TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter One

Context and Related Works

As the method developed in this work approaches Graph Classification problem using Random Features, we in this chapter introduce Graph networks, how they arose from real world problems, what is the Graph Classification problem, in which applications it can be used, state-of-the-art methods and its limitations, and we finish be stating our contribution.

1.1 Graph Structures and its presence in real world

Graph structures are used to represent a set of objects and the interactions/relations that link between different pairs of these objects, where the nature of these objects and their interactions with each other is application-dependent. Generally in graph structures, objects are reduced to what is called nodes and a relation between two objects is reduced to an edge between the corresponding two nodes.

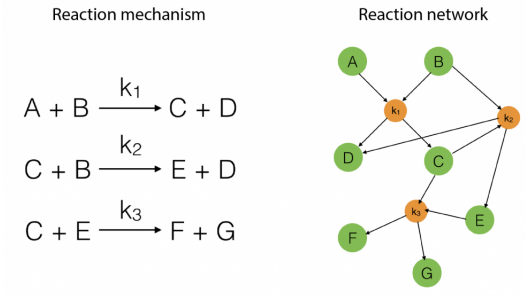

Figure 1.1 Graph structures in representing chemical reactions mechanisms

Network	Nodes	Node Attributes	Edges	Edge Attributes
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

The main advantage of using such representation is that unlike using words to describe the relations between objects, Graphs are more aligned with our human brain, it provides us with a better ability to analyse and conclude from the data structures, and actually this is similar to images which are more informative than arrays of pixels values. To better understand how graphs are built in real life, we throw in Table.1.1 some examples of these graphs in different domains.

1.2 Graph classification Problem

There are many varieties of the graph classification problem, where in some cases the task is to do the classification on the Graph level, but in others it is to be done on the nodes level. From another point of view, in some applications each node/edge in the graph dataset has its own features vector that can be used along with the graph structure to classify different graphs into different classes, while in other applications all we have is the graph structure or it happens that some nodes/edges have their features but others don't. In this project, we consider the one case of graph-level classification with the absence of any node/edge features, and that is what we refer to by ***Graph Classification*** from now on unless the opposite is indicated. On the ground, this problem is drastically being addressed in many fields, as in:

- ***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.

- ***Banking Security:*** graph classification is used to catch unusual patterns of fraudulent transactions.

- ***Biology and Genomics:*** graphs based on protein-protein interactions are analyzed and classified to determine the stage of evolutionary process of Genes that compound these proteins.

1.3 State-of-the-art Methods for Graph Classification

we here present the traditional algorithms deployed in graph classification problem in its wide sense (with or without node/edge features) which is explained in Section 1.2, we simultaneously supply that with the limitations of each of them. In general, these algorithms can be traced back into 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 particularly applied on graphs whose nodes/edges are supplied with features or attributes, so each graph is reduced to a set of nodes, edges or both. Then a distance function of interest between the graphs is computed based on the similarity between pairs of edges/nodes in the corresponding sets. 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 of one graph are similar to the ones of another, we can have two graphs with the same set of edges, which will lead to maximum similarity, but we in reality ignore other important information that can make these graphs completely different such that how many connected communities of nodes these edges form in each graph, how many circles of nodes these edges promote in each graph, etc. On the other hand, a strength point of these algorithms compared to others

is the low computations cost that is usually linear or quadratic in the number of nodes and edges (Shervashidze et al., 2009).

Frequent Sub-graph based algorithms: these algorithms can be applied in two stages, the graph dataset (all the graphs we have in an application) is first analyzed to pick the frequent sub-graphs that occur in different graphs. Then, another analysis is done to choose the most discriminative sub-graphs out of the ones resulted from the first stage. The disadvantage of using this method is the computational cost that grows exponentially with the graph size (Shervashidze et al., 2009).

Graph kernels based algorithms: it is a middle ground between both previous methodologies, where the graph structure (topology) 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 (Shervashidze et al., 2009). However, some effective and competitive kernels still require exponential time, and this is in short the problem we approached in this work using random features to approximate these kernels or to compete them in notably lower computational time.

Graph neural networks (GNNs) based algorithms: GNNs revolutionized learning on graph structures, as it computes a representation vector (embedding vector) for every node in the graph, where this vector is recursively computed by aggregating the representation vectors of neighbor nodes. The goal of this aggregation technique is that nodes that are neighbor (or even close) to each other in the graph are more likely to have a similar representations (with respect to some similarity function) and vise versa. On the graph level, a representation vector is computed by aggregating its nodes representation vectors, and then this vector is used as the features 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 failed to provide high performance classifying graphs

(even the ones with simple topology) whose node/edges don't include any original features vectors (Xu et al., 2018). However, another GNN structure was developed to overcome this weakness point, 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 Context and Our Contribution

k-graphlet kernel is one of the aforementioned graph kernels which had proven a competitive performance in graphs 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. But, the computational cost is massively 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.

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 sub-graphs, number of random features, etc).

Chapter Two

Preliminaries and Necessary background

In this chapter we present the necessary background and the already-investigated analysis done separately in graph kernels, random features, and maximum mean discrepancy. However, we show our analysis combining these different notions in one algorithm in the next chapter, so readers can skip this chapter in case they already have a strong footing in these topics.

2.1 Necessary Notations

Before introducing the aforementioned topics, we first introduce the necessary notations related to graph definition and graph kernels. A graph of size n is by definition a pair $G = (V, E)$, where V is the set of the graph nodes (vertices) $V = \{v_1, ..., v_n\}$, and $E \in V \times V$ is the set of edges between these nodes, i.e. $(v_i, v_j) \in E$ means that the graph has an edge from node v_i to node v_j (and vice versa since we consider undirected graphs in this work). $H = (V_H, E_H)$ is said to be a subgraph (graphlet) of G and we write $H \sqsubseteq G$, if and only if there exists an injective function $\mathcal{M} : V_H \to V$ such that $(v, w) \in E_H \Leftrightarrow (\mathcal{M}(v), \mathcal{M}(w)) \in E$. Any edge (v_i, v_i) is called a self loop. In a general graph two vertices v_i and v_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 A of size $n \times n$. The $(i,j) - th$ entry of A is 1 if an edge (v_i, v_j) exists and zero otherwise.
Two graphs $G = (V, E)$ and $G' = (V', E')$ are isomorphic and we write $G' \cong G$ if there exists a bijective function $\mathcal{M} : V \to V'$ such that $(v_i, v_j) \in E$ iff $(\mathcal{M}(v_i), \mathcal{M}(v_j)) \in E'$

+ In our setting of undirected simple graphs, all adjacency matrices are sym.

2.2 Graph Kernels

In general, the kernel method is used to generate features for learning algorithms that is in its nature a function of the inner product of data points in the dataset (as in linear regression for instance). The mathematical basis behind it is that any positive definite function $\mathcal{K}(x, y)$ with $x, y \in \mathcal{R}^d$ defines an inner product and a lifting function ϕ so that the kernel value is equal to the inner product between lifted data points:

$$\mathcal{K}(x, y) =< \phi(x), \phi(y) > \tag{2.1}$$

The convenience one gets deploying kernels-based models is that there is no need to have the formula or the evaluations of the lifting function ϕ. Instead, it is sufficient to have the evaluations of the kernel itself $\mathcal{K}(x, y)$. To better illustrate this benefit, we take the Gaussian kernel as an example, where:

$$\mathcal{K}_G(x, y) = exp(-\frac{\|x - y\|^2}{2\sigma^2}) \tag{2.2}$$

where σ is called the bandwidth parameter of the kernel. The lifting function ϕ_G of this kernel is located in a Hilbert space of infinite dimension, but the kernel can be easily evaluated for any pair (x,y).

do we really need to talk about the "kernel trick" here?

2.2.1 Graph kernels design

Key

Traditional kernel machines approach problems with vector-valued input data, where it compare different data points ($x, y \in \mathcal{R}^d$) using the difference between correspondent pairs of

vector entries. Based on that, these kernels are imperfect to be used with graphs, since the graph structure is permutation invariant, i.e. isomorphic graphs have different adjacency matrices but they represent the same structure. So in this case distance-kernels between graph representation vectors (adjacency matrices for example) are uninformative. As a result it is necessary to measure distance between graphs in ways that are permutation invariant as well. Here the concept of isomorphism is critical in learning algorithms on graphs, not only because there is no known polynomial-time algorithm for testing graph isomorphism (except for graphs with specific structures), but isomorphism is also too strict for learning in a similar way to learning with equality operator (Kriege, Johansson, and Morris, 2020). Most of graph kernels developed for graph learning problems are convolution kernels, where given two graphs, the trick is to divide each into smaller subgraphs and then to pairwise compute the 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:*

$$K_{CV}(X, Y) = \sum_{x \in R^{-1}(X)} \sum_{y \in R^{-1}(Y)} \underbrace{\prod_{i=1}^{d} k_i(x_i, y_i)}_{k(x,y)} \tag{2.3}$$

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

Applying this definition on graphs, $R^{-1}(G)$ includes all the components in graph G that we want to compare with the components $R^{-1}(H)$ in graph H. One example of these kernels is the node label kernel, where for two graphs G, H, the mapping function R maps the features $x_u \in \mathcal{R}$ of each node $u \in G \cup H$ to the graph that u is a member of. Another example that is mainly related to our work is the k-graphlet kernel, where R here maps the subgraphs of size k to the graph in which it occur. The advantage of using convolution

kernel framework with graphs is that kernels are permutation invariant on the graphs level as long as they are permutation invariant on the components level. As a drawback, the sum in Eq. 2.3 iterates over every possible pair of components. As a result, when we choose our components to be more specific such that the kernel value is high 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 so this problem is resolved.

2.2.2 Graphlet Kernel

Referring by $\mathcal{G} = \{graphlet(1), ..., graphlet(\binom{n}{k})\}$ to the set of all graphs of size k, we define for a graph G the vector $f_G \in \mathcal{R}^{N_k}$, the i-th entry equals the normalized-number of occurrences of $graphlet(i)$ in G ($\#(graphlet(i) \sqsubseteq G)$). What should be noticed based on this definition is that no two different graphlets in $\mathcal{G}$ are isomorphic. f_G is usually referred to by the k-spectrum of G, and this vector is the key idea behind graphlet kernel.

Definition 2 (Graphlet Kernel)**.** *Given two graphs G and H of size $n_G, n_H \geq k$, the graphlet kernel $\mathcal{K}_g$ is defined as (Shervashidze et al., 2009):*

$$\mathcal{K}_g(G, H) = f_G^T f_H. \tag{2.4}$$

Which naturally involves an associated Euclidean metric $d_{\mathcal{K}}(G, H) = \|f_G - f_H\|_2$. The drawback of this kernel is that computing the k-spectrum vector costs huge computational time, as there are $\binom{n}{k}$ subgraphs of size k in a graph G of size n. 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, some techniques are used in order to resolve this limitation as sampling from graph technique (section 2.2.3).

2.2.3 Graph Sampling to approximate k-Graphlet Spectrum

The problem of Graph Sampling arises when we deal with a large-scale graph and the task is to pick a small-size sample subgraph that would be similar to the original graph with respect to some important properties.

Sampling from graph techniques are used to resolve the processing cost limitation of graphlet kernel, and it can deployed in two different manners:

1. directly sample m sugraphs $\{H_1, ..., H_m\}$ of size k, and then estimate the k-spectrum vector empirically: $f_G(i) = \frac{1}{m}\Sigma_{j=1}^{m}\mathbb{1}[H_j = graphlet(i)]$
2. sample m subgraphs $\{H_1, ..., H_m\}$ of size n' such that $n \gg n' > k$, then estimate f_G as follows: $f_G = \frac{1}{m}\Sigma_{j=1}^{m} f_{H_j}$, this method is usually being referred to by Mean Kernel Method and is used in problems with large-scale or random distribution-based infinity-size graphs (see 2.5).

The important thing here is whether a sufficiently large number of random samples will lead to an empirical k-spectrum vector close to the actual vector, and this is to be analyzed in chapter 3.

2.3 Random Features

Random features is a method developed to approximate kernel machines efficiently with a reasonable computational time. The idea is that instead of considering the true lifting function ϕ in Eq. 2.1, we explicitly map the data points to an Euclidean inner product space of low dimensionality. This mapping is done using an appropriate randomized feature map $z : \mathcal{R}^d \to \mathcal{R}^D$. Then we approximate the kernel of two data points x, y by the inner product of their random features:

$$\mathcal{K}(x, y) =< \phi(x), \phi(y) >\approx z(x)^* z(y) \tag{2.5}$$

Considering this approximation, we can transform the input with z and then apply a fast linear learning method to have a similar learning power as the original non-linear kernel-based algorithm.

The question here is how to construct z for a specific kernel so that Eq. 2.5 is satisfied. In what follows, Random Fourier Features method to construct the random feature mapping function z is presented.

2.3.1 Random Fourier Features

The following theorem represents the key idea behind this mapping method.

Theorem 1 (Bochner's theorem)**.** *A continuous and shift-invariant kernel $\mathcal{K}(x, y) = \mathcal{K}(x - y)$ on $\mathcal{R}^d$ is positive definite if and only if $\mathcal{K}(\delta)$ is the Fourier transform of a non-negative measure.*

As a straight result, we can easily scale a shift-invariant kernel so that its Fourier transform $p(w)$ is a correct probability distribution, since it is non-negative measure and integral-bounded function, and we write:

$$\mathcal{K}(x - y) = \int_{\mathcal{R}^d} p(w) e^{jw^T(x-y)} dw = E_w[e^{jw^T x} e^{jw^T y^*}] \tag{2.6}$$

But both $p(w)$ and $\mathcal{K}(\delta)$ are real-valued functions, thus from Eq. 2.6 we can prove that:

$$\mathcal{K}(x - y) = \int_{\mathcal{R}^d} p(w) cos(w^T(x - y)) dw = E_w[z_w(x) z_w(y)] \tag{2.7}$$

where $z_w(x) = \sqrt{2} cos(w^T x + b)$ such that w is drawn from $p(w)$ and b is drawn uniformly from $[0, 2\pi]$.

As a result, $z_w(x) z_w(y)$ is an unbiased estimate of $\mathcal{K}(x, y)$. Moreover, we can achieve lower variance estimation to the expectation (Eq. 2.7) by averaging m instances of the estimator with different random frequencies w. i.e. the low-variance estimator can be written as $z(x)'z(y) = \frac{1}{m} \Sigma_{j=1}^{m} z_{w_j}(x) z_{w_j}(y)$. This estimator and based on Hoeffding's inequality

guarantees exponentially fast convergence in m between $z(x)'z(y)$ and the kernel true value:

$$Pr(|z(x)'z(y) - \mathcal{K}(x,y)| \geq \epsilon) \leq 2e^{\frac{-m\epsilon^2}{4}} \tag{2.8}$$

2.4 Random Projections with Optical Processing Units (OPU's)

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 W and a huge computational time to project the input data points x, i.e. to compute Wx. Optical processing units (OPU's) is the developed 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 where 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 et al., 2016):

$$Y = \phi(WX + b);\ W \in \mathcal{R}^{m \times d}, b \in \mathcal{R}^m, U \in \mathcal{R}^d \tag{2.9}$$

Where b is a bias vector, X is an input point, m is the number of random features and d is the input space dimension. Also, W is a random i.i.d complex matrix with Gaussian real and imaginary parts and ϕ is the non-linear mapping function.

In the limit where the number of random features $r \to \infty$, it can be proven by the concentration of the measure that the inner product between the projected data points ($X_i \in \mathcal{R}^r$) in the new feature space tends to a kernel function that depends only on the input points in the original feature space ($U_i \in \mathcal{R}^d$)

2.4.1 OPU structure and functionality

Eq. 2.9 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 et al., 2016). 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.

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 et al., 2016). 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

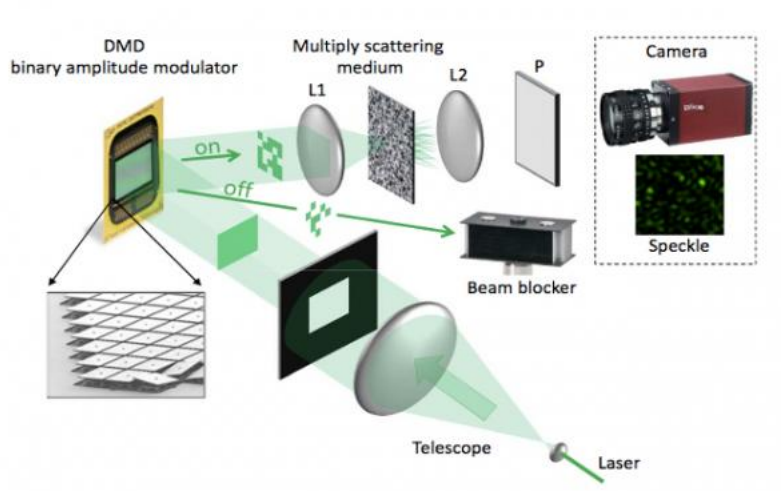

Figure 2.1 OPU's Experimental setup (Saade et al., 2016): A monochromatic laser is expanded by a telescope, then illuminates a digital micromirror device (DMD), able to spatially encode digital information on the light beam by amplitude modulation. The light beam carrying the signal is then focused on 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 .

is the principle concept behind OPU's functionality as seen in Fig 2.1.

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.

2.5 Mean kernel and Maximum Mean Discrepancy

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

$$\mathcal{K}(P,Q) = \mathbb{E}_{x\sim P, y\sim Q}\mathcal{K}(x,y) \tag{2.10}$$

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

$$MMD(P,Q) = \sqrt{\mathcal{K}(P,P) + \mathcal{K}(Q,Q) - 2\mathcal{K}(P,Q)} \tag{2.11}$$

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

Chapter Three

Theoretical Analysis

3.1 Computational Cost Of Graphlet Kernel With Graph Sampling

As indicated in section 2.2.2, to evaluate the k-graphlet kernel for a graph dataset we need to compute the k-spectrum vector for each graph G in this dataset. We also pointed out that to do so, $\binom{n}{k}$ subgraphs of size k must be enumerated from each graph G of size n in the dataset, which is extremely expensive. However, we can use graph sampling proposed in section 2.2.3 to accelerate this process by estimating the k-spectrum vector for each graph. We can do that by just randomly enumerate N_s subgraph samples of size k instead of exhaustive enumeration. The question here is how many samples we should consider in order to have a desired certainty in our estimation.

Theorem 2. *Let f be a probability distribution on the finite set of k-nodes graphlets $\mathcal{G} = \{g_1, ..., g_{N_k}\}$, $(F_j)_{j=1}^{N_s}$ be a set of independent identically distributed (iid) random variables drawn from f, and $\hat{f}(g_i) = \frac{1}{N_s}\Sigma_{j=1}^{N_s}\mathbb{1}(F_j = g_i)$. Then for a given $\epsilon > 0$ and $\delta > 0$ we have (Shervashidze et al., 2009):*

$$N_{s,min} = \left\lceil \frac{2(log(2)N_k + log(\frac{1}{\delta}))}{\epsilon^2} \right\rceil \tag{3.1}$$

samples suffice to ensure that $P(MMD(f, \hat{f}) \geq \epsilon) = P(\|f - \hat{f}\|_2 \geq \epsilon) \leq \delta$

We denote by $\phi_k^{hist.}$ the function on size-k graphs that identifies the corresponding graphlet and serves to construct the corresponding histogram:

$$\phi_k^{hist.}(F) = \left[1_{(F=g_i)}\right]_{i=1}^{N_k} \in \{0,1\}^{N_k}$$

In other words, $\phi_k^{hist.}$ puts a 1 in the coordinate i if $F = H_i$ and 0 otherwise. Here, equality between F and H_i is to be taken up-to-isomorphism. And we recall here that MMD(P,Q) is the maximum mean discrepancy metric explained in section 2.5. Therefore this theorem gives a lower bound on the number of size-k graphlet samples needed to approximate the k-spectrum vector with respect to MMD metric, with certainty ϵ and probability $(1-\delta)$. When ϵ, δ are reasonably chosen, this bound is better than exhaustively enumerating all possible $\binom{n}{k}$ graphlets, especially that N_k is exponential function of the graphlets size k not of the graph size n. However, when certainty intervals are required to be narrow with high probability (i.e. small values of ϵ, δ), or/and when high value of k is required then this method is still computationally expensive. In addition, we must not forget the computational cost of mapping the sampled graphlets to their isomorphic correspondent in the list $\mathcal{G}$ (isomorphism test).

Issues. In short, we identify two issues with the traditional graphlet kernel:

- For general k, N_k is at least exponential in k and the function $\phi_k^{hist.}$ itself is expensive to compute. Moreover, graphlet kernel still expensive when using graph sampling to approximate k-spectrum vector with high certainty.
- The inner product in Eq. 2.4 and its associated Euclidean metric (not MMD metric) do not take into account a notion of *similarity* between the graphlet themselves, they just compare the frequency counts for each graphlet, independently from the other.

We address both question by replacing the function ϕ_k^{hist} used to compute the k-spectrum vector f_G with an efficient, randomized high-dimensional mapping.

3.2 Proposed Algorithm

The proposed algorithm can be seen as the combination of the notions presented in chapter 2, which are mean kernel associated with MMD metric, size-k graphlets spectrum and random features. We first combine the mean kernel with random features, then we combine the mean kernel with discrete probability distributions (graphlets distribution), and finally we integrate all in the final algorithm.

We recall that for two probability distributions P, Q, the mean kernel is written as follows:

$$\mathcal{K}(P,Q) = \mathbb{E}_{x\sim P, y\sim Q}\mathcal{K}(x,y) \tag{3.2}$$

If the kernel has the following form (as in the case of continuous and shift-invariant kernels in Eq. 2.7):

$$\mathcal{K}(x,y) = \mathbb{E}_{\omega\sim\Lambda} z_\omega(x)^* z_\omega(y) \tag{3.3}$$

Then it is immediate that we have:

$$MMD(P,Q)^2 = \mathbb{E}_\omega\Big(|\mathbb{E}_P z_\omega(x) - \mathbb{E}_Q z_\omega(x)|^2\Big) \tag{3.4}$$

Now given data points $(x_1,\ldots,x_{N_s})$ drawn *iid* (identically and independently) from P and $(y_1,\ldots,y_{N_s})$ drawn *iid* from Q, the kernel in Eq. 3.2 can naturally be approximated by:

$$\mathcal{K}(P,Q) \approx \frac{1}{N_s^2}\sum_{i,j=1}^{N_s}\mathcal{K}(x_i,y_j) \tag{3.5}$$

And the corresponding approximate MMD is (other variants exist):

$$MMD(P,Q) \approx \sqrt{\frac{1}{{N_s}^2}\sum_{i,j=1}^{N_s}\mathcal{K}(x_i,x_j)+\mathcal{K}(y_i,y_j)-\mathcal{K}(x_i,y_j)-\mathcal{K}(x_j,y_i)}$$

3.2.1 MMD And Random Features

Mean kernel works especially well with random features. Combining (2.5) and (3.5), it is not hard to see that using random features the mean kernel can be further approximated by:

$$\mathcal{K}(P,Q) \approx \frac{1}{{N_s}^2}\sum_{i,j=1}^{N_s} z(x_i)^* z(y_j) = \left(\frac{1}{N_s}\sum_i z(x_i)\right)^* \left(\frac{1}{N_s}\sum_i z(y_i)\right) \tag{3.6}$$

So the computation can be drastically improved by first computing the *averaged random features* (also called random *generalized moments*, also called *sketch*) $\frac{1}{N_s}\sum_i z(x_i)$, and taking a linear kernel between them. The corresponding MMD is then just the Euclidean metric between the averaged random features

$$MMD(P,Q) \approx \|\frac{1}{N_s}\sum_i z(x_i) - \frac{1}{N_s}\sum_i z(y_i)\|_2$$

3.2.2 MMD For Discrete Distributions

For a discrete space of objects $\{g_1,\ldots,g_N\}$ with discrete probability distributions $P = \{P_1,\ldots,P_N\}$ and $Q = \{Q_1,\ldots,Q_N\}$ on them, the mean kernel 3.2 takes a particular form:

$$\mathcal{K}(P,Q) = \sum_{i,j=1}^{N} P_i Q_j k(g_i,g_j)$$

3.2.3 MMD With Random Features On Discrete Distributions

To combine all notions in the final algorithm, one can see the link with graphlet sampling, where f_G is the (discrete) probability distribution of the graphlets. If we define $\mathcal{K}(F,F') \approx z(F)^* z(F')$ where z is a random feature map that replaces ϕ_k^{hist}, then the feature map is exactly what appears in (3.6). So, now, all that left is to construct the aforementioned feature map $z(F)$ for graphlets. The induced MMD metric between graphs is the MMD between graphlets probability distributions f_G:

$$d(G,G') = MMD(f_G,f_{G'}) = \sqrt{\mathcal{K}(f_G,f_G) + \mathcal{K}(f_{G'},f_{G'}) - 2\mathcal{K}(f_G,f_{G'})} \approx \|\frac{1}{N_s}\sum_i z(F_i) - \frac{1}{N_s}\sum_i z(F_i')\|_2$$

where F_i are graphlets drawn from G and F_i' are graphlets drawn from G'.

. . (here there is a missing part on how to theoretically connect OPUs random features to this algorithm).Beside, the algorithm in its latex algorithmic form to be stated).

We need a big picture. The reader is very much lost here.

3.3 Concentration Analysis

For us, it is necessary to see how much, given two graphs G and G', $\|\frac{1}{N_s}\sum_i z(F_i) - \frac{1}{N_s}\sum_i z(F'_i)\|_2$ is close to $d(G,G') = MMD(G,G')$. We assume that when we replace ϕ_k^{hist} of the graphlet kernel with the random features map z, then the Λ distribution in (3.3) satisfies that for each graph F we have:

$$0 \leq z_\omega(F) \leq 1, \forall \omega \sim \Lambda \tag{3.7}$$

This is a reasonable assumption since by definition $\forall F, F'; \phi_k^{hist}(F)^T \phi_k^{hist}(F') \in [0,1]$.

Lemma 1. *Let G and G' be two graphs, $\{F_i\}_{i=1}^{N_s}$ (resp. $\{F_i\}_{i=1}^{N_s}$) be iid size-k graphlet samples drawn from G (resp. G'). We have that $\forall \epsilon > 0$:*

$$Pr\Big(\Big|\mathbb{E}_\omega\Big(|\mathbb{E}_{f_G} z_\omega(F) - \mathbb{E}_{f_{G'}} z_\omega(F')|^2\Big) - \|\frac{1}{N_s}\sum_i z(F_i) - \frac{1}{N_s}\sum_i z(F'_i)\|\Big| \geq \frac{2}{\sqrt{N_s}} + \epsilon\Big) \leq e^{-\frac{N_s\epsilon^2}{4}}$$

Proof. We decompose the proof in two steps.

Step 1: infinite N_s, finite m (number of random features). Based on our assumption on z_ω in (3.7), it is a straight forward result of Hoeffding's inequality that $d(G,G')^2$ is close to $\frac{1}{m}\sum_{j=1} |\mathbb{E}_{F\sim f_G} z_{\omega_j}(F) - \mathbb{E}_{F'\sim f_{G'}} z_{\omega_j}(F')|^2 = \|\mathbb{E}_{F\sim f_G} z(F) - \mathbb{E}_{F'\sim f_{G'}} z(F')\|^2$.

Lemma 2 (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:*

$$Pr(|\mathbb{E}\overline{X} - \overline{X}| \geq \epsilon) \leq 2\ exp(-\frac{2m^2\epsilon^2}{\Sigma_{i=1}^m (b_i - a_i)^2}) \tag{3.8}$$

In our case, and for a finite number of random features (m) we have the variables $x_j = |\mathbb{E}_{F\sim f_G} z_{\omega_j}(F) - \mathbb{E}_{F'\sim f_{G'}} z_{\omega_j}(F')|^2$ are independent and bounded by the interval $[0,1]$ too, thus it is an easy result to see that:

$$Pr\Big(\Big|\frac{1}{m}\sum_{j=1}^m |\mathbb{E}_{F\sim f_G} z_{\omega_j}(F) - \mathbb{E}_{F'\sim f_{G'}} z_{\omega_j}(F')|^2 - \mathbb{E}_\omega |\mathbb{E}_P z_\omega(x) - \mathbb{E}_Q z_\omega(x)|^2\Big| \geq \epsilon\Big) \leq 2\, e^{-2m\epsilon^2} \tag{3.9}$$

Step 2: finite N_s and m. We show that for any *fixed* set of random features ω_j, we have $\|\mathbb{E}_{F\sim f_G} z(F) - \mathbb{E}_{F'\sim f_{G'}} z(F')\|$ close to $\|\frac{1}{N_s}\sum_i z(F_i) - \frac{1}{N_s}\sum_i z(F_i')\|$.

Let us consider a fixed set of random variables $\{\omega_j\}_{j\in\{1,\dots,m\}}$ drawn independently from Λ, thus the random features map of a graph F equals: $z(F) = \frac{1}{\sqrt{m}}\left[z_{\omega_j}(F)\right]_{j=1}^{m}$.

For every graph G , let $F_1, \dots, F_{N_s}$ be N_s random subgraphs drawn independently from G, we clearly have:

$$\mathbb{E}_{F\sim f_G} z(F) = \mathbb{E}(\ \frac{1}{N_s}\sum_i z(F_i)\ ) \tag{3.10}$$

What should be noticed now to be used later is that $\forall F \sim f_G, z(F)$ is in a ball $\mathcal{H}$ of radius $M = \frac{\sqrt{m}}{\sqrt{m}} = 1$.

Lemma 3. *let $X = \{x_1, \dots, x_{N_s}\}$ be iid random variables in a ball $\mathcal{H}$ of radius M centered around the origin in a Hilbert space. Denote their average by $\overline{X} = \frac{1}{N_s}\sum_{i=1}^{N_s} x_i$. Then for any $\delta > 0$, with probability at lest $1-\delta$,*

$$\|\overline{X} - \mathbb{E}\overline{X}\| \leq \frac{M}{\sqrt{N_s}}(1 + \sqrt{2\ log\frac{1}{\delta}} \tag{3.11}$$

Proof. Defining the function $f(x) = \|\overline{X} - \mathbb{E}\overline{X}\|$, and $\widetilde{X} = x_1, \dots, \widetilde{x_i}, \dots, x_{N_s}$ 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:

$$|f(X) - f(\widetilde{X})| = \left|\|\overline{X} - \mathbb{E}\overline{X}\| - \|\overline{\widetilde{X}} - \mathbb{E}\overline{X}\|\right| \leq \|\overline{X} - \overline{\widetilde{X}}\| \leq \frac{\|x_i - \widetilde{x_i}\|}{N_s} \leq \frac{2M}{N_s} \tag{3.12}$$

Therefor, $f(X)$ is insensitive to the ith component of X, $\forall i \in \{1, \dots, N_s\}$ which is an important requirement to apply 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:

$$\mathbb{E}\|\overline{X} - \mathbb{E}\overline{X}\|^2 = \frac{1}{n}(\mathbb{E}\|x\|^2 - \|\mathbb{E}x\|^2) \tag{3.13}$$

Also:

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

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

$$Pr_x\Big[f(X) - \frac{M}{\sqrt{N_s}} \geq \epsilon\Big] \leq Pr_x\Big[f(X) - \mathbb{E}f(X) \geq \epsilon\Big] \leq exp\Big(-\frac{N_s\epsilon^2}{2M^2}\Big) \tag{3.14}$$

Which is equivalent to (3.11) by setting $\delta = exp(-\frac{N_s\epsilon^2}{2M^2})$ and solving for ϵ. □

Now back to Eq. (3.10) and its corresponding assumptions that we made, we can directly apply lemma 3 (and more especially Eq.(3.14)) to get that:

$$Pr(\|\mathbb{E}_{F\sim f_G} z(F) - \frac{1}{n}\sum_i z(F_i)\| \geq \frac{1}{\sqrt{n}} + \epsilon) \leq e^{-\frac{n\epsilon^2}{2}} \tag{3.15}$$

Now applying the triangle inequality again yields:

$$\Big|\|\mathbb{E}_{F\sim f_G} z(F) - \mathbb{E}_{F'\sim f_{G'}} z(F')\| - \|\frac{1}{N_s}\sum_i z(F_i) - \frac{1}{N_s}\sum_i z(F_i')\|\Big| \leq$$

$$\|(\mathbb{E}_{F\sim f_G} z(F) - \frac{1}{N_s}\sum_i z(F_i)) - (\mathbb{E}_{F'\sim f_{G'}} z(F') - \frac{1}{N_s}\sum_i z(F_i'))\| \leq$$

$$\|\mathbb{E}_{F\sim f_G} z(F) - \frac{1}{N_s}\sum_i z(F_i)\| + \|\mathbb{E}_{F'\sim f_{G'}} z(F') - \frac{1}{N_s}\sum_i z(F_i')\|$$

Thus, since the two variables $\|\mathbb{E}_{F\sim f_G} z(F) - \frac{1}{N_s}\sum_i z(F_i)\|$ and $\|\mathbb{E}_{F'\sim f_{G'}} z(F') - \frac{1}{N_s}\sum_i z(F_i')\|$ are independent (as a direct result of our aforementioned assumption of independent random sampling), $\forall\epsilon > 0$ we have:

$$Pr(\|\mathbb{E}_{F\sim f_G} z(F) - \frac{1}{N_s}\sum_i z(F_i)\| \geq \frac{1}{\sqrt{N_s}} + \frac{\epsilon}{2}, \|\mathbb{E}_{F'\sim f_{G'}} z(F') - \frac{1}{N_s}\sum_i z(F_i')\| \geq \frac{1}{\sqrt{N_s}} + \frac{\epsilon}{2}) =$$

$$Pr(\|\mathbb{E}_{F\sim f_G} z(F) - \frac{1}{N_s}\sum_i z(F_i)\| \geq \frac{1}{\sqrt{N_s}} + \frac{\epsilon}{2})\ Pr(\|\mathbb{E}_{F'\sim f_{G'}} z(F') - \frac{1}{N_s}\sum_i z(F_i')\| \geq \frac{1}{\sqrt{N_s}} + \frac{\epsilon}{2}) \leq e^{-\frac{N_s\epsilon^2}{4}}$$

finally, we get as a straight result from above:

$$Pr(\Big|\|\mathbb{E}_{F\sim f_G} z(F) - \mathbb{E}_{F'\sim f_{G'}} z(F')\| - \|\frac{1}{N_s}\sum_i z(F_i) - \frac{1}{N_s}\sum_i z(F_i')\|\Big| \geq \frac{2}{\sqrt{N_s}} + \epsilon) \leq e^{-\frac{n\epsilon^2}{4}}$$

And that is true for any fixed set of random variables $\{\omega_j\}_{j\in\{1,\ldots,m\}}$ drawn independently from Λ. 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*. So we can write:

$$Pr(\Big|\mathbb{E}_\omega\Big(|\mathbb{E}_{f_G} z_\omega(F) - \mathbb{E}_{f_{G'}} z_\omega(F')|^2\Big) - \|\frac{1}{N_s}\sum_i z(F_i) - \frac{1}{N_s}\sum_i z(F_i')\|\Big| \geq \frac{2}{\sqrt{N_s}} + \epsilon) \leq e^{-\frac{N_s\epsilon^2}{4}}$$

□

3.4 Complexity analysis

Here is the place to mention what part of the original complexity the the OPUs reduce to O(1).

Chapter Four

Results and Discussion

In this chapter we introduce the experiments conducted in this work with a discussion of the results. At first, we present the results of a Synthetic Graph Dataset created based on Stochastic Block Model (SBM), then the results of some real-world datasets (DD, Mutag,..etc).

4.1 Results of Synthetic SBM Dataset

4.1.1 Stochastic Block Model (SBM)

SBM is commonly known in social sciences to model group structures in friendship graph networks. As a combination of the strict block model with a stochastic element, it was able to deal with imperfect group structures and noise of real world networks. The standard SBM does not only determine the likelihood of a specific group structure belonging to a certain network. The model is based on a generative model, which enables the user to generate other network instances from a given structure or allows the prediction of missing edges (Funke and Becker, 2019).

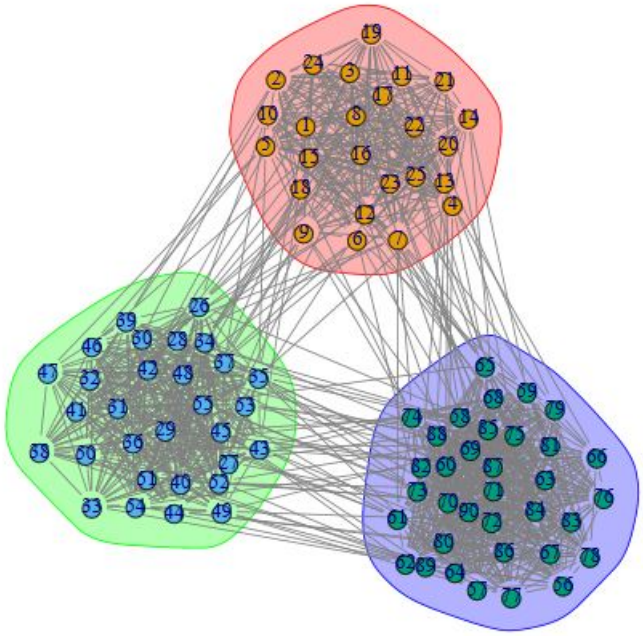

Figure 4.1 An example of a graph generated using SBM model, the graph has 90 nodes divided into three communities of size 25, 30 and 35 nodes. An edge between two nodes within the same community has a probability 0.8, while it has a probability 0.5 if the two nodes belong to different communities.

The basic idea of the standard SBM is that the neighborhood relations of each node only depend on the probabilities assigned to the model. Roughly speaking, the nodes are clustered in a way so that the neighbors of nodes in a group (community) have a similar neighbor pattern as well. To generate a graph G of size n using SBM model, the following parameters should be given: The number of communities (groups) in the graph L, node to community assignments $b_1, \ldots, b_n$ such that node i belongs to community b_i, edge probability matrix $(p_{i,j})_{i,j \in \{1,\ldots,L\}}$. Then the graph is generated by independently add an edge between any two pair of nodes (u, v) with probability p_{b_u, b_v}.

An easy thing to compute here is the average degree d of each node u in the graph $G = SBM(V_G, L, b_u, p_{i,j})$, which is equal to:

$$d_u = \sum_{b \neq b_u} p_{b_u,b} * (\#\{v \in V_G, b_v = b\}) + p_{b_u,b_u} * (\#\{v \in V_G, b_v = b_u\} - 1) \qquad (4.1)$$

In our case and almost in every experiment, unless the opposite is mentioned, the dataset

consists of 300 graphs constructed by SBM model each, each graph has $n = 60$ nodes divided equally between two communities $L = 2$. Other parameters $(p_{i,j})$ take different values in different graphs, and actually graphs are divided into different classes based on the corresponding values of these parameters. We consider only the case where the probabilities $p_{1,1} = p_{2,2} = p_{in}$. However, it is obvious that $p_{1,2} = p_{2,1} = p_{out}$ since we want an indirect graphs dataset as indicated previously.

The first class of graphs corresponds to a fixed pair $(p_{in,1}, p_{out,1})$ and similarly the second one corresponds to $(p_{in,2}, p_{out,2})$. These two pairs are always chosen so that any node in any graph in any dataset has an expected average degree equal to 10 (to preserve some difficulty in the classification problem). we refer to $r = (p_{in,1}/p_{in,2})$ by inter-classes similarity parameter, where the closer to one it is the more similar both classes are and thus the harder it is to discriminate them.

4.1.2 Graphlet kernel results

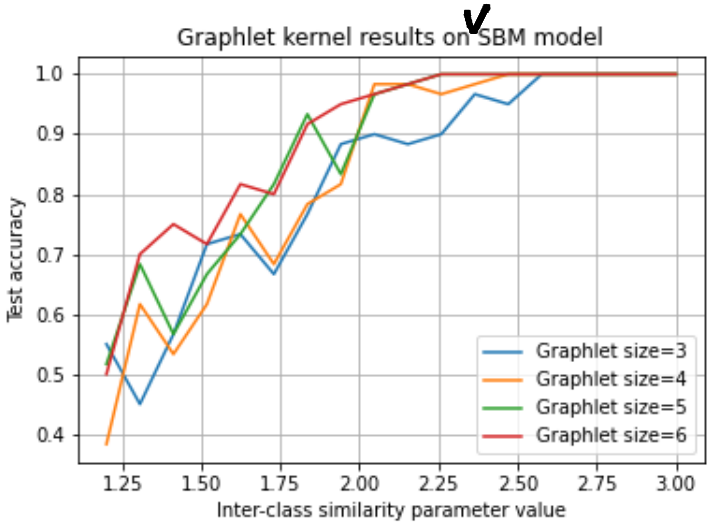

Figure 4.2 Graphlet kernel classification test accuracy with respect to Inter-classes similarity parameter r. where per graph G, 2000 graphlet samples are considered to compute its graphlet spectrum vector.

	3-GRAPHLETS	4-GRAPHLETS	5-GRAPHLETS	6-GRAPHLETS
Computational Time (Sec)	80	120	150	320

Table 4.1 Computational time per epoch of k-graphlet kernel with different k values.

4.1.3 OPUs' Random Features Results

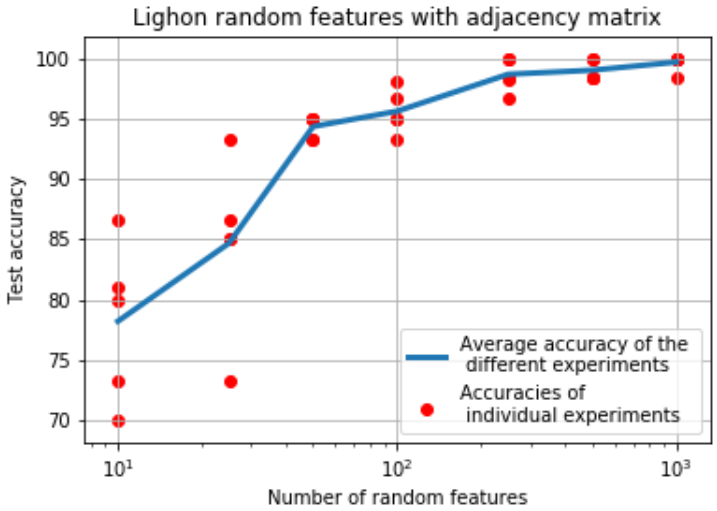

Figure 4.3 Classification test accuracy with respect to the number of LightOn random features. The SVM model is trained on SBM 240-sized labeled dataset. Per graph G, 2000 graphlet samples (Uniform sampling) of size 6 are considered to compute its features map $\phi(G)$. As expected, the accuracy variance drastically decrease as the number of random features increase.

Fixed inter-classes similarity and varying number of random features (Fig. 4.3) The interesting thing here with OPUs is that the computational time does not really depend on the number of random features as long as it is within the capacity of the OPU when we consider fixed graphlet size, in this experiment processing time was as specified in Table. 4.2.

COMPUTATIONAL TIME (SEC)	400 Sec

Table 4.2 Computational time per epoch of OPUs' random features based method.

Graphlet Size	3	4	5
Computational Time (Sec)	630	720	810

Table 4.3 Computational time per epoch of OPUs' random features method using Induced Random walk

Varying inter-classes similarity and fixed number of random features (Fig. 4.4) The computational time in this case is presented in Table. 4.4. We notice that there are slight differences between time values when the graphlet size change, this is due to two reasons, the first one is that OPUs' processing time does not depend on the dimension of the input nor on the number of random features as we respect its capacity, the second one is that we consider Uniform Sampling technique to sub-sample k-size graphlets, which is unlike Random Walk based sampling techniques fast and doesn't require significantly larger time when the graphlet size increases.

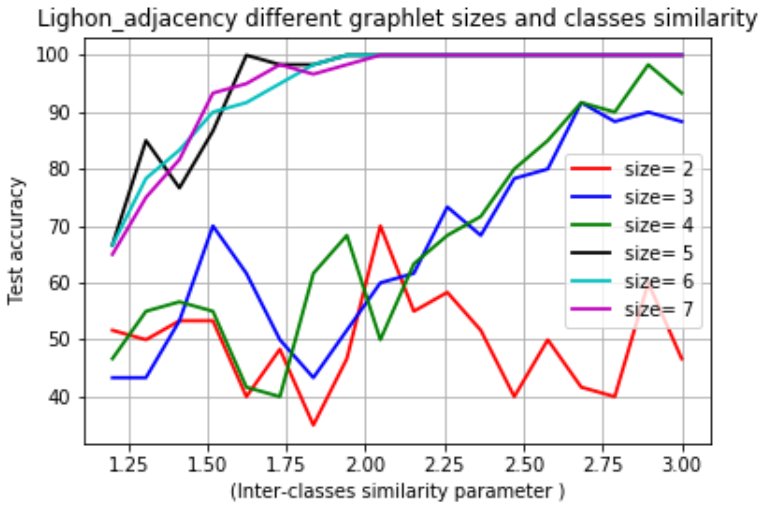

Figure 4.4 Classification test accuracy with respect to Inter-classes similarity parameter when the number of random features is fixed to 5000 but with different sizes of the graphlet to be sampled. The SVM model is trained on SBM 240-sized labeled dataset. Per graph G, 2000 graphlet samples (Uniform sampling) of the corresponding size are considered to compute its features map $\phi(G)$.

Graphlet Size	2	3	4	5	6	7
Computational Time (Sec)	340	357	362	378	400	412

Table 4.4 Computational time per epoch of OPUs' random features based method using uniform sampling technique.

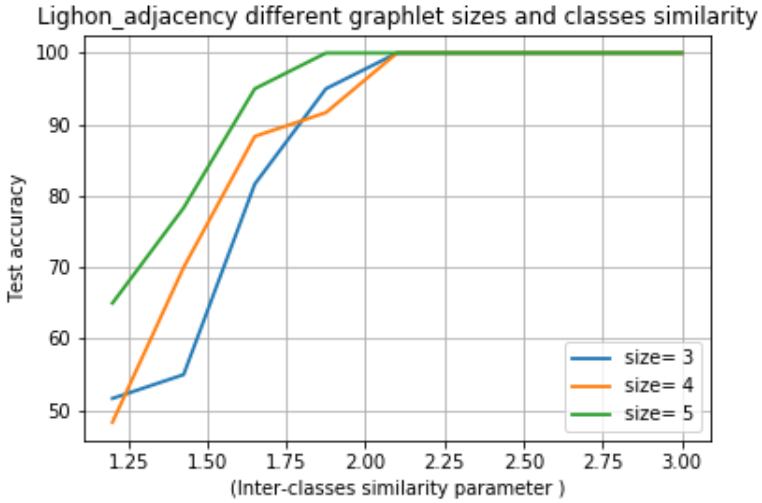

Figure 4.5 Classification test accuracy with respect to Inter-classes similarity parameter when the number of random features is fixed to 5000 but with different sizes of the graphlet to be sampled. The SVM model is trained on SBM 240-sized labeled dataset. Per graph G, 2000 graphlet samples (Random Walk sampling) of the corresponding size are considered to compute its features map $\phi(G)$. This experiment is done to check if the uniform sampling technique is the reason behind the gap between accuracy curves of graphlet sizes 4 and 5 in Fig. 4.4

However, one unexpected thing in Fig. 4.4 is that there is notably a gap between 4-graphlet and 5-graphlet curves, to detect the reason behind this gap we ran the same experiment with the same settings but using Induced Random Walk Sampling Technique , which tends to sample nodes that appear in a random walk starting from randomly chosen node (will be further explained in the report later). results are shown in Fig. 4.5 and the corresponding computational time is shown in Table. 4.5.

Varying number of graphlet samples and fixing other parameters .

Epoch processing time (Sec)	total training time (Sec)
0.29	87

Table 4.5 Computational time per epoch of the GCN GIN-based model.

..

.

4.1.4 Graph Convolution Network GCN

In order to benchmark the results of Random Features-based methods with other common methods used in Graph classification, we modified and trained one of the proposed models built based on GIN (Graph Isomorphism Network), which has been reported to give brilliant results (even better than most of state-of-art classical Graph Convolutional Networks) in classifying graphs based only on their structural information in the absence of node features just like the case of our SBM dataset (Xu et al., 2018).

Figure 4.6 GCN model's classification test accuracy with respect to Inter-classes similarity parameter. The model is trained on SBM 240-sized labeled dataset.

The model consists of 5 GIN layers followed by two fully connected layers where the

dimensions of hidden layers are equal to 4. Comparing GCN performance to the one we got using OPUs random features (Fig. 4.5 and Fig. 4.4) we can say that the performance of Random Features based methods is slightly better when the graphlet size is greater than 4, especially using Induced Random Walk sampling technique. However, the training time of GCN model still notably better (I should check if I can do something with LightOn platform to enhance its time, so the report will be fixed later).

4.2 DD Real-world Data set

4.2.1 Graphlet Kernel Results

.

.

.

4.2.2 OPUs' Random Features Results

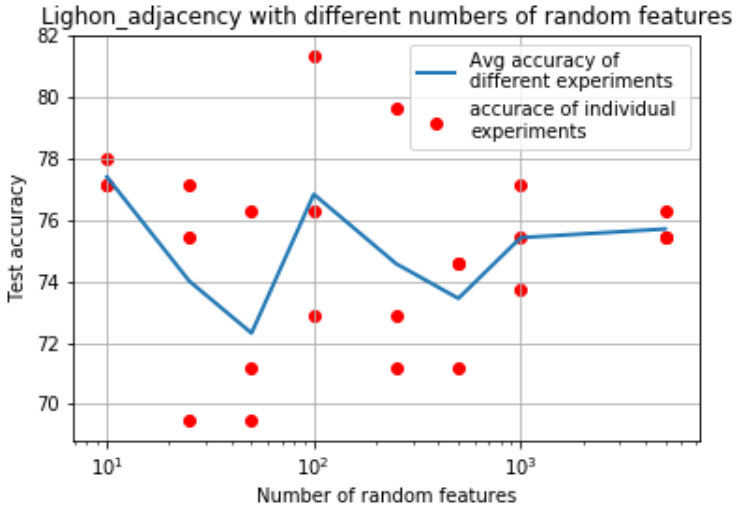

Figure 4.7 Classification test accuracy with respect to the number of random features on DD Dataset

4.2.3 Graph Convolutional Networks GCNs Results

.

.

.

REFERENCES

Funke, Thorben and Till Becker (2019). "Stochastic block models: A comparison of variants and inference methods". In: *PloS one* 14.4.

Kriege, Nils M, Fredrik D Johansson, and Christopher Morris (2020). "A survey on graph kernels". In: *Applied Network Science* 5.1, pp. 1–42.

Saade, Alaa et al. (2016). "Random projections through multiple optical scattering: Approximating kernels at the speed of light". In: *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, pp. 6215–6219.

Shervashidze, Nino et al. (2009). "Efficient graphlet kernels for large graph comparison". In: *Artificial Intelligence and Statistics*, pp. 488–495.

Xu, Keyulu et al. (2018). "How powerful are graph neural networks?" In: *arXiv preprint arXiv:1810.00826*.