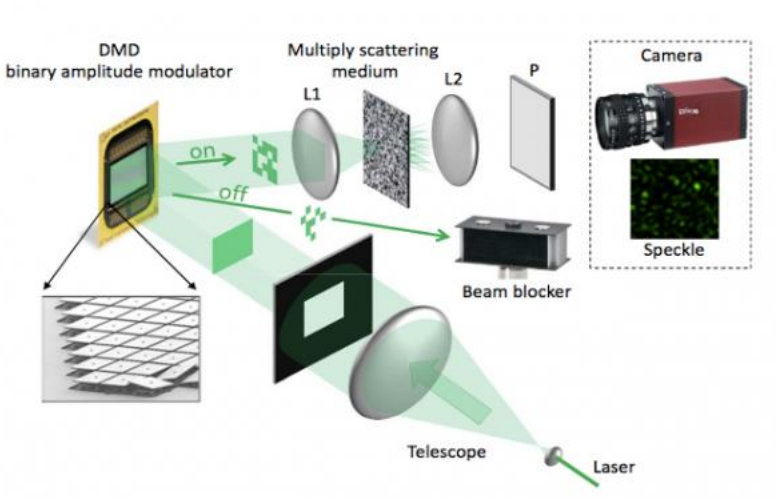

Figure 3.1 OPU's Experimental setup (**saade_opu**): A monochromatic laser is expanded by a telescope, then illuminates a digital micromirror device (DMD), able to spatially encode digital information on the light beam by amplitude modulation. The light beam carrying the signal is then focused on a random medium by means of a lens. The transmitted light is collected on the far side by a second lens, passes through a polarizer, and is measured by a standard monochrome CCD camera for example .

When the incident light is coherent, it promotes complex interference patterns, called

speckles, due to the scattering process. These speckles don't only characterize the propagation medium but also the incident light, and this can be modeled by $y = Wx$, where y and x are the vector amplitudes between a set of spatial modes at the output and at the input of the medium respectively. In OPUs, the transmission matrix W of the propagation medium can be approximately considered as a Gaussian i.i.d matrix, it was also shown that even without W being known, but it is guaranteed to be stable as long as the propagation medium is stable as a paint layer for instance (**saade_opu**). So if we use a spatial light modulator and a laser to send an appropriate set of illuminations to the propagation medium, we can acquire the output intensity $|y|^2$ with a CCD or CMOS camera, and that is the principle concept behind OPU's functionality as seen in Fig 3.1.

The DMD (digital micromirror device) used in OPU's is a binary amplitude modulator consisting of an array of micro-mirrors, Each mirror can be lit or not, thus it can represent a binary value. In order to represent grey values, each value is encoded on a square sub-array (4×4 for example) in the DMD, where the number of lit mirrors reflects the desired level of grey. DMD reflects the data and send the reflected light through the disordered medium, then a snapshot of the resulting random projection is acquired using a standard camera. all that is completed in a very high speed compared to the traditional random features techniques.

3.3.3 Fast $GSA - \varphi_{OPU}$ algorithm with complexity discussion

After what we presented about OPUs and optical random features projections, we can efficiently deploy φ_{OPU} random feature map in $GSA - \varphi$ framework. The resulted pseudo code is presented below and we refer to the resulted algorithm with $GSA - \varphi_{OPU}$.

Algorithm 2: GSA-φ_{OPU}

Input: 2-Classes labelled graph dataset $\mathcal{X} = (\mathcal{G}_i, y_i)_{i=1,\dots,n}$

Output: Trained model to classify graphs

Tools *Graph random sampler S_k, optical processing unit (OPU), linear classifier (ex. SVM)*

Hyperparameters *k:graphlet size, s:#graphlet samples per graph*

Algorithm

Random initialization of the classifier weights

for $\mathcal{G}_i$ *in* $\mathcal{X}$ **do**

  $\boldsymbol{\varphi}_{OPU}(i) = 0$

  for $j = 1 : s$ **do**

    $F_{i,j} \leftarrow S_k(\mathcal{G}_i)$

    $\mathbf{A} \leftarrow adjacency\_matrix(F_{i,j})$

    $\boldsymbol{\varphi}_{OPU}(i) \leftarrow \boldsymbol{\varphi}_{OPU}(i) + \frac{1}{s}\boldsymbol{\varphi}_{OPU}(F_{i,j})$

$\mathcal{D}_{\varphi_{OPU}} \leftarrow (\boldsymbol{\varphi}_{OPU}(i), y_i)_{i=1,\dots,n}$

Train the linear classifier on the new vector-valued dataset $\mathcal{D}_{\varphi_{OPU}}$

The computational cost of this innovative version $GSA-\varphi_{OPU}$, and more specifically of computing $\boldsymbol{\varphi}_{OPU}(\mathbf{A})$, is $O(1)$ in both the number of random features and the graphlet size. One can reasonably argue that an OPU has its limits, as the DMD mirror array ,where the input $\mathbf{A}$ is encrypted, has a limited dimensions (width $\times$ length). Moreover, the camera used to acquire the light intensity in an OPU has a limited number of pixels thus it introduces a limit on the maximum number of random features. However, our argument is that when we respect these limits, which are usually large ones, the computational cost is a constant in both m and k. To better illustrate the advantage of $GSA-\varphi_{OPU}$ over other discussed methods in this work, we show a comparison between them with respect to the computational time per graph $\mathcal{G}$:

- Graphlet kernel: $C_{gk} = O(\binom{v}{k} N_k C_k)$
- Graphlet kernel with sampling: $C_{gk+gs} = O(C_s s N_k C_k)$
- Gaussian features $GSA_{\varphi_{Gs}}$: $C_{Gs} = O(smk^2)$
- Gaussian features $GSA_{\varphi_{Gs}}$ with Eigenvalue preprocessing: $C_{Gs+Eig} = O(s(mk + k^3))$
- $GSA_{\varphi_{OPU}}$: $C_{OPU} = O(1)$

3.4 $GSA - \varphi$ concentration inequality proof

Proof. Here, we prove theorem 5. We decompose the proof in two steps.

Step 1: infinite s, finite m (number of random features). Based on our assumption $|\xi_w| \leq 1$, it is a straightforward result of Hoeffding's inequality that $MMD(\mathcal{G}, \mathcal{G}')^2$ is close to $\frac{1}{m}\sum_{j=1} |\mathbb{E}_{F\sim S_k(\mathcal{G})}\xi_{w_j}(F) - \mathbb{E}_{F'\sim S_k(\mathcal{G}')}\xi_{w_j}(F')|^2 = |\mathbb{E}_{F\sim S_k(\mathcal{G})}\varphi(F) - \mathbb{E}_{F'\sim S_k(\mathcal{G}')}\varphi(F')|^2$. We recall Hoeffding's inequality below.

Lemma 1 (Hoeffding's inequality)**.** *Let $(x_1, \ldots, x_m)$ be independent random variables such that the variable x_i is strictly bounded by the interval $[a_i, b_i]$, and let $\overline{X} = \frac{1}{m}\Sigma_{i=1}^m x_i$ then we have:*

$$\mathbb{P}(|\mathbb{E}\overline{X} - \overline{X}| \geq \epsilon) \leq 2\ \exp\left(-\frac{2m^2\epsilon^2}{\Sigma_{i=1}^m (b_i - a_i)^2)}\right) \tag{3.11}$$

In our case, we define the variables $x_j = |\mathbb{E}_{F\sim S_k(\mathcal{G})}\xi_{w_j}(F) - \mathbb{E}_{F'\sim S_k(\mathcal{G}')}\xi_{w_j}(F')|^2$. They are independent, have expectation $MMD(\mathcal{G}, \mathcal{G}')^2$, and are bounded by the interval $[0, 4]$, thus we have:

$$\mathbb{P}\left(\left|\frac{1}{m}\sum_{j=1}^{m} |\mathbb{E}_{F\sim S_k(\mathcal{G})}\xi_{w_j}(F) - \mathbb{E}_{F'\sim S_k(\mathcal{G}')}\xi_{w_j}(F')|^2 - MMD(\mathcal{G}, \mathcal{G}')^2\right| \geq \epsilon\right) \leq 2\ e^{-2m\epsilon^2/16}$$

Or, in other words, with probability $1 - \delta$,

$$\left|\frac{1}{m}\sum_{j=1}^{m} |\mathbb{E}_{F\sim S_k(\mathcal{G})}\xi_{w_j}(F) - \mathbb{E}_{F'\sim S_k(\mathcal{G}')}\xi_{w_j}(F')|^2 - MMD(\mathcal{G}, \mathcal{G}')^2\right| \leq \frac{4\sqrt{\log(2/\delta)}}{\sqrt{m}} \tag{3.12}$$

Step 2: finite s and m. We show that for any *fixed* set of random features w_j, we have $\|\mathbb{E}_{F\sim S_k(\mathcal{G})}\varphi(F) - \mathbb{E}_{F'\sim S_k(\mathcal{G}')}\varphi(F')\|$ close to $\|\frac{1}{s}\sum_i \varphi(F_i) - \frac{1}{s}\sum_i \varphi(F_i')\|$. By definition of $F_1, \ldots, F_s$ random subgraphs drawn independently from $S_k(\mathcal{G})$, we clearly have:

$$\mathbb{E}_{F\sim S_k(\mathcal{G})}\varphi(F) = \mathbb{E}(\ \frac{1}{s}\sum_i \varphi(F_i)\ ) \tag{3.13}$$

Moreover by our assumptions $\varphi(F)$ is in a ball of radius $M = \frac{\sqrt{m}}{\sqrt{m}} = 1$. We the apply the following vector version of Hoeffding's inequality.

Lemma 2. *let $X = \{x_1, \ldots, x_s\}$ be iid random variables in a ball of radius M centered around the origin in a Hilbert space $\mathcal{H}$. Denote their average by $\overline{X} = \frac{1}{s}\sum_{i=1}^{s} x_i$. Then for any $\delta > 0$, with probability at lest $1 - \delta$,*

$$\|\overline{X} - \mathbb{E}\overline{X}\|_{\mathcal{H}} \leq \frac{M}{\sqrt{s}}\left(1 + \sqrt{2\log\frac{1}{\delta}}\right) \tag{3.14}$$

Proof. Although this lemma is known in the literature, we reproduce its proof for completeness. Defining the function $f(x) = \|\overline{X} - \mathbb{E}\overline{X}\|$, and $\widetilde{X} = x_1, \ldots, \widetilde{x}_i, \ldots, x_s$ to be a copy of X with the ith element replaced by an arbitrary element of $\mathcal{H}$, we can prove using the triangle inequality:

$$|f(X) - f(\widetilde{X})| = \left|\|\overline{X} - \mathbb{E}\overline{X}\| - \|\overline{\widetilde{X}} - \mathbb{E}\overline{X}\|\right| \leq \|\overline{X} - \overline{\widetilde{X}}\| \leq \frac{\|x_i - \widetilde{x}_i\|}{s} \leq \frac{2M}{s} \tag{3.15}$$

Therefore, $f(X)$ is insensitive to the ith component of X, $\forall i \in \{1, \ldots, s\}$ which suggests applying McDiarmid's inequality on f.

To bound the expectation of f, we use the familiar identity about the variance of the average of *iid* random variables:

$$\mathbb{E}\|\overline{X} - \mathbb{E}\overline{X}\|^2 = \frac{1}{n}(\mathbb{E}\|x\|^2 - \|\mathbb{E}x\|^2) \tag{3.16}$$

Also:

$$\mathbb{E}f(X) \leq \sqrt{\mathbb{E}f^2(X)} = \sqrt{\mathbb{E}\|\overline{X} - \mathbb{E}\overline{X}\|^2} \leq \frac{M}{\sqrt{s}}$$

This bound for the expectation of f and McDiarmid's inequality give:

$$\mathbb{P}_x\Big[f(X) - \frac{M}{\sqrt{s}} \geq \epsilon\Big] \leq \mathbb{P}_x\Big[f(X) - \mathbb{E}f(X) \geq \epsilon\Big] \leq \exp\Big(-\frac{s\epsilon^2}{2M^2}\Big) \tag{3.17}$$

Which is equivalent to (3.14) by setting $\delta = exp(-\frac{s\epsilon^2}{2M^2})$ and solving for ϵ. □

Now back to Eq. (3.13) and its corresponding assumptions that we made, we can directly apply lemma 2 (and more especially Eq.(3.17)) to get that with probability $1-\delta$:

$$\|\mathbb{E}_{F\sim S_k(\mathcal{G})}\varphi(F) - \frac{1}{s}\sum_i \varphi(F_i)\| \geq \frac{1}{\sqrt{s}}\left(1 + \sqrt{2\log\frac{1}{\delta}}\right) \tag{3.18}$$

Now, by a union bound and triangle inequality: with probability $1-2\delta$,

$$\begin{aligned}
&\Big|\|\mathbb{E}_{F\sim S_k(\mathcal{G})}\varphi(F) - \mathbb{E}_{F'\sim S_k(\mathcal{G}')}\varphi(F')\| - \|\frac{1}{s}\sum_i \varphi(F_i) - \frac{1}{s}\sum_i \varphi(F_i')\|\Big| \\
&\qquad \leq \|\mathbb{E}_{F\sim S_k(\mathcal{G})}\varphi(F) - \frac{1}{s}\sum_i \varphi(F_i)\| + \|\mathbb{E}_{F'\sim S_k(\mathcal{G}')}\varphi(F') - \frac{1}{s}\sum_i \varphi(F_i')\| \\
&\qquad \leq \frac{2}{\sqrt{s}}\left(1 + \sqrt{2\log\frac{1}{\delta}}\right)
\end{aligned}$$

Moreover, since $\|\mathbb{E}_{F\sim S_k(\mathcal{G})}\varphi(F) - \mathbb{E}_{F'\sim S_k(\mathcal{G}')}\varphi(F')\| + \|\frac{1}{s}\sum_i \varphi(F_i) - \frac{1}{s}\sum_i \varphi(F_i')\| \leq 4$, with the same probability:

$$\Big|\|\mathbb{E}_{F\sim S_k(\mathcal{G})}\varphi(F) - \mathbb{E}_{F'\sim S_k(\mathcal{G}')}\varphi(F')\|^2 - \|\frac{1}{s}\sum_i \varphi(F_i) - \frac{1}{s}\sum_i \varphi(F_i')\|^2\Big| \leq \frac{8}{\sqrt{s}}\left(1 + \sqrt{2\log\frac{1}{\delta}}\right) \tag{3.19}$$

Since it is valid for any fixed set of random features, it is also valid with *joint* probability on random features and samples, by the law of total probability.

Finally, combining (3.12), (3.19), a union bound and a triangular inequality, we have: with probability $1-3\delta$,

$$\Big|\|\varphi(\mathfrak{F}_{\mathcal{G}}) - \varphi(\mathfrak{F}_{\mathcal{G}'})\|^2 - MMD(\mathcal{G},\mathcal{G}')^2\Big| \leq \frac{4\sqrt{\log(2/\delta)}}{\sqrt{m}} + \frac{8}{\sqrt{s}}\left(1 + \sqrt{2\log\frac{1}{\delta}}\right) \tag{3.20}$$

which concludes the proof by taking δ as $\delta/3$.

□